Cyme Documentation

Release 0.0.4

VMWare, Inc.

March 20, 2012

CONTENTS

Contents:

CHAPTER

ONE

INTRODUCTION

- Synopsis
- Requirements
- Getting started
- API Reference
  - Applications
  - Instances
  - Queues
  - Consumers
  - Queueing Tasks
  - Querying Task State
  - Instance details and statistics
  - Autoscale
- Components
  - Programs
  - Models
    - App
    - Broker
    - Instance
    - Queue
- Supervisor
- Controller
- HTTP

1.1 Synopsis

Cyme is the Celery instance manager, a distributed application that manages clusters of Celery worker instances. Every machine (physical or virtual) runs the `cyme-branch` service.

1.2 Requirements

- Python 2.6 or later.
- cl
- eventlet

- django
- django-celery

Note: An updated list of requirements can always be found in the `requirements/` directory of the Cyme distribution. This directory contains pip requirements files for different scenarios.

1.3 Getting started

You can run as many branches as needed: one or multiple. It is also possible to run multiple branches on the same machine by specifying a custom HTTP port, and root directory for each branch.

Start one or more branches:

```
$ cyme-branch :8001 -i branch1 -D cyme/branch1/

$ cyme-branch :8002 -i branch2 -D cyme/branch2/
```

The default HTTP port is 8000, and the default root directory is `instances/`. The root directory must be writable by the user the `cyme-branch` application is running as. The logs and pid files of every worker instance will be stored in this directory.

Create a new application named `foo`:

```
$ curl -X POST -i http://localhost:8001/foo/
HTTP/1.1 201 CREATED
Content-Type: application/json
Date: Mon, 15 Aug 2011 22:06:43 GMT
Transfer-Encoding: chunked

{"name": "foo", "broker": {"password": "guest",
                           "hostname": "127.0.0.1",
                           "userid": "guest",
                           "port": 5672,
                           "virtual_host": "/"}}
```

Note that we can edit the broker connection details here by using a POST request:

```
$ curl -X POST -i http://localhost/bar/ -d \
    'hostname=w1&userid=me&password=me&vhost=/'
```

Note: For convenience and full client support **PUT** can be replaced with a **POST** instead, and it will result in the same action being performed.

Also, for **POST**, **PUT** and **DELETE** the query part of the URL can be used instead of actual post data.

Create a new Celery worker instance:

```
$ curl -X PUT -i http://localhost:8001/foo/instances/
HTTP/1.1 201 CREATED
Content-Type: application/json
Date: Mon, 15 Aug 2011 15:25:11 GMT
Transfer-Encoding: chunked

{"is_enabled": true,
 "name": "a35f2518-13bb-4403-bbdf-dd8751077712",
```

```
 "queues": [],
 "broker": {"password": "guest",
            "userid": "guest",
            "hostname": "127.0.0.1",
            "virtual_host": "/",
            "port": 5672},
 "max_concurrency": 1,
 "min_concurrency": 1}
```

Note that this instance is created on a random branch, not necessarily the branch that you are currently speaking to over HTTP. If you want to edit the data on a specific branch, please do so by using the admin interface of the branch, at http://localhost:8001/admin/.

In the logs of the affected branch you should now see something like this:

```
{582161d7-1187-4242-9874-32cd7186ba91} --> Instance.add(name=None)
{Supervisor} wake-up
{Supervisor} a35f2518-13bb-4403-bbdf-dd8751077712 instance.restart
celeryd-multi restart --suffix="" --no-color a35f2518-13bb-4403-bbdf-dd8751077712
    -Q 'dq.a35f2518-13bb-4403-bbdf-dd8751077712'
    --workdir=cyme/branch1/
    --pidfile=cyme/branch1/a35f2518-13bb-4403-bbdf-dd8751077712/worker.pid
    --logfile=cyme/branch1/a35f2518-13bb-4403-bbdf-dd8751077712/worker.log
    --loglevel=DEBUG --autoscale=1,1
    --broker=amqp://guest:guest@localhost:5672//
celeryd-multi v2.3.1
> a35f2518-13bb-4403-bbdf-dd8751077712: DOWN
> Restarting instance a35f2518-13bb-4403-bbdf-dd8751077712: OK
{Supervisor} a35f2518-13bb-4403-bbdf-dd8751077712 pingWithTimeout: 0.1
{Supervisor} a35f2518-13bb-4403-bbdf-dd8751077712 pingWithTimeout: 0.5
{Supervisor} a35f2518-13bb-4403-bbdf-dd8751077712 pingWithTimeout: 0.9
{Supervisor} a35f2518-13bb-4403-bbdf-dd8751077712 successfully restarted
{Supervisor} wake-up
{582161d7-1187-4242-9874-32cd7186ba91} <-- ok={
    'is_enabled': True,
    'name': 'a35f2518-13bb-4403-bbdf-dd8751077712',
    'queues': [],
    'broker': {'password': u'guest',
               'hostname': u'127.0.0.1',
               'userid': u'guest',
               'port': 5672,
               'virtual_host': u'/'},
    'max_concurrency': 1,
    'min_concurrency': 1}
```

Now that we have created an instance we can list the available instances:

```
$ curl -X GET -i http://localhost:8001/foo/instances/
HTTP/1.1 200 OK
Content-Type: application/json
Date: Mon, 15 Aug 2011 15:28:33 GMT
Transfer-Encoding: chunked

["a35f2518-13bb-4403-bbdf-dd8751077712"]
```

Note that this will list instances for every branch, not just the branch you are currently speaking to over HTTP.

Let's create a queue declaration for a queue named `tasks`. This queue binds the exchange `tasks` with routing key `tasks`. (note that the queue name will be used as both exchange name and routing key if these are not provided).

