RBAC Documentation

Release 0.0.1

Red Hat, Inc.

Apr 06, 2022

Contents:

RBAC, accessed through the application program interface (API), is a role based access control tool. It is designed to identify principals, define roles containing permissions for related principals, and define groups of principals to which those roles apply.

CHAPTER 1

Managing Resources with Role Based Access Control

1.1 Overview

The Role Based Access Control (RBAC) service allows users to control access to Platform services and resources. Management of RBAC resources can only be performed by account/organization administrators. There are three primary resources RBAC uses to control access to services: Principals, Groups, and Roles. In order to give any Principal(user) access to an application resource, they must be added to a Group. The associated group must then be granted access by being bound to a Role or set of Roles. A Role defines access to specific application resources with a specific set of permissions.

1.1.1 Managing Principals

A Principal is an authenticated user that is a member of the Account. Principals can be searched and added to Groups in order to grant access to resources. RBAC itself does not perform any enforcement of that access, instead enforcement is left to the application logic. API endpoints involving Principals may only be accessed by Account Administrators.

1.1.2 Managing Groups

A Group is a collection of Principals used to grant access to a resource. A Principal can be a member of many Groups, and a Group can be associated with multiple Roles.

Permissions for Group API access

A user can only view (read) Groups. Administrators can view (read) and create/update (write) Groups.

1.1.3 Managing Roles

A Role defines a set of access control lists (ACLs). These ACLs define Permissions and contain Resource Definitions.

Permissions

A Permission is a three part object: application, resource type, operation

Application specifies the service or domain for the resource, for example:

```
- catalog
- approval
- cost-management
```

Resource type defines the resource to be controlled, for example:

```
- aws.account
- openshift.cluster
```

Operation defines the application logic action, for example:

```
- read
- write
- order
```

Note that in any of the above stanzas, * is taked to mean "all".

Resource Definitions

Resource Definitions are a somewhat trickier aspect of our implementation of RBAC currently only used by the cost-management service.

In general, ALL roles should be created with a resourceDefinitions stanza of `[]`. This is taken to mean "no additional filtering" and will generally result in the expected behavior.

In specific cases where the application logic has been written to handle them, however, resource access can be limited by specifing an attribute filter in the resourceDefinitions stanza as below.

Specifying a single resource with an attribute filter:

```
"resourceDefinitions": [
    {
        "attributeFilter": {
            "key": "uuid",
            "operation": "equal",
            "value": "39c8cecd-e595-46fb-8908-13365d59d5e8"
        }
    }
]
```

While you can specify resources individually you can also specify a multiple resource with an attribute filter as follows:

```
"resourceDefinitions": [
    {
        "attributeFilter": {
            "key": "uuid",
            "operation": "in",
            "value": "39c8cecd-e595-46fb-8908-13365d59d5e8,9928e33b-e28f-4e82-b996-
↪12e222f08098"
        }
    }
]
```

Permissions for Role API access

Only an account administrator can view (read) roles or create/update (write) roles. Non-administrator can view (read) roles within their scope with scope specified in the API call - `?scope=principal`.

CHAPTER 2

Development Information

2.1 Insights Role Based Access Control README

2.1.1 About

Insights RBAC's goal is to provide an open source solution for storing roles, permissions and groups.

Full documentation is available through readthedocs. More info is available through platformdocs.

Getting Started

This is a Python project developed using Python 3.6. Make sure you have at least this version installed.

Additionally, the development environment installation requires the postgresql-devel package installed for your distribution before running properly.

Development

To get started developing against Insights-rbac first clone a local copy of the git repository.

```
git clone https://github.com/RedHatInsights/insights-rbac.git
```

Developing inside a virtual environment is recommended. A Pipfile is provided. Pipenv is recommended for combining virtual environment (virtualenv) and dependency management (pip). To install pipenv, use pip

```
pip3 install pipenv
```

Then project dependencies and a virtual environment can be created using

```
pipenv install --dev
```

To activate the virtual environment run

```
pipenv shell
```

Preferred Environment

Please refer to Working with Openshift.

Alternative Environment

If deploying with Openshift seems overly complex you can try an alternate local environment where you will need to install and setup some of the dependencies and configuration.

Configuration

This project is developed using the Django web framework. Many configuration settings can be read in from a *.env* file. An example file *.env.example* is provided in the repository. To use the defaults simply

```
cp .env.example .env
```

Modify as you see fit.

