ATMO Documentation

Release 2018.4.0

Mozilla Foundation

Aug 14, 2018

Contents:

Welcome to the documentation of **ATMO**, the code that runs Mozilla's Telementry Analysis Service.

ATMO is a self-service portal to launch on-demand AWS EMR clusters with Apache Spark, Apache Zeppelin and Jupyter installed. Additionally it allows to schedule Spark jobs to run regularly based on uploaded Jupyter (and soon Zeppelin) notebooks.

It provides a management UI for public SSH keys when launching on-demand clusters, login via Google auth and flexible adminstration interfaces for users and admins.

Behind the scenes it's shipped as Docker images and uses Python 3.6 for the web UI (Django) and the task management (Celery).

CHAPTER 1

Overview

This is a quick overview of how ATMO works, both from the perspective of code structure as well as an architecture overview.

CHAPTER 2

Workflows

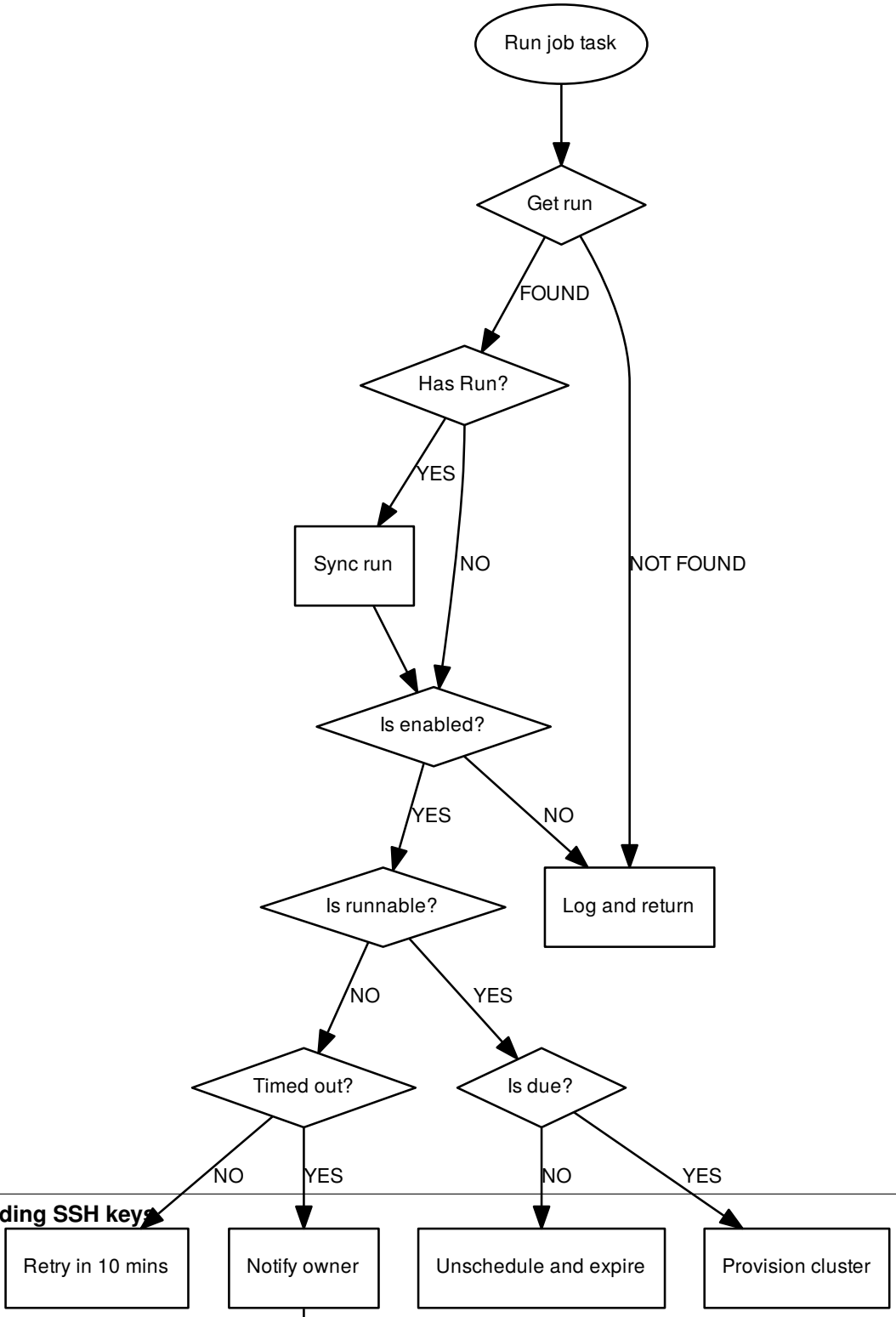
There are a few workflows in the ATMO code base that are of interest and decide how it works:

- Adding SSH keys
- Creating an on-demand cluster
- Scheduling a Spark job

2.1 Adding SSH keys

2.2 Creating an on-demand cluster

2.3 Scheduling a Spark job



CHAPTER 3

Maintenance

3.1 EMR releases

3.2 Dependency upgrades

ATMO uses a number of dependencies for both the backend as well as the frontend web UI.

For the first we're using pip requirements.txt files to manage dependencies, whose version should always be pinned to an exact version and have hashes attached for the individual release files for pip's hash-checking mode.

For the frontend dependencies we're using NPM's default `package.json` and NPM >= 5's `package-lock.json`.

See below for guides to update both.

3.2.1 Python dependencies

Python dependencies are installed using pip during the Docker image build process. As soon as you build the docker image using `make build` it'll check if the appropriate requirements file has changed and rebuilds the container image if needed.

To add a new Python dependency please:

- Log in into the web container with `make shell`.
- Change into the requirements folder with `cd /app/requirements`
- Then, depending on the area in which the dependency you're about to add/update, chose one of the following files to update:
  - `build.txt` - dependencies for when the Docker image is built, the default requirements file, basically.
  - `docs.txt` - dependencies for building the Sphinx based docs.
  - `tests.txt` - dependencies for running the *test suite*.

- Add/update the dependency in the file you chose, including a hash for pip's hash-checking mode. You may want to use the tool hashin to do that, e.g. `hashin -r /app/requirements/docs.txt Sphinx`.
- Leave the container again with `exit`.
- Run `make build` on the host machine.

That will rebuild the images used by docker-compose.

3.2.2 NPM ("front-end") dependencies

The front-end dependencies are installed when building the Docker images just like Python dependencies.

To add a new dependency to ATMO, please:

- Log in into the web container with `make shell`.
- Install the new dependency with `npm install --save-exact <name>`
- Delete the temporary `node_modules` folder: `rm -rf /app/node_modules`.
- Leave the container again with `exit`.
- Run `make build` on the host machine
- Extend the `NPM_FILE_PATTERNS` setting in the `settings.py` file with the files that are needed to be copied by Django's `collectstatic` management command.

That will rebuild the images used by docker-compose.

CHAPTER 4

Development

ATMO is maintained on GitHub in its own repository at:

> https://github.com/mozilla/telemetry-analysis-service

Please clone the Git repository using the git command line tool or any other way you're comfortable with, e.g.:

```
git clone https://github.com/mozilla/telemetry-analysis-service
```

ATMO also uses Docker for local development and deployment. Please make sure to install Docker and Docker Compose on your computer to contribute code or documentation changes.