Create the queue by performing the following request:

```
$ curl -X POST -d 'exchange=tasks&routing_key=tasks' \
    -i http://localhost:8001/foo/queues/tasks/
HTTP/1.1 201 CREATED
Content-Type: application/json
Date: Mon, 15 Aug 2011 16:03:07 GMT
Transfer-Encoding: chunked

{"exchange": "t2",
 "routing_key": "t2",
 "options": null,
 "name": "t2",
 "exchange_type": null}
```

The queue declaration should now have been stored inside one of the branches, and we can verify that by retrieving a list of all queues defined on all branches:

```
$ curl -X GET -i http://localhost:8001/foo/queues/
HTTP/1.1200 OK
Content-Type: application/json
Date: Mon, 15 Aug 2011 16:08:37 GMT
Transfer-Encoding: chunked

["tasks"]
```

Now we can make our worker instance consume from the `tasks` queue to process tasks sent to it:

```
$ curl -X PUT -i \
    http://localhost:8001/foo/instances/a35f2518-13bb-4403-bbdf-dd8751077712/queues/t2
HTTP/1.1 201 CREATED
Content-Type: application/json
Date: Mon, 15 Aug 2011 16:06:32 GMT
Transfer-Encoding: chunked

{"ok": "ok"}
```

In the logs for the branch that this is instance is a member of you should now see:

```
[2011-08-15 16:06:32,226: WARNING/MainProcess]
    {Supervisor} a35f2518-13bb-4403-bbdf-dd8751077712: instance.consume_from: tasks
```

If the test was successful you can clean up after yourself by,

- Cancelling consuming from the `tasks` queue:

  ```
  $ curl -X DELETE -i \
      http://localhost:8001/foo/instances/a35f2518-13bb-4403-bbdf-dd875107772/queues/tasks
  ```

- Deleting the `tasks` queue:

  ```
  $ curl -X DELETE -i http://localhost:8001/foo/queues/
  ```

- and finally, deleting the worker instance:

  ```
  $ curl -X DELETE -i http://localhost:8001/instances/a35f2518-13bb-4403-bbdf-dd8751077712/
  ```

The worker instance should now be shutdown by the branch supervisor.

1.4 API Reference

1.4.1 Applications

- Create new named application

```
[PUT|POST] http://branch:port/<name>/?hostname=str
                                     ?port=int
                                     ?userid=str
                                     ?password=str
                                     ?virtual_host=str
```

If `hostname` is not provided, then any other broker parameters will be ignored and the default broker will be used.

- List all available applications

```
GET http://branch:port/
```

- Get the configuration for app by name

```
GET http://branch:port/name/
```

1.4.2 Instances

- Create and start an anonymous instance associated with app

```
[PUT|POST] http://branch:port/<app>/instances/
```

This will return the details of the new id, including the instance name (which for anonymous instances is an UUID).

- Create and start a named instance associated with app:

```
[PUT|POST] http://branch:port/<app>/instances/<name>/
```

- List all available instances associated with an app

```
GET http://branch:port/<app>/
```

- Get the details of an instance by name

```
GET http://branch:port/<app>/instances/<name>/
```

- Delete an instance by name.

```
DELETE http://branch:port/<app>/instances/<name>/
```

1.4.3 Queues

- Create a new queue declaration by name

```
[PUT|POST] http://branch:port/<app>/queues/<name>/?exchange=str
                                                  ?exchange_type=str
                                                  ?routing_key=str
                                                  ?options=json dict
```

`exchange` and `routing_key` will default to the queue name if not provided, and `exchange_type` will default to `direct`. `options` is a json encoded mapping of additional queue, exchange and binding options, for a full list of supported options see `kombu.compat.entry_to_queue()`.

- Get the declaration for a queue by name

```
GET http://branch:port/<app>/queues/<name>/
```

- Get a list of available queues

```
GET http://branch:port/<app>/queues/
```

1.4.4 Consumers

Every instance can consume from one or more queues. Queues are referred to by name, and there must exist a full declaration for that name.

- Tell an instance by name to consume from queue by name

```
[PUT|POST] http://branch:port/<app>/instances/<instance>/queues/<queue>/
```

- Tell an instance by name to stop consuming from queue by name

```
DELETE http://branch:port/<app>/instances/<instance>/queues/<queue>/
```

1.4.5 Queueing Tasks

Queueing an URL will result in one of the worker instances to execute that request as soon as possible.

```
[verb] http://branch:port/<app>/queue/<queue>/<url>?get_data
post_data
```

The `verb` can be any supported HTTP verb, such as `HEAD`, `GET`, `POST`, `PUT`, `DELETE`, `TRACE`, `OPTIONS`, `CONNECT`, and `PATCH`. The worker will then use the same verb when performing the request. Any get and post data provided will also be forwarded.

When you queue an URL a unique identifier is returned, you can use this identifier (called an UUID) to query the status of the task or collect the return value. The return value of the task is the HTTP response of the actual request performed by the worker.

Examples:

```
GET http://branch:port/<app>/queue/tasks/http://m/import_contacts?user=133


POST http://branch:port/<app>/queue/tasks/http://m/import_user
username=George Costanza
company=Vandelay Industries
```

1.4.6 Querying Task State

- To get the current state of a task

```
GET http://branch:port/<app>/query/<uuid>/state/
```

- To get the return value of a task

```
GET http://branch:port/<app>/query/<uuid>/result/
```

- To wait for a task to complete, and return its result.

```
GET http://branch:port/<app>/query/<uuid>/wait/
```

1.4.7 Instance details and statistics

To get configuration details and statistics for a particular instance:

```
GET http://branch:port/<app>/instance/<name>/stats/
```

1.4.8 Autoscale

- To set the max/min concurrency settings of an instance

```
POST http://branch:port/<app>/instance/<name>/autoscale/?max=int
                                                        ?min=int
```

- To get the max/min concurrency settings of an instance

```
GET http://branch:port/<app>/instance/<name>/autoscale/
```

1.5 Components

1.5.1 Programs

- `cyme <cyme.management.commands.cyme`.

