ecsjobs Documentation

Release 1.0.0

Jason Antman

Aug 23, 2021

Contents

- Documentation: http://ecsjobs.readthedocs.io/en/latest/
- Builds: https://app.travis-ci.com/github/jantman/ecsjobs

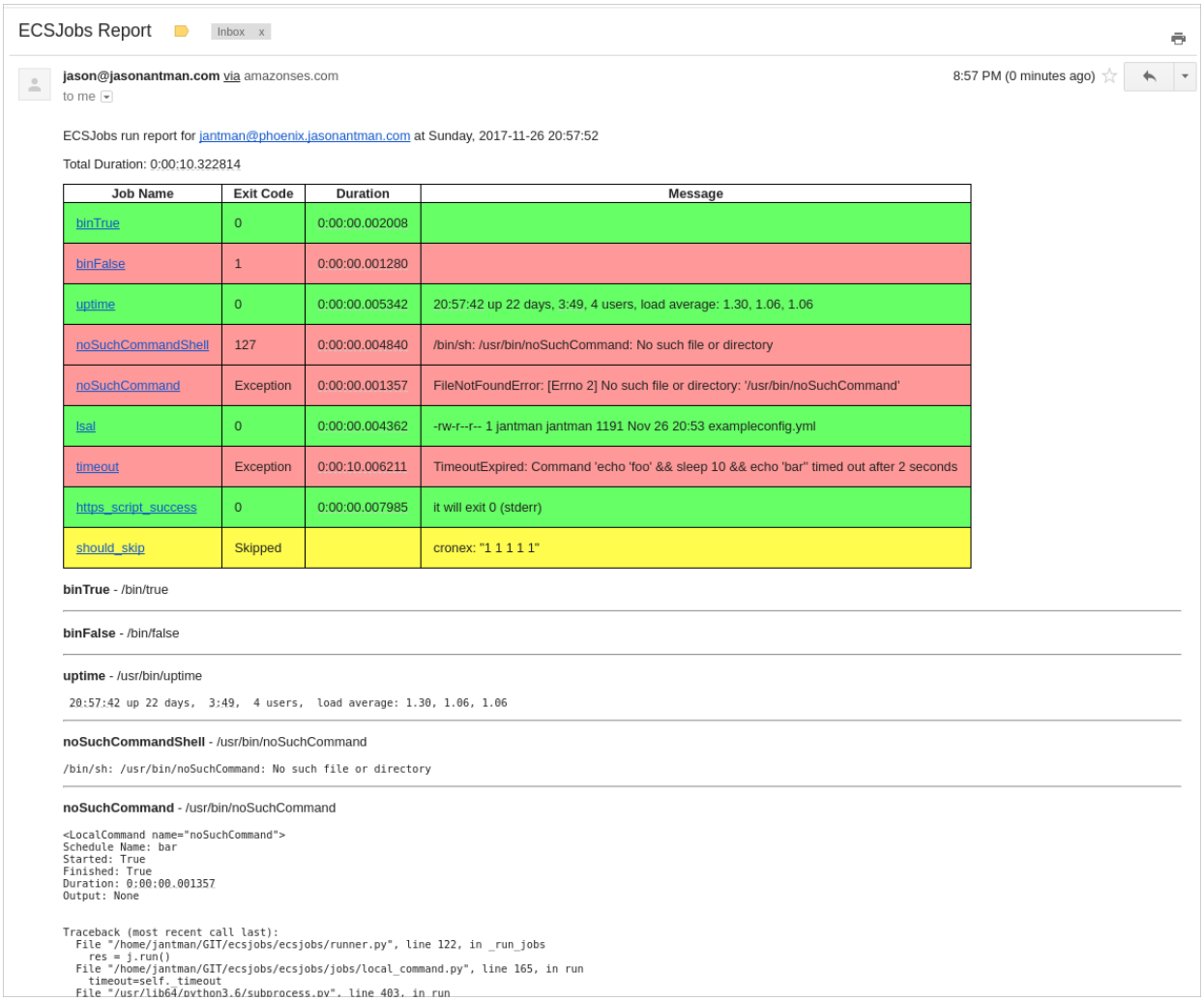
A scheduled job wrapper for ECS, focused on email reporting and adding docker exec and local command abilities.

This is a very, very esoteric project with a really niche use case.