4.1 Configuration

To set the application up, please copy the `.env-dist` file to one named `.env` and then update the variables starting with `AWS_` with the appropriate value.

Set the `DJANGO_SECRET_KEY` variable using the output of the following command after logging into the Docker container with `make shell`:

```
python -c "import secrets; print(secrets.token_urlsafe(50))"
```

To start the application, run:

```
make up
```

4.2 Run the tests

There's a sample test in `tests/test_users.py` for your convenience, that you can run using the following command on your computer:

```
make test
```

This will spin up a Docker container to run the tests, so please set up the development setup first.

The default options for running the test are in `pytest.ini`. This is a good set of defaults.

Alternatively, e.g. when you want to only run part of the tests first open a console to the web container..

```
make shell
```

and then run pytest directly:

```
pytest
```

Some helpful command line arguments to pytest (won't work on `make test`):

`--pdb`: Drop into pdb on test failure.

`--create-db`: Create a new test database.

`--showlocals`: Shows local variables in tracebacks on errors.

`--exitfirst`: Exits on the first failure.

`--lf, --last-failed`: Run only the last failed tests.

See `pytest --help` for more arguments.

4.2.1 Running subsets of tests and specific tests

There are a bunch of ways to specify a subset of tests to run:

- all the tests that have "foobar" in their names:

  ```
  pytest -k foobar
  ```

- all the tests that don't have "foobar" in their names:

  ```
  pytest -k "not foobar"
  ```

- tests in a certain directory:

  ```
  pytest tests/jobs/
  ```

- specific test:

  ```
  pytest tests/jobs/test_views.py::test_new_spark_job
  ```

See http://pytest.org/latest/usage.html for more examples.

4.3 Troubleshooting

Docker-Compose gives an error message similar to "ERROR: client and server don't have same version (client : 1.21, server: 1.18)"

> Make sure to install the latest versions of both Docker and Docker-Compose. The current versions of these in the Debian repositories might not be mutually compatible.

Django gives an error message similar to `OperationalError: SOME_TABLE does not exist`

> The database likely isn't set up correctly. Run `make migrate` to update it.

Django gives some other form of `OperationalError`, and we don't really care about the data that's already in the database (e.g., while developing or testing)

> Database errors are usually caused by an improper database configuration. For development purposes, recreating the database will often solve the issue.

Django gives an error message similar to `'NoneType' object has no attribute 'get_frozen_credentials'`.

- The AWS credentials on the current machine are likely not correctly set.
- Set them in your **ENVIRONMENT VARIABLES** (these environment variables are transferred to the docker container, from definitions in `.env`).
- See the [relevant section of the Boto3 docs](https://boto3.readthedocs.org/en/latest/guide/configuration.html#environment-variables) for more details.

CHAPTER 5

Deployment

Releasing ATMO happens by tagging a CalVer based Git tag with the following pattern:

    YYYY.M.N

`YYYY` is the four-digit year number, `M` is a single-digit month number and `N` is a single-digit zero-based counter which does **NOT** relate to the day of the release. Valid versions numbers are:

- 2017.10.0
- 2018.1.0
- 2018.12.12
- 1970.1.1

Once the Git tag has been pushed to the main GitHub repository using `git push origin --tags`, Circle CI will automatically build a tagged Docker image after the tests have passed and push it to Docker Hub. From there the Mozilla CloudOPs team has configured a stage/prod deployment pipeline.

Stage deployments happen automatically when a new release is made. Prod deployments happen on demand by the CloudOPs team.

CHAPTER 6

Reference

Here you'll find the automated code documentation for the ATMO code:

6.1 atmo

These are the submodules of the `atmo` package that don't quite fit "topics", like the *atmo.clusters*, *atmo.jobs* and *atmo.users* packages.

6.1.1 atmo.celery

class atmo.celery.**AtmoCelery**(*main=None*, *loader=None*, *backend=None*, *amqp=None*, *events=None*, *log=None*, *control=None*, *set_as_current=True*, *tasks=None*, *broker=None*, *include=None*, *changes=None*, *config_source=None*, *fixups=None*, *task_cls=None*, *autofinalize=True*, *namespace=None*, *strict_typing=True*, ***kwargs*)

A custom Celery class to implement exponential backoff retries.

backoff(*n*, *cap=3600*)

Return a fully jittered backoff value for the given number.

send_task(**args*, ***kwargs*)

Send task by name.

Supports the same arguments as `@-Task.apply_async()`.

Arguments: name (str): Name of task to call (e.g., *"tasks.add"*). result_cls (AsyncResult): Specify custom result class.

class atmo.celery.**ExpoBackoffFullJitter**(*base*, *cap*)

Implement fully jittered exponential retries.

See for more infos:

- https://www.awsarchitectureblog.com/2015/03/backoff.html

- https://github.com/awslabs/aws-arch-backoff-simulator

backoff(*n*)
Return the exponential backoff value for the given number.

expo(*n*)
Return the exponential value for the given number.

atmo.celery.**celery = <AtmoCelery atmo>**
The Celery app instance used by ATMO, which auto-detects Celery config values from Django settings prefixed with "CELERY_" and autodiscovers Celery tasks from `tasks.py` modules in Django apps.

6.1.2 atmo.context_processors

atmo.context_processors.**alerts**(*request*)
Here be dragons, for who are bold enough to break systems and lose data

This adds an alert to requests in stage and development environments.

atmo.context_processors.**settings**(*request*)
Adds the Django settings object to the template context.

atmo.context_processors.**version**(*request*)
Adds version-related context variables to the context.

6.1.3 atmo.decorators

atmo.decorators.**add_permission_required**(*model*, ***params*)
Checks add object permissions for the given model and parameters.

atmo.decorators.**change_permission_required**(*model*, ***params*)
Checks change object permissions for the given model and parameters.

atmo.decorators.**delete_permission_required**(*model*, ***params*)
Checks delete object permissions for the given model and parameters.

atmo.decorators.**modified_date**(*view_func*)
A decorator that when applied to a view using a TemplateResponse will look for a context variable (by default "modified_date") to set the header (by default "X-ATMO-Modified-Date") with the ISO formatted value.

This is useful to check for modification on the client side. The end result will be a header like this:

```
X-ATMO-Modified-Date: 2017-03-14T10:48:53+00:00
```

atmo.decorators.**permission_required**(*perm*, *klass*, ***params*)
A decorator that will raise a 404 if an object with the given view parameters isn't found or if the request user does not have the given permission for the object.

E.g. for checking if the request user is allowed to change a user with the given username:

```
@permission_required('auth.change_user', User)
def change_user(request, username):
    # can use get() directly since get_object_or_404 was already called
    # in the decorator and would have raised a Http404 if not found
    user = User.objects.get(username=username)
    return render(request, 'change_user.html', context={'user': user})
```

atmo.decorators.**view_permission_required**(*model*, ***params*)
Checks view object permissions for the given model and parameters.

6.1.4 atmo.models

class atmo.models.**CreatedByModel**(**args*, ***kwargs*)
An abstract data model that has a relation to the Django user model as configured by the AUTH_USER_MODEL setting. The reverse related name is created_<name of class>s, e.g. user.created_clusters.all() where user is a User instance that has created various Cluster objects before.