  This is the management application, speaking HTTP with the clients. See `cyme --help` for full description and command line arguments.

- `cyme-branch`.

  Creates a new branch and starts the service to manage it. See `cyme-branch --help` for full description and command line arguments.

1.5.2 Models

The branch manager uses an SQLite database to store state, but this can also be another database system (MySQL, PostgreSQL, Oracle, DB2).

App

see `cyme.models.App`.

Every instance belongs to an application, and the application contains the default broker configuration.

Broker

see `cyme.models.Broker`.

The connection parameters for a specific broker (`hostname`, `port`, `userid`, `password`, `virtual_host`)

Instance

see `cyme.models.Instance`.

This describes a Celery worker instance that is a member of this branch. And also the queues it should consume from and its max/min concurrency settings. It also describes what broker the instance should be connecting to (which if not specified will default to the broker of the app the instance belongs to).

Queue

see `cyme.models.Queue`.

A queue declaration: name, exchange, exchange type, routing key, and options. Options is a json encoded mapping of queue, exchange and binding options supported by `kombu.compat.entry_to_queue()`.

1.6 Supervisor

see `cyme.supervisor`.

The supervisor wakes up at intervals to monitor for changes in the model. It can also be requested to perform specific operations, e.g. restart an instance, add queues to instance, and these operations can be either async or sync.

It is responsible for:

- Stopping removed instances.
- Starting new instances.
- Restarting unresponsive/killed instances.
- Making sure the instances consumes from the queues specified in the model, sending add_consumer/- cancel_consumer broadcast commands to the instances as it finds inconsistencies.
- Making sure the max/min concurrency setting is as specified in the model, sending autoscale broadcast commands to the instances as it finds inconsistencies.

The supervisor is resilient to intermittent connection failures, and will auto-retry any operation that is dependent on a broker.

Since workers cannot respond to broadcast commands while the broker is off-line, the supervisor will not restart affected instances until the instance has had a chance to reconnect (decided by the wait_after_broker_revived attribute).

1.7 Controller

see `cyme.controller`.

The controller is a series of cl actors to control applications, instances and queues. It is used by the HTTP interface, but can also be used directly.

1.8 HTTP

The http server currently serves up an admin instance where you can add, remove and modify instances.

The http server can be disabled using the `--without-http` option.

CHAPTER

TWO

CHANGE HISTORY

- 0.0.1

2.1 0.0.1

status in development

- Initial commit.

CHAPTER

THREE

API REFERENCE

Release 0.0

Date March 20, 2012

3.1 cyme.client

- Branches
- Applications
- Queues
- Instances
- Consumers
- Deleting

cyme.client

- Python client for the Cyme HTTP API.

3.1.1 Branches

```
>>> client = Client("http://localhost:8000")
>>> client.branches
["cyme1.example.com", "cyme2.example.com"]

>>> client.branch_info("cyme1.example.com")
{'sup_interval': 5, 'numc': 2, 'loglevel': 'INFO',
 'logfile': None, 'id': 'cyme1.example.com', 'port': 8000}
```

3.1.2 Applications

```
>>> app = client.get("foo")
>>> app
<Client: 'http://localhost:8016/foo'>
>>> app.info
{'name': 'foo', 'broker': 'amqp://guest:guest@localhost:5672//'}
```

3.1.3 Queues

```
>>> app.queues.add("myqueue", exchange="myex", routing_key="x")
>>> <Queue: u'myqueue'>
>>> my_queue = app.queues.get("my_queue")

>>> app.queues
[<Queue: u'myqueue'>]
```

3.1.4 Instances

```
>>> i = app.instances.add()
>>> i
<Instance: u'd87798f3-0bb0-4161-8e0b-a5f069b1d58b'>
>>> i.name
u'd87798f3-0bb0-4161-8e0b-a5f069b1d58b'

>>> i.broker  # < inherited from app
u'amqp://guest:guest@localhost:5672//'

>>> app.instances
[<Instance: u'd87798f3-0bb0-4161-8e0b-a5f069b1d58b'>]

>>> i.autoscale()   # current autoscale settings
{'max': 1, 'min': 1}

>>> i.autoscale(max=10, min=10)  # always run 10 processes
{'max': 10, 'min': 10}

>>> i.stats()
{'total': {},
 'consumer': {'prefetch_count': 80,
              'broker': {'transport_options': {},
                         'login_method': 'AMQPLAIN',
                         'hostname': '127.0.0.1',
                         'userid': 'guest',
                         'insist': False,
                         'connect_timeout': 4,
                         'ssl': False,
                         'virtual_host': '/',
                         'port': 5672,
                         'transport': 'amqp'}},
'pool': {'timeouts': [None, None],
         'processes': [76003],
         'max-concurrency': 1,
         'max-tasks-per-child': None,
         'put-guarded-by-semaphore': True},
'autoscaler': {'current': 1, 'max': 1, 'min': 1, 'qty': 0}}
```

3.1.5 Consumers

```
>>> instance.consumers.add(my_queue)
{"ok": "ok"}
```

```
>>> instance_consumers.delete(my_queue)
{"ok": "ok"}

>>> instance.consumers
#... consumers with full declarations ...
```

3.1.6 Deleting

This will delete the queue and eventually force all worker instances to stop consuming from it:

```
>>> my_queue.delete()
```

This will shutdown and delete an instance:

```
>>> i.delete()
```

class cyme.client.**Client**(*url=None*, *app=None*, *info=None*)

class **Instances**(*client*)

class **Model**(**args*, ***kwargs*)

class **Consumers**(*client*, *name*)

create_model(*data*, **args*, ***kwargs*)

class Client.Instances.Model.**LazyQueues**(*instance*)