Database

PostgreSQL is used as the database backend for Insights-rbac. A docker-compose file is provided for creating a local database container. If modifications were made to the .env file the docker-compose file will need to be modified to ensure matching database credentials. Several commands are available for interacting with the database.

```
# This will launch a Postgres container
make start-db

# This will run Django's migrations against the database
make run-migrations

# This will stop and remove a currently running database and run the above commands
make reinitdb
```

Assuming the default .env file values are used, to access the database directly using psql run

```
psql rbac -U rbacadmin -h localhost -p 15432
```

There is a known limitation with docker-compose and Linux environments with SELinux enabled. You may see the following error during the postgres container deployment:

```
"mkdir: cannot create directory '/var/lib/pgsql/data/userdata': Permission denied"␣
↪can be resolved by granting ./pg_data ownership permissions to uid:26 (postgres␣
↪user in centos/postgresql-96-centos7)
```

If a docker container running Postgres is not feasible, it is possible to run Postgres locally as documented in the Postgres tutorial. The default port for local Postgres installations is *5432*. Make sure to modify the *.env* file accordingly. To initialize the database run

```
make run-migrations
```

You may also run migrations explicitly, and in parallel, by specifying *TENANT_PARALLEL_MIGRATION_MAX_PROCESSES* (the number of concurrent processes to run migrations) and/or *TENANT_PARALLEL_MIGRATION_CHUNKS* (the number of migrations for each process to run at a time). Both of these values default to 2. *Be mindful of the fact that bumping these values will consume more database connections:*

TENANT_PARALLEL_MIGRATION_MAX_PROCESSES=4 TENANT_PARALLEL_MIGRATION_CHUNKS=2 ./rbac/manage.py migrate

Seeds

Default roles and groups are automatically seeded when the application starts by default unless either of the following environment variables are set to 'False' respectively:

```
PERMISSION_SEEDING_ENABLED
ROLE_SEEDING_ENABLED
GROUP_SEEDING_ENABLED
```

Locally these are sourced from */rbac/management/role/definitions/*.json*, while the config maps in deployed instances are source from our RBAC config repo. **If any changes to default roles/groups are required, they should be make there.**

You can also execute the following Django command to run seeds manually. It's recommended that you disable db signals while running seeds with *ACCESS_CACHE_CONNECT_SIGNALS=False*. Caching will be busted after seeding for each tenant has processed. You may also specify the number of concurrent threads in which seeds should be run, by setting *MAX_SEED_THREADS* either in the process, or the app environment. The default value is 2. *Be mindful of the fact that bumping this value will consume more database connections:*

```
ACCESS_CACHE_CONNECT_SIGNALS=False MAX_SEED_THREADS=2 ./rbac/manage.py seeds [--
↪roles|--groups|--permissions]
```

Server

To run a local dev Django server you can use

```
make serve
```

To run the local dev Django on a specific port use:

```
make PORT=8111 serve
```

Making Requests

You can make requests to RBAC locally to mimic traffic coming from the gateway, or locally within the same cluster from another internal service.

Basic/JWT Auth with an Identity Header

By default, with the *DEVELOPMENT* variable set to *True*, the *dev_middleware.py* will be used. This will ensure that a mock identity header will be set on all requests for you. You can modify this header to add new users to your tenant by changing the *username*, create new tenants by changing the *account_number*, and toggling between admin/non-admins by flipping *is_org_admin* from *True* to *False*.

This will allow you to simulate a JWT or basic-auth request from the gateway.

Serivce to Service Requests

RBAC also allows for service-to-service requests. These requests require a PSK, and some additional headers in order to authorize the request as an "admin". To test this locally, do the following:

First disable the local setting of the identity header in *dev_middleware.py* by [commenting this line out](https://github.com/RedHatInsights/insights-rbac/blob/master/rbac/rbac/dev_middleware.py#L53)

Next, start the server with:

```
make serve SERVICE_PSKS='{"catalog": {"secret": "abc123"}}'
```

Verify that you cannot access any endpoints requiring auth:

```
curl http://localhost:8000/api/rbac/v1/roles/ -v
```

Verify that if you pass in the correct headers/values, you _can_ access the endpoint:

```
curl http://localhost:8000/api/rbac/v1/roles/ -v -H 'x-rh-rbac-psk: abc123' -H 'x-rh-
↪rbac-account: 10001' -H 'x-rh-rbac-client-id: catalog'
```

Change the 'x-rh-rbac-client-id', 'x-rh-rbac-psk' and 'x-rh-rbac-account' header values to see that you should get back a 401 (or 400 with an account that doesn't exist).

You can also send a request _with_ the identity header explicitly in the curl command along with the service-to-service headers to verify that the identity header will take precedence.