I've migrated my very small personal AWS infrastructure to a single t2.micro ECS instance. I'm also trying to migrate some of my personal stuff from my desktop computer to that instance. I need a way to run scheduled tasks and report on their success or failure, and maybe some output (I have a cron wrapper script that does this on my desktop). But my AWS spend is about $15/month and I don't want to go over that just because of a bunch of CloudWatch alarms. Also, sometimes the scheduled things I want to run are really `docker exec` into existing task containers.

This is a Python project (distributed as an ECS-ready Docker image) that aims to handle running scheduled things and then sending an email report on their success or failure. The main shortcomings this intends to address are the lack of simple built-in failure monitoring for Scheduled ECS Tasks, the lack of a built-in way to execute a command in a running (ECS Service) container, and the lack of useful email reports.

The generated email reports look like (this one for `exampleconfig.yml`):



ECSJobs Report

Inbox x

jason@jasonantman.com via amazonses.com
to me

8:57 PM (0 minutes ago)

ECSJobs run report for jantman@phoenix.jasonantman.com at Sunday, 2017-11-26 20:57:52

Total Duration: 0:00:10.322814

Job Name	Exit Code	Duration	Message
binTrue	0	0:00:00.002008	
binFalse	1	0:00:00.001280	
uptime	0	0:00:00.005342	20:57:42 up 22 days, 3:49, 4 users, load average: 1.30, 1.06, 1.06
noSuchCommandShell	127	0:00:00.004840	/bin/sh: /usr/bin/noSuchCommand: No such file or directory
noSuchCommand	Exception	0:00:00.001357	FileNotFoundError: [Errno 2] No such file or directory: '/usr/bin/noSuchCommand'
lsal	0	0:00:00.004362	-rw-r--r-- 1 jantman jantman 1191 Nov 26 20:53 exampleconfig.yml
timeout	Exception	0:00:10.006211	TimeoutExpired: Command 'echo 'foo' && sleep 10 && echo 'bar'' timed out after 2 seconds
https_script_success	0	0:00:00.007985	it will exit 0 (stderr)
should_skip	Skipped		cronex: "1 1 1 1 1"

binTrue - /bin/true

binFalse - /bin/false

uptime - /usr/bin/uptime

```
20:57:42 up 22 days,  3:49,  4 users,  load average: 1.30, 1.06, 1.06
```

noSuchCommandShell - /usr/bin/noSuchCommand

```
/bin/sh: /usr/bin/noSuchCommand: No such file or directory
```

noSuchCommand - /usr/bin/noSuchCommand

```
<LocalCommand name="noSuchCommand">
Schedule Name: bar
Started: True
Finished: True
Duration: 0:00:00.001357
Output: None


Traceback (most recent call last):
  File "/home/jantman/GIT/ecsjobs/ecsjobs/runner.py", line 122, in _run_jobs
    res = j.run()
  File "/home/jantman/GIT/ecsjobs/ecsjobs/jobs/local_command.py", line 165, in run
    timeout=self._timeout
  File "/usr/lib64/python3.6/subprocess.py", line 403, in run
```

CHAPTER 1

Contents

1.1 Configuration

ecsjobs is configured via YAML files stored in S3 or locally. The paths to these files are specified via environment variables.

1.1.1 S3 Configuration

S3 is used as the source for the configuration files when the `ECSJOBS_BUCKET` and `ECSJOBS_KEY` environment variables are set. The former specifies the name of the S3 bucket that configuration will be retrieved from. The latter specifies the name of a key within the bucket to retrieve the configuration files from. If the key name ends in `.yaml` or `.yml` it will be assumed to be a file, and used as a single configuration file. If it does not, it will be assumed to be a "directory", and all `.yml` or `.yaml` files directly below it will be used.

1.1.2 Local File Configuration

Local file configuration is controlled via the `ECSJOBS_LOCAL_CONF_PATH` environment variable. While it's recommended to use S3 for production use, local file configuration is useful in testing or to validate config files before uploading them to S3. the `ECSJOBS_BUCKET` and `ECSJOBS_KEY` environment variables take precedence over `ECSJOBS_LOCAL_CONF_PATH`. If the path specified by this variable is a directory, all `.yml` and `.yaml` files under it (recursively) will be loaded as configuration. Otherwise, it will be assumed to be a single YAML file.

