Coalition Documentation

Release 4.0

Mercenaries Engineering SARL

Dec 06, 2022

Contents

Welcome to *"Coalition, a small but beautiful task manager"* documentation page. We hope you will find here all the information you need. If it's not the case, or if you want to contribute, you may contact the project's maintainers on the the project development platform.

CHAPTER 1

Introduction

Full online documentation is availlble on ReadTheDocs.

CHAPTER 2

Coalition

Coalition is a lightweight open source **job manager** client-server application whose role is to control **job execution in a set of computers**. A computer is acting as a **server** centralizing the list of jobs to be done. A set of physical (or virtual, eg. in the cloud) computers acting as **workers** shall be deployed, raising the global grid system ressources.

The server waits for incoming workers connections. Workers ask the server for a job to do. When the server is asked by a worker for a job, he decides which job to attribute according to simple **affinity rules**. The worker is now aware of which job it has to do. The worker executes the job. When the job is done, the worker informs the server of the job's execution status and ask for a new job.

Coalition should not be used on the public Internet but on **private LANs**, **cloud VLANs** or **VPN** for security reasons.

Coalition has been successfully used in production notably for **renderfarms**.

Coalition provides:

- **Broadcast discovery** for workers to find the server whithout configuration;
- **RESTfull python API** based on Twisted matrix for program to program communication;
- **Cloud ready** configuration to manage starting/termination of workers in the cloud;
- **Web interface** for humans to control jobs, workers, affinities and view status and logs;
- **Database** interface for sqlite and mysql;
- **Logging** system;
- **Email notification** system;
- **Access Control List** when connected to a **LDAP** server;
- **Unittests** of critical code parts;
- **Source code** and **documentation** on the development platform.

The current stable version are 3.8 and 3.10.

The development version is **|current-version|**.

CHAPTER 3

Installation

3.1 Installing the server

The *Coalition server* can be installed on a localhost or on a remote host. Please remember that communication between server and workers is not encrypted, so if the server is installed on localhost and workers on remote machines, using a VPN or a VLAN is a good idea.

Coalition works with python2.7.

3.1.1 Debian like, Ubuntu, etc. via system packages

Logged as a priviledged user, in a shell prompt, run:

```
apt-get install -y \
    python2.7 \
    python-httplib2 \
    python-configparser \
    python-twisted \
    python-mysqldb \
    python-ldap \
    python-sphinx \
    python-sphinxcontrib-httpdomain

cd /usr/local/bin
git clone https://github.com/MercenariesEngineering/coalition.git
cd coalition
cp _coalition.ini coalition.ini
```

Edit the section *[server]* in the file *coalition.ini* according to your needs.

You may want to fine tune the installation using:

- a dedicated system user and group to isolate the process and file ownership;
- a systemd service definition file;

- any system service monitoring daemon.

3.1.2 Via pip, the python package manager

Using a python virtual environment is advised in this case, although not mandatory.

Logged as a priviledged user, in a shell prompt, run:

```
cd /usr/local/bin
git clone https://github.com/MercenariesEngineering/coalition.git
cd coalition
pip install -r requirements.txt
cp _coalition.ini coalition.ini
```

Edit the section *[server]* in the file *coalition.ini* according to your needs.

You may want to fine tune the installation using:

- a dedicated system user and group to isolate the process and file ownership;
- a systemd service definition file;
- any system service monitoring daemon.

3.1.3 coalition.ini configuration file

This configuration file contains two sections: **[server]** that will be used in server mode, and **[worker]** that will be used while running in worker mode.

```
[server]
# Server configuration

# Type of database to use. "sqlite" for a file based database, "mysql" for an␣
↪external mysql server.
#db_type=sqlite

# The sqlite database file
#db_sqlite_file=coalition.db

# The mysql server
#db_mysql_host=127.0.0.1
#db_mysql_user=
#db_mysql_password=
#db_mysql_base=base
#db_mysql_install=1

# Server port (default is 19211)
#port=19211

# Server mode [normal|aws|gcloud] (default is normal)
#  If cloud mode is selected (all but "normal"), the corresponding
#  configuration file has to be edited.
#  eg. the file "aws_cloud.ini" for servermode="aws".
servermode=normal

# Worker time out in seconds, time lapse after a worker missing heartbeats is␣
↪considered out (default is 10)
```

(continues on next page)

