ara-server Documentation

Release 1.0.0.0a3.dev1

Red Hat

Jan 30, 2019

Contents

1 Table of Contents **3**
1.1 Installing ara-server 3
1.2 ara-server configuration 3
1.3 Authentication and security 10
1.4 Using API clients with ara-server 14
1.5 Architecture and Workflows 15

CHAPTER 1

Table of Contents

1.1 Installing ara-server

`ara-server` requires a Linux distribution with python 3 in order to work.

It is recommended to use a python virtual environment in order to avoid conflicts with your Linux distribution python packages:

```
# Create a virtual environment
python3 -m venv ~/.ara/venv

# Install ara-server from source
~/.ara/venv/bin/pip install git+https://git.openstack.org/openstack/ara-server

# or install it from PyPi
~/.ara/venv/bin/pip install ara-server
```

1.2 ara-server configuration

ara-server ships with sane defaults, supports the notion of different environments (*such as dev, staging, prod*) and allows you to customize the configuration with files, environment variables or a combination of both.

ara-server is a Django application that leverages django-rest-framework. Both Django and django-rest-framework have extensive configuration options which are not necessarily exposed or made customizable by ARA for the sake of simplicity.

1.2.1 Overview

This is a brief overview of the different configuration options for ara-server. For more details, click on the configuration parameters.

Environment Variable	Usage	default
ARA_BASE_DIR	Default directory for storing data and configuration	`~/.ara/server`
ARA_SETTINGS	Path to an ara-server configuration file	`None`
ARA_ENV	Environment to load configuration for	`default`
ARA_READ_LOGIN_REQUIRED	Whether authentication is required for reading data	`False`
ARA_WRITE_LOGIN_REQUIRED	Whether authentication is required for writing data	`False`
ARA_ENV	Environment to load configuration for	`development`
ARA_LOG_LEVEL	Log level of the different components	`INFO`
ARA_LOGGING	Logging configuration	See *ARA_LOGGING*
ARA_CORS_ORIGIN_WHITELIST	django-cors-headers's CORS_ORIGIN_WHITELIST setting	`["127.0.0.1:8000", "localhost:3000"]`
ARA_ALLOWED_HOSTS	Django's ALLOWED_HOSTS setting	`["127.0.0.1", "localhost", "::1"]`
ARA_STATIC_ROOT	Django's STATIC_ROOT setting	`~/.ara/server/www/static`
ARA_DEBUG	Django's DEBUG setting	`false`
ARA_SECRET_KEY	Django's SECRET_KEY setting	Randomized token, see *ARA_SECRET_KEY*
ARA_DATABASE_ENGINE	Django's ENGINE database setting	`django.db.backends.sqlite3`
ARA_DATABASE_NAME	Django's NAME database setting	`~/.ara/server/ansible.sqlite`
ARA_DATABASE_USER	Django's USER database setting	`None`
ARA_DATABASE_PASSWORD	Django's PASSWORD database setting	`None`
ARA_DATABASE_HOST	Django's HOST database setting	`None`
ARA_DATABASE_PORT	Django's PORT database setting	`None`

1.2.2 Configuration variables

ARA_BASE_DIR

- **Environment variable**: `ARA_BASE_DIR`
- **Configuration file variable**: `BASE_DIR`
- **Type**: `string`
- **Default**: `~/.ara/server`

The directory where data will be stored by default.

Changing this location influences the default root directory for the `ARA_STATIC_ROOT` and `ARA_DATABASE_NAME` parameters.

This is also used to determine the location where the default configuration file, `settings.yaml`, will be generated by ara-server.

ARA_SETTINGS

- **Environment variable**: `ARA_SETTINGS`
- **Configuration file variable**: None, this variable defines the configuration file itself.

- **Type**: `string`
- **Default**: `None`
- **Provided by**: dynaconf

Location of an ara-server configuration file to load settings from. `ara-server` generates a default configuration file at `~/.ara/server/settings.yaml` that you can use to get started.

