ara Documentation

Release 1.1.0

Red Hat

Jul 02, 2019

Contents

ara

CHAPTER 1

Table of Contents

1.1 FAQ

1.1.1 What is ARA ?

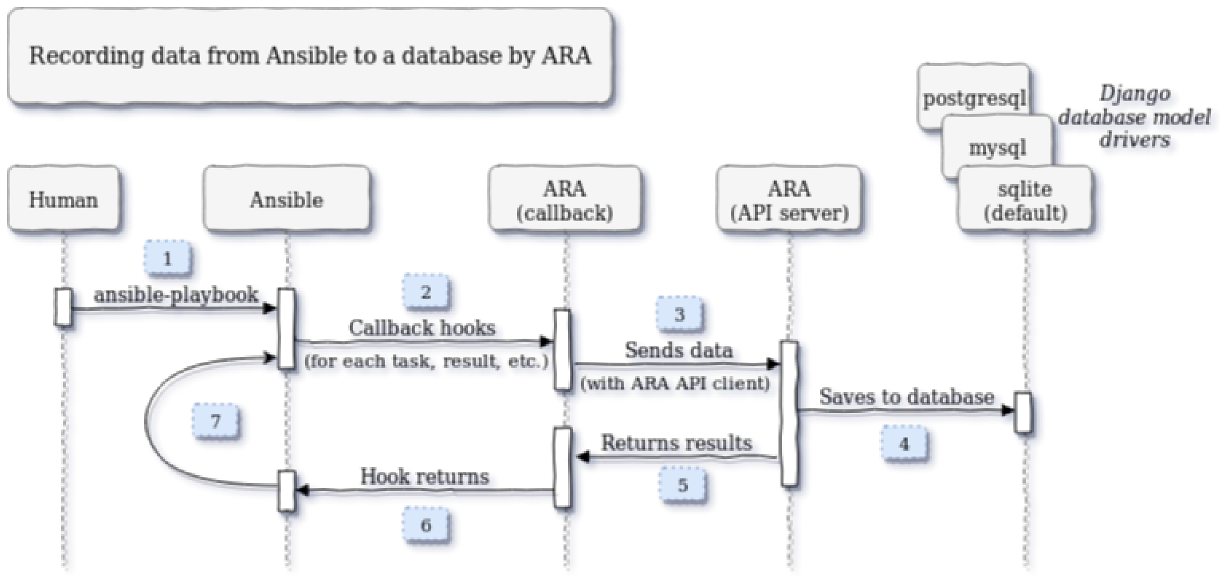
ARA Records Ansible playbooks and makes them easier to understand and troubleshoot.

ARA is currently composed of three different free and open source projects:

- https://github.com/ansible-community/ara for the REST API server and Ansible plugins
- https://github.com/ansible-community/ara-web for the standalone web interface
- https://github.com/ansible-community/ara-infra for project-specific infrastructure needs (such as the ara.recordsansible.org website)

1.1.2 How does it work ?

ARA Records Ansible playbooks through an Ansible callback plugin.



0. ARA is installed and Ansible is configured to use the callback plugin
1. An `ansible-playbook` command is executed
2. Ansible triggers the callback plugin for every event (`v2_playbook_on_start`, `v2_runner_on_failed`, etc.)
3. The relevant information is retrieved from the Ansible playbook execution context and is sent to the API server
4. The API server validates and serializes the data before storing it the configured database backend
5. The API server sends a response back to the API client with the results
6. The callback plugin returns, ending the callback hook
7. Ansible continues running the playbook until it fails or is completed (back to step 2)

Once the data has been saved in the database, it is made available for query by the API.

1.1.3 What's an Ansible callback ?

Ansible Callbacks are essentially hooks provided by Ansible. Ansible will send an event and you can react to it with a callback. You could use a callback to do things like print additional details or, in the case of ARA, record the playbook run data in a database.

1.1.4 Are there live demos available ?

Yes, you can find persistent and up-to-date live demos at api.demo.recordsansible.org for the API and web.demo.recordsansible.org for the ara-web standalone interface.

1.1.5 What versions of Ansible are supported ?

The upstream Ansible community and maintainers provide support for the latest three major stable releases and ARA follows the same support cycle.

For example, if the latest version of Ansible is 2.8, then the latest release of ARA will support 2.8 as well as 2.7 and 2.6.

For more information on Ansible's release and maintenance cycle, you can refer to the Ansible documentation.

If you are using a release of Ansible that is no longer supported, we strongly encourage you to upgrade as soon as possible in order to benefit from the latest features and security fixes.

Older unsupported versions of Ansible can contain unfixed security vulnerabilities (*CVE*).

1.1.6 What versions of Python are supported ?

Before version 1.0 of ARA, both python2 and python3 were supported. Versions of ARA after 1.0 are not designed to support python2 in consideration that python2 will reach end of life in January 2020.

1.1.7 Why ARA instead of <X> ?

Ansible is an awesome tool. It can be used for a lot of things.

Reading and interpreting the output of an `ansible-playbook` run, especially one that is either long running, involves a lot of hosts or prints a lot of output can be hard to understand and troubleshoot.

This is especially true when you happen to be running Ansible hundreds of times during the day, through automated means – for example when doing continuous integration or continuous delivery.

ARA aims to do one thing and do it well: Record Ansible playbooks and provide you with the tools you need to make your playbook results intuitive for you and for your systems.

The great thing about ARA is that it is not mutually exclusive with other software and systems you might already be using Ansible with today.

There is nothing preventing you from using ARA with other tools such as Ansible Tower (or AWX), Zuul, Jenkins or Rundeck since all you need to get started is to install and enable the ARA Ansible callback plugin.

1.1.8 Can I set up the different components of ARA on different servers ?

Yes.

The defaults are set to have the callback use the offline API client which expects the server dependencies installed and the data is saved to a local sqlite database.

However, the callback can also be configured to send data to a specified API server address and the API server can be configured to use a remote database server such as PostgreSQL or MySQL.

The web client interface provided by ara-web is stateless and requires an API server address to connect to. It can be installed anywhere that has access to the API server.

1.2 Installing ARA

1.2.1 Requirements and dependencies

ARA should work on any Linux distributions as long as python3 is available.

Since ARA provides Ansible plugins to record data, it should be installed wherever Ansible is running from so that Ansible can use those plugins. The API server does not require to run on the same machine as Ansible.

By default, only the client and plugin dependencies are installed. This lets users record and send data to a remote API server without requiring that the API server dependencies are installed.

If you are standing up an API server or if your use case is about recording data locally or offline, the required dependencies can be installed automatically by suffixing `[server]` to your pip install commands.

1.2.2 Installing from PyPi or from source

First, it is recommended to use a python virtual environment in order to avoid conflicts with your Linux distribution python packages.

```
python3 -m venv ~/.ara/virtualenv
```

To install the latest release of ARA from PyPi:

```
# With the API server dependencies
~/.ara/virtualenv/bin/pip install ara[server]

# Without the API server dependencies
~/.ara/virtualenv/bin/pip install ara
```

Installing from source is possible by using the git repository:

```
# With the API server dependencies
# (Suffixes don't work when supplying the direct git URL)
git clone https://github.com/ansible-community/ara ara-src
pip install ./ara-src[server]

# Without the API server dependencies
~/.ara/virtualenv/bin/pip install git+https://github.com/ansible-community/ara
```

1.2.3 With Ansible roles

Two roles are built-in to help users configure their API and web deployments:

- *ansible-role-ara-api* to install the python parts (Ansible plugins, API server)
- *ansible-role-ara-web* to install the web client (nodejs, react+patternfly)

1.3 Configuring Ansible to use ARA

To begin using ARA, you'll first need to tell Ansible where it is located.

Since this location will be different depending on your operating system and how you are installing ARA, there are convenient python modules to help you figure out the right paths.

Once you've set up the `callback_plugins` configuration or the `ANSIBLE_CALLBACK_PLUGINS` environment variable, Ansible will automatically use the ARA callback plugin to start recording data.

If you'd like to use the `ara_record` action plugin to record arbitrary data during your playbook, you'll need to set `action_plugins` and `ANSIBLE_ACTION_PLUGINS` as well.

1.3.1 Using setup helper modules

The modules can be used directly on the command line:

```
$ python3 -m ara.setup.path
/usr/lib/python3.7/site-packages/ara

$ python3 -m ara.setup.plugins
/usr/lib/python3.7/site-packages/ara/plugins

$ python3 -m ara.setup.action_plugins
/usr/lib/python3.7/site-packages/ara/plugins/action

$ python3 -m ara.setup.callback_plugins
/usr/lib/python3.7/site-packages/ara/plugins/callback

# Note: This doesn't export anything, it only prints the commands.
# If you want to export directly from the command, you can use:
#     source <(python3 -m ara.setup.env)
$ python3 -m ara.setup.env
export ANSIBLE_CALLBACK_PLUGINS=/usr/lib/python3.7/site-packages/ara/plugins/callback
export ANSIBLE_ACTION_PLUGINS=/usr/lib/python3.7/site-packages/ara/plugins/action

$ python3 -m ara.setup.ansible
[defaults]
callback_plugins=/usr/lib/python3.7/site-packages/ara/plugins/callback
action_plugins=/usr/lib/python3.7/site-packages/ara/plugins/action
```

Or from python, for example:

```
>>> from ara.setup import callback_plugins
>>> print(callback_plugins)
/usr/lib/python3.7/site-packages/ara/plugins/callback

>>> from ara.setup import action_plugins
>>> print(action_plugins)
/usr/lib/python3.7/site-packages/ara/plugins/action
```

1.4 Configuring the ARA Ansible plugins

ARA plugins uses the same mechanism and configuration files as Ansible to retrieve it's configuration. It comes with sane defaults that can be customized if need be.