(continued from previous page)

```
#timeout=10

# Run the server as service (Windows only)
#service=0

# Display verbose logs
#verbose=0

# Notify the user after the N first children jobs have been finished. 0 disables this␣
↪notification.
#notifyafter=10

# Decrease the priority of a parent job after N errors.
#decreaseprorityafter=10

# SMTP server hostname, emails disabled if empty
#smtphost=

# SMTP server port
#smtpport=587

# SMTP use a TLS connection
#smtptls=1

# SMTP sender email
#smtpsender=

# SMTP server login, no authentification if empty
#smtplogin=

# SMTP server password, no authentification if empty
#smtppasswd=

### LDAP configuration ###
# LDAP server to use for authentication
# If not empty, coalition will require a login and a password at every requests
; ldaphost=ldap://localhost

# Set to True to prevent password validation for API requests.
# Useful for scripts using API as it prevents hard-writing passwords.
# Authentification is still required while serving index.html to force web frontend␣
↪users login.
; ldapunsafeapi=True

# LDAP base (used for searches)
; ldapbase=dc=ldap,dc=localhost,dc=lan

# LDAP template used to validate the user, eg.
# uid=cn=__login__,ou=people,dc=example,dc=com
# where __login__ will be replaced by the user login
; ldaptemplatelogin = cn=__login__,dc=ldap,dc=localhost,dc=lan

### Group permissions ###
# Permissions are defined following the generic CRUD actions.
# Two major modes are predefined: per user or global.
# Per user mode offers CRUD actions only for jobs owned by the user.
# Global mode offers CRUD actions to the user for any job.
```

(continues on next page)

(continued from previous page)

```
#
# Per user actions:
# createjob: User can create jobs owned by himself.
# viewjob: User can see his jobs.
# editjob: User can edit his jobs.
# deletejob: User can delete his jobs.
#
# Global actions:
# createjobglobal: User can create a job owned by any other user.
# viewjobglobal: User can see any job.
# editjobglobal: User can edit any job.
# deletejobglobal: User can delete any job.
#
# LDAP template are used to validate that the user belongs to a specific group
#
# For instance, 3 permission groups can be defined in LDAP this way:
# 1. administrators: Can create, view, edit and delete any job.
# 2. wranglers: Can create, view and edit any job (but not delete).
# 3. artists: Can create, view, edit and delete only his own jobs.
#
# __login__ is replaced by the username.

; ldaptemplatecreatejob=(& (cn=artists) (member=cn=__login__,dc=ldap,dc=localhost,
↪dc=lan) )
; ldaptemplateviewjob=  (& (cn=artists) (member=cn=__login__,dc=ldap,dc=localhost,
↪dc=lan) )
; ldaptemplateeditjob=  (& (cn=artists) (member=cn=__login__,dc=ldap,dc=localhost,
↪dc=lan) )
; ldaptemplatedeletejob=(& (cn=artists) (member=cn=__login__,dc=ldap,dc=localhost,
↪dc=lan) )

; ldaptemplatecreatejobglobal=(& (| (cn=administrators) (cn=wranglers) ) (member=cn=__
↪login__,dc=ldap,dc=localhost,dc=lan) )
; ldaptemplateviewjobglobal=  (& (| (cn=administrators) (cn=wranglers) ) (member=cn=__
↪login__,dc=ldap,dc=localhost,dc=lan) )
; ldaptemplateeditjobglobal=  (& (| (cn=administrators) (cn=wranglers) ) (member=cn=__
↪login__,dc=ldap,dc=localhost,dc=lan) )
; ldaptemplatedeletejobglobal=(& (cn=administrators) (member=cn=__login__,dc=ldap,
↪dc=localhost,dc=lan) )

# Command white list.
# For global permission, use "global" as first parameter
# For per user permission, use "<username>" as first paramater

# Global and per user command while list.
#commandwhitelist=global command1 regexp
# global command2 regexp
# @user1
# user1 command1 regexp The command white list can be global or per user.
# user1 command2 regexp
# @user2
# user2 command1 regexp
# user2 command2 regexp

[worker]
# Worker configuration
```

(continues on next page)

(continued from previous page)

```
# Server URL, like http://serverhost:19211, let it blank to use autodetection by␣
↪broadcasting.
#serverUrl=

# Number of simultaneous workers on this system (default is 1)
#workers=1

# Worker name (default is host name)
#name=MyWorker

# Sleep time between two heartbeats in seconds (default is 2)
#sleep=2

# Maximum number of cpus per worker, will override the number of workers when defined␣
↪(Windows only)
#cpus=None

# Command to execute at worker startup
#startup=

# Display verbose logs
#verbose=0

# Customize the command used to run the job.
#
# The command set in runcommand is responsible for :
# * changing the user
# * changing the working directory
# * run the job command with the current environment
#
# The following pattern will be replaced:
# __user__ : the job user name
# __dir__ : the job directory
# __cmd__ : the job command
#
# If runcommand is blank, the worker set the working directory to the job directory
# and run the command using the worker permissions.

# Default run command
#runcommand=

# Run the jobs using sudo
#runcommand=sudo -u __user__ -E -- sh -c 'cd __dir__; __cmd__'

# Workers log file
#logfile=./worker.log
```

3.1.4 Cloud mode

A coalition server must be installed and the cloud provider needs configuration. See *cloud mode documentation page* for details.

3.1.5 Database

A database must be setup for the coalition server. To initialize it the firest time, run:

```
python server.py --verbose --init
```

The database can be reset on demand. All data are lost:

```
python server.py --verbose --reset
```

See also the next section about migrations.