1.1.3 Single File

If configuring with a single file (`ECSJOBS_KEY` ends in `.yml` or `.yaml` and is a file), the top-level of a file must be a mapping with keys `global` and `jobs`. The value of `global` must be a mapping following the schema described below. The value of `jobs` must be a list of mappings, each following the schema described below.

For single file configurations, Jobs within a Schedule will be executed in the order they appear in the `jobs` array.

1.1.4 Multiple Files

If configuring with multiple files (`ECSJOBS_KEY` does not end in `.yml` or `.yaml` and is used as a prefix/directory), all `.yml` or `.yaml` keys in the bucket beginning with (prefixed by) `ECSJOBS_KEY` will be used for configuration. There must be one file named `global.yml` or `global.yaml` corresponding to the global schema described below. All other `.yml` or `.yaml` files will be treated as job configurations, one job per file, each corresponding to the schema described below.

For multi-file configurations, Jobs within a Schedule will be executed in the lexicographic order of the files each Job is defined in.

1.1.5 Global Schema

The global configuration file or mapping should match the following:

- **from_email** - String, email address to set as FROM.
- **to_email** - List of Strings, email notification recipients.
- **inter_poll_sleep_sec** - *(optional)* how many seconds to sleep between each poll cycle to check the status of asynchronous jobs. Defaults to 10 seconds.
- **max_total_runtime_sec** - *(optional)* Maximum runtime for each ecsjobs invocation, in seconds. If invocation runs longer than this amount, it will die with an error. Default is 3600 seconds (1 hour).
- **email_subject** - *(optional)* a string to use for the email report subject, instead of "ECSJobs Report".
- **failure_html_path** - *(optional)* a string absolute path to write the HTML email report to on disk, if sending via SES fails. If not specified, a temporary file will be used (via Python's `tempfile.mkstemp`) and its path included in the output. If specified, the string `{date}` in this setting will be replaced with the current datetime (at time of config load) in `%Y-%m-%dT%H-%M-%S` format.
- **failure_command** - *(optional)* Array. A command to call if sending via SES fails. This should be an array beginning with the absolute path to the executable, suitable for passing to Python's `subprocess.Popen()`. The content of the HTML report will be passed to the process on STDIN.

1.1.6 Job Schema

Each job configuration file or mapping should match the following:

- **name** - A unique name for the job.
- **class_name** - The name of a *ecsjobs.jobs.base.Job* subclass.
- **schedule** - A string to identify which jobs to run at which times.
- **summary_regex** - A String regular expression to use for extracting a string from the job output for use in the summary table. If there is more than one match, the last one will be used.
- **cron_expression** - A string cron-like expression parsable by cronex specifying when the job should run. This has the effect of causing runs to skip this job unless the expression matches. It's recommended not to use any minute specifiers and not to use any hour specifiers if the total runtime of all jobs is more than an hour.

The rest of the Job keys depend on the class. See the documentation of each Job subclass for the required configuration.

1.1.7 Example Configuration

1.1.7.1 Global

The content of `global.yml` might look like:

Email reports can also be sent to multiple recipients:

1.1.7.2 All Job Classes

All Job classes require the `name`, `schedule` and `class_name` properties:

```
name: jobName
schedule: scheduleName
class_name: SomeJobSubclassName
```

They also support two optional properties, `summary_regex` and `cron_expression`. See the documentation for the *Job* class for more information.

1.1.7.3 Local Commands

Commands can be specified as a string:

```
name: jobName
schedule: scheduleName
class_name: LocalCommand
command: /bin/true
```

Or as an array:

```
name: jobName
schedule: scheduleName
class_name: LocalCommand
command: ['/bin/echo', 'foo']
```

1.2 Running

1.2.1 Locally via Docker

To pull the Docker image

```
docker pull jantman/ecsjobs:latest
```

To run locally via Docker to validate a configuration directory `./conf`:

```
docker run -it --rm \
  -e ECSJOBS_LOCAL_CONF_PATH=/tmp/conf \
  -v $(pwd)/conf:/tmp/conf \
  jantman/ecsjobs:latest \
  validate
```

To run the "foo" schedule locally in a detached/background container (i.e. as a cron job) and allow it to run Docker execs, assuming your Docker socket is at `/var/run/docker.sock`, your configuration directory is at `./conf`, and you want to use AWS credentials from `~/.aws/credentials`:

```
docker run --rm -d \
  -e ECSJOBS_LOCAL_CONF_PATH=/tmp/conf \
  -e DOCKER_HOST=unix:///tmp/docker.sock \
  -v $(pwd)/conf:/tmp/conf \
  -v /var/run/docker.sock:/tmp/docker.sock \
  -v $(readlink -f ~/.aws/credentials):/root/.aws/credentials \
  jantman/ecsjobs:latest \
  run foo
```

Note that when running in this method, the `LocalCommand` class runs commands inside the ecsjobs Docker container, not on the host system.