Client.Instances.Model.**autoscale**(*max=None*, *min=None*)

Client.Instances.Model.**queues**

Client.Instances.Model.**stats**()

Client.Instances.**add**(*name=None*, *broker=None*, *arguments=None*, *config=None*, *nowait=False*)

Client.Instances.**autoscale**(*name*, *max=None*, *min=None*)

Client.Instances.**create_model**(*data*, **args*, ***kwargs*)

Client.Instances.**stats**(*name*)

class Client.**Queues**(*client*)

class **Model**(*parent*, **args*, ***kwargs*)

Client.Queues.**add**(*name*, *exchange=None*, *exchange_type=None*, *routing_key=None*, *nowait=False*, ***options*)

Client.**add**(*name*, *broker=None*, *arguments=None*, *extra_config=None*, *nowait=False*)

Client.**all**()

Client.**app = 'cyme'**

Client.**branch_info**(*id*)

Client.**branches**

Client.**build_url**(*path*)

Client.**clone**(*app=None*, *info=None*)

Client.**create_model**(*name*, *info*)

Client.**delete**(*name=None*)

Client.**get**(*name=None*)

class cyme.client.**Instance**(*parent*, **args*, ***kwargs*)

arguments
A unicode string field.

broker
A unicode string field.

extra_config
A unicode string field.

is_enabled
A boolean field type.

max_concurrency
A field that validates input as an Integer

min_concurrency
A field that validates input as an Integer

name
A unicode string field.

pool
A unicode string field.

queue_names

class cyme.client.**Queue**(*parent*, **args*, ***kwargs*)

exchange
A unicode string field.

exchange_type
A unicode string field.

name
A unicode string field.

options
A unicode string field.

routing_key
A unicode string field.

3.2 cyme.client.base

cyme.client.base

class cyme.client.base.**Base**

deserialize(*text*)

keys()

maybe_async(*name*, *nowait*)

serialize(*obj*)

class cyme.client.base.**Client**(*url=None*)

DELETE(*path*, *params=None*, *data=None*, *type=None*)

GET(*path*, *params=None*, *type=None*)

POST(*path*, *params=None*, *data=None*, *type=None*)

PUT(*path*, *params=None*, *data=None*, *type=None*)

default_url = 'http://127.0.0.1:8000'

headers

request(*method*, *path*, *params=None*, *data=None*, *type=None*)

root(*method*, *path=None*, *params=None*, *data=None*)

class cyme.client.base.**Model**(*parent*, **args*, ***kwargs*)

delete(*nowait=False*)

id
A field that stores a valid UUID value and optionally auto-populates empty values with new UUIDs.

class cyme.client.base.**Path**(*s=None*, *stack=None*)

class cyme.client.base.**Section**(*client*)

class Model(*parent*, **args*, ***kwargs*)

delete(*nowait=False*)

id
A field that stores a valid UUID value and optionally auto-populates empty values with new UUIDs.

Section.**add**(*name*, *nowait=False*, ***data*)

Section.**all**()

Section.**all_names**()

Section.**create_model**(**args*, ***kwargs*)

Section.**delete**(*name*, *nowait=False*)

Section.**get**(*name*)

Section.**maybe_async**(*name*, *nowait*)

Section.**name = None**

Section.**path = None**

Section.**proxy = ['GET', 'POST', 'PUT', 'DELETE']**

3.3 cyme.branch

cyme.branch

- This is the Branch thread started by the **cyme-branch** program.

  It starts the HTTP server, the Supervisor, and one or more controllers.

class cyme.branch.**Branch**(*addrport=''*, *id=None*, *loglevel=20*, *logfile=None*, *without_httpd=False*, *numc=2*, *sup_interval=None*, *ready_event=None*, *colored=None*, ***kwargs*)

about()

after()

controller_cls = '.controller.Controller'

httpd_cls = '.httpd.HttpServer'

intsup_cls = '.intsup.gSup'

on_ready(***kwargs*)

prepare_signals()

run()

stop()

supervisor_cls = '.supervisor.Supervisor'

class cyme.branch.**MockSup**(*thread*, **args*)

start()

stop()

3.4 cyme.branch.controller

3.5 cyme.branch.managers

cyme.branch.managers

- Contains the LocalInstanceManager instance, which is the preferred API used to control and manage worker instances handled by this branch. I.e. it can be used to do synchronous actions that don't return until the supervisor has performed them.

class cyme.branch.managers.**LocalInstanceManager**

Brokers = <cyme.models.managers.BrokerManager object at 0x4d08f90>

Instances = <cyme.models.managers.InstanceManager object at 0x4d18290>

add(*name=None*, *queues=None*, *max_concurrency=1*, *min_concurrency=1*, *broker=None*, *pool=None*, *app=None*, *arguments=None*, *extra_config=None*, *nowait=False*, ***kwargs*)

add_consumer(*name*, *queue*, *nowait=False*)

cancel_consumer(*name*, *queue*, *nowait=False*)

disable(*name*, *nowait=False*)

enable(*name*, *nowait=False*)

get(*name*)

maybe_wait(*fun*, *instances*, *nowait*)

remove(*name*, *nowait=False*)

remove_queue(*queue*, *nowait=False*)

restart(*name*, *nowait=False*)

3.6 cyme.branch.supervisor

cyme.branch.supervisor

class cyme.branch.supervisor.**Supervisor**(*interval=None*, *queue=None*, *set_as_current=True*)
The supervisor wakes up at intervals to monitor changes in the model. It can also be requested to perform specific operations, and these operations can be either async or sync.

Parameters

- **interval** – This is the interval (in seconds as an int/float), between verifying all the registered instances.
- **queue** – Custom `Queue` instance used to send and receive commands.