The order of priority is the following:

1. Environment variables
2. `./ansible.cfg` (*in the current working directory*)
3. `~/.ansible.cfg` (*in the home directory*)
4. `/etc/ansible/ansible.cfg`

When using the `ansible.cfg` file, the configuration options must be set under the ara namespace, like so:

```
[ara]
variable = value
```

1.4.1 ARA callback plugin

The ARA callback plugin is the component that recovers data throughout the execution of your playbook and sends it to the API.

By default, the callback plugin is set up to use the local API server with the offline API client but you can also send data to a remote API server, specify credentials or customize other parameters:

```
callback: ara
callback_type: notification
requirements:
  - ara
short_description: Sends playbook execution data to the ARA API internally or over␣
↪HTTP
description:
  - Sends playbook execution data to the ARA API internally or over HTTP
options:
  api_client:
    description: The client to use for communicating with the API
    default: offline
    env:
      - name: ARA_API_CLIENT
    ini:
      - section: ara
        key: api_client
    choices: ['offline', 'http']
  api_server:
    description: When using the HTTP client, the base URL to the ARA API server
    default: http://127.0.0.1:8000
    env:
      - name: ARA_API_SERVER
    ini:
      - section: ara
        key: api_server
  api_username:
    description: If authentication is required, the username to authenticate with
    default: null
    env:
      - name: ARA_API_USERNAME
    ini:
      - section: ara
        key: api_username
  api_password:
    description: If authentication is required, the password to authenticate with
    default: null
    env:
      - name: ARA_API_PASSWORD
    ini:
      - section: ara
        key: api_password
  api_timeout:
    description: Timeout, in seconds, before giving up on HTTP requests
    type: integer
    default: 30
    env:
      - name: ARA_API_TIMEOUT
    ini:
      - section: ara
```

(continues on next page)

(continued from previous page)

```
        key: api_timeout
  ignored_facts:
    description: List of host facts that will not be saved by ARA
    type: list
    default: ["ansible_env"]
    env:
      - name: ARA_IGNORED_FACTS
    ini:
      - section: ara
        key: ignored_facts
  ignored_arguments:
    description: List of Ansible arguments that will not be saved by ARA
    type: list
    default: ["extra_vars"]
    env:
      - name: ARA_IGNORED_ARGUMENTS
    ini:
      - section: ara
        key: ignored_arguments
```

For example, a customized callback plugin configuration might look like this in an `ansible.cfg` file:

```
[ara]
api_client = http
api_server = https://api.demo.recordsansible.org
api_username = user
api_password = password
api_timeout = 15
ignored_facts = '["ansible_env", "ansible_all_ipv4_addresses"]'
ignored_arguments = '["extra_vars", "vault_password_files"]'
```

or as environment variables:

```
export ARA_API_CLIENT=http
export ARA_API_SERVER="https://api.demo.recordsansible.org"
export ARA_API_USERNAME=user
export ARA_API_PASSWORD=password
export ARA_API_TIMEOUT=15
export ARA_IGNORED_FACTS='["ansible_env", "ansible_all_ipv4_addresses"]'
export ARA_IGNORED_ARGUMENTS='["extra_vars", "vault_password_files"]'
```

1.4.2 ARA action plugin: ara_record

The `ara_record` action plugin recovers it's configuration from the callback plugin.

It is therefore not necessary to configure it explicitely other than enabling Ansible to find it by setting `action_plugins` in `ansible.cfg` or the `ANSIBLE_ACTION_PLUGINS` environment variable.

1.5 ARA API server configuration

The API 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.

The API 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.

1.5.1 Overview

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

Environment Variable	default	Usage
ARA_ALLOWED_HOSTS	`["127.0.0.1", "localhost", "::1"]`	Django's ALLOWED_HOSTS setting
ARA_BASE_DIR	`~/.ara/server`	Default directory for storing data and configuration
ARA_CORS_ORIGIN_WHITELIST	`["http://127.0.0.1:8000", "http://localhost:3000"]`	django-cors-headers's CORS_ORIGIN_WHITELIST setting
ARA_DATABASE_CONN_MAX_AGE	`0`	Django's CONN_MAX_AGE database setting
ARA_DATABASE_ENGINE	`django.db.backends.sqlite3`	Django's ENGINE database setting
ARA_DATABASE_HOST	`None`	Django's HOST database setting
ARA_DATABASE_NAME	`~/.ara/server/ansible.sqlite`	Django's NAME database setting
ARA_DATABASE_PASSWORD	`None`	Django's PASSWORD database setting
ARA_DATABASE_PORT	`None`	Django's PORT database setting
ARA_DATABASE_USER	`None`	Django's USER database setting
ARA_DEBUG	`False`	Django's DEBUG setting
ARA_DISTRIBUTED_SQLITE	`False`	Whether to enable distributed sqlite backend
ARA_DISTRIBUTED_SQLITE_PREFIX	`ara-api`	Prefix to delegate to the distributed sqlite backend
ARA_DISTRIBUTED_SQLITE_ROOT	`/var/www/logs`	Root under which sqlite databases are expected
ARA_ENV	`default`	Environment to load configuration for
ARA_LOGGING	See *ARA_LOGGING*	Logging configuration
ARA_LOG_LEVEL	`INFO`	Log level of the different components
ARA_PAGE_SIZE	`100`	Amount of results returned per page by the API
ARA_READ_LOGIN_REQUIRED	`False`	Whether authentication is required for reading data
ARA_SECRET_KEY	Randomized token, see *ARA_SECRET_KEY*	Django's SECRET_KEY setting
ARA_SETTINGS	`~/.ara/server/settings.yaml`	Path to an API server configuration file
ARA_WRITE_LOGIN_REQUIRED	`False`	Whether authentication is required for writing data

1.5.2 Configuration variables

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 the API server somewhere, you'll need to add your domain name to this list.

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_DATABASE_NAME` and `ARA_SETTINGS` parameters.

This is also used to determine the location where the default configuration file, `settings.yaml`, will be generated by the API 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="['https://api.ara.example.org', 'https://web.ara.example.org']"`
  - In a YAML configuration file:

```
dev:
  CORS_ORIGIN_WHITELIST:
    - http://127.0.0.1:8000
    - http://localhost:3000
production:
  CORS_ORIGIN_WHITELIST:
    - https://api.ara.example.org
    - https://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_DATABASE_CONN_MAX_AGE

- **Environment variable**: `ARA_DATABASE_CONN_MAX_AGE`
- **Configuration file variable**: `DATABASE_CONN_MAX_AGE`
- **Provided by**: Django's CONN_MAX_AGE database setting
- **Type**: `integer`
- **Default**: `0`

The lifetime of a database connection, in seconds.

The default of 0 closes database connections at the end of each request.

ARA_DATABASE_ENGINE

- **Environment variable**: `ARA_DATABASE_ENGINE`
- **Configuration file variable**: `DATABASE_ENGINE`
- **Provided by**: Django's ENGINE database setting
- **Type**: `string`
- **Default**: `django.db.backends.sqlite3`
- **Examples**:
  - `django.db.backends.sqlite3`
  - `django.db.backends.postgresql`
  - `django.db.backends.mysql`
  - `ara.server.db.backends.distributed_sqlite`

The Django database driver to use.

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

ARA_DATABASE_NAME

- **Environment variable**: `ARA_DATABASE_NAME`
- **Configuration file variable**: `DATABASE_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**: `DATABASE_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**: `DATABASE_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**: `DATABASE_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**: `DATABASE_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.

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_DISTRIBUTED_SQLITE

- **Environment variable**: `ARA_DISTRIBUTED_SQLITE`
- **Configuration file variable**: `DISTRIBUTED_SQLITE`
- **Provided by**: `ara.server.db.backends.distributed_sqlite` and `ara.server.wsgi.distributed_sqlite`
- **Type**: `bool`
- **Default**: `False`

Whether or not to enable the distributed sqlite database backend and WSGI application.

This feature is useful for loading different ARA sqlite databases dynamically based on request URLs.

For more information, see: *distributed sqlite backend*.

ARA_DISTRIBUTED_SQLITE_PREFIX

- **Environment variable**: `ARA_DISTRIBUTED_SQLITE_PREFIX`
- **Configuration file variable**: `DISTRIBUTED_SQLITE_PREFIX`
- **Provided by**: `ara.server.db.backends.distributed_sqlite` and `ara.server.wsgi.distributed_sqlite`
- **Type**: `string`
- **Default**: `ara-api`

Under which URL should requests be delegated to the distributed sqlite wsgi application. `ara-api` would delegate everything under `.*/ara-api/.*`

The path leading to this prefix must contain the `ansible.sqlite` database file, for example: `/var/www/logs/some/path/ara-api/ansible.sqlite`.

For more information, see: *distributed sqlite backend*.

ARA_DISTRIBUTED_SQLITE_ROOT

- **Environment variable**: `ARA_DISTRIBUTED_SQLITE_ROOT`
- **Configuration file variable**: `DISTRIBUTED_SQLITE_ROOT`

- **Provided by**: `ara.server.db.backends.distributed_sqlite` and `ara.server.wsgi.distributed_sqlite`
- **Type**: `string`
- **Default**: `/var/www/logs`

Root directory under which databases will be found relative to the requested URLs.

This will restrict where the WSGI application will go to seek out databases.

For example, the URL `example.org/some/path/ara-api` would translate to `/var/www/logs/some/path/ara-api`.

For more information, see: *distributed sqlite backend*.

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 the API 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:

```yaml
# 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, the API 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_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 the API server.