1.2.2 Locally via pip

To run locally directly on the host OS, i.e. so the `LocalCommand` class will run commands on the host, first setup a virtualenv and install ecsjobs:

```
virtualenv --python=python3.6 .
source bin/activate
pip install ecsjobs
```

To run the "foo" schedule locally using a configuration directory at `./conf`:

```
ECSJOBS_LOCAL_CONF_PATH=$(readlink -f ./conf) ecsjobs run foo
```

1.2.3 In ECS

Note that because of how ECS Scheduled Tasks work, you'll need to create a separate Task Definition for each schedule that you want ecsjobs to run. As an example, if your jobs have two different `schedule` values, "daily" and "weekly", you'd need to create two separate ECS Task Definitions that differ only by the command they run (`run daily` and `run weekly`, respectively).

To run ecsjobs as an ECS Scheduled Task, you'll need to create an ECS Task Definition for the task and an IAM Role to run the task with. You'll also need to create a CloudWatch Event Rule, CloudWatch Event Target, and IAM Role for CloudWatch to trigger the Task, but these are easily done either through the AWS Console or various automation tools.

The IAM Policy that I use on my ecsjobs role is below; it also includes a "AllowSnapshotManagement" statement to allow management of EBS Snapshots, because I do this via a command executed directly in the ecsjobs container.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowDescribeSSMParams",
      "Action": ["ssm:DescribeParameters"],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Sid": "AllowGetSSMParams",
```

(continues on next page)

(continued from previous page)

```
      "Action": ["ssm:GetParameters"],
      "Effect": "Allow",
      "Resource": "arn:aws:ssm:$${aws_region}:$${account_id}:parameter/*"
    },
    {
      "Sid": "AllowS3",
      "Action": ["s3:Get*", "s3:List*", "s3:Head*"],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Sid": "AllowCloudwatch",
      "Action": ["cloudwatch:List*", "cloudwatch:PutMetricData"],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Sid": "AllowECS",
      "Action": ["ecs:RunTask", "ecs:Describe*", "ecs:List*", "ecs:Discover*"],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Sid": "AllowCWLogs",
      "Action": ["logs:FilterLogEvents", "logs:Describe*", "logs:Get*"],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Sid": "AllowSesSend",
      "Action": ["ses:SendEmail"],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Sid": "AllowSnapshotManagement",
      "Action": ["ec2:CreateSnapshot", "ec2:DeleteSnapshot", "ec2:Describe*",
↪"ec2:CreateTags", "ec2:ModifySnapshotAttribute", "ec2:ResetSnapshotAttribute"],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

The container definition that I use in my Task Definition for ecsjobs is as follows:

```
[
  {
    "name": "ecsjobs",
    "image": "jantman/ecsjobs:latest",
    "command": ["run", "${var.schedule}"],
    "cpu": 64,
    "memoryReservation": 64,
    "environment": [
      {"name": "DOCKER_HOST", "value": "unix:///tmp/docker.sock"},
      {"name": "ECSJOBS_BUCKET", "value": "${var.bucket_name}"},
      {"name": "ECSJOBS_KEY", "value": "${var.bucket_key}"},
```

(continues on next page)

(continued from previous page)

```
      {"name": "AWS_REGION", "value": "us-west-2"},
      {"name": "AWS_DEFAULT_REGION", "value": "us-west-2"}
    ],
    "essential": true,
    "mountPoints": [
      {
        "sourceVolume": "dockersock",
        "containerPath": "/tmp/docker.sock"
      }
    ],
    "logConfiguration": {
      "logDriver": "awslogs",
      "options": {
        "awslogs-region": "us-west-2",
        "awslogs-group": "${var.log_group_name}",
        "awslogs-stream-prefix": "${var.cluster_name}"
      }
    }
  }
]
```