Note that while the configuration file is in YAML by default, it is possible to have configuration files written in `ini`, `json` and `toml` as well.

Settings and configuration parsing in ara-server is provided by the dynaconf python module.

ARA_ENV

- **Environment variable**: `ARA_ENV`
- **Configuration file variable**: None, this variable defines which section of a configuration file is loaded.
- **Type**: `string`
- **Default**: `development`
- **Provided by**: dynaconf

If you are using ara-server in different environments and would like keep your configuration in a single file, you can use this variable to select a specific environment's settings.

For example:

```
# Default settings are used only when not provided in the environments
default:
    READ_LOGIN_REQUIRED: false
    WRITE_LOGIN_REQUIRED: false
    LOG_LEVEL: INFO
    DEBUG: false
# Increase verbosity and debugging for the default development environment
development:
    LOG_LEVEL: DEBUG
    DEBUG: true
    SECRET_KEY: dev
# Enable write authentication when using the production environment
production:
    WRITE_LOGIN_REQUIRED: true
    SECRET_KEY: prod
```

With the example above, loading the development environment would yield the following settings:

- READ_LOGIN_REQUIRED: `false`
- WRITE_LOGIN_REQUIRED: `false`
- LOG_LEVEL: `DEBUG`
- DEBUG: `true`
- SECRET_KEY: `dev`

Another approach to environment-specific configuration is to use `ARA_SETTINGS` and keep your settings in different files such as `dev.yaml` or `prod.yaml` instead.

Tip: If it does not exist, ara-server will generate a default configuration file at `~/.ara/server/settings.yaml`. This generated file sets up all the configuration keys in the **default** environment. This lets users override only the parameters they are interested in for specific environments.

ARA_READ_LOGIN_REQUIRED

- **Environment variable**: `ARA_READ_LOGIN_REQUIRED`
- **Configuration file variable**: `READ_LOGIN_REQUIRED`
- **Type**: `bool`
- **Default**: `False`
- **Provided by**: django-rest-framework permissions

Determines if authentication is required before being authorized to query all API endpoints exposed by the server.

There is no concept of granularity: users either have access to query everything or they don't.

Enabling this feature first requires setting up *users*.

ARA_WRITE_LOGIN_REQUIRED

- **Environment variable**: `ARA_WRITE_LOGIN_REQUIRED`
- **Configuration file variable**: `WRITE_LOGIN_REQUIRED`
- **Type**: `bool`
- **Default**: `False`
- **Provided by**: django-rest-framework permissions

Determines if authentication is required before being authorized to post data to all API endpoints exposed by the server.

There is no concept of granularity: users either have access to query everything or they don't.

Enabling this feature first requires setting up *users*.

ARA_LOG_LEVEL

- **Environment variable**: `ARA_LOG_LEVEL`
- **Configuration file variable**: `LOG_LEVEL`
- **Type**: `string`
- **Default**: `INFO`

Log level of the different components from `ara-server`.

`ARA_LOG_LEVEL` changes the log level of the default logging configuration provided by *ARA_LOGGING*.

ARA_LOGGING

- **Environment variable**: *Not recommended, use configuration file*
- **Configuration file variable**: `LOGGING`
- **Type**: `dictionary`
- **Default**:

```
LOGGING:
    disable_existing_loggers: false
    formatters:
    normal:
        format: '%(asctime)s %(levelname)s %(name)s: %(message)s'
    handlers:
    console:
        class: logging.StreamHandler
        formatter: normal
        level: INFO
        stream: ext://sys.stdout
    loggers:
    ara:
        handlers:
        - console
        level: INFO
        propagate: 0
    root:
    handlers:
    - console
    level: INFO
    version: 1
```

The python logging configuration for `ara-server`.