ARA_LOG_LEVEL

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

Log level of the different components from the API server.

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

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 API server configuration file to load settings from. The API server will generate 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 by the API server is provided by the dynaconf python library.

ARA_PAGE_SIZE

- **Environment variable**: `ARA_PAGE_SIZE`
- **Configuration file variable**: `PAGE_SIZE`
- **Type**: `integer`
- **Default**: `50`
- **Provided by**: django-rest-framework pagination

When querying the API server, the amount of items per page returned.

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_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_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*.

1.6 ARA API Server authentication and security

The API 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.

1.6.1 Setting a custom secret key

By default, the API 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.

1.6.2 User management

The API 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.
```

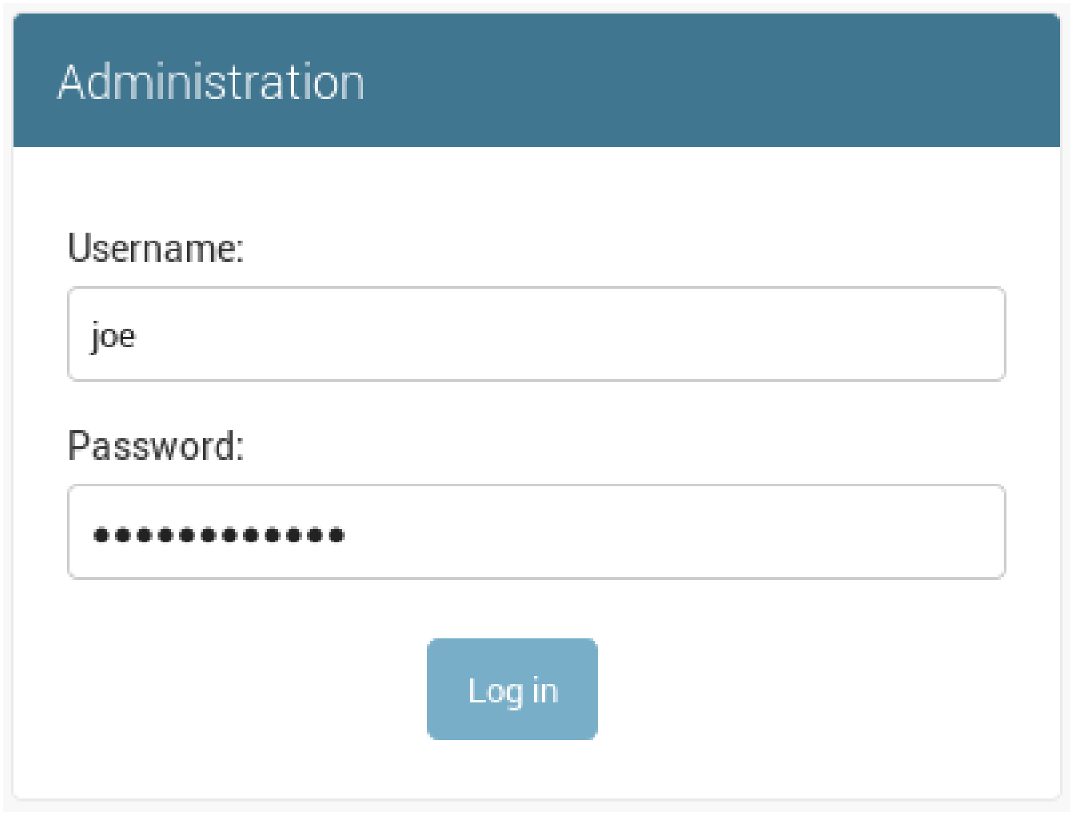
Tip: If you ever need to reset the password of a superuser account, this can be done with the "changepassword" command:

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

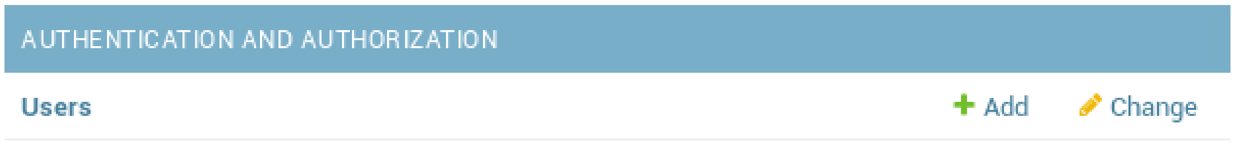
Once the superuser has been created, make sure the API server is started and then login to the Django web administrative interface using the credentials you just set up.

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/`.

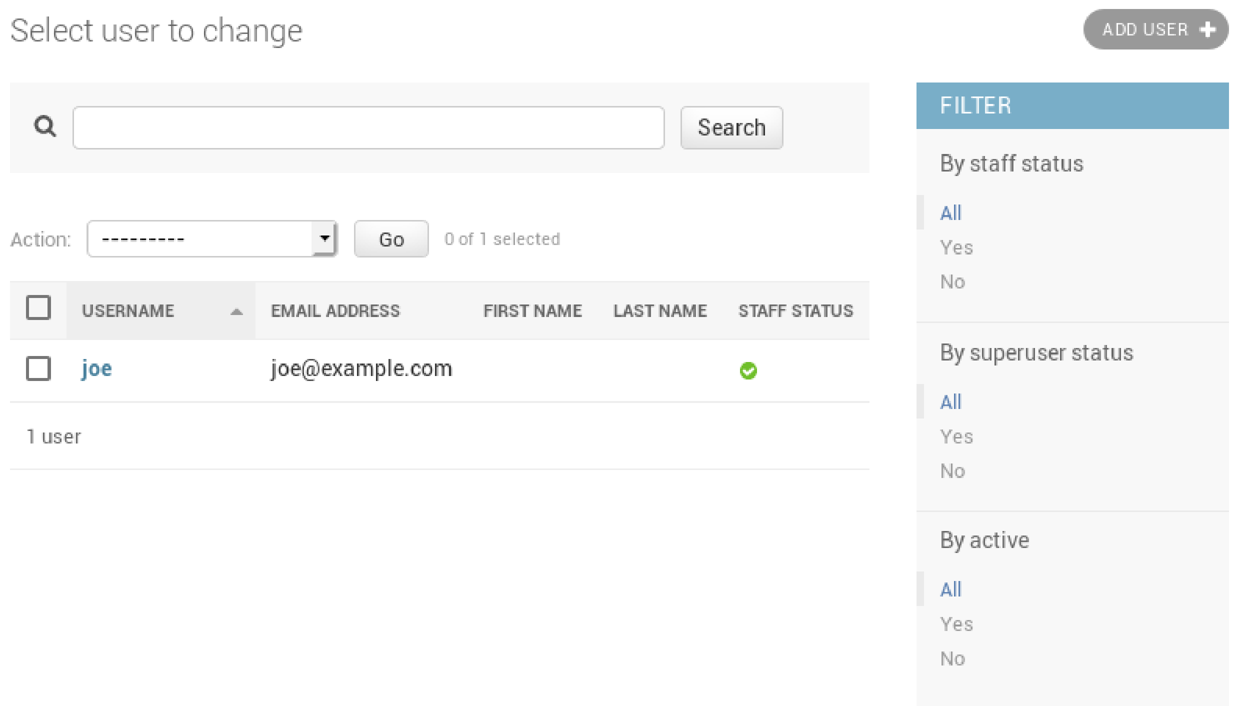
Log in to the admin interface:



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

1.6.3 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.

1.6.4 Setting up authentication for the Ansible plugins

The callback plugin used to record playbooks as well as the `ara_record` action plugin will need to authenticate against the API if authentication is enabled and required.

You can specify the necessary credentials through the `ARA_API_USERNAME` and `ARA_API_PASSWORD` environment variables or through your `ansible.cfg` file:

```
[defaults]
# ...
```

(continues on next page)

(continued from previous page)

```
[ara]
api_client = http
api_server = http://api.example.org
api_username = ara
api_password = password
```

1.6.5 Using authentication with the API clients

To instanciate an authenticated client with the built-in basic HTTP authentication provided by Django:

```
from ara.clients.utils import get_client
client = get_client(
    client="http",
    endpoint="http://api.example.org",
    username="ara",
    password="password"
)
```

If you have a custom authentication that is supported by the python requests library, you can also pass the relevant `auth` object directly to the client:

```
from ara.clients.http import AraHttpClient
from requests_oauthlib import OAuth1
auth = OAuth1(
    "YOUR_APP_KEY",
    "YOUR_APP_SECRET",
    "USER_OAUTH_TOKEN",
    "USER_OAUTH_TOKEN_SECRET"
)
client = AraHttpClient(endpoint="http://api.example.org", auth=auth)
```

1.6.6 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 the API server on another domain, the domain must be part of this list or the application server will deny any requests sent to it.

1.6.7 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 the API server.

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

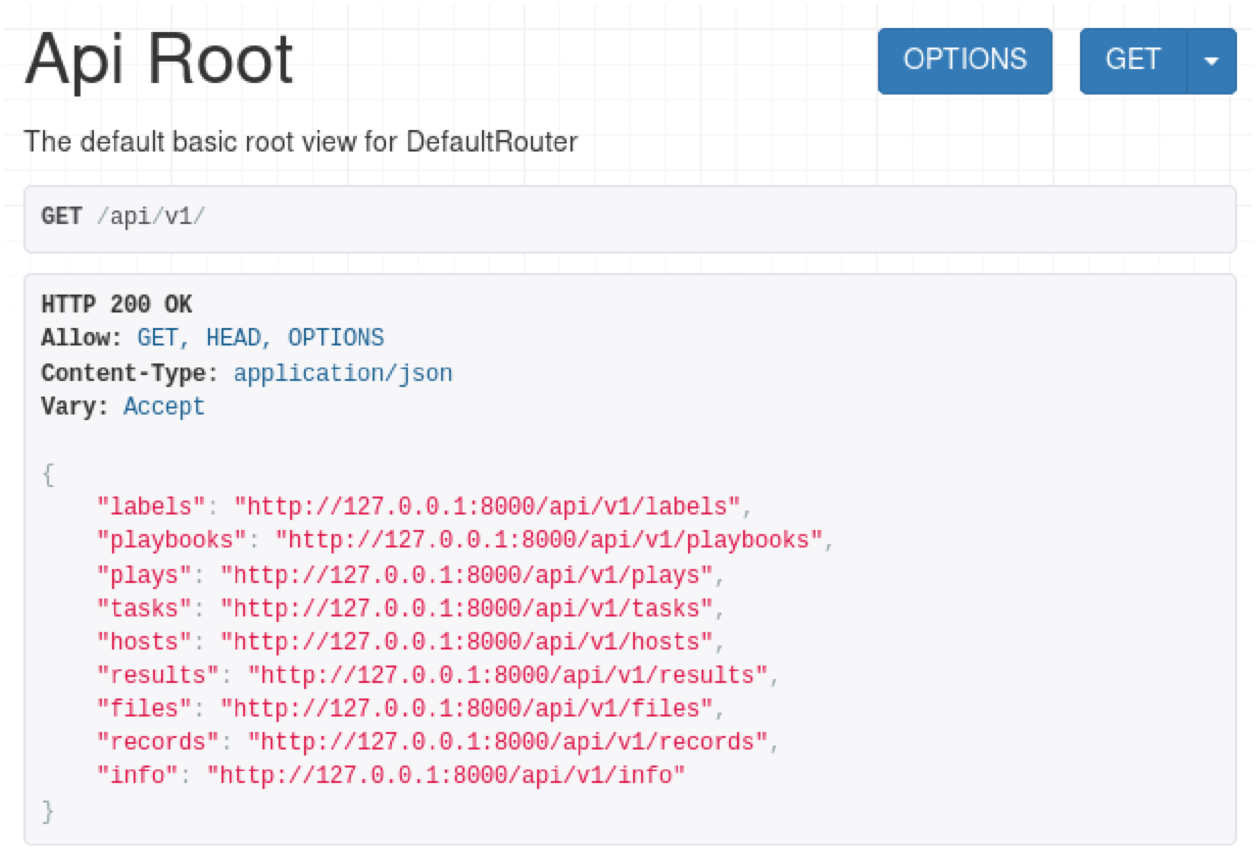
1.7 API Documentation

The API documentation is a work in progress.

1.7.1 Built-in API browser interface

ARA ships with a helpful interface to navigate the API directly from your favorite web browser.

For example, if you run `ara-manage runserver`, this interface would be available at `http://127.0.0.1:8000/api/v1/`:



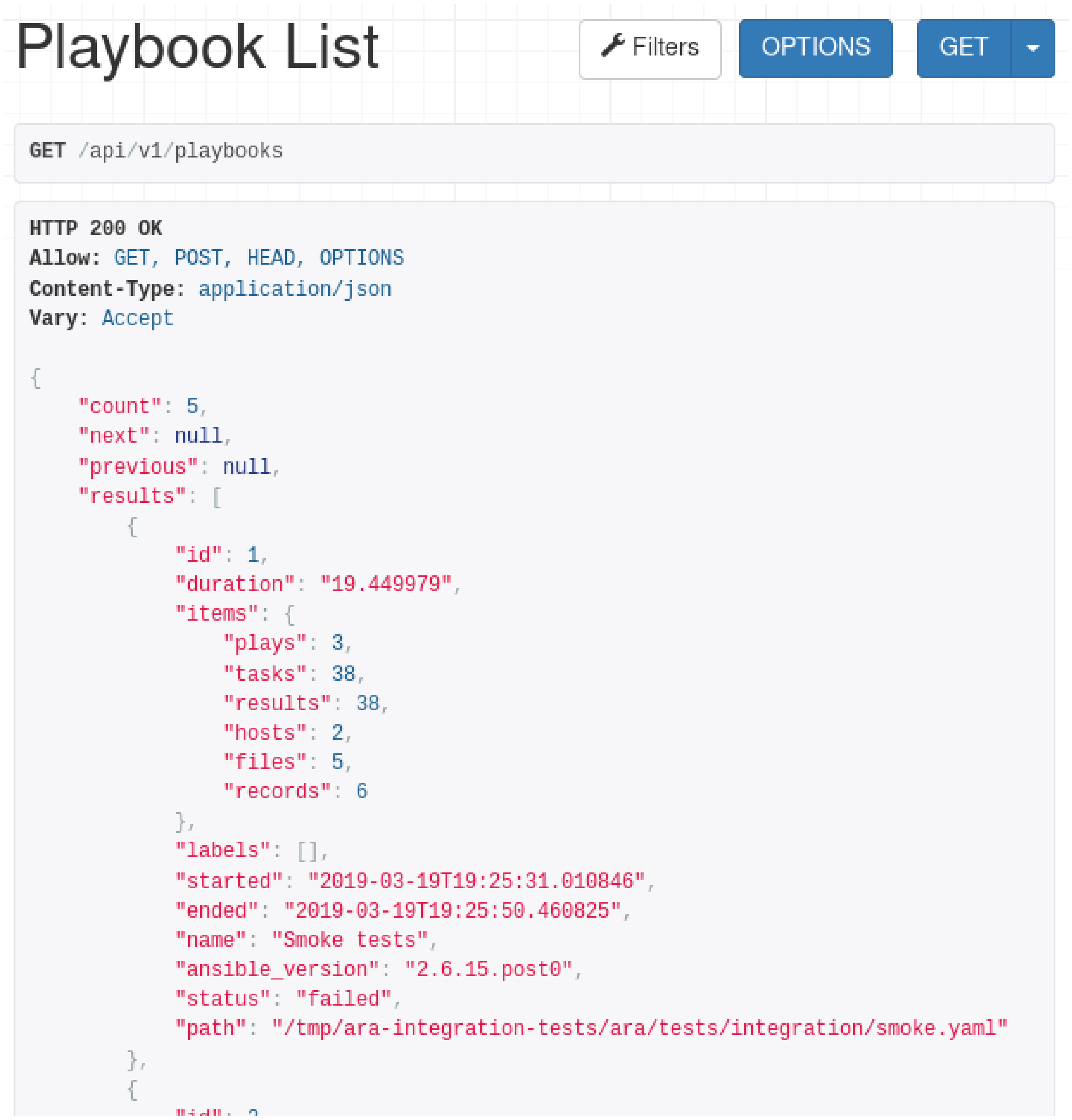
The default basic root view for DefaultRouter

```
GET /api/v1/
```

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "labels": "http://127.0.0.1:8000/api/v1/labels",
    "playbooks": "http://127.0.0.1:8000/api/v1/playbooks",
    "plays": "http://127.0.0.1:8000/api/v1/plays",
    "tasks": "http://127.0.0.1:8000/api/v1/tasks",
    "hosts": "http://127.0.0.1:8000/api/v1/hosts",
    "results": "http://127.0.0.1:8000/api/v1/results",
    "files": "http://127.0.0.1:8000/api/v1/files",
    "records": "http://127.0.0.1:8000/api/v1/records",
    "info": "http://127.0.0.1:8000/api/v1/info"
}
```

You can navigate the interface and drill down to list views, for example:

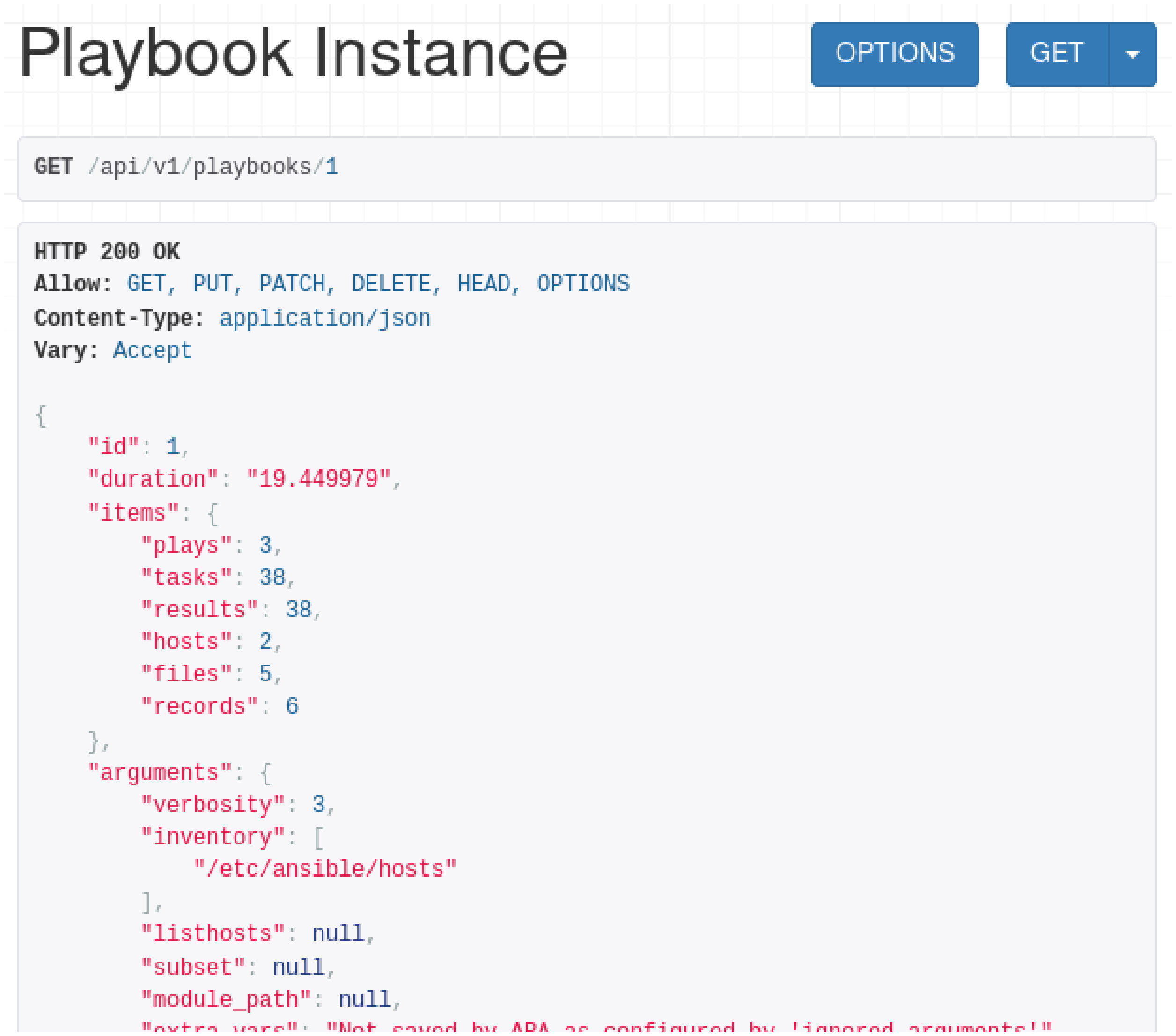


```
GET /api/v1/playbooks
```

```
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "count": 5,
    "next": null,
    "previous": null,
    "results": [
        {
            "id": 1,
            "duration": "19.449979",
            "items": {
                "plays": 3,
                "tasks": 38,
                "results": 38,
                "hosts": 2,
                "files": 5,
                "records": 6
            },
            "labels": [],
            "started": "2019-03-19T19:25:31.010846",
            "ended": "2019-03-19T19:25:50.460825",
            "name": "Smoke tests",
            "ansible_version": "2.6.15.post0",
            "status": "failed",
            "path": "/tmp/ara-integration-tests/ara/tests/integration/smoke.yaml"
        },
        {
```

You can also see what a detailed view looks like by querying a specific object id:



Playbook Instance

```
GET /api/v1/playbooks/1
```

```
HTTP 200 OK
Allow: GET, PUT, PATCH, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "id": 1,
    "duration": "19.449979",
    "items": {
        "plays": 3,
        "tasks": 38,
        "results": 38,
        "hosts": 2,
        "files": 5,
        "records": 6
    },
    "arguments": {
        "verbosity": 3,
        "inventory": [
            "/etc/ansible/hosts"
        ],
        "listhosts": null,
        "subset": null,
        "module_path": null,
        "extra_vars": "Not saved by ARA as configured by 'ignored_arguments'"
```

Alternatively, you may also find an up-to-date live demonstration of the API at `https://api.demo.recordsansible.org`.

1.7.2 Relationship between objects

The relationship between objects in the API should be straightforward with an understanding of how Ansible playbooks are executed.

Generally speaking, the data is organized in the following fashion:

```
 labels
   |
Playbook -> Play -> Task -> Result <- Host
   |                  |       |         |
  file               file   content    facts
```

- Every object is associated to a playbook (except *labels* which are applied to playbooks)
- In a playbook you have plays

- In plays you have tasks
- In tasks you have results
- Results have a relationship to their parent task and the host the task ran on
- Files are only associated to a playbook but tasks have a reference to the file they were executed from
- Records (provided by `ara_record`) are only associated to a playbook

Additional fields may only be available in the detailed views. For example:

- Playbooks arguments with `/api/v1/playbooks/<id>`
- Hosts facts with `/api/v1/hosts/<id>`
- Results content with `/api/v1/results/<id>`
- Files content with `/api/v1/files/<id>`

ARA ships with two built-in API clients to help you get started. You can learn more about those clients in *Using ARA API clients*.

1.8 Using ARA API clients

When installing ARA, you are provided with a REST API server and two API clients out of the box:

- `AraOfflineClient` can query the API without needing an API server to be running
- `AraHttpClient` is meant to query a specified API server over http

1.8.1 ARA Offline API client

If your use case doesn't require a remote or persistent API server, the offline client lets you query the API without needing to start an API server.

In order to use this client, you would instanciate it like this:

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

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

Note that, by default, instanciating an offline client will automatically run SQL migrations.

If you expect the migrations to have already been run when you instanciate the client, you can disable automatic SQL migrations with by specifying `run_sql_migrations=False`:

```python
client = AraOfflineClient(run_sql_migrations=False)
```

1.8.2 ARA HTTP API client

`AraHttpClient` works with the same interface, methods and behavior as `AraOfflineClient`.

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 an API server is listening
client = AraHttpClient(endpoint="https://api.demo.recordsansible.org")
```

1.8.3 Example API usage

For more details on the API endpoints, see *API Documentation*.

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

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

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

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

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

# If there are any results from our query, get more information about the
# failure and print something helpful
template = "{timestamp}: {host} failed '{task}' ({task_file}:{lineno})"
for playbook in playbooks["results"]:
    # Get a detailed version of the playbook that provides additional context
    detailed_playbook = client.get("/api/v1/playbooks/%s" % playbook["id"])

    # Iterate through the playbook to get the context
    # Playbook --> Play --> Task --> Result <- Host
    for play in detailed_playbook["plays"]:
        for task in play["tasks"]:
            for result in task["results"]:
                if result["status"] in ["failed", "unreachable"]:
                    print(template.format(
                        timestamp=result["ended"],
                        host=result["host"]["name"],
                        task=task["name"],
                        task_file=task["file"]["path"],
                        lineno=task["lineno"]
                    ))
```

Running this script would then provide an output that looks like the following:

```
2019-03-20T16:18:41.710765: localhost failed 'smoke-tests : Return false' (tests/
↪integration/roles/smoke-tests/tasks/test-ops.yaml:25)
2019-03-20T16:19:17.332663: localhost failed 'fail' (tests/integration/failed.yaml:22)
```

1.9 Distributed sqlite database backend

The ARA API server provides an optional backend that dynamically loads sqlite databases based on the requested URL with the help of a WSGI application middleware.

In summary, it maps an URL such as `http://example.org/some/path/ara-api` to a location on the file system like `/var/www/logs/some/path/ara-api` and loads an `ansible.sqlite` database from that directory, if it exists.

Note: This backend is not enabled by default and is designed with a specific range of use cases in mind. This documentation attempts to explain if this might be a good fit for you.

1.9.1 Use case

Running at least one Ansible playbook with the ARA Ansible callback plugin enabled will generate a database at `~/.ara/server/ansible.sqlite` by default.

sqlite, in the context of ARA, is good enough for most use cases:

- it is portable: everything the API server needs is in a single file that you can upload anywhere
- no network dependency or latency: sqlite is on your filesystem and doesn't rely on a remote database server
- relatively lightweight: Ansible's own integration tests used ~13MB for 415 playbooks, 1935 files, 12456 tasks, 12762 results, 586 hosts (and host facts)

However, since write concurrency does not scale very well with sqlite, it might not be a good fit if you plan on having a single API server handle data for multiple `ansible-playbook` commands running at the same time.

The distributed sqlite database backend and WSGI middleware provide an alternative to work around this limitation.

This approach works best if it makes sense to logically split your playbooks into different databases. One such example is in continuous integration (CI) where you might have multiple jobs running Ansible playbooks concurrently.

If each CI job is recording to its own database, you probably no longer have write concurrency issues and the database can be uploaded in your logs or as an artifact after the job has been completed.