This is actually a snippet from a terraform configuration. A few notes about it:

- The "command" in the container definition references a `${var.schedule}` variable that defines the schedule name. I have two task definitions, one for my daily schedule and one for my weekly schedule.
- In order to be able to run Docker Execs on the ECS host, i.e. against another ECS container, we mount `/var/run/docker.sock` from the host into the container at `/tmp/docker.sock`. The `DOCKER_HOST` environment variable must be set to the path of the socket (prefixed with `unix://` to denote that it's a socket).
- The `ECSJOBS_BUCKET` and `ECSJOBS_KEY` environment variables specify the bucket name and key (in that bucket) to retrieve configuration from.
- The `${var.log_group_name}` and `${var.cluster_name}` variables specify settings for the `awslogs` Docker logging driver, to send container logs to CloudWatch Logs.

1.2.4 Suppressing Reports for Successful Runs

If you do not wish to send an email report if all jobs ran successfully, you can pass the `-m` / `--only-email-if-problems` command line argument to ecsjobs.

1.3 ecsjobs

1.3.1 ecsjobs package

1.3.1.1 Subpackages

ecsjobs.jobs package

ecsjobs.jobs.**get_job_classes**()

ecsjobs.jobs.**schema_for_job_class**(*cls*)
Given a *ecsjobs.jobs.base.Job* subclass, return the final JSONSchema for it.

Parameters **cls** (class) – Class to get schema for

Returns final combined JSONSchema for the class

Return type dict

Submodules

ecsjobs.jobs.base module

class ecsjobs.jobs.base.**Job**(*name*, *schedule*, *summary_regex=None*, *cron_expression=None*)

Bases: object

Base class for all Job types/classes.

Parameters

- **name** (*str*) – unique name for this job
- **schedule** (*str*) – the name of the schedule this job runs on
- **summary_regex** (string or None) – A regular expression to use for extracting a string from the job output for use in the summary table. If there is more than one match, the last one will be used.
- **cron_expression** (*str*) – A cron-like expression parsable by cronex specifying when the job should run. This has the effect of causing runs to skip this job unless the expression matches. It's recommended not to use any minute specifiers and not to use any hour specifiers if the total runtime of all jobs is more than an hour.

_schema_dict = {'properties': {'class_name': {'type': 'string'}, 'cron_expression':

Dictionary describing the configuration file schema, to be validated with jsonschema.

duration

Return the duration/runtime of the job, or None if the job did not run.

Returns job duration

Return type datetime.timedelta or None

error_repr

Return a detailed representation of the job state for use in error reporting.

Returns detailed representation of job in case of error

Return type str

exitcode

For Job subclasses that result in a command exit code, return the integer exitcode. For Job subclasses that result in a boolean (success / failure) status, return 0 on success or 1 on failure. Returns -1 if the Job has not completed.

Returns Job exit code or (0 / 1) status

Return type int

is_finished

Return whether or not the Job is finished.

Returns whether or not the Job is finished

Return type bool

is_started
Return whether or not the Job has been started.

Returns whether or not the Job has been started

Return type bool

name
Return the Job Name.

Returns Job name

Return type str

output
Return the output of the Job as a string, or None if the job has not completed.

Returns Job output

Return type str

poll()
For asynchronous jobs (*is_started* is True but *is_finished* is False), check if the job has finished yet. If not, return `False`. If the job has finished, update `self._finish_time`, `self._exit_code`, `self._output` and `self._finished` and then return `True`.

This method should **never** raise exceptions; recoverable exceptions should be handled via internal retry logic on subsequent poll attempts. Retries should be done on the next call of this method; we never want to sleep during this method. Unrecoverable exceptions should set `self._exit_code`, `self._output` and `self._finished`.

Returns *is_finished*

Return type bool

report_description()
Return a one-line description of the Job for use in reports.

Return type str

run()
Run the job.

This method sets `self._started` and `self._start_time`. If the Job runs synchronously, this method also sets `self._finished`, `self._exit_code`, `self._finish_time` and `self._output`.

In the case of an exception, this method must still set those attributes as appropriate and then raise the exception.

Returns True if job finished successfully, False if job finished but failed, or None if the job is still running in the background.

schedule_name
Return the configured schedule name for this job.

Returns schedule name

Return type str

skip
Either None if the job should not be skipped, or a string reason describing why the Job should be skipped.