ARA_CORS_ORIGIN_WHITELIST

- **Environment variable**: `ARA_CORS_ORIGIN_WHITELIST`
- **Configuration file variable**: `CORS_ORIGIN_WHITELIST`
- **Provided by**: django-cors-headers
- **Type**: `list`
- **Default**: `["127.0.0.1:8000", "localhost:3000"]`
- **Examples**:
  - `export ARA_CORS_ORIGIN_WHITELIST="['api.ara.example.org', 'web.ara.example.org']"`
  - In a YAML configuration file:

```
dev:
  CORS_ORIGIN_WHITELIST:
    - 127.0.0.1:8000
    - localhost:3000
production:
  CORS_ORIGIN_WHITELIST:
```

(continues on next page)

(continued from previous page)

```
    - api.ara.example.org
    - web.ara.example.org
```

Hosts in the whitelist for Cross-Origin Resource Sharing.

This setting is typically used in order to allow the API and a web client (such as ara-web) to talk to each other.

ARA_ALLOWED_HOSTS

- **Environment variable**: ARA_ALLOWED_HOSTS
- **Configuration file variable**: ALLOWED_HOSTS
- **Type**: list
- **Provided by**: Django's ALLOWED_HOSTS
- **Default**: ["127.0.0.1", "localhost", "::1"]

A list of strings representing the host/domain names that this Django site can serve.

If you are planning on hosting an instance of ara-server somewhere, you'll need to add your domain name to this list.

ARA_DEBUG

- **Environment variable**: ARA_DEBUG
- **Configuration file variable**: DEBUG
- **Provided by**: Django's DEBUG
- **Type**: string
- **Default**: false

Whether or not Django's debug mode should be enabled.

The Django project recommends turning this off for production use.

ARA_SECRET_KEY

- **Environment variable**: ARA_SECRET_KEY
- **Configuration file variable**: SECRET_KEY
- **Provided by**: Django's SECRET_KEY
- **Type**: string
- **Default**: Randomized with django.utils.crypto.get_random_string()

A secret key for a particular Django installation. This is used to provide cryptographic signing, and should be set to a unique, unpredictable value.

If it is not set, a random token will be generated and persisted in the default configuration file.

ARA_STATIC_ROOT

- **Environment variable**: `ARA_STATIC_ROOT`
- **Configuration file variable**: `STATIC_ROOT`
- **Provided by**: Django's STATIC_ROOT
- **Type**: `string`
- **Default**: `~/.ara/server/www/static`

The absolute path to the directory where Django's collectstatic command will collect static files for deployment.

The static files are required for the built-in API browser by django-rest-framework.

ARA_DATABASE_ENGINE

- **Environment variable**: `ARA_DATABASE_ENGINE`
- **Configuration file variable**: `DATABASES["default"]["ENGINE"]`
- **Provided by**: Django's ENGINE database setting
- **Type**: `string`
- **Default**: `django.db.backends.sqlite3`
- **Examples**: - `django.db.backends.postgresql` - `django.db.backends.mysql`

The Django database driver to use.

When using anything other than sqlite3 default driver, make sure to set the other database settings to allow ara-server to connect to the database.

ARA_DATABASE_NAME

- **Environment variable**: `ARA_DATABASE_NAME`
- **Configuration file variable**: `DATABASES["default"]["NAME"]`
- **Provided by**: Django's NAME database setting
- **Type**: `string`
- **Default**: `~/.ara/server/ansible.sqlite`

The name of the database.

When using sqlite, this is the absolute path to the sqlite database file. When using drivers such as MySQL or PostgreSQL, it's the name of the database.

ARA_DATABASE_USER

- **Environment variable**: `ARA_DATABASE_USER`
- **Configuration file variable**: `DATABASES["default"]["USER"]`
- **Provided by**: Django's USER database setting
- **Type**: `string`
- **Default**: `None`

The username to connect to the database.

Required when using something other than sqlite.

ARA_DATABASE_PASSWORD

- **Environment variable**: `ARA_DATABASE_PASSWORD`
- **Configuration file variable**: `DATABASES["default"]["PASSWORD"]`
- **Provided by**: Django's PASSWORD database setting
- **Type**: `string`
- **Default**: `None`

The password to connect to the database.