Parameters **created_by_id** (ForeignKey to User) – User that created the instance.

assign_permission(*user*, *perm*)
Assign permission to the given user, e.g. 'clusters.view_cluster',

save(**args*, ***kwargs*)
Saves the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

class atmo.models.**EditedAtModel**(**args*, ***kwargs*)
An abstract data model used by various other data models throughout ATMO that store timestamps for the creation and modification.

Parameters

- **created_at** (DateTimeField) – Created at
- **modified_at** (DateTimeField) – Modified at

save(**args*, ***kwargs*)
Saves the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

class atmo.models.**PermissionMigrator**(*apps*, *model*, *perm*, *user_field=None*, *group=None*)
A custom django-guardian permission migration to be used when new model classes are added and users or groups require object permissions retroactively.

assign()
The primary method to assign a permission to the user or group.

remove()
The primary method to remove a permission to the user or group.

class atmo.models.**URLActionModel**(**args*, ***kwargs*)
A model base class to be used with URL patterns that define actions for models, e.g. /foo/bar/1/edit, /foo/bar/1/delete etc.

url_actions = []
The list of actions to be used to reverse the URL patterns with

url_delimiter = '-'
The delimiter to be used for the URL pattern names.

url_field_name = 'id'
The field name to be used with the keyword argument in the URL pattern.

url_kwarg_name = 'id'
The keyword argument name to be used in the URL pattern.

url_prefix = None
The prefix to be used for the URL pattern names.

atmo.models.**next_field_value**(*model_cls*, *field_name*, *field_value*, *start=2*, *separator='-'*, *max_length=0*, *queryset=None*)
For the given model class, field name and field value provide a "next" value, which basically means a counter appended to the value.

6.1.5 atmo.names

atmo.names.**adjectives = ['admiring', 'adoring', 'affectionate', 'agitated', 'amazing', 'ang**
The adjectives to be used to generate random names.

atmo.names.**random_scientist**(*separator=None*)
Generate a random scientist name using the given separator and a random 4-digit number, similar to Heroku's random project names.

atmo.names.**scientists = ['albattani', 'allen', 'almeida', 'agnesi', 'archimedes', 'ardinghe**
The scientists to be used to generate random names.

6.1.6 atmo.provisioners

class atmo.provisioners.**Provisioner**
A base provisioner to be used by specific cases of calling out to AWS EMR. This is currently storing some common code and simplifies testing.

Subclasses need to override there class attributes:

- *log_dir*
- *name_component*

job_flow_params(*user_username*, *user_email*, *identifier*, *emr_release*, *size*)
Given the parameters returns the basic parameters for EMR job flows, and handles for example the decision whether to use spot instances or not.

log_dir = None
The name of the log directory, e.g. 'jobs'.

name_component = None
The name to be used in the identifier, e.g. 'job'.

spark_emr_configuration()
Fetch the Spark EMR configuration data to be passed as the Configurations parameter to EMR API endpoints.

We store this in S3 to be able to share it between various Telemetry services.

6.1.7 atmo.tasks

Task: atmo.tasks.**cleanup_permissions**
A Celery task that cleans up old django-guardian object permissions.

6.1.8 atmo.templatetags

atmo.templatetags.**full_url**(*url*)
A Django template filter to prepend the given URL path with the full site URL.

atmo.templatetags.**markdown**(*content*)
A Django template filter to render the given content as Markdown.

atmo.templatetags.**url_update**(*url*, ***kwargs*)
A Django template tag to update the query parameters for the given URL.

6.1.9 atmo.views

class atmo.views.**DashboardView**(***kwargs*)
The dashboard view that allows filtering clusters and jobs shown.

active_cluster_filter = 'active'
Active filter for clusters

clusters_filters = ['active', 'terminated', 'failed', 'all']
Allowed filters for clusters

default_cluster_filter = 'active'
Default cluster filter

http_method_names = ['get', 'head']
No need to accept POST or DELETE requests

maintainer_group_name = 'Spark job maintainers'
Name of auth group that is checked to display Spark jobs

template_name = 'atmo/dashboard.html'
Template name

atmo.views.**permission_denied**(*request*, *exception*, *template_name='403.html'*)
Permission denied (403) handler.

Template `403.html`

If the template does not exist, an Http403 response containing the text "403 Forbidden" (as per RFC 7231) will be returned.

atmo.views.**server_error**(*request*, *template_name='500.html'*)
500 error handler.

Template `500.html`

6.2 atmo.clusters

The code base to manage AWS EMR clusters.

6.2.1 atmo.clusters.forms

class atmo.clusters.forms.**EMRReleaseChoiceField**(**args*, ***kwargs*)
A `ModelChoiceField` subclass that uses *`EMRRelease`* objects for the choices and automatically uses a "radioset" rendering – a horizontal button group for easier selection.

label_from_instance(*obj*)
Append the status of the EMR release if it's experimental or deprecated.

class atmo.clusters.forms.**ExtendClusterForm**(**args*, ***kwargs*)

class atmo.clusters.forms.**NewClusterForm**(**args*, ***kwargs*)
A form used for creating new clusters.

Parameters

- **identifier** (RegexField) – A unique identifier for your cluster, visible in the AWS management console. (Lowercase, use hyphens instead of spaces.)
- **size** (IntegerField) – Number of workers to use in the cluster, between 1 and 30. For testing or development 1 is recommended.
- **lifetime** (IntegerField) – Lifetime in hours after which the cluster is automatically terminated, between 2 and 24.
- **ssh_key** (ModelChoiceField) – Ssh key
- **emr_release** (*EMRReleaseChoiceField*) – Different AWS EMR versions have different versions of software like Hadoop, Spark, etc. See what's new in each.

6.2.2 atmo.clusters.models

class atmo.clusters.models.**Cluster**(*id*, *created_at*, *modified_at*, *created_by*, *emr_release*, *identifier*, *size*, *lifetime*, *lifetime_extension_count*, *ssh_key*, *expires_at*, *started_at*, *ready_at*, *finished_at*, *jobflow_id*, *most_recent_status*, *master_address*, *expiration_mail_sent*)

Parameters

- **id** (AutoField) – Id
- **created_at** (DateTimeField) – Created at
- **modified_at** (DateTimeField) – Modified at
- **created_by_id** (ForeignKey to User) – User that created the instance.
- **emr_release_id** (ForeignKey to *EMRRelease*) – Different AWS EMR versions have different versions of software like Hadoop, Spark, etc. See what's new in each.
- **identifier** (CharField) – Cluster name, used to non-uniqely identify individual clusters.
- **size** (IntegerField) – Number of computers used in the cluster.
- **lifetime** (PositiveSmallIntegerField) – Lifetime of the cluster after which it's automatically terminated, in hours.
- **lifetime_extension_count** (PositiveSmallIntegerField) – Number of lifetime extensions.
- **ssh_key_id** (ForeignKey to *SSHKey*) – SSH key to use when launching the cluster.
- **expires_at** (DateTimeField) – Date/time that the cluster will expire and automatically be deleted.
- **started_at** (DateTimeField) – Date/time when the cluster was started on AWS EMR.
- **ready_at** (DateTimeField) – Date/time when the cluster was ready to run steps on AWS EMR.
- **finished_at** (DateTimeField) – Date/time when the cluster was terminated or failed on AWS EMR.