API Documentation Generation

To generate and host the API documentation locally you need to Install APIDoc.

Generate the project API documenttion by running the following command

```
make gen-apidoc
```

In order to host the docs locally you need to collect the static files

```
make collect-static
```

Now start the server with as described above and point your browser to **http://127.0.0.1:8000/apidoc/index.html**.

Testing and Linting

Insights-rbac uses tox to standardize the environment used when running tests. Essentially, tox manages its own virtual environment and a copy of required dependencies to run tests. To ensure a clean tox environment run

```
tox -r
```

This will rebuild the tox virtual env and then run all tests.

To run unit tests specifically:

```
tox -e py36
```

To lint the code base

```
tox -e lint
```

Caveats

For all requests to the Insights RBAC API, it is assumed and required that principal information for the request be sent in a header named: *x-rh-identity*. The information in this header is used to determine the tenant, principal and other account-level information for the request.

Consumers of this API through cloud.redhat.com should not be concerned with adding this header, as it will be overwritten by the gateway. All traffic to the Insights RBAC API comes through Akamai and the Insights 3scale Gateway. The gateway is responsible for adding the *x-rh-identity* header to all authenticated requests.

Any internal, service-to-service requests which do **not** go through the gateway will need to have this header added to each request.

This header requirement is not reflected in the openapi.json spec, as it would cause spec-based API clients to require the header, which would be superfluously added to all requests on cloud.redhat.com.

Contributing

This repository uses [pre-commit](https://pre-commit.com) to check and enforce code style. It uses [Black](https://github.com/psf/black) to reformat the Python code and [Flake8](http://flake8.pycqa.org) to check it afterwards. Other formats and text files are linted as well.

Install pre-commit hooks to your local repository by running:

    $ pre-commit install

After that, all your commited files will be linted. If the checks don't succeed, the commit will be rejected. Please make sure all checks pass before submitting a pull request. Thanks!

Repositories of the roles to be seeded

Default roles can be found in the RBAC config repo.

For additional information please refer to Contributing.

2.2 Contributing to insights-rbac

Thank you for your interest in contributing to this project!

The following are a set of guidelines for contributing to insights-rbac. These are guidelines, not rules. Use your best judgement. Feel free to suggest changes to this document in a pull-request.

This document uses RFC 2119 keywords to indicate requirement levels.

2.2.1 Reporting Bugs & Requesting Features

We use Github Issues to track bug reports and feature requests.

When submitting a bug report, please be as detailed as possible. Include as much of these items as you have:

1. steps to reproduce the bug
2. error messages with stacktraces
3. logs
4. any relevant configuration settings
5. environment details

When submitting a feature request, please submit them in the form of a user story with acceptance criteria:

> As a [user], I want [a thing], So that [some goal].
>
> When complete, I will be able to:
>
> 1. [do this]
> 2. [do that]
> 3. [do another]

2.2.2 Contributing Code (Pull Requests)

All code contributions MUST come in the form of a pull-request. Pull-requests will be reviewed for a variety of criteria. This section attempts to capture as much of that criteria as possible.

2.2.3 Readability and Style considerations

In general, we believe that code is read just as much as it is executed. So, writing readable code is just as important as writing functional code.

Pull-requests MUST follow Python style conventions (e.g. PEP 8 and PEP 20 and conform to generally recognized best practices. Pull-requests MAY also choose to conform to additional style guidelines, e.g. Google's Python Style Guide.

We do use automation whenever possible to ensure a basic level of acceptability. Pull-requests MUST pass a linter (pylint or flake8) without errors.

We do recognize that sometimes linters can get things wrong. They are useful tools, but they are not perfect tools. Pull-requests SHOULD pass a linter without warnings.

Pull-requests MAY include disabling a specific linter check. If your pull-request disables linting it MUST include a comment block detailing why that particular check was disabled and it MUST be scoped as narrowly as possible. i.e. Don't disable linting on an entire class or method when disabling the check for a single statement will do.

2.2.4 Code testing considerations

We believe that well-tested code is a critical component to every successful project. For this reason, all pull-requests MUST include unit test cases and those unit tests MUST pass when run.

The unit tests SHOULD cover all of the code in the pull-request. Our goal is to maintain at least 90% test coverage.

In general, the test cases SHOULD cover both success and failure conditions.

An attempt SHOULD be made to cover all code branches. You SHOULD also attempt to include tests for all class and method parameters. e.g. If a method accepts a boolean, there should be tests for when that boolean is True, False, and None.

2.3 Working with OpenShift

We are currently developing using OpenShift version 3.11. There are different setup requirements for Mac OS and Linux (instructions are provided for Fedora).