It is responsible for:

•Stopping removed instances.

•Starting new instances.

•Restarting unresponsive/killed instances.

•Making sure the instances consumes from the queues specified in the model, sending `add_consumer`/-`cancel_consumer` broadcast commands to the instances as it finds inconsistencies.

•Making sure the max/min concurrency setting is as specified in the model, sending `autoscale` broadcast commands to the noes as it finds inconsistencies.

The supervisor is resilient to intermittent connection failures, and will auto-retry any operation that is dependent on a broker.

Since workers cannot respond to broadcast commands while the broker is off-line, the supervisor will not restart affected instances until the instance has had a chance to reconnect (decided by the `wait_after_broker_revived` attribute).

before()

interval = 60.0
Default interval (time in seconds as a float to reschedule).

pause()
Pause all timers.

paused = False
Connection errors pauses the supervisor, so events does not accumulate.

restart(*instances*)
Restart one or more instances.

Parameters **instances** – List of instances to restart.

This operation is asynchronous, and returns a `Greenlet` instance that can be used to wait for the operation to complete.

restart_max_rate = '1/m'
Limit instance restarts to 1/m, so out of control instances will be disabled

resume()
Resume all timers.

run()

shutdown(*instances*)
Shutdown one or more instances.

Parameters **instances** – List of instances to stop.

This operation is asynchronous, and returns a `Greenlet` instance that can be used to wait for the operation to complete.

> **Warning:** Note that the supervisor will automatically restart any stopped instances unless the corresponding `Instance` model has been marked as disabled.

verify(*instances*, *ratelimit=False*)
Verify the consistency of one or more instances.

Parameters **instances** – List of instances to verify.

This operation is asynchronous, and returns a `Greenlet` instance that can be used to wait for the operation to complete.

wait_after_broker_revived = 35.0
Default interval_max for ensure_connection is 30 secs.

cyme.branch.supervisor.**get_current**()

cyme.branch.supervisor.**set_current**(*sup*)

3.7 cyme.branch.httpd

cyme.branch.httpd

- Our embedded WSGI server used to serve the HTTP API.

class cyme.branch.httpd.**HttpServer**(*addrport=None*)

create_http_protocol()

create_log()

joinable = False

logger_name

run()

server(*sock*, *handler*)

url

3.8 cyme.branch.signals

cyme.branch.signals

cyme.branch.signals.**branch_ready = <Signal: Signal>**
Sent when the branch and all its components are ready to serve.

Arguments:

sender is the Branch instance.

cyme.branch.signals.**controller_ready = <Signal: Signal>**
Sent when a controller is ready.

Arguments:

sender is the Controller instance.

cyme.branch.signals.**httpd_ready = <Signal: Signal>**
Sent when the http server is ready to accept requests. Arguments:

sender the HttpServer instance.

addrport the (hostname, port) tuple.

handler the WSGI handler used.

sock the socket used.

cyme.branch.signals.**supervisor_ready = <Signal: Signal>**
Sent when the supervisor is ready. Arguments:

sender is the Supervisor instance.

3.9 cyme.branch.state

cyme.branch.state

- Global branch state.
- Used to keep track lost connections and so on, which is used by the supervisor to know if an instance is actually down, or if it is just the connection being shaky.

class cyme.branch.state.**State**

broker_last_revived = None

is_branch = False
set to true if the process is a cyme-branch

on_broker_revive(**args*, ***kwargs*)

supervisor

time_since_broker_revived

3.10 cyme.branch.metrics

cyme.branch.metrics

class cyme.branch.metrics.**df**(*path*)

available

capacity

stat

total_blocks

cyme.branch.metrics.**load_average**()

3.11 cyme.branch.thread

cyme.branch.thread

- Utilities for working with greenlets.

exception cyme.branch.thread.**AlreadyStartedError**
Raised if trying to start a thread instance that is already started.

class cyme.branch.thread.**gThread**

AlreadyStarted
alias of AlreadyStartedError

exception Timeout(*seconds=None*, *exception=None*)
Raises *exception* in the current greenthread after *timeout* seconds.

When *exception* is omitted or None, the Timeout instance itself is raised. If *seconds* is None, the timer is not scheduled, and is only useful if you're planning to raise it directly.

Timeout objects are context managers, and so can be used in with statements. When used in a with statement, if *exception* is False, the timeout is still raised, but the context manager suppresses it, so the code outside the with-block won't see it.

cancel()
If the timeout is pending, cancel it. If not using Timeouts in with statements, always call cancel() in a finally after the block of code that is getting timed out. If not canceled, the timeout will be raised later on, in some unexpected section of the application.

pending
True if the timeout is scheduled to be raised.

start()
Schedule the timeout. This is called on construction, so it should not be called explicitly, unless the timer has been canceled.

gThread.**after**()
Call after the thread has shut down.

gThread.**before**()
Called at the beginning of start().

gThread.**extra_shutdown_steps** = 0

gThread.**extra_startup_steps** = 0

gThread.**g** = None
Greenlet instance of the thread, set when the thread is started.

gThread.**join**(*timeout=None*)
Wait until the thread exits.

Parameters **timeout** – Timeout in seconds (int/float).

Raises eventlet.Timeout if the thread can't be joined before the provided timeout.

gThread.**joinable = True**
Set this to False if it is not possible to join the thread.

gThread.**kill**()
Kill the green thread.

gThread.**logger_name**

gThread.**name = None**
Name of the thread, used in logs and such.

gThread.**ping**(*timeout=None*)

gThread.**respond_to_ping**()

gThread.**run**()

gThread.**should_stop = False**
Set when the thread is requested to stop.

gThread.**spawn**(*fun*, **args*, ***kwargs*)

gThread.**start**()
Spawn green thread, and returns `GreenThread` instance.

gThread.**start_periodic_timer**(*interval*, *fun*, **args*, ***kwargs*)
Apply function every `interval` seconds.