The file hierarchy on your log or artifact server might end up looking like this:

```
/var/www/logs/
├── 1
│   ├── ara-api
│   │   └── ansible.sqlite
│   └── console.txt
├── 2
│   ├── logs.tar.gz
│   └── some
│       └── path
│           └── ara-api
│               └── ansible.sqlite
└── 3
    ├── builds.txt
    ├── dev
    │   └── ara-api
    │       └── ansible.sqlite
    └── prod
```

(continues on next page)

(continued from previous page)

```
        └── ara-api
            └── ansible.sqlite
```

With the above example file tree, a single instance of the API server with the distributed sqlite backend enabled would be able to respond to queries at the following endpoints:

- http://example.org/1/ara-api
- http://example.org/2/some/path/ara-api
- http://example.org/3/dev/ara-api
- http://example.org/3/prod/ara-api

1.9.2 Configuration

For enabling and configuring the distributed sqlite backend, see:

- *ARA_DISTRIBUTED_SQLITE*
- *ARA_DISTRIBUTED_SQLITE_PREFIX*
- *ARA_DISTRIBUTED_SQLITE_ROOT*

When recording data to a sqlite database, the location of the database can be defined with *ARA_DATABASE_NAME*.

1.10 Setting playbook names and labels

ARA provides the ability for users to specify playbook names and labels in order to better distinguish playbooks run in different environments or purposes.

Names and labels are also searchable by the ARA API, allowing you to find playbooks matching your query.

Names and labels are set as regular Ansible variables:

- `ara_playbook_name`
- `ara_playbook_labels`

These variables can be provided by your Ansible inventory, directly in your playbook file, as extra-vars or any other way supported by Ansible.

For example, in an inventory:

```
[dev]
host1
host2

[dev:vars]
ara_playbook_name=deploy-dev
ara_playbook_labels='["deploy", "dev"]'
```

In a playbook:

```
- name: Deploy dev environment
  hosts: dev
  vars:
    ara_playbook_name: deploy-dev
```

(continues on next page)

(continued from previous page)

```
    ara_playbook_labels:
        - deploy
        - dev
  roles:
    - application
```

Or as extra-vars:

```
ansible-playbook -i hosts playbook.yaml \
    -e ansible_playbook_name=deploy-dev \
    -e ansible_playbook_labels='["deploy", "dev"]'
```

1.11 Recording arbitrary data in playbooks

ARA comes with a built-in Ansible action plugin called `ara_record`.

This module can be used as an action for a task in your Ansible playbooks in order to register whatever you'd like in a key/value format, for example:

```
- name: Test playbook
  hosts: localhost
  tasks:
    - name: Get git version of playbooks
      command: git rev-parse HEAD
      register: git_version

    - name: Record git version
      ara_record:
        key: "git_version"
        value: "{{ git_version.stdout }}"
      register: version

    - name: Print recorded data
      debug:
        msg: "{{ version.playbook_id }} - {{ version.key }}: {{ version.value }}
```

It also supports different types of data which will have an impact on how a value might later be parsed or displayed:

```
- name: Record different things
  ara_record:
    key: "{{ item.key }}"
    value: "{{ item.value }}"
    type: "{{ item.type }}"
  loop:
    - { key: "log", value: "error", type: "text" }
    - { key: "website", value: "http://domain.tld", type: "url" }
    - { key: "data", value: '{ "key": "value" }', type: "json" }
    - { key: "somelist", value: ['one', 'two'], type: "list" }
    - { key: "somedict", value: {'key': 'value' }, type: "dict" }
```

1.11.1 Recording data for playbooks after completion

It is possible to run an `ara_record` task on a specific playbook that might already be completed by specifying a playbook. This is particularly useful for recording data that might only be available or computed after your playbook run has been completed:

```
---
# Write data to a specific (previously run) playbook
- ara_record:
    playbook: 14
    key: logs
    value: "{{ lookup('file', '/var/log/ansible.log') }}"
    type: text
```

Or as an ad-hoc command:

```
ansible localhost -m ara_record \
        -a "playbook=14 key=logs value={{ lookup('file', '/var/log/ansible.log') }}"
```

This data will be recorded inside ARA's database and associated with the particular playbook run that was executed.

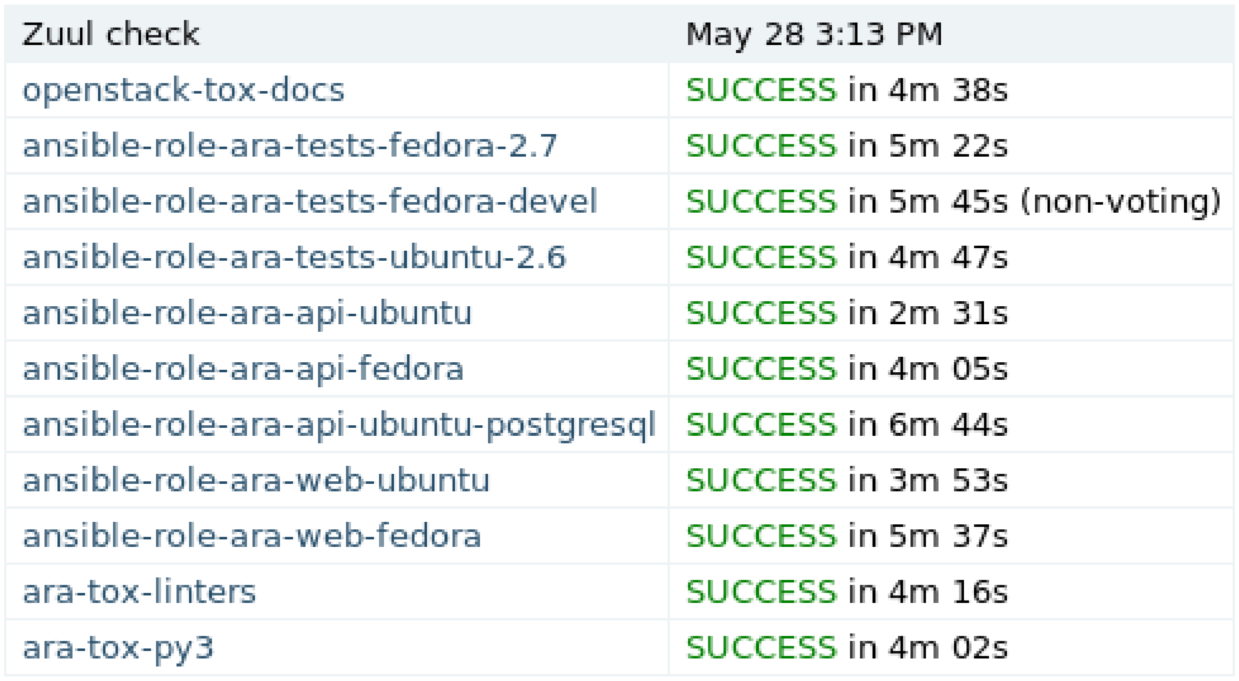
These records can later be retrieved through the API or through a web interface.

1.12 How to contribute

ARA Records Ansible is an open source community project and welcomes contributions, whether they are in the form of feedback, comments, suggestions, bugs, documentation, code, or code reviews.

The ARA code review and CI infrastructure is hosted by opendev.org which provides Gerrit for code review, Zuul for CI/CD as well as many other systems.

All new patches are automatically tested with lint, unit and a variety of integration test scenarios. The end result is higher standards, better code, more testing, less regressions and more stability.



Zuul check	May 28 3:13 PM
openstack-tox-docs	SUCCESS in 4m 38s
ansible-role-ara-tests-fedora-2.7	SUCCESS in 5m 22s
ansible-role-ara-tests-fedora-devel	SUCCESS in 5m 45s (non-voting)
ansible-role-ara-tests-ubuntu-2.6	SUCCESS in 4m 47s
ansible-role-ara-api-ubuntu	SUCCESS in 2m 31s
ansible-role-ara-api-fedora	SUCCESS in 4m 05s
ansible-role-ara-api-ubuntu-postgresql	SUCCESS in 6m 44s
ansible-role-ara-web-ubuntu	SUCCESS in 3m 53s
ansible-role-ara-web-fedora	SUCCESS in 5m 37s
ara-tox-linters	SUCCESS in 4m 16s
ara-tox-py3	SUCCESS in 4m 02s

1.12.1 Running tests locally

- Unit tests: `tox -e py37`
- Linters tests (pep8/flake8/bandit/bashate/black/isort/etc): `tox -e linters`
- Documentation tests (builds to `docs/build/html`): `tox -e docs`
- Integration tests: `tox -e ansible-integration`

1.12.2 Issues and pull requests

ARA has several projects that are mirrored to GitHub:

- https://opendev.org/recordsansible/ara -> https://github.com/ansible-community/ara
- https://opendev.org/recordsansible/ara-web -> https://github.com/ansible-community/ara-web
- https://opendev.org/recordsansible/ara-infra -> https://github.com/ansible-community/ara-infra

While new issues, bugs and feature requests should be filed on GitHub, we are unable to accept pull requests on GitHub at this time.

We would appreciate the opportunity to accept them in the future by trying Gerritbot to synchronize pull requests to Gerrit.

In the meantime, what follows are the required steps in order to send a patch to Gerrit.