Return type `None` or `str`

summary()
Retrieve a simple one-line summary of the Job output/status.

Returns Job one-line summary.

Return type str

ecsjobs.jobs.docker_exec module

class ecsjobs.jobs.docker_exec.**DockerExec**(*name*, *schedule*, *summary_regex=None*, *cron_expression=None*, *container_name=None*, *command=None*, *tty=False*, *stdout=True*, *stderr=True*, *privileged=False*, *user='root'*, *environment=None*)
Bases: *ecsjobs.jobs.base.Job*, *ecsjobs.jobs.docker_exec_mixin.DockerExecMixin*

Class to run a command in an existing Docker container via `exec`. Captures combined STDOUT and STDERR to *output* and sets *exitcode* to the exit code of the command/process.

Parameters

- **name** (*str*) – unique name for this job
- **schedule** (*str*) – the name of the schedule this job runs on
- **summary_regex** (`string` or `None`) – A regular expression to use for extracting a string from the job output for use in the summary table. If there is more than one match, the last one will be used.
- **cron_expression** (*str*) – A cron-like expression parsable by cronex specifying when the job should run. This has the effect of causing runs to skip this job unless the expression matches. It's recommended not to use any minute specifiers and not to use any hour specifiers if the total runtime of all jobs is more than an hour.
- **container_name** (*str*) – The name of the Docker container to run the exec in. Required. This can also be a container ID, but that's much less useful in a scheduled job.
- **command** (`str` or `list`) – The command to execute as either a String or a List of Strings, as used by `docker.api.exec_api.ExecApiMixin.exec_create()`.
- **tty** (*bool*) – Whether or not to allocate a TTY when reading output from the command; passed through to `docker.api.exec_api.ExecApiMixin.exec_start()`.
- **stdout** (*bool*) – Whether or not to attach to/capture STDOUT. Passed through to `docker.api.exec_api.ExecApiMixin.exec_create()`.
- **stderr** (*bool*) – Whether or not to attach to/capture STDERR. Passed through to `docker.api.exec_api.ExecApiMixin.exec_create()`.
- **privileged** (*bool*) – Whether or not to run the command as privileged. Passed through to `docker.api.exec_api.ExecApiMixin.exec_create()`.
- **user** (*str*) – The username to run the command as. Default is "root".
- **environment** (`dict` or `list`) – A dictionary or list of string environment variables to set. Passed through to `docker.api.exec_api.ExecApiMixin.exec_create()`.

_schema_dict = {'properties': {'command': {'oneOf': [{'type': 'string'}, {'type':
Dictionary describing the configuration file schema, to be validated with jsonschema.

error_repr
Return a detailed representation of the job state for use in error reporting.

Returns detailed representation of job in case of error

Return type str

report_description()
Return a one-line description of the Job for use in reports.

Return type str

run()
Run the command for the job. Either raise an exception or return True if the command exited 0, False if it exited non-zero.

Returns True if command exited 0, False otherwise.

ecsjobs.jobs.docker_exec_mixin module

class ecsjobs.jobs.docker_exec_mixin.**DockerExecMixin**
Bases: object

Mixin class to be used in other classes for Docker Exec.

_docker_run()
Run `self._command` in `self._container_name`. Set class attributes as appropriate.

ecsjobs.jobs.ecs_docker_exec module

class ecsjobs.jobs.ecs_docker_exec.**EcsDockerExec**(*name*, *schedule*, *summary_regex=None*, *cron_expression=None*, *task_definition_family=None*, *container_name=None*, *command=None*, *tty=False*, *stdout=True*, *stderr=True*, *privileged=False*, *user='root'*, *environment=None*)
Bases: *ecsjobs.jobs.base.Job*, *ecsjobs.jobs.docker_exec_mixin.DockerExecMixin*

Subclass of *DockerExec* that runs the `exec` against a Docker container that is part of an ECS Task, using the ECS Agent Introspection metadata to identify the container to `exec` against.

Note that the functionality of this class depends on the Docker container labels set by the Amazon ECS Container Agent, specifically the `com.amazonaws.ecs.task-definition-family` and `com.amazonaws.ecs.container-name` labels as set in version 1.16.0

Parameters

- **name** (*str*) – unique name for this job
- **schedule** (*str*) – the name of the schedule this job runs on
- **summary_regex** (`string` or `None`) – A regular expression to use for extracting a string from the job output for use in the summary table. If there is more than one match, the last one will be used.