3.1.6 Update coalition to a new release

If you update the coalition source code with a more recent coalition release, the new coalition features may need a database schema update. If it's the case, you will be informed by a message while trying to run the server:

```
python server.py --verbose --init
# ...
# The database requires migration
```

In this case, you should use the *–migrate* option to explicitely reconfigure the database:

```
python server.py --verbose --init
# ...
# Migration was sucessful
```

3.1.7 Running the server

When all has been set up, run:

```
python server.py
```

To see available command line arguments, run:

```
python server.py --help
```

On windows, use one those options:

```
python server.py --console
python server.py --service
```

3.2 Installing a worker

The same procedure than above in *installing a server* applies, except for configuration and running.

3.2.1 Configuration

Edit the section *[worker]* of the configuration file *coalition.ini* according to your needs.

You may want to fine tune the installation using:

- a dedicated system user and group to isolate the process and file ownership;
- a systemd service definition file;
- any system service monitoring daemon.

3.2.2 Running a worker

Run:

```
python worker.py --verbose
```

CHAPTER 4

Cloud mode

In this setup, the coalition server is allowed to start and delete instances so that all the jobs get done with minimum costs. Here, we install the coalition server on a dedicated cloud instance (instead of locahost). This way we simplify the network setup as we don't need a VPN or VLAN.

4.1 Configuration

4.1.1 Amazon Cloud

First, an initial setup is required on the cloud provider side. We provide here a minimal working setup. It can of course be enriched by your specificid needs and policy.

1. **Amazon account**

   To be allowed to manage cloud instances (ie. starting and terminating), the coalition server needs authentication.

   - from you amazon cloud account, visit the section *Manage security credentials*
   - get an access Keys (ID and Secret key) as text file

   You might prefer to create a dedicated user instead of your global user account.

2. **Virtual Private Cloud (VPC)**

   For the workers and the server to communicate securely, we use a common VPC:

   - create a new VPC

3. **Security Groups**

   The coalition's server and workers communication port defaults to 19211. The server should be accessible by the user (to interact with the API and/or web frontend) and from workers. The server can be hosted in the office or in the cloud, according to your network policy. Here, the server is instanciated in the cloud, belongs to the security group *sg-coalition* and the workers belong to the security group *sg-worker*. The workers should be accessible from the server only. So, the security groups and inbound rules are like those:

   - create a security group *sg-coalition*

  - Inbound Rules: TCP 19211 sg-worker
  - Inbound Rules: TCP 19211 office-public-IP
- create security group *sg- worker*
  - Inbound Rules: TCP 19211 sg-coalition

4. **Setup Coalition server as a cloud instance**

Now that the cloud provider has been set up, the coalition server has to be configured accordingly.

- install a coalition server on a cloud instance as explained in *Installation* documentation page
- edit the file **coalition.ini** in the **[Server]** section and set:

```
servermode = aws
```

- copy the file **_cloud_aws.ini** to **cloud_aws.ini**
- edit the file **cloud_aws.ini**

The configuration file **cloud_aws.ini** is self-explanatory. Set the options with your own amazon parameters:

```
# Configuration file for aws cloud

[authentication]
# Aws ssh key pair name
keyname=
# Accesskey
accesskey=
# Secretkey
secretaccesskey=

[storage]
# Storage name
name=
# Mountpoint in the worker
mountpoint=/mnt/bucket
# Location of the guerilla installer in the storage
guerillarenderfilename=srv/guerilla_render_2.0.0a13_linux64.tar.gz
# Location of the coalition installer in the storage
coalitionfilename=srv/coalition.tar.gz

[coalition]
# Coalition server IP
ip =
# Coalition server port
port = 19211
# Maximum number of simultaneous workers
workerinstancemax=3
# Delay in seconds between creation of instances.
# This prevents massive instances creation for big list of short time jobs.
# Default is 30 seconds.
workerinstancestartdelay=30
# Minimum lifetime in seconds before allowing the termination of useless
# worker instances. Since an instance requires several minutes to start,
# this option offers the possibility of keeping instances ready even during
# a short time without jobs.
# Default is 900 seconds = 15 minutes.
workerinstanceminimumlifetime=900
```

(continues on next page)

(continued from previous page)

```
[worker]
# Prefix for the new instance name
nameprefix=cloud-
spot=true
# Instance type
# https://aws.amazon.com/ec2/instance-types/
# http://www.ec2instances.info/
instancetype=m3.medium
# Aws image, for instance debian-stretch-amd64-hvm-2016-09-23-08-48-ebs
imageid=ami-2f40bd40
# Aws subnet
subnetid=
# Aws instance profile
iaminstanceprofile=
# Aws security group
securitygroupid=
availabilityzone=
[spot]
# http://docs.aws.amazon.com/cli/latest/reference/ec2/request-spot-instances.html
# https://aws.amazon.com/ec2/spot/pricing/
spotprice=10
instancecount=1
type=one-time
```

5. **Bucket**

   As workers are instanciated on demand, they need to fetch startup configuration files somewhere. Besides, as the workers might produce some data files (for example in a renderfarm usecase), those files must be saved in a filer. We create a bucket for that:

   - create a bucket
   - prepare the startup configuration files in the bucket
     - create a directory **srv**
     - copy the coalition source code into the **srv** directory:
       - download coalition source code as a zip file (or use the git source you got while installing the server)
       - unzip the file
       - copy **_coalition.ini** into **coalition.ini** and edit the **[worker]** section
       - recompress and pack it as a tar compressed file
       - copy **coalition.tar.gz** to the bucket: **srv/coalition.tar.gz**
     - in this setup, we build a guerilla render cloud renderfarm, so the worker needs the guerilla render binary:
       - copy **guerilla_render_2.0.0a13_linux64.tar.gz** to **srv/guerilla_render_2.0.0a13_linux64.tar.gz**

4.1.2 Google cloud

1. **Google cloud account**

- login on *google cloud console <https://console.cloud.google.com>*
- create a new project eg. **guerilla-cloud**
- get the json key file for the service account (menu IAM & Admin > Service accounts > Options > Create keys)

2. **Networking**

   We want to be able to visit the coalition server web frontend, so we need to allow remote connection from our office.

   - add a firewall rule allowing office IP on port tcp:19211

3. **Setup Coalition server as a google cloud instance**

Now that the cloud provider has been set up, the coalition server has to be configured accordingly.

- install a coalition server in a compute cloud instance as explained in *Installation* documentation page
  - as the server will create and delete cloud instances, set the instance **access scope** to **Allow full acees to all Cloud APIs**
  - use a dedicated IP instead of an ephemeral one for permanent reachability
  - ssh access for copying coalition files can be done via google credentials:

  ssh -i ~/.ssh/google_compute_engine <coalition_server_ip>

- edit the file **coalition.ini** in the **[Server]** section and set:

```
servermode = gcloud
```

- copy the file **_cloud_gcloud.ini** to **cloud_gcloud.ini**
- edit the file **cloud_gcloud.ini**

The configuration file **cloud_gcloud.ini** is self-explanatory. Set the options with your own google parameters:

```
# Configuration file for google cloud

[authentication]
# Project name
;project=guerilla-cloud
# Location of json key file for service user got from developper interface
;keyfile=guerilla-cloud-34bf64e0149b.json
# Service account
;serviceaccount=19254862847-compute@developer.gserviceaccount.com
;scopes=default


[storage]
# Storage name
;name=guerilla-cloud-bucket
# Mountpoint in the worker
;mountpoint=/mnt/bucket
# Location of the coalition installer in the storage
;coalitionpackage=srv/coalition.tar.gz

[coalition]
# Coalition server IP
;ip = 10.132.0.2
# Coalition server port
;port = 19211
```

(continues on next page)

(continued from previous page)

```
# Maximum number of simultaneous workers
;workerinstancemax=3
# Delay in seconds between creation of instances.
# This prevents massive instances creation for big list of short time jobs.
# Default is 30 seconds.
;workerinstancestartdelay=30
# Minimum lifetime in seconds before allowing the termination of useless
# worker instances. Since an instance requires several minutes to start,
# this option offers the possibility of keeping instances ready even during
# a short time without jobs.
# Default is 900 seconds = 15 minutes.
;workerinstanceminimumlifetime=900

[main_program]
;package=srv/guerilla_render_2.0.0a13_linux64.tar.gz
;environment=GUERILLA=/usr/local/bin/guerillarender/data/usr/local/guerilla GUERILLA_
↪CLOUD_ROOT=/mnt/bucket

[worker]
# Install dir fr coalition and main program
;installdir=/usr/local/bin
# Prefix for the new instance name
;nameprefix=cloud-
;zone=europe-west1-d
;machinetype=f1-micro
;subnet=default
;preemptible=true
# maintenancepolicy must be TERMINATE if preemptible is true
;maintenancepolicy=TERMINATE
;image=debian-8-jessie-v20170308
;imageproject=debian-cloud
;bootdisksize=10
;bootdisktype=pd-standard
```

4. **Storage**

   As workers are instanciated on demand, they need to fetch startup configuration files somewhere. Besides, as the workers might produce some data files (for example in a renderfarm usecase), those files must be saved in a filer. We create a bucket for that:

   - create a bucket
   - prepare the startup configuration files in the bucket
     - create a directory **srv**
     - copy the coalition source code into the **srv** directory:
       - download coalition source code as a zip file (or use the git source you got while installing the server)
       - unzip the file
       - copy the service user json key file into the coalition directory
       - copy **_coalition.ini** into **coalition.ini** and edit the **[worker]** section
       - recompress and pack it as a tar compressed file
       - copy **coalition.tar.gz** to the bucket: **srv/coalition.tar.gz**

  - in this setup, we build a guerilla render cloud renderfarm, so the worker needs the guerilla render binary:
    * copy **guerilla_render_2.0.0a13_linux64.tar.gz** to **srv/guerilla_render_2.0.0a13_linux64.tar.gz**

4.2 Running coalition

The coalition server is now ready to manage workers in the cloud:

- start the server
- visit the web interface **http://<server_adress>:19211**
- add affinities
- add some jobs

Workers will automagically be instanciated, getting jobs, working and terminated according to the configuration until there are no more jobs in waiting state on the server.

4.2.1 Changing coalition server or worker configuration while running

On the server instance, edit the concerned configuration files **coalition.py** and **cloud_<cloud_provider>.py** and restart the server.