- **jobflow_id** (CharField) – AWS cluster/jobflow ID for the cluster, used for cluster management.
- **most_recent_status** (CharField) – Most recently retrieved AWS status for the cluster.
- **master_address** (CharField) – Public address of the master node.This is only available once the cluster has bootstrapped
- **expiration_mail_sent** (BooleanField) – Whether the expiration mail were sent.

exception DoesNotExist

exception MultipleObjectsReturned

deactivate()
Shutdown the cluster and update its status accordingly

extend(*hours*)
Extend the cluster lifetime by the given number of hours.

info
Returns the provisioning information for the cluster.

is_active
Returns whether the cluster is active or not.

is_expiring_soon
Returns whether the cluster is expiring in the next hour.

is_failed
Returns whether the cluster has failed or not.

is_ready
Returns whether the cluster is ready or not.

is_terminated
Returns whether the cluster is terminated or not.

is_terminating
Returns whether the cluster is terminating or not.

save(**args*, ***kwargs*)
Insert the cluster into the database or update it if already present, spawning the cluster if it's not already spawned.

sync(*info=None*)
Should be called to update latest cluster status in *self.most_recent_status*.

class atmo.clusters.models.**EMRRelease**(*created_at*, *modified_at*, *version*, *changelog_url*, *help_text*, *is_active*, *is_experimental*, *is_deprecated*)

Parameters

- **created_at** (DateTimeField) – Created at
- **modified_at** (DateTimeField) – Modified at
- **version** (CharField) – Version
- **changelog_url** (TextField) – The URL of the changelog with details about the release.
- **help_text** (TextField) – Optional help text to show for users when creating a cluster.
- **is_active** (BooleanField) – Whether this version should be shown to the user at all.

- **is_experimental** (BooleanField) – Whether this version should be shown to users as experimental.
- **is_deprecated** (BooleanField) – Whether this version should be shown to users as deprecated.

exception DoesNotExist

exception MultipleObjectsReturned

6.2.3 atmo.clusters.provisioners

class atmo.clusters.provisioners.**ClusterProvisioner**

The cluster specific provisioner.

format_info(*cluster*)

Formats the data returned by the EMR API for internal ATMO use.

format_list(*cluster*)

Formats the data returned by the EMR API for internal ATMO use.

info(*jobflow_id*)

Returns the cluster info for the cluster with the given Jobflow ID with the fields start time, state and public IP address

job_flow_params(**args*, ***kwargs*)

Given the parameters returns the extended parameters for EMR job flows for on-demand cluster.

list(*created_after*, *created_before=None*)

Returns a list of cluster infos in the given time frame with the fields: - Jobflow ID - state - start time

start(*user_username*, *user_email*, *identifier*, *emr_release*, *size*, *public_key*)

Given the parameters spawns a cluster with the desired properties and returns the jobflow ID.

stop(*jobflow_id*)

Stops the cluster with the given JobFlow ID.

6.2.4 atmo.clusters.queries

class atmo.clusters.queries.**ClusterQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

A Django queryset that filters by cluster status.

Used by the *Cluster* model.

active()

The clusters that have an active status.

failed()

The clusters that have an failed status.

terminated()

The clusters that have an terminated status.

class atmo.clusters.queries.**EMRReleaseQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

A Django queryset for the *EMRRelease* model.

active()

deprecated()
The EMR releases that are deprecated.

experimental()
The EMR releases that are considered experimental.

natural_sort_by_version()
Sorts this queryset by the EMR version naturally (human-readable).

stable()
The EMR releases that are considered stable.

6.2.5 atmo.clusters.tasks

Task: atmo.clusters.tasks.**deactivate_clusters**
Deactivate clusters that have been expired.

Task: atmo.clusters.tasks.**send_expiration_mails**
Send expiration emails an hour before the cluster expires.

Task: atmo.clusters.tasks.**update_clusters**
Update the cluster metadata from AWS for the pending clusters.

- To be used periodically.
- Won't update state if not needed.
- Will queue updating the Cluster's public IP address if needed.

Task: atmo.clusters.tasks.**update_master_address**(*cluster_id*, *force=False*)
Update the public IP address for the cluster with the given cluster ID

6.2.6 atmo.clusters.views

atmo.clusters.views.**detail_cluster**(*request*, *id*)
View to show details about an existing cluster.

atmo.clusters.views.**extend_cluster**(*request*, *id*)
View to extend the lifetime an existing cluster.

atmo.clusters.views.**new_cluster**(*request*)
View to create a new cluster.

atmo.clusters.views.**terminate_cluster**(*request*, *id*)
View to terminate an existing cluster.

6.3 atmo.jobs

The code base to manage scheduled Spark job via AWS EMR clusters.

6.3.1 atmo.jobs.forms

class atmo.jobs.forms.**BaseSparkJobForm**(**args*, ***kwargs*)
A base form used for creating new jobs.

Parameters

- **identifier** (RegexField) – A unique identifier for your Spark job, visible in the AWS management console. (Lowercase, use hyphens instead of spaces.)
- **description** (CharField) – A brief description of your Spark job's purpose. This is intended to provide extra context for the data engineering team.
- **result_visibility** (ChoiceField) – Whether notebook results are uploaded to a public or private S3 bucket.
- **size** (IntegerField) – Number of workers to use when running the Spark job (1 is recommended for testing or development).
- **interval_in_hours** (ChoiceField) – Interval at which the Spark job should be run.
- **job_timeout** (IntegerField) – Number of hours that a single run of the job can run for before timing out and being terminated.
- **start_date** (DateTimeField) – Date and time of when the scheduled Spark job should start running.
- **end_date** (DateTimeField) – Date and time of when the scheduled Spark job should stop running - leave this blank if the job should not be disabled.

clean_notebook()
Validate the uploaded notebook file if it ends with the ipynb file extension.

field_order
Copy the defined model form fields and insert the notebook field at the second spot

save(*commit=True*)
Store the notebook file on S3 and save the Spark job details to the datebase.

class atmo.jobs.forms.**EditSparkJobForm**(**args*, ***kwargs*)
A *BaseSparkJobForm* subclass used for editing jobs.

Parameters

- **identifier** (RegexField) – A unique identifier for your Spark job, visible in the AWS management console. (Lowercase, use hyphens instead of spaces.)
- **description** (CharField) – A brief description of your Spark job's purpose. This is intended to provide extra context for the data engineering team.
- **result_visibility** (ChoiceField) – Whether notebook results are uploaded to a public or private S3 bucket.
- **size** (IntegerField) – Number of workers to use when running the Spark job (1 is recommended for testing or development).
- **interval_in_hours** (ChoiceField) – Interval at which the Spark job should be run.
- **job_timeout** (IntegerField) – Number of hours that a single run of the job can run for before timing out and being terminated.
- **start_date** (DateTimeField) – Date and time of when the scheduled Spark job should start running.
- **end_date** (DateTimeField) – Date and time of when the scheduled Spark job should stop running - leave this blank if the job should not be disabled.

class atmo.jobs.forms.**NewSparkJobForm**(**args*, ***kwargs*)
A *BaseSparkJobForm* subclass used for creating new jobs.