1.12.3 Set up your Ubuntu Launchpad account

OpenDev's Gerrit instance currently uses Launchpad for authentication. If you do not already have a Launchpad account, you will need to create one here.

1.12.4 Set up your Gerrit code review account

If you'll be contributing code or code reviews, you'll need to set up your Gerrit code review account.

Once you have your Launchpad account, you will be able to sign in to review.opendev.org.

To be able to submit code, Gerrit needs to have your public SSH key in the same way Github does. To do that, click on your name at the top right and go to the settings where you will see the tab to set up your SSH key.

Note that if the username from your local machine differs from the one in Gerrit, you might need to set it up in your local `~/.ssh/config` file like this:

1.12.5 Installing Git Review

Git Review is a python module that adds a "git review" command that wraps around the process of sending a commit for review in Gerrit. You need to install it to be able to send patches for code reviews.

There are different ways to install git-review, choose your favorite.

1.12.6 Sending a patch for review

The process looks a bit like this:

```
$ git clone https://opendev.org/recordsansible/ara
# or git clone https://github.com/ansible-community/ara
$ cd ara
# Create a new local branch
$ git checkout -b super_cool_feature
# hack on super_cool_feature
$ git commit -a --message="This is my super cool feature"
$ git review
```

When you send a commit for review, it'll create a code review request in Gerrit for you. When that review is created, it will automatically be tested by a variety of jobs that the ARA maintainers have set up to test every patch that is sent.

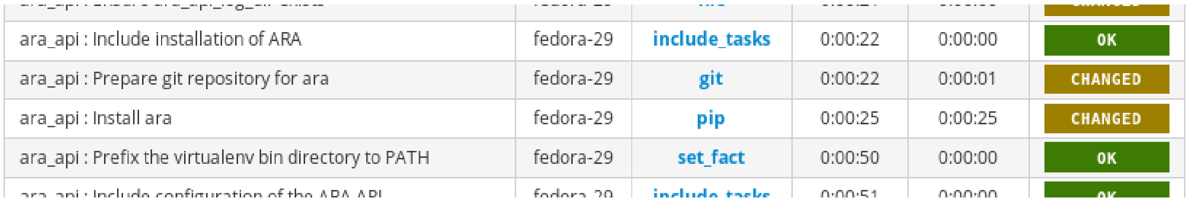
We'll check for things like code quality (pep8/flake8), run unit tests to catch regressions and we'll also run both integration tests on different operating systems to make sure everything really works.

The result of the tests are added as a comment in the review when all of them are completed. If you're interested in digging into the logs for a particular test, clicking on the results of the test will take you to console, debug logs and a built version of ARA's web interface.

If you get a failed test result and you believe you have fixed the issue, add the files, amend your commit (`git commit --amend`) and send it for review once again. This will create a new patchset that will be up for review and testing.

To be able to merge a patch, the tests have to come back successful and the core reviewers must provide their agreement with the patch.

1.13 ansible-role-ara-api



ara_api : Include installation of ARA	fedora-29	include_tasks	0:00:22	0:00:00	OK
ara_api : Prepare git repository for ara	fedora-29	git	0:00:22	0:00:01	CHANGED
ara_api : Install ara	fedora-29	pip	0:00:25	0:00:25	CHANGED
ara_api : Prefix the virtualenv bin directory to PATH	fedora-29	set_fact	0:00:50	0:00:00	OK

This Ansible role provides a framework for installing one or many instances of ARA Records Ansible in a variety of opinionated deployment topologies.

It is currently tested and supported against Ubuntu 18.04 and Fedora 29.

1.13.1 Role Variables

See defaults/main.yaml.

```
# Root directory where every file for the ARA installation are located
ara_api_root_dir: "{{ ansible_user_dir }}/.ara"

# Directory where logs are written to
ara_api_log_dir: "{{ ara_api_root_dir }}/logs"

# Whether or not ara should be installed in a virtual environment.
# This defaults to true to prevent conflicting with system or distribution
```

(continues on next page)

(continued from previous page)

```
# python packages.
ara_api_venv: true

# When using a virtualenv, path to where it will be installed
ara_api_venv_path: "{{ ara_api_root_dir }}/virtualenv"

# How ARA will be installed
# - source [default]: installs from a local or remote git repository
# - pypi [planned]: installs from pypi
ara_api_install_method: source

# When installing from source, the URL or filesystem path where the git source
# repository can be cloned from.
ara_api_source: "https://opendev.org/recordsansible/ara"

# When installing from source, location where the source repository will be checked␣
↪out to.
ara_api_source_checkout: "{{ ara_api_root_dir }}/git/ara"

# Version of ARA to install
# When installing from source, this can be a git ref (tag, branch, commit, etc)
# When installing from PyPi, it would be a version number that has been released.
# When using "latest" as the source version, HEAD will be used
# When using "latest" as the pypi version, the latest release will be used
ara_api_version: master

# The frontend/web server for serving the ARA API
# It is recommended to specify a web server when deploying a production environment.
# - null [default]: No frontend server will be set up.
# - nginx: Nginx will be configured in front of the WSGI application server.
# - apache [planned]
ara_api_frontend_server: null

# Path to a custom vhost configuration jinja template
# The vhost configuration templates provided by the role are simple by design
# and are not sufficient to cover every use cases.
# Use this variable if you need to have your own custom nginx or apache configuration.
ara_api_frontend_vhost: null

# The WSGI server for running ARA's API server
# - null [default]: No persistent WSGI application server will be set up. Only the␣
↪offline API client will work.
# - gunicorn: gunicorn will be installed and set up to run the API as a systemd␣
↪service.
# - mod_wsgi [planned]
ara_api_wsgi_server: null

# Address and port on which the wsgi server will bind
# Changing this value means you might need to adjust "ara_api_allowed_hosts" and
# "ara_api_cors_origin_whitelist".
ara_api_wsgi_bind: "127.0.0.1:8000"

# When using a frontend server, the domain it will be listening on
ara_api_fqdn: "{{ ansible_default_ipv4['address'] }}"

####################################
# ara API configuration settings
```

(continues on next page)

(continued from previous page)

```
# For more information, see documentation: https://ara.readthedocs.io
####################################

# ARA_BASE_DIR - Default directory for storing data and configuration
ara_api_base_dir: "{{ ara_api_root_dir }}/server"

# ARA_SETTINGS - Path to an ARA API configuration file
ara_api_settings: "{{ ara_api_base_dir }}/settings.yaml"

# ARA_ENV - Environment to load configuration for
ara_api_env: default

# ARA_READ_LOGIN_REQUIRED - Whether authentication is required for reading data
ara_api_read_login_required: false

# ARA_WRITE_LOGIN_REQUIRED - Whether authentication is required for writing data
ara_api_write_login_required: false

# ARA_PAGE_SIZE - Amount of results returned per page by the API
ara_api_page_size: 100

# ARA_LOG_LEVEL - Log level of the different components
ara_api_log_level: INFO

# ARA_LOGGING - Python logging configuration
ara_api_logging:
  disable_existing_loggers: false
  formatters:
    normal:
      format: '%(asctime)s %(levelname)s %(name)s: %(message)s'
  handlers:
    console:
      class: logging.handlers.TimedRotatingFileHandler
      formatter: normal
      level: "{{ ara_api_log_level }}"
      filename: "{{ ara_api_log_dir }}/server.log"
      when: 'midnight'
      interval: 1
      backupCount: 30
  loggers:
    ara:
      handlers:
        - console
      level: "{{ ara_api_log_level }}"
      propagate: 0
  root:
    handlers:
      - console
    level: "{{ ara_api_log_level }}"
  version: 1

# ARA_CORS_ORIGIN_ALLOW_ALL - django-cors-headers's CORS_ORIGIN_WHITELIST_ALLOW_ALL␣
↪setting
ara_api_cors_origin_allow_all: false

# ARA_CORS_ORIGIN_WHITELIST - django-cors-headers's CORS_ORIGIN_WHITELIST setting
ara_api_cors_origin_whitelist:
```

(continues on next page)

(continued from previous page)

```
  - "http://127.0.0.1:8000"
  - "http://localhost:3000"

# ARA_SERVER_ALLOWED_HOSTS - Django’s ALLOWED_HOSTS setting
ara_api_allowed_hosts:
  - "127.0.0.1"
  - "localhost"
  - "::1"
  - "{{ ansible_default_ipv4['address'] }}"

# ARA_DEBUG - Django's DEBUG setting
# It is not recommended to run with debug enabled in production.
ara_api_debug: false

# ARA_SECRET_KEY - Django's SECRET_KEY setting
# Note: If no key is provided, a random one will be generated once and persisted
ara_api_secret_key: null

# ARA_DISTRIBUTED_SQLITE - Whether to enable distributed sqlite backend
ara_api_distributed_sqlite: false

# ARA_DISTRIBUTED_SQLITE_PREFIX - Prefix to delegate to the distributed sqlite backend
ara_api_distributed_sqlite_prefix: ara-api

# ARA_DISTRIBUTED_SQLITE_ROOT - Root under which sqlite databases are expected
ara_api_distributed_sqlite_root: /var/www/logs

# ARA_DATABASE_ENGINE - Django’s ENGINE database setting
ara_api_database_engine: "{{ ara_api_distributed_sqlite | ternary('ara.server.db.
↪backends.distributed_sqlite', 'django.db.backends.sqlite3') }}"

# ARA_DATABASE_NAME - Django’s NAME database setting
ara_api_database_name: "{{ ara_api_base_dir }}/ansible.sqlite"

# ARA_DATABASE_USER - Django’s USER database setting
ara_api_database_user: null

# ARA_DATABASE_PASSWORD - Django’s PASSWORD database setting
ara_api_database_password: null

# ARA_DATABASE_HOST - Django’s HOST database setting
ara_api_database_host: null

# ARA_DATABASE_PORT - Django’s PORT database setting
ara_api_database_port: null

# ARA_DATABASE_CONN_MAX_AGE - Django's CONN_MAX_AGE database setting
ara_api_database_conn_max_age: 0
```