As the configuration for workers is set up in the **bucket**, edit the configuration file **coalition.py** and re-upload the **coalition.tar.gz** to the bucket. Newly started instances will immediately use the new configuration. You might want to manually terminate previous instances. The coalition server does not need restarting in this case since the file names in the bucket are unchanged.

4.3 Monitoring the cloud deployment

The coalition server limits the number of simultaneous instances to the configuration parameter **workerinstancemax** in **coalition.ini**. But if there is a configuration problem (for instance in the workers starting scripts located in the bucket), coalition server might not be reached by the workers. In this case, coalition server will keep starting instances. So, as long as the configuration is not confirmed, you are advised to check in your cloud provider console the the effective number of starting instances. Some limits can also be setup directly in the cloud provider preventing any excessive cloud usage.

On the web frontend, in the workers tab, clicking the button **Terminate** destroys the selected instances after confirmation.

4.4 Additional documentation for programmers

4.4.1 python cloud module

cloud.common

This module contains functions common to various cloud providers.

`cloud.common.`**`createWorkerInstanceName`**(*prefix*)
Return a unique name based on prefix and timestamp.

cloud.aws

This module provides functions used for aws service.

`cloud.aws.`**`startInstance`**(*name*, *config*)
Run the aws command to start a worker instance. Return the created instanceid in case of dedicated ec2 instance or the spotinstancerequestid in case of a spot instance.

`cloud.aws.`**`stopInstance`**(*name*, *config*)
Run the aws command to terminate the instance.

4.4.2 Amazon specific templates

cloud/aws_worker_cloud_init.template

```
#cloud-config

# This cloud-init template is used for aws workers's startup configuration.
# http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/user-data.html
# http://cloudinit.readthedocs.io/

fqdn: $hostname

repo_update: true
repo_upgrade: all

packages:
  - python2.7
  - python-pip
  - python-twisted
  - python-twisted-web
  - python-mysqldb
  - curl
  - s3fs

runcmd:
  - pip install awscli
  - AWS_ACCESS_KEY_ID=$access_key AWS_SECRET_ACCESS_KEY=$secret_access_key aws ec2 --
↪region $region create-tags --resources $$(curl http://instance-data/latest/meta-
↪data/instance-id) --tags Key=Name,Value=$hostname
  - mkdir -p $mount_point
  - chmod a+w $mount_point
  - echo $bucket_name:$access_key:$secret_access_key > /etc/passwd-s3fs
  - chmod 0640 /etc/passwd-s3fs
  - s3fs -o url=https://s3.amazonaws.com,enable_content_md5 $bucket_name $mount_point
  - cat $mount_point/$guerilla_render_filename | tar xzf - -C /tmp/
  - mv /tmp/guerillarender/data/usr/local/guerilla /usr/local/bin/
  - rm -rf /tmp/guerillarender
  - cat $mount_point/$coalition_filename | tar xzf - -C /tmp/
  - mv /tmp/coalition /usr/local/bin/
  - GUERILLA=/usr/local/bin/guerilla GUERILLA_CLOUD_ROOT=$mount_point /usr/bin/
↪python2.7 /usr/local/bin/coalition/worker.py http://$coalition_server_ip:$coalition_
↪server_port
```

cloud/aws_worker_spot_launchspecification.json.template

```
{
  "ImageId": "$image_id",
  "KeyName": "$keyname",
  "SecurityGroupIds": [ "$security_group_id" ],
  "InstanceType": "$instance_type",
  "UserData": "$user_data"
}
```

CHAPTER 5

Web interface

With a web browser visit:

```
http://<the-coalition-server-IP-or-hostname>:19211
```

If LDAP is configured in **coalition.ini**, you will be asked for a username and a password. Once logged in, some permissions are granted to you according to the LDAP policies.

By defaut no LDAP is required and any action are authorized.

You can select multiple lines while pressing <control> or <shift> key and mouse clic.

A double clic on a job will get you to the job log page.

The job page will remember your last jobs filtering criteria via localstorage so that you can refresh the page without loosing your customisations. Clear you browser's localstorage or click the button "reset filter" to revert to the default display.

In cloud mode, you can manually terminate worker instances with the "terminate" button in the workers tab.

CHAPTER 6

Python API

CHAPTER 7

REST API

The coalition server provides a REST API using json data.

7.1 Jobs

GET /api/jobs
Returns the root jobs (with parent=0).