- **cron_expression** (*str*) – A cron-like expression parsable by cronex specifying when the job should run. This has the effect of causing runs to skip this job unless the expression matches. It's recommended not to use any minute specifiers and not to use any hour specifiers if the total runtime of all jobs is more than an hour.
- **task_definition_family** (*str*) – The ECS Task Definition "family" to use to find the container to execute in. **Required.**
- **container_name** (*str*) – The name of the Docker container (within the specified Task Definition Family) to run the exec in. **Required.** If more than one running container is found with a matching family and container name, the first match will be used.
- **command** (`str` or `list`) – The command to execute as either a String or a List of Strings, as used by `docker.api.exec_api.ExecApiMixin.exec_create()`.
- **tty** (*bool*) – Whether or not to allocate a TTY when reading output from the command; passed through to `docker.api.exec_api.ExecApiMixin.exec_start()`.
- **stdout** (*bool*) – Whether or not to attach to/capture STDOUT. Passed through to `docker.api.exec_api.ExecApiMixin.exec_create()`.
- **stderr** (*bool*) – Whether or not to attach to/capture STDERR. Passed through to `docker.api.exec_api.ExecApiMixin.exec_create()`.
- **privileged** (*bool*) – Whether or not to run the command as privileged. Passed through to `docker.api.exec_api.ExecApiMixin.exec_create()`.
- **user** (*str*) – The username to run the command as. Default is "root".
- **environment** (`dict` or `list`) – A dictionary or list of string environment variables to set. Passed through to `docker.api.exec_api.ExecApiMixin.exec_create()`.

_find_container()
Using `self._family` and `self._task_container_name`, find the name of the first currently-running Docker container for that task.

Returns name of first matching running Docker container

Return type str

_schema_dict = {'properties': {'command': {'oneOf': [{'type': 'string'}, {'type':
Dictionary describing the configuration file schema, to be validated with jsonschema.

error_repr
Return a detailed representation of the job state for use in error reporting.

Returns detailed representation of job in case of error

Return type str

report_description()
Return a one-line description of the Job for use in reports.

Return type str

run()
Run the command for the job. Either raise an exception or return True if the command exited 0, False if it exited non-zero.

Returns True if command exited 0, False otherwise.

ecsjobs.jobs.ecs_task module

class ecsjobs.jobs.ecs_task.**EcsTask**(*name*, *schedule*, *summary_regex=None*, *cron_expression=None*, *cluster_name=None*, *task_definition_family=None*, *overrides=None*, *network_configuration=None*)

Bases: *ecsjobs.jobs.base.Job*

Class to run an ECS Task asynchronously; starts the task with the *run()* method and then uses *poll()* to wait for it to finish. Sets *exitcode* according to:

- if only one container in the task, the exit code of that container
- otherwise, the maximum exit code of all containers

Parameters

- **name** (*str*) – unique name for this job
- **schedule** (*str*) – the name of the schedule this job runs on
- **summary_regex** (string or None) – A regular expression to use for extracting a string from the job output for use in the summary table. If there is more than one match, the last one will be used.
- **cron_expression** (*str*) – A cron-like expression parsable by cronex specifying when the job should run. This has the effect of causing runs to skip this job unless the expression matches. It's recommended not to use any minute specifiers and not to use any hour specifiers if the total runtime of all jobs is more than an hour.
- **cluster_name** (*str*) – name of the ECS cluster to run the task on
- **task_definition_family** (*str*) – Name of the Task Definition family to run
- **overrides** (*dict*) – RunTask overrides hash/mapping/dict to pass to ECS RunTask API call, as specified in the documentation for ECS.Client.run_task()
- **networkConfiguration** (*dict*) – RunTask networkConfiguration parameter to pass to ECS API call, as specified in the documentation for ECS.Client.run_task()

_log_info_for_task(*task_family*)

Return a dictionary of container name to 2-tuple of Log Group Name and Log Stream Prefix, for each container in the specified Task Definition that uses the awslogs log driver.

Parameters **task_family** (*str*) – task family name to return log settings for

Returns dictionary of container name to 2-tuple of Log Group Name and Log Stream Prefix, for each container in the specified Task Definition that uses the awslogs log driver

Return type dict

_output_for_task_container(*taskid*, *cont_name*)

Update self.output with the CloudWatch logs for the containers in the task.