1.13.2 TL;DR

Playbook that runs the role with defaults:

```
- name: Install ARA with default settings and no persistent API server
  hosts: all
```

(continues on next page)

(continued from previous page)

```yaml
  gather_facts: yes
  roles:
    - ara_api
```

What the role ends up doing by default:

- Installs required packages (`git`, `virtualenv`, etc.) if superuser privileges are available
- Stores everything in the home directory of the user in `~/.ara`
- Retrieves ARA from source
- Installs ARA in a virtualenv
- Generates a random secret key if none are already configured or provided
- Sets up API configuration in `~/.ara/server/settings.yaml`
- Runs the API SQL migrations (`ara-manage migrate`)

1.13.3 About deployment topologies

This Ansible role is designed to support different opinionated topologies that can be selected with role variables.

For example, the following role variables are used to provide the topology from the `TL;DR` above:

- `ara_api_install_method: source`
- `ara_api_wsgi_server: null`
- `ara_api_database_engine: django.db.backends.sqlite3`
- `ara_api_web_server: null`

The intent is that as the role gains support for other install methods, wsgi servers, database engines or web servers, it will be possible to mix and match according to preference or requirements.

Perhaps ARA could be installed from pypi and run with gunicorn, nginx and mysql. Or maybe it could be installed from distribution packages and set up to run with apache, mod_wsgi and postgresql. Or any combination of any of those.

1.13.4 Example playbooks

Install ARA and set up the API to be served by a persistent gunicorn service:

```yaml
- name: Install ARA and set up the API to be served by gunicorn
  hosts: all
  gather_facts: yes
  vars:
    ara_api_wsgi_server: gunicorn
  roles:
    - ara_api
```

Install ARA and set up the API to be served by nginx in front of gunicorn:

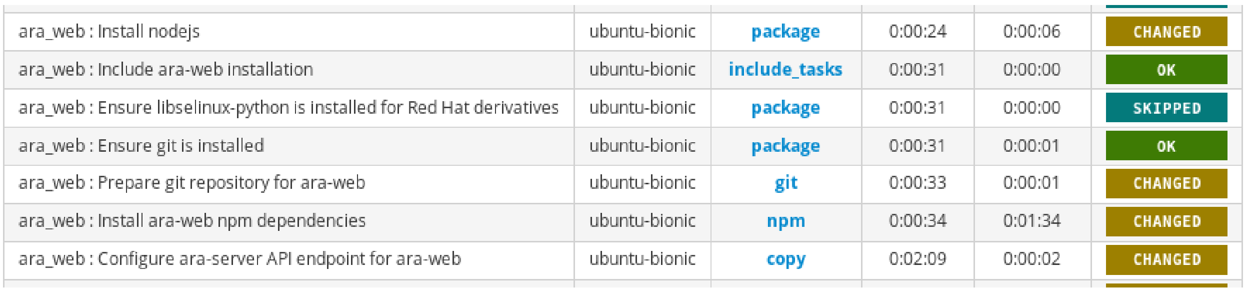
```yaml
# Requires superuser privileges to set up nginx and the ara-api service
# The API will be reachable at http://api.ara.example.org
- name: Install ARA and set up the API to be served by nginx in front of gunicorn
  hosts: all
```

(continues on next page)

(continued from previous page)

```
  gather_facts: yes
  vars:
     ara_api_frontend_server: nginx
     ara_api_wsgi_server: gunicorn
     ara_api_fqdn: api.ara.example.org
     ara_api_allowed_hosts:
       - api.ara.example.org
     ara_api_frontend_vhost: custom_vhost.conf.j2
   roles:
     - ara_api
```

1.14 ansible-role-ara-web



ara_web : Install nodejs	ubuntu-bionic	package	0:00:24	0:00:06	CHANGED
ara_web : Include ara-web installation	ubuntu-bionic	include_tasks	0:00:31	0:00:00	OK
ara_web : Ensure libselinux-python is installed for Red Hat derivatives	ubuntu-bionic	package	0:00:31	0:00:00	SKIPPED
ara_web : Ensure git is installed	ubuntu-bionic	package	0:00:31	0:00:01	OK
ara_web : Prepare git repository for ara-web	ubuntu-bionic	git	0:00:33	0:00:01	CHANGED
ara_web : Install ara-web npm dependencies	ubuntu-bionic	npm	0:00:34	0:01:34	CHANGED
ara_web : Configure ara-server API endpoint for ara-web	ubuntu-bionic	copy	0:02:09	0:00:02	CHANGED

This Ansible role provides a framework for installing one or many instances of ara-web in a variety of opinionated deployment topologies.

It is currently tested and supported against Ubuntu 18.04 and Fedora 29.

1.14.1 Role Variables

See defaults/main.yaml.

```
# Root of where files will be stored for ara-web
ara_web_root_dir: "{{ ansible_user_dir }}/.ara"

# When using static builds without the dev server, path to ara-web static assets
ara_web_static_dir: "{{ ara_web_root_dir }}/www/ara-web"

# How ara-web will be installed
# - source (default): installs from a local or remote git repository specified by ara_
↪web_source
# - npm (planned): installs from npm
ara_web_install_method: source

# When installing from source, the location of the remote or local git repository
ara_web_source: "https://opendev.org/recordsansible/ara-web"

# Location where ara-web will be checked out
ara_web_source_checkout: "{{ ara_web_root_dir }}/git/ara-web"

# Location where node_modules will be installed
ara_web_node_modules_dir: "{{ ara_web_source_checkout }}"
```

(continues on next page)

(continued from previous page)

```
# Version of ara-web to install
# This can be a git ref (tag, branch, commit) when installed from source
# When using "latest" as the source version, HEAD will be used
ara_web_version: latest

# Whether to use the embedded react web server or not
# Setting this to false means ara-web will be statically built instead
ara_web_dev_server: true

# When the development server is enabled, the address it will be listening on
ara_web_dev_server_bind_address: 127.0.0.1

# When the development server is enabled, the port it will be listening on
ara_web_dev_server_bind_port: 3000

# Version of nodesource nodejs repositories to install
ara_web_nodejs_version: 10

# ara-server API endpoint to use
ara_web_api_endpoint: "http://127.0.0.1:8000"

# The frontend server for serving ara-web
# - null (default): none, users are expected to use the development server directly␣
↪or deploy their own web server
# - nginx: when performance of the development server is an issue
# - apache (planned)
ara_web_frontend_server: null

# When using a frontend server, you can override the default vhost configuration
# template by specifying the path to your own template file.
ara_web_frontend_vhost: null

# When using a frontend server, the hostname to listen on
ara_web_fqdn: "{{ ansible_default_ipv4['address'] }}"
```

1.14.2 TL;DR

This is what the role does by default out of the box:

- Retrieves ara-web from source
- Installs nodejs LTS (v10)
- Installs ara-web dependencies with npm
- Configures an ara-server API endpoint in ara-web's `public/config.json` file
- Sets up a systemd unit file for running ara-web with the embedded development server

1.14.3 About deployment topologies

This Ansible role is designed to support different opinionated topologies that can be selected with role variables.

For example, the following role variables are defaults used to provide the topology from the `TL;DR` above:

- `ara_web_install_method: source`

- `ara_web_dev_server: true`
- `ara_web_frontend_server: null`

The intent is that as the role gains support for other install methods or frontend servers, it will be possible to mix and match according to preference or requirements.

1.14.4 Example playbooks

Deploy the ARA API and web client on the same machine with defaults:

```yaml
- name: Deploy ARA API and web client
  hosts: all
  gather_facts: yes
  vars:
    # ara_api
    ara_api_fqdn: api.ara.example.org
    ara_api_wsgi_server: gunicorn
    ara_api_allowed_hosts:
    - api.ara.example.org
    ara_api_cors_origin_whitelist:
    - web.ara.example.org
    # ara_web
    ara_web_fqdn: web.ara.example.org
    ara_web_api_endpoint: "http://api.ara.example.org"
roles:
    - ara_api
    - ara_web
```

Deploy only ara-web behind nginx and point it to a remote API endpoint:

```yaml
# Note: Don't forget to add the web fqdn in the remote cors_origin_whitelist.
# Otherwise, the web client might not be authorized to query the API.
- name: Deploy ara-web for remote API endpoint
  hosts: all
  gather_facts: yes
  vars:
    ara_web_fqdn: web.ara.example.org
    ara_web_api_endpoint: "http://api.remoteara.example.org"
    ara_web_frontend_server: nginx
    ara_web_frontend_vhost: custom-web-vhost.conf.j2
roles:
    - ara_web
```