Required when using something other than sqlite.

ARA_DATABASE_HOST

- **Environment variable**: `ARA_DATABASE_HOST`
- **Configuration file variable**: `DATABASES["default"]["HOST"]`
- **Provided by**: Django's HOST database setting
- **Type**: `string`
- **Default**: `None`

The host for the database server.

Required when using something other than sqlite.

ARA_DATABASE_PORT

- **Environment variable**: `ARA_DATABASE_PORT`
- **Configuration file variable**: `DATABASES["default"]["PORT"]`
- **Provided by**: Django's PORT database setting
- **Type**: `string`
- **Default**: `None`

The port to use when connecting to the database server.

It is not required to set the port if you're using default ports for MySQL or PostgreSQL.

1.3 Authentication and security

ara-server ships with a default configuration that emphasizes simplicity to let users get started quickly.

By default:

- A random SECRET_KEY will be generated once if none are supplied
- No users are created

- API authentication and permissions are not enabled
- ALLOWED_HOSTS and CORS_ORIGIN_WHITELIST are configured for use on localhost

These default settings can be configured according to the requirements of your deployments.

1.3.1 Setting a custom secret key

By default, ara-server randomly generates a token for the *ARA_SECRET_KEY* setting if none have been supplied by the user. This value is persisted in the server configuration file in order to prevent the key from changing on every instanciation of the server.

The default location for the server configuration file is `~/.ara/server/settings.yaml`.

You can provide a custom secret key by supplying the `ARA_SECRET_KEY` environment variable or by specifying the `SECRET_KEY` setting in your server configuration file.

1.3.2 User management

ara-server leverages Django's user management but doesn't create any user by default.

Note: Creating users does not enable authentication on the API. In order to make authentication required for using the API, see *Enabling authentication for read or write access*.

In order to create users, you'll need to create a superuser account before running the API server:

```
$ ara-manage createsuperuser --username=joe --email=joe@example.com
Password:
Password (again):
Superuser created successfully.
```

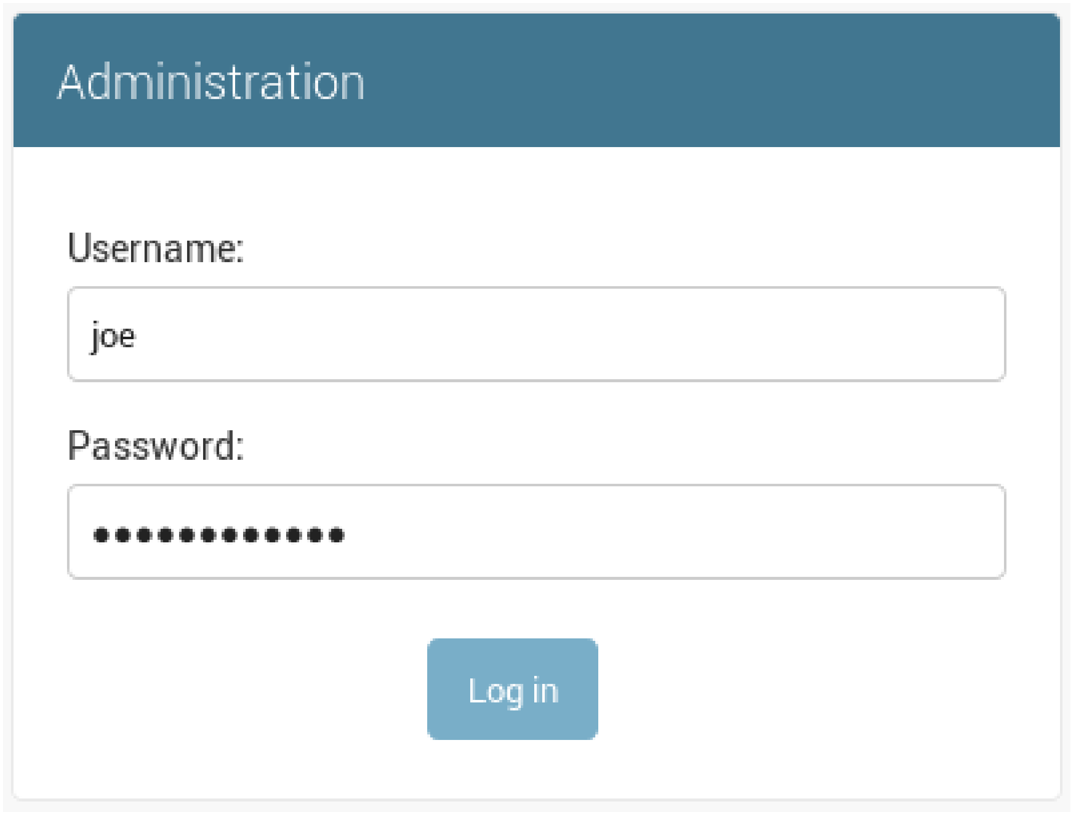
Tip: If you ever need to reset the password of a superuser account, this can be done with the "changepassword" command:

```
$ ara-manage changepassword joe
Changing password for user 'joe'
Password:
Password (again):
Password changed successfully for user 'joe'
```