Example request:

```
GET /api/jobs HTTP/1.1
Host: localhost
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
        {
                "id": 123,
                "parent": 0,
                "title": "Job 123",
                "command": "echo test",
                "dir": ".",
                "environment": "",
                "state": "WAITING",
                "worker": "worker-1",
                "start_time": 123456,
                "duration": 12,
                "ping_time": 123456,
                "run_done": 1,
                "retry": 10,
```

(continues on next page)

(continued from previous page)


```
                "timeout": 10000,
                "priority": 1000,
                "affinity": "LINUX",
                "user": "render",
                "finished": 0,
                "errors": 0,
                "working": 0,
                "total": 10,
                "total_finished": 0,
                "total_errors": 0,
                "total_working": 0,
                "url": "http://localhost/image.png",
                "progress": 0.5,
                "progress_pattern": "#%percent"
        },
        {
                "id": 124,
                "parent": 0,
                "title": "Job 124",
                "command": "echo test",
                "dir": ".",
                "environment": "",
                "state": "WAITING",
                "worker": "worker-1",
                "start_time": 123456,
                "duration": 12,
                "ping_time": 123456,
                "run_done": 1,
                "retry": 10,
                "timeout": 10000,
                "priority": 1000,
                "affinity": "LINUX",
                "user": "render",
                "finished": 0,
                "errors": 0,
                "working": 0,
                "total": 10,
                "total_finished": 0,
                "total_errors": 0,
                "total_working": 0,
                "url": "http://localhost/image.png",
                "progress": 0.5,
                "progress_pattern": "#%percent"
        }
]
```


Status Codes

- 200 OK – no error
- 500 Internal Server Error – error

GET /api/jobs/(**int:** *id*)
Returns the job (*id*) object.

Example request:

```
GET /api/jobs/123 HTTP/1.1
Host: localhost
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
      "id": 123,
      "parent": 0,
      "title": "Job 123",
      "command": "echo test",
      "dir": ".",
      "environment": "",
      "state": "WAITING",
      "worker": "worker-1",
      "start_time": 123456,
      "duration": 12,
      "ping_time": 123456,
      "run_done": 1,
      "retry": 10,
      "timeout": 10000,
      "priority": 1000,
      "affinity": "LINUX",
      "user": "render",
      "finished": 0,
      "errors": 0,
      "working": 0,
      "total": 10,
      "total_finished": 0,
      "total_errors": 0,
      "total_working": 0,
      "url": "http://localhost/image.png",
      "progress": 0.5,
      "progress_pattern": "#%percent"
}
```

Status Codes

- 200 OK – no error
- 500 Internal Server Error – error

PUT /api/jobs

Create a job. Returns the new job id.

Example request:

```
PUT /api/jobs HTTP/1.1
Host: localhost

{
      "parent": 0,
      "title": "Job 1",
      "command": "echo test",
      "dir": ".",
      "environment": "",
```

(continues on next page)

(continued from previous page)

```
      "state": "WAITING",
      "retry": 10,
      "timeout": 10000,
      "priority": 1000,
      "affinity": "LINUX",
      "user": "render",
      "url": "http://localhost/image.png",
      "progress_pattern": "#%percent"
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

123
```

Status Codes

- 200 OK – no error
- 500 Internal Server Error – error

`POST /api/jobs`

Modify the jobs properties.

Example request:

```
POST /api/jobs HTTP/1.1
Host: localhost

{
      123:
      {
              "title": "Job renamed 123",
              "command": "echo renamed",
      },
      124:
      {
              "title": "Job renamed 124",
              "command": "echo renamed",
      }
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

1
```

Status Codes

- 200 OK – no error
- 500 Internal Server Error – error

GET /api/jobs/(int: *id*)/children
Returns the job (*id*) children objects.

Example request:

```
GET /api/jobs/123/children HTTP/1.1
Host: localhost
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
      {
               "id": 124,
               "parent": 123,
               "title": "Job 124",
               "command": "echo test",
               "dir": ".",
               "environment": "",
               "state": "WAITING",
               "worker": "worker-1",
               "start_time": 123456,
               "duration": 12,
               "ping_time": 123456,
               "run_done": 1,
               "retry": 10,
               "timeout": 10000,
               "priority": 1000,
               "affinity": "LINUX",
               "user": "render",
               "finished": 0,
               "errors": 0,
               "working": 0,
               "total": 10,
               "total_finished": 0,
               "total_errors": 0,
               "total_working": 0,
               "url": "http://localhost/image.png",
               "progress": 0.5,
               "progress_pattern": "#%percent"
      },
      {
               "id": 125,
               "parent": 123,
               "title": "Job 125",
               "command": "echo test",
               "dir": ".",
               "environment": "",
               "state": "WAITING",
               "worker": "worker-1",
               "start_time": 123456,
               "duration": 12,
               "ping_time": 123456,
               "run_done": 1,
               "retry": 10,
               "timeout": 10000,
```

(continues on next page)

(continued from previous page)

```
                "priority": 1000,
                "affinity": "LINUX",
                "user": "render",
                "finished": 0,
                "errors": 0,
                "working": 0,
                "total": 10,
                "total_finished": 0,
                "total_errors": 0,
                "total_working": 0,
                "url": "http://localhost/image.png",
                "progress": 0.5,
                "progress_pattern": "#%percent"
        },
]
```

GET /api/jobs/(**int:** *id*)**/dependencies**
Returns the job objects on which the job (*id*) depends.