Parameters

- **interval** – Interval in seconds (int/float).
- **fun** – The function to apply.
- ***args** – Additional arguments to pass.
- ****kwargs** – Additional keyword arguments to pass.

Returns entry object, with `cancel` and `kill` methods.

gThread.**stop**(*join=True*, *timeout=1e+100*)
Shutdown the thread.

This will also cancel+kill any periodic timers registered by the thread.

Parameters

- **join** – Given that the thread is `joinable`, if true will also wait until the thread exits (by calling `join()`).
- **timeout** – Timeout for join (default is 1e+100).

3.12 cyme.branch.intsup

cyme.branch.intsup

- Internal supervisor used to ensure our threads are still running.

class cyme.branch.intsup.**gSup**(*thread*, *signal*, *interval=5*, *timeout=600*)

logger_name

run()

start_wait_child()

stop()

3.13 cyme.api.views

3.14 cyme.api.web

cyme.api.web

- Contains utilities for creating our HTTP API.

class cyme.api.web.**ApiView**(***kwargs*)

Accepted(**args*, ***kwargs*)

Created(*data*, **args*, ***kwargs*)

NotImplemented(**args*, ***kwargs*)

Ok(*data*, **args*, ***kwargs*)

Response(**args*, ***kwargs*)

dispatch(*request*, **args*, ***kwargs*)

get_or_post(*key*, *default=None*)

get_param(*key*, *type=None*)

nowait = False

params(**keys*)

typemap = {<type 'int'>: <function <lambda> at 0x4a84ed8>, <type 'float'>: <function <lambda> at 0x4a84f50>}

class cyme.api.web.**HttpResponseNotImplemented**(*content=''*, *mimetype=None*, *status=None*, *content_type=None*)

The requested action is not implemented. Used for async requests when the operation is inherently sync.

status_code = 501

class cyme.api.web.**HttpResponseTimeout**(*content=''*, *mimetype=None*, *status=None*, *content_type=None*)

The operation timed out.

status_code = 408

cyme.api.web.**JsonResponse**(*data*, *status=200*, *access_control=None*, ***kwargs*)

Returns a JSON encoded response.

cyme.api.web.**set_access_control_options**(*response*, *options=None*)

cyme.api.web.**simple_get**(*fun*)

3.15 cyme.models

cyme.models

class cyme.models.**Broker**(**args*, ***kwargs*)
Broker connection arguments.

url
AMQP (kombu) url.

connection()
Return a new Connection to this broker.

as_dict()
Returns this broker as a dictionary that can be Json encoded.

pool
A connection pool with connections to this broker.

objects
The manager for this model is BrokerManager.

class cyme.models.**Queue**(**args*, ***kwargs*)
An AMQP queue that can be consumed from by one or more instances.

name
Queue name (unique, max length 128)

exchange
Exchange name (max length 128)

exchange_type
Exchange type (max length 128)

routing_key
Routing key/binding key (max length 128).

options
Additional JSON encoded queue/exchange/binding options. see kombu.compat for a list of options supported.

is_enabled
Not in use.

created_at
Timestamp created.

as_dict()
Returns dictionary representation of this queue that can be Json encoded.

objects
The manager for this model is QueueManager.

class cyme.models.**Instance**(**args*, ***kwargs*)
A celeryd instance.

name
Name of the instance.

queues
Queues this instance should consume from (many to many relation to Queue).

max_concurrency
Autoscale setting for max concurrency. (maximum number of processes/threads/green threads when the worker is active).

min_concurrency
Autoscale setting for min concurrency. (minimum number of processes/threads/green threads when the worker is idle).

is_enabled
Flag set if this instance should be running.

created_at
Timestamp of when this instance was first created.

broker
The broker this instance should connect to. (foreign key to `Broker`).

Broker
Broker model class used (default is `Broker`)

Queue
Queue model class used (default is `Queue`)

MultiTool
Class used to start/stop and restart celeryd instances. (Default is `celery.bin.celeryd_multi.MultiTool`).

objects
The manager used for this model is `InstanceManager`.

cwd
Working directory used by all instances. (Default is `/var/run/cyme`).

as_dict()
Returns dictionary representation of this instance that can be Json encoded.

enable()
Enables this instance (model-only).

disable()
Disables this instance (model-only).

start(***kwargs*)
Starts the instance.

stop(***kwargs*)
Shuts down the instance.

restart(***kwargs*)
Restarts the instance.

alive(***kwargs*)
Returns `True` if the pid responds to signals, and the instance responds to ping broadcasts.

stats(***kwargs*)
Returns instance statistics (like `celeryctl inspect stats`).

autoscale(*max=None*, *min=None*, ***kwargs*)
Set max/min autoscale settings.

responds_to_ping(***kwargs*)
Returns `True` if the instance responds to broadcast ping.

responds_to_signal()
Returns `True` if the pid file exists and the pid responds to signals.

consuming_from(***kwargs*)
Returns the queues the instance is currently consuming from.

add_queue(*q*, ***kwargs*)
Add queue for this instance by name.

cancel_queue(*queue*, ***kwargs*)
Cancel queue for this instance by `Queue`.

getpid()
Get the process id for this instance by reading its pid file.

Returns `None` if the pid file does not exist.

_action(*action*, *multi='celeryd-multi'*)
Execute **celeryd-multi** command.

_query(*cmd*, *args={}*, ***kwargs*)
Send remote control command and wait for this instances reply.