Parameters

- **identifier** (RegexField) – A unique identifier for your Spark job, visible in the AWS management console. (Lowercase, use hyphens instead of spaces.)
- **description** (CharField) – A brief description of your Spark job's purpose. This is intended to provide extra context for the data engineering team.
- **result_visibility** (ChoiceField) – Whether notebook results are uploaded to a public or private S3 bucket.
- **size** (IntegerField) – Number of workers to use when running the Spark job (1 is recommended for testing or development).
- **interval_in_hours** (ChoiceField) – Interval at which the Spark job should be run.
- **job_timeout** (IntegerField) – Number of hours that a single run of the job can run for before timing out and being terminated.
- **start_date** (DateTimeField) – Date and time of when the scheduled Spark job should start running.
- **end_date** (DateTimeField) – Date and time of when the scheduled Spark job should stop running - leave this blank if the job should not be disabled.
- **emr_release** (*EMRReleaseChoiceField*) – Different AWS EMR versions have different versions of software like Hadoop, Spark, etc. See what's new in each.

class atmo.jobs.forms.**SparkJobAvailableForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)

A form used in the views that checks for the availability of identifiers.

6.3.2 atmo.jobs.models

class atmo.jobs.models.**SparkJob**(**args*, ***kwargs*)

A data model to store details about a scheduled Spark job, to be run on AWS EMR.

Parameters

- **id** (AutoField) – Id
- **created_at** (DateTimeField) – Created at
- **modified_at** (DateTimeField) – Modified at
- **created_by_id** (ForeignKey to User) – User that created the instance.
- **emr_release_id** (ForeignKey to *EMRRelease*) – Different AWS EMR versions have different versions of software like Hadoop, Spark, etc. See what's new in each.
- **identifier** (CharField) – Job name, used to uniqely identify individual jobs.
- **description** (TextField) – Job description.
- **notebook_s3_key** (CharField) – S3 key of the notebook after uploading it to the Spark code bucket.
- **result_visibility** (CharField) – Whether notebook results are uploaded to a public or private bucket

- **size** (IntegerField) – Number of computers to use to run the job.
- **interval_in_hours** (IntegerField) – Interval at which the job should run, in hours.
- **job_timeout** (IntegerField) – Number of hours before the job times out.
- **start_date** (DateTimeField) – Date/time that the job should start being scheduled to run.
- **end_date** (DateTimeField) – Date/time that the job should stop being scheduled to run, null if no end date.
- **expired_date** (DateTimeField) – Date/time that the job was expired.
- **is_enabled** (BooleanField) – Whether the job should run or not.

exception DoesNotExist

exception MultipleObjectsReturned

has_finished
Whether the job's cluster is terminated or failed

has_never_run
Whether the job has run before. Looks at both the cluster status and our own record when we asked it to run.

has_timed_out
Whether the current job run has been running longer than the job's timeout allows.

is_due
Whether the start date is in the past and the end date is in the future.

is_runnable
Either the job has never run before or was never finished.

This is checked right before the actual provisioning.

run()
Actually run the scheduled Spark job.

save(**args*, ***kwargs*)
Saves the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

should_run
Whether the scheduled Spark job should run.

terminate()
Stop the currently running scheduled Spark job.

class atmo.jobs.models.**SparkJobRun**(**args*, ***kwargs*)
A data model to store information about every individual run of a scheduled Spark job.

This denormalizes some values from its related data model *SparkJob*.

Parameters

- **id** (AutoField) – Id
- **created_at** (DateTimeField) – Created at
- **modified_at** (DateTimeField) – Modified at

- **spark_job_id** (ForeignKey to *SparkJob*) – Spark job
- **jobflow_id** (CharField) – Jobflow id
- **emr_release_version** (CharField) – Emr release version
- **size** (IntegerField) – Number of computers used to run the job.
- **status** (CharField) – Status
- **scheduled_at** (DateTimeField) – Date/time that the job was scheduled.
- **started_at** (DateTimeField) – Date/time when the cluster was started on AWS EMR.
- **ready_at** (DateTimeField) – Date/time when the cluster was ready to run steps on AWS EMR.
- **finished_at** (DateTimeField) – Date/time that the job was terminated or failed.

exception DoesNotExist

exception MultipleObjectsReturned

sync(*info=None*)
Updates latest status and life cycle datetimes.

class atmo.jobs.models.**SparkJobRunAlert**(**args*, ***kwargs*)
A data model to store job run alerts for later processing by an async job that sends out emails.

Parameters

- **id** (AutoField) – Id
- **created_at** (DateTimeField) – Created at
- **modified_at** (DateTimeField) – Modified at
- **run_id** (ForeignKey to *SparkJobRun*) – Run
- **reason_code** (CharField) – The reason code for the creation of the alert.
- **reason_message** (TextField) – The reason message for the creation of the alert.
- **mail_sent_date** (DateTimeField) – The datetime the alert email was sent.

exception DoesNotExist

exception MultipleObjectsReturned

6.3.3 atmo.jobs.provisioners

class atmo.jobs.provisioners.**SparkJobProvisioner**
The Spark job specific provisioner.

add(*identifier*, *notebook_file*)
Upload the notebook file to S3

get(*key*)
Get the S3 file with the given key from the code S3 bucket.

remove(*key*)
Remove the S3 file with the given key from the code S3 bucket.

results(*identifier*, *is_public*)
Return the results created by the job with the given identifier that were uploaded to S3.

Parameters

- **identifier** – Unique identifier of the Spark job.
- **is_public** – Whether the Spark job is public or not.

Returns A mapping of result prefixes to lists of results.

Return type dict

run(*user_username*, *user_email*, *identifier*, *emr_release*, *size*, *notebook_key*, *is_public*, *job_timeout*)
Run the Spark job with the given parameters

Parameters

- **user_username** – The username of the Spark job owner.
- **user_email** – The email address of the Spark job owner.
- **identifier** – The unique identifier of the Spark job.
- **emr_release** – The EMR release version.
- **size** – The size of the cluster.
- **notebook_key** – The name of the notebook file on S3.
- **is_public** – Whether the job result should be public or not.
- **job_timeout** – The maximum runtime of the job.

Returns AWS EMR jobflow ID

Return type str

6.3.4 atmo.jobs.queries

class atmo.jobs.queries.**SparkJobQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

active()
The Spark jobs that have an active cluster status.

failed()
The Spark jobs that have a failed cluster status.

lapsed()
The Spark jobs that have passed their end dates but haven't been expired yet.

terminated()
The Spark jobs that have a terminated cluster status.

with_runs()
The Spark jobs with runs.

class atmo.jobs.queries.**SparkJobRunQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)

active()
The Spark jobs that have an active cluster status.

6.3.5 atmo.jobs.tasks

class atmo.jobs.tasks.**SparkJobRunTask**
A Celery task base classes to be used by the `run_job()` task to simplify testing.

check_enabled(*spark_job*)
Checks if the job should be run at all

get_spark_job(*pk*)
Load the Spark job with the given primary key.

max_retries = 9
The max number of retries which does not run too long when using the exponential backoff timeouts.

provision_run(*spark_job*, *first_run=False*)
Actually run the given Spark job.

If this is the first run we'll update the "last_run_at" value to the start date of the spark_job so Celery beat knows what's going on.

sync_run(*spark_job*)
Updates the cluster status of the latest Spark job run, if available.

terminate_and_notify(*spark_job*)
When the Spark job has timed out because it has run longer than the maximum runtime we will terminate it (and its cluster) and notify the owner to optimize the Spark job code.

unschedule_and_expire(*spark_job*)
Remove the Spark job from the periodic schedule and send an email to the owner that it was expired.