Example request:

```
GET /api/jobs/123/dependencies HTTP/1.1
Host: localhost
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
        {
                "id": 124,
                "parent": 0,
                "title": "Job 124",
                "command": "echo test",
                "dir": ".",
                "environment": "",
                "state": "WAITING",
                "worker": "worker-1",
                "start_time": 123456,
                "duration": 12,
                "ping_time": 123456,
                "run_done": 1,
                "retry": 10,
                "timeout": 10000,
                "priority": 1000,
                "affinity": "LINUX",
                "user": "render",
                "finished": 0,
                "errors": 0,
                "working": 0,
                "total": 10,
                "total_finished": 0,
                "total_errors": 0,
                "total_working": 0,
                "url": "http://localhost/image.png",
                "progress": 0.5,
```

(continues on next page)

(continued from previous page)


```
                "progress_pattern": "#%percent"
        },
        {
                "id": 125,
                "parent": 0,
                "title": "Job 125",
                "command": "echo test",
                "dir": ".",
                "environment": "",
                "state": "WAITING",
                "worker": "worker-1",
                "start_time": 123456,
                "duration": 12,
                "ping_time": 123456,
                "run_done": 1,
                "retry": 10,
                "timeout": 10000,
                "priority": 1000,
                "affinity": "LINUX",
                "user": "render",
                "finished": 0,
                "errors": 0,
                "working": 0,
                "total": 10,
                "total_finished": 0,
                "total_errors": 0,
                "total_working": 0,
                "url": "http://localhost/image.png",
                "progress": 0.5,
                "progress_pattern": "#%percent"
        },
]
```


POST /api/jobs/(int: *id*)/dependencies

Set the job (*id*) dependencies.

Example request:

```
POST /api/jobs/123/dependencies HTTP/1.1
Host: localhost

[124,125]
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

1
```

GET /api/jobs/(int: *id*)/log

Returns the job (*id*) log file.

Example request:

```
GET /api/jobs/123/log HTTP/1.1
Host: localhost
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

"Job 123 done"
```

Status Codes

- 200 OK – no error
- 500 Internal Server Error – error

DELETE /api/jobs

Delete the jobs.

Example request:

```
DELETE /api/jobs HTTP/1.1
Host: localhost

[123,124,125]
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

1
```

POST /api/resetjobs

Reset the jobs. The job status is set to 'WAITING', all the job counters are set to 0.

Example request:

```
POST /api/resetjobs HTTP/1.1
Host: localhost

[123,124,125]
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

1
```

POST /api/startjobs

Start the jobs. The job status is set to 'WAITING'.

Example request:

```
POST /api/startjobs HTTP/1.1
Host: localhost

[123,124,125]
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

1
```

POST /api/pausejobs
Pause the jobs. The job status is set to 'PAUSED'.

Example request:

```
POST /api/pausejobs HTTP/1.1
Host: localhost

[123,124,125]
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

1
```

7.2 Workers

GET /api/workers
Returns the workers.

Example request:

```
GET /api/workers HTTP/1.1
Host: localhost
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
      "name": "worker-1",
      "ip": "127.0.0.1",
      "affinity": "LINUX,WINDOWS",
      "state": "WAITING",
      "ping_time": 123456,
      "finished": 123,
      "error": 21,
      "last_job": 1234,
      "current_event": 1234,
      "cpu": "[0,0,0,0]",
      "free_memory": 123456,
      "total_memory": 1000000,
      "active": 1
}
```

Status Codes

- 200 OK – no error

- 500 Internal Server Error – error

POST /api/workers

Modify the workers properties.

Example request:

```
POST /api/workers HTTP/1.1
Host: localhost

{
      "worker-1":
      {
              "affinity": "LINUX",
              "active": 0,
      },
      "worker-2":
      {
              "affinity": "LINUX",
              "active": 0,
      }
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

1
```

Status Codes

- 200 OK – no error
- 500 Internal Server Error – error

DELETE /api/workers

Delete the workers.

Example request:

```
DELETE /api/workers HTTP/1.1
Host: localhost

["worker-1","worker-2"]
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

1
```

POST /api/stopworkers

Stop the workers.

Example request:

```
POST /api/stopworkers HTTP/1.1
Host: localhost

["worker-1","worker-2"]
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

1
```

POST /api/startworkers

Start the workers.

Example request:

```
POST /api/startworkers HTTP/1.1
Host: localhost

["worker-1","worker-2"]
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

1
```

7.3 Events

GET /api/events

Returns some events.

Parameters

- **job** – returns the events for this job.
- **worker** – returns the events for this worker.
- **howlong** – returns all the events in the last *howlong* seconds.

Example request:

```
GET /api/events?job=123&worker=worker-1&howlong=60 HTTP/1.1
Host: localhost
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
      "id": 123,
      "worker": "worker-1",
      "job_id": 123,
      "job_title": "Job 123",
```

(continues on next page)

(continued from previous page)

```
        "state": "ERROR",
        "start": 123456,
        "duration": 10
}
```

Status Codes

- 200 OK – no error
- 500 Internal Server Error – error

CHAPTER 8

Jobs

A job is a simple command to run on one of the workers. Each job has an ID chosen by the server.

A job can be submitted to the server using control.py or a HTTP request. Job attributes

A job may have different attributes. For example, the job attribute cmd is the command to execute.