3.16 cyme.models.managers

cyme.managers

- These are the managers for our models in `cyme.models`.
- They are not to be used directly, but accessed through the `objects` attribute of a Model.

class cyme.models.managers.**AppManager**

Brokers

add(*name=None*, *broker=None*, *arguments=None*, *extra_config=None*)

from_json(*name=None*, *broker=None*)

get_broker(*url*)

get_default()

instance(*name=None*, *broker=None*)

recreate(*name=None*, *broker=None*, *arguments=None*, *extra_config=None*)

class cyme.models.managers.**BrokerManager**

default_url

get_default()

class cyme.models.managers.**InstanceManager**

add(*name=None*, *queues=None*, *max_concurrency=1*, *min_concurrency=1*, *broker=None*, *pool=None*, *app=None*, *arguments=None*, *extra_config=None*)

add_queue_to_instances(*queue*, ***query*)

disable(*name*)

enable(*name*)

enabled()

remove(*name*)

remove_queue_from_instances(*queue*, ***query*)

class cyme.models.managers.**QueueManager**

add(*name*, *exchange=None*, *exchange_type=None*, *routing_key=None*, ***options*)

enabled()

3.17 cyme.status

cyme.status

class cyme.status.**Status**

all_instances()

ib(*fun*, **args*, ***kwargs*)
Shortcut to `self.insured( fun.im_self, fun(*args, **kwargs))`

insured(*instance*, *fun*, **args*, ***kwargs*)
Ensures any function performing a broadcast command completes despite intermittent connection failures.

pause()

paused = False

respond_to_ping()

restart_all()

restart_max_rate = '100/s'

resume()

shutdown_all()

start_all()

3.18 cyme.tasks

cyme.tasks

- We have our own version of the webhook task.
- It simply forwards the original request, not depending on any semantics present in the query string of the request, nor in the data returned in the response.

cyme.tasks.**DEFAULT_HEADERS = {'User-Agent': 'Celery/cyme v0.0.4'}**
Default HTTP headers to pass to the dispatched request.

cyme.tasks.**UA = 'Celery/cyme v0.0.4'**
Cyme User Agent string.

cyme.tasks.**response_to_dict**(*r*)

3.19 cyme.management.commands.cyme

- cyme
  - Options

3.19.1 cyme

Cyme management utility

Options

-a, --app
Application to use. Required for all operations except for when creating, deleting or listing apps.

-n, --nowait
Don't want for operations to complete (async).

-F, --format
Output format: pretty (default) or json.

-L, --local
Flag that if set means that operations will be performed locally for the current branch only. This means the instance directory must be properly set.

-l, --loglevel
Set custom log level. One of DEBUG/INFO/WARNING/ERROR/CRITICAL. Default is INFO.

-f, --logfile
Set custom logfile path. Default is `<stderr>`

-D, --instance-dir
Custom instance directory (default is `instances/`) Must be readable by the current user.

This needs to be properly specified if the `-L` option is used.

-b, --broker
Broker to use for a local *branches* request.

class cyme.management.commands.cyme.**Command**(*env=None*, **args*, ***kwargs*)

args = 'type command [args]\nE.g.:\n cyme apps\n cyme apps.add <name> [broker URL] [arguments] [extra config]\n cy

create_superuser()

drop_into_shell()

handle(**args*, ***kwargs*)

help = 'Cyme management utility'

name = 'cyme'

option_list = (<Option at 0x2d62320: -v/--verbosity>, <Option at 0x2d623b0: --settings>, <Option at 0x2d62368: --pyt

restart_all()

shutdown_all()

start_all()

status

class cyme.management.commands.cyme.**I**(*app=None*, *format=None*, *nowait=False*, *url=None*, ***kwargs*)

DISPATCH(*fqdn*, **args*)

add_app(**args*, ***kwargs*)

add_consumer(**args*, ***kwargs*)

add_instance(**args*, ***kwargs*)

add_queue(**args*, ***kwargs*)

all_apps(**args*, ***kwargs*)

all_consumers(**args*, ***kwargs*)

all_instances(**args*, ***kwargs*)

all_queues(**args*, ***kwargs*)

delete_app(**args*, ***kwargs*)

delete_consumer(**args*, ***kwargs*)

delete_instance(**args*, ***kwargs*)

delete_queue(**args*, ***kwargs*)

format_optargs(*optargs*)

format_response(*ret*)

get_app(**args*, ***kwargs*)

get_instances(**args*, ***kwargs*)

get_queue(**args*, ***kwargs*)

getsig(*fun*, *opt_args=None*)

instance_autoscale(**args*, ***kwargs*)

instance_stats(**args*, ***kwargs*)

prepare_response(*ret*)

class cyme.management.commands.cyme.**LocalI**(**args*, ***kwargs*)

add_app(*name*, *broker=None*, *arguments=None*, *extra_config=None*)

add_consumer(*instance_name*, *queue_name*)

add_instance(*name=None*, *broker=None*, *arguments=None*, *extra_config=None*)

add_queue(*name*, *exchange=None*, *exchange_type=None*, *routing_key=None*, *options=None*)

all_apps()

all_branches()

all_consumers(*instance_name*)

all_instances()

all_queues()

delete_app(*name*)

delete_consumer(*instance_name*, *queue_name*)

delete_instance(*name*)

delete_queue(*name*)

get_app()

get_instance(*name*)

get_queue(*name*)

instance_autoscale(*name*, *max=None*, *min=None*)

instance_stats(*name*)

class cyme.management.commands.cyme.**WebI**(*app=None*, *format=None*, *nowait=False*, *url=None*, ***kwargs*)

add_app(*name*, *broker=None*, *arguments=None*, *extra_config=None*)

add_consumer(*instance_name*, *queue_name*)

add_instance(*name=None*, *broker=None*, *arguments=None*, *extra_config=None*)

add_queue(*name*, *exchange=None*, *exchange_type=None*, *routing_key=None*, *options=None*)

all_apps()

all_branches()

all_consumers(*instance_name*)

all_instances()

all_queues()

client

delete_app(*name*)

delete_consumer(*instance_name*, *queue_name*)

delete_instance(*name*)

delete_queue(*name*)

get_app(*name*)

get_instance(*name*)

get_queue(*name*)

instance_autoscale(*name*, *max=None*, *min=None*)

instance_stats(*name*)

prepare_response(*ret*)

3.20 cyme.management.commands.cyme_branch

- cyme-branch
  - Options

3.20.1 cyme-branch

Starts the cyme branch service.