6.3.6 atmo.jobs.views

atmo.jobs.views.**check_identifier_available**(*request*)
Given a Spark job identifier checks if one already exists.

atmo.jobs.views.**delete_spark_job**(*request*, *id*)
View to delete a scheduled Spark job and then redirects to the dashboard.

atmo.jobs.views.**detail_spark_job**(*request*, *id*)
View to show the details for the scheduled Spark job with the given ID.

atmo.jobs.views.**detail_zeppelin_job**(*request*, *id*)
View to show the details for the scheduled Zeppelin job with the given ID.

atmo.jobs.views.**download_spark_job**(*request*, *id*)
Download the notebook file for the scheduled Spark job with the given ID.

atmo.jobs.views.**edit_spark_job**(*request*, *id*)
View to edit a scheduled Spark job that runs on AWS EMR.

atmo.jobs.views.**new_spark_job**(*request*)
View to schedule a new Spark job to run on AWS EMR.

atmo.jobs.views.**run_spark_job**(*request*, *id*)
Run a scheduled Spark job right now, out of sync with its actual schedule.

This will actively ask for confirmation to run the Spark job.

6.4 atmo.keys

The code base to manage public SSH keys to be used with *ATMO clusters*.

6.4.1 atmo.keys.forms

class atmo.keys.forms.**SSHKeyForm**(**args*, ***kwargs*)

The form to be used when uploaded new SSH keys.

Parameters

- **title** (CharField) – Name to give to this public key
- **key** (CharField) – Should start with one of the following prefixes: ssh-rsa, ssh-dss, ecdsa-sha2-nistp256, ecdsa-sha2-nistp384, ecdsa-sha2-nistp521
- **key_file** (FileField) – This can usually be found in ~/.ssh/ on your computer.

clean_key()

Checks if the submitted key data:

- isn't larger than 100kb
- is a valid SSH public key (e.g. dismissing if it's a private key)
- does not match any of the `valid key data prefixes`
- already exists in the database

6.4.2 atmo.keys.models

class atmo.keys.models.**SSHKey**(**args*, ***kwargs*)

A Django data model to store public SSH keys for logged-in users to be used in the *on-demand clusters*.

Parameters

- **id** (AutoField) – Id
- **created_at** (DateTimeField) – Created at
- **modified_at** (DateTimeField) – Modified at
- **created_by_id** (ForeignKey to User) – User that created the instance.
- **title** (CharField) – Name to give to this public key
- **key** (TextField) – Should start with one of the following prefixes: ssh-rsa, ssh-dss, ecdsa-sha2-nistp256, ecdsa-sha2-nistp384, ecdsa-sha2-nistp521
- **fingerprint** (CharField) – Fingerprint

exception DoesNotExist

exception MultipleObjectsReturned

VALID_PREFIXES = ['ssh-rsa', 'ssh-dss', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384',

The list of valid SSH key data prefixes, will be validated on save.

prefix

The prefix of the key data, one of the *VALID_PREFIXES*.

save(**args*, ***kwargs*)
Saves the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

SSHKey.**VALID_PREFIXES = ['ssh-rsa', 'ssh-dss', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384**
The list of valid SSH key data prefixes, will be validated on save.

6.4.3 atmo.keys.utils

atmo.keys.utils.**calculate_fingerprint**(*data*)
Calculate the hexadecimal fingerprint for the given key data.

Parameters **data** – str - The key data to calculate the fingerprint for.

Returns The fingerprint.

Return type str

6.4.4 atmo.keys.views

atmo.keys.views.**delete_key**(*request*, *id*)
View to delete an SSH key with the given ID.

atmo.keys.views.**detail_key**(*request*, *id*, *raw=False*)
View to show the details for the SSH key with the given ID.

If the optional `raw` parameter is set it'll return the raw key data.

atmo.keys.views.**list_keys**(*request*)
View to list all SSH keys for the logged-in user.

atmo.keys.views.**new_key**(*request*)
View to upload a new SSH key for the logged-in user.

6.5 atmo.news

The code base to show the "News" section to users.

6.5.1 atmo.news.views

class atmo.news.views.**News**
Encapsulate the rendering of the news document `NEWS.md`.

ast
Return (and cache for repeated querying) the Markdown AST of the `NEWS.md` file.

current(*request*)
Return the latest seen version or nothing.

latest
Return the latest version found in the `NEWS.md` file.

render()
Render the `NEWS.md` file as a HTML.

update(*request*, *response*)
Set the cookie for the given request with the latest seen version.

uptodate(*request*)
Return whether the current is newer than the last seen version.

atmo.news.views.**check_news**(*request*)
View to check if the current user has seen the latest "News" section and return either *'ok'* or *'meh'* as a string.

atmo.news.views.**list_news**(*request*)
View to list all news and optionally render only part of the template for AJAX requests.

6.6 atmo.settings

Django settings for atmo project.

For more information on this file, see https://docs.djangoproject.com/en/1.9/topics/settings/

For the full list of settings and their values, see https://docs.djangoproject.com/en/1.9/ref/settings/

class atmo.settings.**AWS**
AWS settings

AWS_CONFIG = {'ACCOUNTING_APP_TAG': 'telemetry-analysis', 'ACCOUNTING_TYPE_TAG': 'work
The AWS config values.

PUBLIC_DATA_URL = 'https://s3-us-west-2.amazonaws.com/telemetry-public-analysis-2/'
The URL of the S3 bucket with public job results.

PUBLIC_NB_URL = 'https://nbviewer.jupyter.org/url/s3-us-west-2.amazonaws.com/telemetry
The URL to show public Jupyter job results with.

class atmo.settings.**Base**
Configuration that may change per-environment, some with defaults.

LOGGING()
dict() -> new empty dictionary dict(mapping) -> new dictionary initialized from a mapping object's

(key, value) pairs

dict(iterable) -> new dictionary initialized as if via: d = {} for k, v in iterable:

d[k] = v

dict(kwargs) -> new dictionary initialized with the name=value pairs** in the keyword argument list.
For example: dict(one=1, two=2)

SITE_URL = 'http://localhost:8000'
The URL under which this instance is running

class atmo.settings.**Build**
Configuration to be used in build (!) environment

CONSTANCE_CONFIG
Dictionary that remembers insertion order

DATABASES
dict() -> new empty dictionary dict(mapping) -> new dictionary initialized from a mapping object's

(key, value) pairs

dict(iterable) -> new dictionary initialized as if via: d = {} for k, v in iterable:

d[k] = v

dict(kwargs) -> new dictionary initialized with the name=value pairs** in the keyword argument list. For example: dict(one=1, two=2)

LOGGING()

dict() -> new empty dictionary dict(mapping) -> new dictionary initialized from a mapping object's

(key, value) pairs

dict(iterable) -> new dictionary initialized as if via: d = {} for k, v in iterable:

d[k] = v

dict(kwargs) -> new dictionary initialized with the name=value pairs** in the keyword argument list. For example: dict(one=1, two=2)

class atmo.settings.**CSP**

CSP settings

class atmo.settings.**Celery**

The Celery specific Django settings.

CELERY_BEAT_MAX_LOOP_INTERVAL = 5

redbeat likes fast loops

CELERY_BEAT_SCHEDULE = {'clean_orphan_obj_perms': {'schedule': <crontab: 30 3 * * *

The default/initial schedule to use.