Run *oc cluster up* once before running the make commands to generate the referenced config file.

Openshift does offer shell/tab completion. It can be generated for either bash/zsh and is available by running *oc completion bash|zsh* The following example generates a shell script for completion and sources the file.

```
oc completion zsh > $HOME/.oc/oc_completion.sh
source $HOME/.oc/oc_completion.sh
```

2.3.1 Local Development Cluster

The following commands can be used to manually create an OpenShift cluster with the necessary components to run RBAC.

```
# bring up a new dev cluster
oc cluster up \
      --image=$(OC_SOURCE) \
      --version=$(OC_VERSION) \
      --host-data-dir=$(OC_DATA_DIR)

# log in as cluster admin
oc login -u system:admin

# import postgresql-9.6 imagestream
oc create -n openshift istag postgresql:9.6 --from-image=centos/postgresql-96-centos7

# import python-3.6 imagestream
oc create -n openshift istag python:3.6 --from-image=centos/python-36-centos7

# create the app
oc new-app openshift/templates/django-postgresql-persistent.json
```

Alternatively, make commands are provided as a convenience.

```
# Start the OpenShift cluster
make oc-up

# Terminate the OpenShift cluster
```

(continues on next page)

(continued from previous page)

```
make oc-down

# Clean out local data
make oc-clean
```

There are a few ways to use OpenShift while developing RBAC. It is possible to spin up the entire application and its dependent services, or just the dependent services can be spun up while using the local Django dev server.

```
# Run everything through OpenShift
make oc-up-all

# Run *just* a database in Openshift, while running the server locally
make oc-up-db
# Run Django migrations to initialize the database
make oc-run-migrations
# Run the Django server locally with access to the OpenShift database
make oc-serve
```

To gain temporary access to the database within OpenShift, port forwarding is used.

```
# Port forward to 15432
make oc-forward-ports

psql rbac -U rbacadmin -p 15432 -h localhost

# Stop port forwarding
make oc-stop-forwarding-ports
```

2.3.2 Fedora

The setup process for Fedora is well outlined in two articles. First, get Docker up and running. Getting Started with Docker on Fedora.

Then follow these instructions to get OpenShift setup OpenShift — Fedora Developer Portal.

2.3.3 Mac OS

There is a known issue with Docker for Mac ignoring *NO_PROXY* settings which are required for OpenShift. (https://github.com/openshift/origin/issues/18596) The current solution is to use a version of Docker prior to 17.12.0-ce, the most recent of which can be found at docker-community-edition-17091-ce-mac42-2017-12-11

Docker needs to be configured for OpenShift. A local registry and proxy are used by OpenShift and Docker needs to be made aware.

Add *172.30.0.0/16* to the Docker insecure registries which can be accomplished from Docker -> Preferences -> Daemon. This article details information about insecure registries Test an insecure registry | Docker Documentation

Add *172.30.1.1* to the list of proxies to bypass. This can be found at Docker -> Preferences -> Proxies

2.3.4 Troubleshooting

OpenShift uses Docker to run containers. When running a cluster locally for developement, deployment can be strained by low resource allowances in Docker. For development it is recommended that Docker have at least 4 GB of memory available for use.

Also, if Openshift services misbehave or do not deploy properly, it can be useful to spin the cluster down, restart the Docker service and retry.

2.4 Generating the Template

To generate a new template from a running configuration, use this command.

```
oc export all -o yaml --as-template=my-new-template > openshift/my-new-template.yaml
```

CHAPTER 3

Installation

Insights RBAC provides a web server for its API interaction.

This guide will focus on deploying Insighs RBAC into an existing OpenShift cluster.

3.1 Deploying the RBAC API

The RBAC application contains two components - a web service and database.

OpenShift

A basic deployment configuration is contained within the application's openshift template file `openshift/rbac-template.yaml`. This template should be acceptable for most use cases. It provides parameterized values for most configuration options.

To deploy the RBAC API application using the provided templates, you can use the provided `Makefile`:

```
make oc-create-all
```

To deploy individual components, there are also `make` commands provided for your convenience:

> Deploy the API web application: `make oc-create-rbac` Deploy the PostgreSQL database: `make oc-create-db`

Docker Compose

The RBAC API can also be deployed with Docker Compose with the following steps. Before these steps can complete, the postgresql-devel package for your distribution must be installed.

- Create a Docker bridge network named `rbac-network`: `docker network create rbac-network`
- Start RBAC server and database: `make docker-up`

This command will run database migraitons and start the API server. Once complete the API server will be running on port 8000 on your localhost.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search