Options

-i, -id
Set branch id, if not provided one will be automatically generated.

-without-httpd
Disable the HTTP server thread.

-l, -loglevel
Set custom log level. One of DEBUG/INFO/WARNING/ERROR/CRITICAL. Default is INFO.

-f, -logfile
Set custom log file path. Default is `<stderr>`

-D, -instance-dir
Custom instance directory (default is `instances/ `') Must be writeable by the user cyme-branch runs as.

-C, -numc
Number of controllers to start, to handle simultaneous requests. Each controller requires one AMQP connection. Default is 2.

-sup-interval
Supervisor schedule Interval in seconds. Default is 5.

class `cyme.management.commands.cyme_branch.`**Command**(*env=None*, **args*, ***kwargs*)

args = '[optional port number, or ipaddr:port]'

banner()

branch_cls = 'cyme.branch.Branch'

connect_signals()

default_detach_logfile = 'b ranch.log'

default_detach_pidfile = 'branch.pid'

handle(**args*, ***kwargs*)

help = 'Starts a cyme branch'

install_signal_handlers()

name = 'cyme-branch'

on_branch_ready(*sender=None*, ***kwargs*)

on_branch_shutdown(*sender=None*, ***kwargs*)

option_list = (<Option at 0x2d62320: -v/–verbosity>, <Option at 0x2d623b0: –settings>, <Option at 0x2d62368: –pyt

repr_controller_id(*c*)

set_process_title(*info*)

setup_default_env(*env*)

setup_shutdown_progress(*sender=None*, ***kwargs*)

setup_startup_progress(*sender=None*, ***kwargs*)

signals

stop()

3.21 cyme.bin.base

cyme.bin.base

- Utilities used by command line applications, basically just to set up Django without having an actual project.

class cyme.bin.base.**BaseApp**

env = None

get_version()

instance_dir = None

needs_eventlet = False

run_from_argv(*argv=None*)

class cyme.bin.base.**Env**(*needs_eventlet=False*, *instance_dir=None*)

configure()

management

setup_eventlet()

setup_pool_limit(***kwargs*)

syncdb(*interactive=True*)

cyme.bin.base.**app**(***attrs*)

3.22 cyme.bin.cyme

cyme.bin.cyme

3.23 cyme.bin.cyme_branch

cyme.bin.cyme_branch

- This is the script run by the **cyme-branch** script installed by the Cyme distribution (defined in setup.py's `entry_points`).
- It in turn executes the cyme-branch management command.

3.24 cyme.utils

cyme.utils

class cyme.utils.**LazyProgressBar**(*size*, *description=None*, *endtext=None*)

finish(***kwargs*)

step(*i=1*, ***kwargs*)

class cyme.utils.**Path**

unipath.Path version that can use the / operator to combine paths:

```
>>> p = Path("foo")
>>> p / "bar" / "baz"
Path("foo/bar/baz")
```

cyme.utils.**find_package**(*mod*, *_s=None*)

Find the package a module belongs to.

if you have structure:

```
package/__init__.py
       /foo.py
       /bar/__init__.py
       /bar/baz.py
```

Then the following examples returns:

```
>>> find_package(import_module("package"))
"package"
>>> find_package(import_module("package.foo"))
"package"
>>> find_package(import_module("package.bar.baz"))
>>> package.bar
```

Note that this does not look at the file system, but rather uses the __package__ attribute of a module.

cyme.utils.**find_symbol**(*origin*, *sym*)

Find symbol in module relative by origin.

E.g. if origin is an object in the module package.foo, then:

```
>>> find_symbol(origin, ".bar.my_symbol")
```

will return the object my_symbol from module package.bar.

cyme.utils.**force_list**(*obj*)

Force object to be a list.

If obj is a scalar value then a list with that value as sole element is returned, or if obj is a tuple then it is coerced into a list.

cyme.utils.**imerge_settings**(*a*, *b*)

Merge two django settings modules, keys in b have precedence.

cyme.utils.**instantiate**(*origin*, *sym*, **args*, ***kwargs*)

Like find_symbol() but instantiates the class found using *args and **kwargs.

cyme.utils.**redirect_stdouts_to_logger**(*loglevel='INFO'*, *logfile=None*, *redirect_level='WARNING'*, *stdout=False*, *stderr=True*)

See celery.log.Log.redirect_stdouts_to_logger().

cyme.utils.**setup_logging**(*loglevel='INFO'*, *logfile=None*)

Setup logging using loglevel and logfile.

stderr will be used if not logfile provided.

CHAPTER

FOUR

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

C

cyme.api.web, **??**
cyme.bin.base, **??**
cyme.bin.cyme, **??**
cyme.bin.cyme_branch, **??**
cyme.branch, **??**
cyme.branch.httpd, **??**
cyme.branch.intsup, **??**
cyme.branch.managers, **??**
cyme.branch.metrics, **??**
cyme.branch.signals, **??**
cyme.branch.state, **??**
cyme.branch.supervisor, **??**
cyme.branch.thread, **??**
cyme.client, **??**
cyme.client.base, **??**
cyme.management.commands.cyme, **??**
cyme.management.commands.cyme_branch, **??**
cyme.models, **??**
cyme.models.managers, **??**
cyme.status, **??**
cyme.tasks, **??**
cyme.utils, **??**