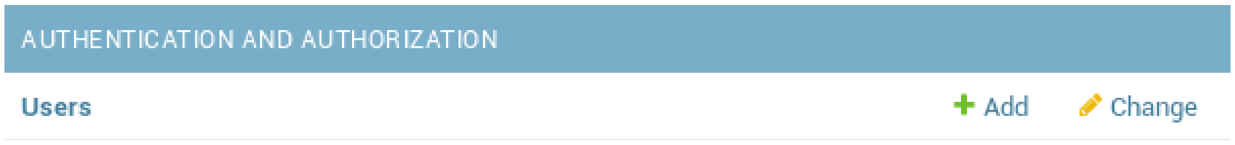
Once the superuser has been created, make sure the API server is started and then login to the Django web administrative interface using the credentials you just set up.

By default, you can start the API server with `ara-manage runserver` and access the admin interface at `http://127.0.0.1:8000/admin/`.

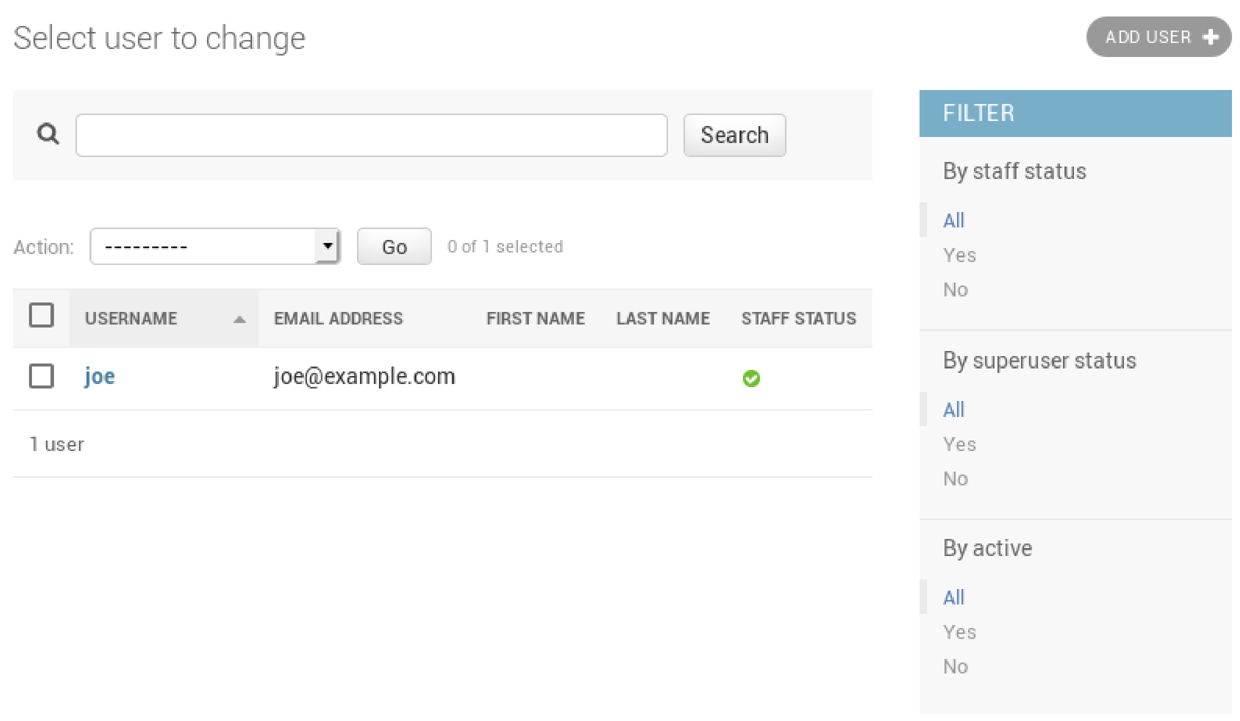
Log in to the admin interface:



Access the authentication and authorization configuration:



And from here, you can manage existing users or create new ones:



USERNAME	EMAIL ADDRESS	FIRST NAME	LAST NAME	STAFF STATUS
joe	joe@example.com			✓

1 user

1.3.3 Enabling authentication for read or write access

Once you have created your users, you can enable authentication against the API for read (ex: GET) and write (ex: DELETE, POST, PATCH) requests.

This is done with the two following configuration options:

- *ARA_READ_LOGIN_REQUIRED* for read access
- *ARA_WRITE_LOGIN_REQUIRED* for write access

These settings are global and are effective for all API endpoints.

1.3.4 Managing hosts allowed to serve the API

By default, *ARA_ALLOWED_HOSTS* authorizes `localhost`, `::1` and `127.0.0.1` to serve requests for the API server.

In order to host an instance of ara-server on another domain, the domain must be part of this list or the application server will deny any requests sent to it.

1.3.5 Managing CORS (cross-origin resource sharing)

The *ARA_CORS_ORIGIN_WHITELIST* default is designed to allow a local development instance of an ara-web dashboard to communicate with a local development instance of ara-server.

The whitelist must contain the domain names where you plan on hosting instances of ara-web.

1.4 Using API clients with ara-server

Once you've *installed* ara-server, you need to know how you're going to use it.

Typically, ara-server is consumed by ara-clients which currently provides two python clients for the API.

1.4.1 ARA Offline REST API client

The default client, `AraOfflineClient`, is meant to be used to query the API without requiring users to start or host an instance of `ara-server`.

To use the offline client, first install `ara-server` and `ara-clients`, for example:

```
# Install ara-server and ara-clients
python3 -m venv ~/.ara/venv
~/.ara/venv/bin/pip install ara-server ara-clients
```

Then you can use it like this:

```
#!/usr/bin/env python3
# Import the client
from ara.clients.offline import AraOfflineClient

# Instanciate the offline client
client = AraOfflineClient()
```

1.4.2 ARA HTTP REST API client

`AraHttpClient` works with the same interface, methods and behavior as `AraOfflineClient`. The HTTP client does not require `ara-server` to be installed in order to be used but expects a functional API endpoint at a specified location.

You can set your client to communicate with a remote `ara-server` API by specifying an endpoint parameter:

```
#!/usr/bin/env python3
# Import the client
from ara.clients.http import AraHttpClient

# Instanciate the HTTP client with an endpoint where ara-server is listening
client = AraHttpClient(endpoint="https://api.demo.recordsansible.org")
```

1.4.3 Example API usage

Note: API documentation is a work in progress.

