Accounting Center Documentation

Release 0.3

Pim Witlox

Dec 07, 2018

Contents

The Accounting Center (API) aims to provide a stable rest interface for accounting resource usage of clusters and private clouds and managing users of a group/tenant. The idea is that users authenticate on the service using an LDAP compliant account, and can see their own resource usage, and group resource usage. If users are group administrators, they can add/remove users from the group.

In some magical way to be defined (probably with a couple of hacky scripts and cron), cluster controllers and cloud controllers will regularly update resource usage using a service account. These controllers will also check (and modify) existing group structures within their respective systems.

CHAPTER 1

Installing the package

You need python 3.5 or higher to run this package. The package can be installed using `pip install acpy`.

CHAPTER 2

Using the cli

Our package is self contained, so you can start and stop the server using the cli. Simply calling `acpy start` starts a development server for the API. When running in production we recommend using gevent.

acpy [OPTIONS] COMMAND [ARGS]...

This CLI allows you to manage the Accounting Center API, this service will run in the background. The service is started direct by default (don't do this in production, use gevent). Check your config file for settings, the default location is in your home folder under *~/.acpy/api.config*.

Options:

-c, --config-file TEXT	Specify configuration file path (creates if not exists)
-v, --verbosity LVL	Either CRITICAL, ERROR, WARNING, INFO or DEBUG
--help	Show this message and exit.

2.1 start

acpy start [OPTIONS]

start api

Options:

-g, --gevent	use gevent as server
-u, --ui	enable swagger ui (url/api/v1/ui)
-d, --debug	enable debug mode
-f, --force	force start ignoring recorded state
--help	Show this message and exit.

2.2 stop

acpy stop [OPTIONS]

stop api

Options:

-f, --force	kill instead of terminate
--help	Show this message and exit.

2.3 info

acpy info [OPTIONS]

rest service information

Options:

--help	Show this message and exit.

info output will have the following information:

- gevent : started as gevent
- ui : enabled or not
- debug : enabled or not
- pid : process id

CHAPTER 3

Table of contents

3.1 Configuration

All the information regarding the system is stored in a configuration file. The default configuration file is stored in your home directory: `~/.acpy/api.config` but you can specify a different location from the command line with the *-c* option.

If no configuration file is found, it will copy an example configuration file in `~/.acpy/api.config`. The example contains placeholders for all required values.

3.1.1 General settings

The following parameters are available as general settings:

1. CORS can be enabled, by default it is disabled
2. `secret` for Flask, autogenerated when the config file is created
3. `port` for the api, default is 8080
4. `run_time` stores the runtime information in a file, default is `~/.acpy/run_time.data`

3.1.2 Logging settings

The following parameters are available for logging:

1. `log_file` specifies the location of the logfile, default is `~/.acpy/acpy.log`. If left empty, no log file is generated.
2. `max_bytes` is the maximum size of a logfile before it is rotated, default is 2.5MB.
3. `backup_count` is the maximum amount of logfiles to keep of max_bytes, default is 5.

3.1.3 The admin account

The default admin account behaves as a service account:

1. `access` is the access code for the admin account (analog to username)
2. `secret` is the secret code for the admin account (analog to password) the password needs to be passed as sha256.

3.1.4 JWT token settings

Users and services request a token by calling the `login` service. This service returns a JWT token.

1. `issuer` is the name of the token issuer, this should reflect the URI of the API.
2. `secret` is the secret that is used for encoding the token, if left empty the FLASK shared secret is used.
3. `lifetime` is the token lifetime in seconds, default is 3600.
4. `algorithm` is the encryption algorithm for the token, default is HS256.

3.1.5 Database settings

All data is stored in a database, the connection needs to be specified in the configuration. By default we store to memory.

1. `connection` specifies the database connection, default is sqlite://

- example of a mariadb connection: `mysql://scott:tiger@localhost/test`
- example of a postgres connection: `postgresql://scott:tiger@localhost/mydatabase`

We use SQLAlchemy, so for all options see engines

3.1.6 LDAP settings

For authenticating users configure an LDAP compliant connection.

1. `host` ldap server without protocol.
2. `port` ldap port
3. `ssl` secure connection
4. `base_dn` root for the users
5. `rdn_attr` relative distinguished name (usually `uid` or `cn`)
6. `login_attr` what field to use for username.
7. `bind_user` user account that has read access on the `base_dn`
8. `bind_pass` password for `bind_user`

3.2 We recommend the following production configuration

1. NGINX reverse proxy with SSL (no plain HTTP due to login mechanism).
2. systemd service for managing our service.

3.2.1 NGINX

Make sure NGINX is installed on your system.

Get a Certificate

You will need to purchase or create an SSL certificate. These commands are for a self-signed certificate, but you should get an officially signed certificate if you want to avoid browser warnings.

Move into the proper directory and generate a certificate:

> cd /etc/nginx sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/nginx/cert.key -out /etc/nginx/cert.crt

You will be prompted to enter some information about the certificate. You can fill this out however you'd like; just be aware the information will be visible in the certificate properties. We've set the number of bits to 2048 since that's the minimum needed to get it signed by a CA. If you want to get the certificate signed, you will need to create a CSR. Edit the Configuration

Next you will need to edit the default Nginx configuration file.

> sudo nano /etc/nginx/sites-enabled/default

Here is what the final config might look like; the sections are broken down and briefly explained below. You can update or replace the existing config file, although you may want to make a quick copy first.

In our configuration, the cert.crt and cert.key settings reflect the location where we created our SSL certificate. You will need to update the servername and *proxyredirect* lines with your own domain name. There is some additional Nginx magic going on as well that tells requests to be read by Nginx and rewritten on the response side to ensure the reverse proxy is working.

The first section tells the Nginx server to listen to any requests that come in on port 80 (default HTTP) and redirect them to HTTPS.

Next we have the SSL settings. This is a good set of defaults but can definitely be expanded on. For more explanation, please read this tutorial.

The final section is where the proxying happens. It basically takes any incoming requests and proxies them to the instance that is bound/listening to port 8080 on the local network interface. This is a slightly different situation, but this tutorial has some good information about the Nginx proxy settings.

3.2.2 systemd

If you've installed acpy in system space, you should be able to create the file `/etc/systemd/system/acpy.service` with the following contents:

In order to start the service, first run `systemctl daemon-reload`. Now test if `systemctl start acpy` works, if so you can enable it `systemctl enable acpy`.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search