The different job attributes used to initialise a job are:

- **cmd**: the command to run
- **title**: the title to display in the user interface
- **dir**: the job's working directory
- **env**: an environment for the command
- **parent**: the parent job ID
- **priority**: the job's priority
- **dependencies**: the job's dependencies on other jobs
- **timeout**: the job's timeout in seconds
- **retry**: how many time to retry this job
- **affinity**: the job affinities to match
- **url**: the url to open with the Open link
- **user**: the user name/email of the owner of this job
- **globalprogress**: the job progression pattern
- **localprogress**: the second job progression pattern

8.1 Job execution

A new job is in the **WAITING** state.

When a worker run a job, it set the current working directory to dir and run the command cmd. The job is then in the **WORKING** state. If the command returns with the exit code 0, the job is put in the state **FINISHED**. If not, the job is put in the state **ERROR**.

If the job is in the **ERROR** state, the server will retry to run this job up to retry times.

If the job duration exceeds timeout, the server will kill this job and set it in the **ERROR** state.

8.2 Job display

The title attribute is displayed in the user interface.

The url attribute is the url to open with the Open link in the user interface.

By default, the web browser blocks the URLs on local files.

On Firefox, *it is possible to override this behavior <http://kb.mozillazine.org/Links_to_local_pages_don%27t_work>*.

8.3 Job environment

An environment can be provided with a job using the **env** attribute. An environment is a string containing all the variables and their values. The separator is the string **"n"** (a '' character followed be a 'n' character, not an end of line character).

Here is a string you can use with the **env** attribute:

```
"USER=mylogin\nPATH=mypath"
```

8.4 Job hierarchy

The hierarchy is useful to organize and schedule the different jobs.

A job can be the parent of some children jobs. In this case, the parent job won't run any command. Even if the attribute **cmd** has been provided.

The parent attribute can be specified to create a job into a previously created parent job.

The parent attribute can be the parent job ID (an integer), a job title (a string) or a path of job titles. Exemples:

```
parent=12345 : add the new job to the job #12345
parent="departments" : add the new job to the job named "departments"
parent="departments|render" : add the new job to the job named "render" inside the␣
↪job named "departments"
```

8.5 Job dependencies

The dependencies attributes is a job ID list of the different jobs to finish before to run this job.

A list can be provided like this : "1,3,5".

8.6 Job affinities

Affinities are used to associate some jobs to a subset of workers.

The affinity attribute is a list of strings, separated by comas.

If a job has an affinity attribute, only the workers with the affinities matching all the job's affinities will be able to run this job.

For example, let's say a job has the following affinity : "LINUX,24GB". Here is a summary of which worker affinities configuration match the job's one:

```
| Job affinites | Worker affinities | Match | |:------------------|:------------------
↪----|:----------| | "LINUX,24GB" | "LINUX" | NO | | "LINUX,24GB" | "24GB" | NO | |
↪"LINUX,24GB" | "LINUX,24GB" | YES | | "LINUX,24GB" | "LINUX,24GB,GL" | YES |
```

8.7 Job owner

The user attributes is the user name of the owner of the job. If the emails are activated, the emails regarding this job will be sent at user.

If LDAP is configured, the job will be executed with the user rights.

8.8 Job log

The job's log is the output of the command's stdout and stderr streams. The log is sent by the worker to the server.

The server stores the log in the logs/ directory, in a file named ID.log with ID the ID of the job.

8.9 Job progression

The globalprogress attribute is a pattern that is used to extract the job progression out of the job's logs. Here are some examples of logs and patterns:

```
| Log | Pattern | Progression | |:--------|:------------|:----------------| | 25 |
↪%percent | 25% | | (50) | (%percent) | 50% | | 0.75 | %one | 75% | | P:1 | P:%one |␣
↪100% |
```

The localprogress attribute can be used with globalprogress to specify a second level of the job progression.

CHAPTER 9

Server

The coalition server collects the jobs and distributes them to the differents workers.

Run the server

9.1 Add a job

It is possible to add a job to the server using control.py or a HTTP request.

If the job is added, the new job ID is returned. Using control.py

You can use control.py to add jobs to the server:

```
python control.py --cmd="echo toto" --priority=1000 --affinity="linux" --retry=10␣
↪http://127.0.0.1:19211 add
```

9.2 Using a HTTP request

To add a job using the HTTP interface, simply GET or POST the url http://host:port/json/addjob with any job attributes.

Example:

```
http://127.0.0.1:19211/json/addjob?title=job&cmd=echo toto&priority=1000&
↪affinity=linux&retry=10
```

CHAPTER 10

Contribute

10.1 Development platform

Coalition is free software LGPL licensed and hosted on github. Feel free to participate via a github account.

10.2 Running tests

The test suite requires a database. To prevent a database overwriting, you should **first backup your current database or change the database reference in the coalition.ini** configuration file.

Run:

```
# Intialize a fresh database
python server.py --init

# Run the tests
python tests/main_tests.py
```

The status of the tests must show no errors.

A **.travis.yml** file is provided for automated testing via travis testing platform.

10.3 Build documentation

Go to the **coalition/doc** directory and run:

```
./build.sh
```

- genindex
- modindex

- search

HTTP Routing Table

/api

Python Module Index

C

Index

C

S