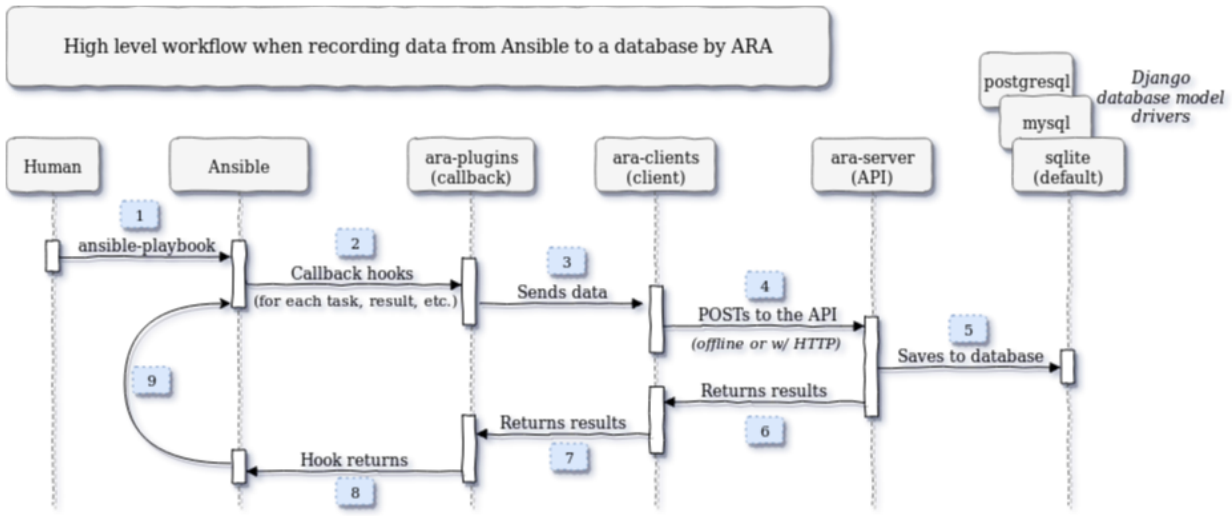
Once you've instanciated your client, you're ready to query the API.

Here's a code example to help you get started:

```
# Get a list of failed playbooks
# /api/v1/playbooks?status=failed
playbooks = client.get("/api/v1/playbooks", status="failed")

# If there are any failed playbooks, retrieve their failed results
# and provide some insight.
for playbook in playbooks["results"]:
    # Retrieve results for this playbook
    # /api/v1/results?playbook=<:id>&status=failed
    results = client.get("/api/v1/results", playbook=playbook["id"], status="failed")

    # Iterate over failed results to get meaningful data back
    for result in results["results"]:
        # Get the task that generated this result
        # /api/v1/tasks/<:id>
        task = client.get(f"/api/v1/tasks/{result['task']}")

        # Get the file from which this task ran from
        # /api/v1/files/<:id>
        file = client.get(f"/api/v1/files/{task['file']}")

        # Get the host on which this result happened
        # /api/v1/hosts/<:id>
        host = client.get(f"/api/v1/hosts/{result['host']}")

        # Print something useful
        print(f"Failure on {host['name']}: '{task['name']}' ({file['path']}:{task[
↪'lineno']})")
```

1.5 Architecture and Workflows

1.5.1 Recording data from Ansible



0. A human (*or a system, script, etc.*) installs ARA and configures Ansible to use the ARA callback
1. A human (*or a system, script, etc.*) executes an `ansible-playbook` command

2. Ansible sends hooks for every event to callback plugins (`v2_playbook_on_start`, `v2_runner_on_failed`, etc.)
3. The callback plugin, provided by ara-plugins, organizes the data sent by Ansible and sends it to the API client
4. The API client, provided by ara-clients, takes care of actually sending the data to the API over HTTP or locally offline through an internal implementation
5. The API server, provided by ara-server, receives the POST from the client, validates it and sends it to the database model backend
6. The API server sends a response back to the client with the results
7. The API client sends the response back to the callback with the results
8. The callback plugin returns, ending the callback hook
9. Ansible continues running until it is complete (back to step 2)