CELERY_BEAT_SCHEDULER = 'redbeat.RedBeatScheduler'

The scheduler to use for periodic and scheduled tasks.

CELERY_BROKER_TRANSPORT_OPTIONS = {'fanout_patterns': True, 'fanout_prefix': True, '

The Celery broker transport options

CELERY_REDBEAT_LOCK_TIMEOUT = 25

Unless refreshed the lock will expire after this time

CELERY_RESULT_BACKEND = 'django-db'

Use the django_celery_results database backend.

CELERY_RESULT_EXPIRES = datetime.timedelta(14)

Throw away task results after two weeks, for debugging purposes.

CELERY_TASK_SEND_SENT_EVENT = True

Send SENT events as well to know when the task has left the scheduler.

CELERY_TASK_SOFT_TIME_LIMIT = 300

Add a 5 minute soft timeout to all Celery tasks.

CELERY_TASK_TIME_LIMIT = 600

And a 10 minute hard timeout.

CELERY_TASK_TRACK_STARTED = True

Track if a task has been started, not only pending etc.

CELERY_WORKER_DISABLE_RATE_LIMITS = True

Completely disable the rate limiting feature since it's costly

CELERY_WORKER_HIJACK_ROOT_LOGGER = False

Stop hijacking the root logger so Sentry works.

class atmo.settings.**Constance**
Constance settings

CONSTANCE_ADDITIONAL_FIELDS = {'announcement_styles': [<class 'django.forms.fields.Ch
Adds custom widget for announcements.

CONSTANCE_CONFIG = {'ANNOUNCEMENT_CONTENT': ('', 'The announcement content.'), 'ANNOUN
The default config values.

CONSTANCE_CONFIG_FIELDSETS = {'AWS': ('AWS_USE_SPOT_INSTANCES', 'AWS_SPOT_BID_CORE', '
Some fieldsets for the config values.

CONSTANCE_REDIS_CONNECTION_CLASS = 'django_redis.get_redis_connection'
Using the django-redis connection function for the backend.

class atmo.settings.**Core**
Configuration that will never change per-environment.

BASE_DIR = '/home/docs/checkouts/readthedocs.org/user_builds/atmo/envs/latest/lib/pyth
Build paths inside the project like this: os.path.join(BASE_DIR, ...)

INSTALLED_APPS = ['atmo.apps.AtmoAppConfig', 'atmo.clusters', 'atmo.jobs', 'atmo.apps.
The installed apps.

SITE_ID = 1
Using the default first site found by django.contrib.sites

THIS_DIR = '/home/docs/checkouts/readthedocs.org/user_builds/atmo/envs/latest/lib/pyth
The directory in which the settings file reside.

VERSION = None
The current ATMO version.

class atmo.settings.**Dev**
Configuration to be used during development and base class for testing

LOGGING()
dict() -> new empty dictionary dict(mapping) -> new dictionary initialized from a mapping object's

(key, value) pairs

dict(iterable) -> new dictionary initialized as if via: d = {} for k, v in iterable:

d[k] = v

dict(kwargs) -> new dictionary initialized with the name=value pairs** in the keyword argument list.
For example: dict(one=1, two=2)

class atmo.settings.**Docs**
Configuration to be used in the documentation environment

LOGGING()
dict() -> new empty dictionary dict(mapping) -> new dictionary initialized from a mapping object's

(key, value) pairs

dict(iterable) -> new dictionary initialized as if via: d = {} for k, v in iterable:

d[k] = v

dict(kwargs) -> new dictionary initialized with the name=value pairs** in the keyword argument list.
For example: dict(one=1, two=2)

class atmo.settings.**Prod**
Configuration to be used in prod environment

CONSTANCE_CONFIG
Dictionary that remembers insertion order

DATABASES
dict() -> new empty dictionary dict(mapping) -> new dictionary initialized from a mapping object's

(key, value) pairs

dict(iterable) -> new dictionary initialized as if via: d = {} for k, v in iterable:

d[k] = v

dict(kwargs) -> new dictionary initialized with the name=value pairs** in the keyword argument list.
For example: dict(one=1, two=2)

LOGGING()
dict() -> new empty dictionary dict(mapping) -> new dictionary initialized from a mapping object's

(key, value) pairs

dict(iterable) -> new dictionary initialized as if via: d = {} for k, v in iterable:

d[k] = v

dict(kwargs) -> new dictionary initialized with the name=value pairs** in the keyword argument list.
For example: dict(one=1, two=2)

class atmo.settings.**Stage**
Configuration to be used in stage environment

DATABASES
dict() -> new empty dictionary dict(mapping) -> new dictionary initialized from a mapping object's

(key, value) pairs

dict(iterable) -> new dictionary initialized as if via: d = {} for k, v in iterable:

d[k] = v

dict(kwargs) -> new dictionary initialized with the name=value pairs** in the keyword argument list.
For example: dict(one=1, two=2)

LOGGING()
dict() -> new empty dictionary dict(mapping) -> new dictionary initialized from a mapping object's

(key, value) pairs

dict(iterable) -> new dictionary initialized as if via: d = {} for k, v in iterable:

d[k] = v

dict(kwargs) -> new dictionary initialized with the name=value pairs** in the keyword argument list.
For example: dict(one=1, two=2)

class atmo.settings.**Test**
Configuration to be used during testing

LOGGING()
dict() -> new empty dictionary dict(mapping) -> new dictionary initialized from a mapping object's

(key, value) pairs

dict(iterable) -> new dictionary initialized as if via: d = {} for k, v in iterable:

d[k] = v

dict(kwargs) -> new dictionary initialized with the name=value pairs** in the keyword argument list. For example: dict(one=1, two=2)

6.7 atmo.users

The code base to handle user sign ups and logins.

6.7.1 atmo.users.utils

atmo.users.utils.**generate_username_from_email**(*email*)

Use the unique part of the email as the username for mozilla.com and the full email address for all other users.

CHAPTER 7

Changelog

Welcome to the running release notes of ATMO!

- You can use this document to see high-level changes done in each release that is git tagged.
- Backward-incompatible changes or other notable events that have an impact on users are noted individually.
- The order of this changelog is descending (newest first).
- Dependency updates are only mentioned when they require user attention.

7.1 2018.4.0

date 2018-03-07

Updated link to documentation about custom analysis with Spark.

7.2 2018.3.0

date 2018-03-07

Naturally sort EMR releases by version instead of alphabetically when launching EMR clusters or scheduling Spark jobs.

Stop overwriting the owner on save in the admin when the object already exists.

7.3 2017.11.0

date 2017-01-01

Fixed login and logout issues with new auth mechanism.

7.4 2017.10.2

date 2017-10-31

Switched from Google based authentication to Auth0 based authentication (via their OpenID Connect API).

Removed leftovers from old Heroku deploy method.

7.5 2017.10.1

date 2017-10-23

Fix an issue when recording the Spark job run time metric.

7.6 2017.10.0

date 2017-10-05

Add ability to upload Zeppelin notebooks.

Remove name generator for scheduled Spark jobs to reduce confusion.

Record Spark job metrics.

Fix recording metrics in database transactions.

7.7 2017.8.1

date 2017-08-17

Fix metric duplicates.

7.8 2017.8.0

date 2017-08-16

Add more cluster metrics.

7.9 2017.7.2

date 2017-07-25

Add metrics for EMR version and cluster extensions.

7.10 2017.7.1

date 2017-07-18

Make EMR profile configurable via environment variable.

7.11 2017.7.0

date 2017-07-12

Allow EMR bootstrap bucket to be configurable for improved environment specific setup.

Add description of schedule Spark job to alert email body.

Add documentation under https://atmo.readthedocs.io/

7.12 2017.6.1

date 2017-06-20

Filter out inactive EMR releases from dropdown and minor UI tweaks.

7.13 2017.6.0

date 2017-06-06

Add Zeppelin examples to cluster detail.

7.14 2017.5.7

date 2017-05-30

Fix regression introduced when the backoff feature for task retries was improved in 2017.5.5.

7.15 2017.5.[5,6]

date 2017-05-24

Fix more race conditions in sending out emails.

Fix duplicate job runs due to job scheduling race conditions.

Store and show datetimes from EMR status updates for better monitoring.

Add job history details to job detail page.

Improved backoff patterns by inlining the Celery task retries.

7.16 2017.5.[3,4]

date 2017-05-18

Fix issue with Celery monitoring.

7.17 2017.5.2

date 2017-05-17

Fix race conditions in email sending.

Add ability to run job right now.

UI fixes to the cluster and Spark job detail pages.

Upgrade to Django 1.11 and Python 3.6.

Add a responsive admin theme.

Add ability to show a site-wide announcement on top of every page.

Update the status of all past Spark job runs not only the last one.

Better unique cluster identifiers based on scientist names.

7.18 2017.5.1

date 2017-05-11

Add status and visual indicators to scheduled Spark jobs listings.

Fix issue with running scheduled Celery tasks multiple times.

7.19 2017.5.0

date 2017-05-03

Use user part of email addresses as username (e.g. "jdoe" in "jdoe@mozilla.com) instead of first name.

Add Celery monitoring to Django admin.

7.20 2017.4.3

date 2017-04-27

UX updates to job detail page.

Minor fixes for Celery schedule refactoring.

7.21 2017.4.2

date 2017-04-26

Updated Celery timeout.

Populate new Celery schedules for all scheduled Spark jobs.

7.22 2017.4.1

date 2017-04-25

Add a Celery task for running a Spark job.

This task is used of Redbeat to schedule the Spark jobs using the Celery beat. We add/remove Spark jobs from the schedule on save/delete and can restore the schedule from the database again.

Send emails for Spark jobs when expired and when they have timed out and need to be modified.

Refactored and extended tests.

7.23 2017.4.0

date 2017-04-04

Moved EMR releases into own data model for easy maintenance (including deprecation and experimental tags).

Add ability to define a lifetime on cluster start.

Change default lifetime to 8 hours (~a work day), maximum stays at 24 hours.

Add ability to extend the lifetime of clusters on demand. The cluster expiration email will notify cluster owners about that ability, too.

7.24 2017.3.[6,7]

date 2017-03-28/2017-03-29

Show all scheduled Spark jobs for admin users in the Spark job maintainers group.

Fix logging for Celery and RedBeat.

7.25 2017.3.5

date 2017-03-22

Switch to Celery as task queue to improve stability and processing guarentees.

Wrap more tasks in Django database transactions to reduce risk of race conditions.

Only updates the cluster master address if the cluster isn't ready.

Pins Node dependencies and use Greenkeeper for dependency CI.

7.26 2017.3.4

date 2017-03-20

Fixing an inconsistency with how the run alert status message is stored with values from Amazon, extending the length of the column.

Check and run jobs only every 5 minutes instead of every minute to reduce API access numbers.

7.27 2017.3.3

date 2017-03-17

Regression fixes to the email alerting feature introduced in 2017.3.2 that prevented scheduled jobs to run successfully.

7.28 2017.3.2

date 2017-03-15

BACKWARD INCOMPATIBLE: Removes EMR release 4.5.0.

BACKWARD INCOMPATIBLE: Make clusters persist the home directory between runs.

Adds a changelog (this file) and a "What's new?" section (in the footer).

Adds email alerting if a scheduled Spark job fails.

Replaced automatic page refresher with in-page-alerts when page changes on server.

Moved project board to Waffle: https://waffle.io/mozilla/telemetry-analysis-service

Run flake8 automatically as part of test suite.

7.29 2017.3.[0,1]

date 2017-03-07/2017-03-08

Selects the SSH key automatically if only one is present.

Uses ListCluster API endpoint for updating Spark job run states instead of DescribeCluster to counteract AWS API throtteling.

7.30 2017.2.[9,10,11,12,13]

date 2017-02-23

Regression fixes for the Python 3 migration and Zeppeling integration.

7.31 2017.2.[6,7,8]

date 2017-02-20/2017-02-21

Adds the ability to store the history of scheduled Spark job for planned features such as alerting and cost calculations.

7.32 2017.2.[4,5]

date 2017-02-17

Adds experimental support for Apache Zeppelin, next to Jupyter a second way to manage notebooks.

Improves client side form validation dramaticlly and changes file selector to better suited system.

Adds exponential backoff retries for the worker system to counteract AWS API throtteling for jobs that update cluster status or run scheduled Spark jobs.

Moves from Python 2 to 3.

7.33 2017.2.[1,2,3]

date 2017-02-07/2017-02-10

Uses AWS EC2 spot instances for scheduled Spark jobs with more than one node.

Moves issue management from Bugzilla to GitHub.

7.34 2017.1.[11,12]

date 2017-01-31

Self-dogfoods the newly implemented python-dockerflow.

Fix many UX issues in the various forms.

7.35 2017.1.[7,8,9,10]

date 2017-01-24

Adds ability to upload personal SSH keys to simplify starting clusters.

Adds a new required description field to Spark job to be able to debug jobs easily.

Adds EMR 5.2.1 to list of available EMR versions.

Uses new shared public SSH key that is used by the hadoop user on EMR.

7.36 2017.1.[0,1,2,3,4,5,6]

date 2017-01-20

First release of 2017 that comes with a lot of changes around deployment, UI and UX. o/

Adopts NPM as a way to maintain frontend dependencies.

Adds a object level permission system to be able to share CRUD permissions per user or user group, e.g. admins can see clusters and Spark jobs of other users now.

Makes the cluster and Spark job deletion confirmation happen in place instead of redirecting to separate page that asks for confirmation.

Extends tests and adds test coverage reporting via Codecov.

Drops Travis-CI in favor of Circle CI.

Allows enabling/disabling AWS EC2 spot instances via the Django admin UI in the Constance section.

7.37 2016.11.5

date 2016-11-21

Fix job creation edge case.

More NewRelic fixes.

7.38 2016.11.[2,3,4]

date 2016-11-17

Fixes logging related to Dockerflow.

Turned off NewRelic's "high_security" mode.

Increases the job timeouts for less job kills.

Removes the need for Newrelic deploys to Heroku.

7.39 2016.11.1

date 2016-11-14

Implements Dockerflow health checks so it follows the best practices of Mozilla's Dockerflow. Many thanks to @mythmon for the inspiration in the Normandy code.

7.40 2016.11.0

date 2016-11-11

The first release of ATMO V2 under the new release system that ports the majority of the V1 to a new codebase.

This is a major milestone after months of work of many contributors, finishing the work of Mozilla community members and staff.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index

D

E

F

T

U

V

W