Parameters

- **taskid** (*str*) – ECS Task ID
- **cont_name** (*str*) – container name in the task

Returns CloudWatch logs for the container

Return type str

_schema_dict = {'properties': {'cluster_name': {'type': 'string'}, 'network_configu
Dictionary describing the configuration file schema, to be validated with jsonschema.

poll()
Poll to check status on the task. If STOPPED, set this Job as finished and collect report information.

Returns whether or not the Task is finished

Return type bool

report_description()
Return a one-line description of the Job for use in reports.

Return type str

run()
Run the command for the job. Output and exit code will be captured by *poll()*, according to `self._task_id`.

Returns `None`

ecsjobs.jobs.local_command module

class ecsjobs.jobs.local_command.**LocalCommand**(*name*, *schedule*, *summary_regex=None*, *cron_expression=None*, *command=None*, *shell=False*, *timeout=None*, *script_source=None*)

Bases: *ecsjobs.jobs.base.Job*

Job class to run a local command via `subprocess.run()`. The *output* property of this class contains combined STDOUT and STDERR.

Parameters

- **name** (*str*) – unique name for this job
- **schedule** (*str*) – the name of the schedule this job runs on
- **summary_regex** (`string` or `None`) – A regular expression to use for extracting a string from the job output for use in the summary table. If there is more than one match, the last one will be used.
- **cron_expression** (*str*) – A cron-like expression parsable by cronex specifying when the job should run. This has the effect of causing runs to skip this job unless the expression matches. It's recommended not to use any minute specifiers and not to use any hour specifiers if the total runtime of all jobs is more than an hour.
- **command** (`str` or `list`) – The command to execute as either a String or a List of Strings, as used by `subprocess.run()`. If `script_source` is specified and this parameter is not an empty string or empty list, it will be passed as arguments to the downloaded script.
- **shell** (*bool*) – Whether or not to execute the provided command through the shell. Corresponds to the `shell` argument of `subprocess.run()`.
- **timeout** (*int*) – An integer number of seconds to allow the command to run. Coorresponds to the `timeout` argument of `subprocess.run()`.
- **script_source** (*str*) – A URL to retrieve an executable script from, in place of `command`. This currently supports URLs with `http://`, `https://` or `s3://` schemes. HTTP and HTTPS URLs must be directly retrievable without any authentication. S3 URLs will use the same credentials already in use for the session. **Note** that this setting will cause ecsjobs to download and execute code from a potentially untrusted location.

_get_script(*script_url*)
Download a script from HTTP/HTTPS or S3 to a temporary path, make it executable, and return the command to execute.

Parameters **script_url** (*str*) – URL to download - HTTP/HTTPS or S3

Returns path to the downloaded executable script if `self._command` is an empty string, empty array, or None. Otherwise, a list whose first element is the path to the downloaded executable script, and then contains `self._command`.

Return type str

_schema_dict = {'properties': {'command': {'oneOf': [{'type': 'string'}, {'type':
Dictionary describing the configuration file schema, to be validated with jsonschema.

report_description()
Return a one-line description of the Job for use in reports.

Return type str

run()
Run the command for the job. Either raise an exception or return True if the command exited 0, False if it exited non-zero.

Returns True if command exited 0, False otherwise.

1.3.1.2 Submodules

ecsjobs.config module

class ecsjobs.config.**Config**
Bases: object

YAML_EXTNS = ['.yml', '.yaml']
File extensions to consider as YAML config files.

_get_multipart_config(*bucket*, *prefix*)
Retrieve each piece of a multipart config from S3; return the combined configuration (i.e. the corresponding single-dict config).

Parameters

- **bucket** (S3.Bucket) – the S3 bucket to retrieve configs from
- **prefix** (*str*) – prefix for configuration files

Returns combined configuration dict

Return type dict

_get_yaml_from_s3(*bucket*, *key*)
Retrieve the contents of a file from S3 and deserialize the YAML.

Parameters

- **bucket** (S3.Bucket) – the S3 bucket to retrieve the file from
- **key** (*str*) – key/path of the file

Returns deserialized YAML file contents

Return type dict

_global_defaults = {'email_subject': 'ECSJobs Report', 'failure_command': None, 'fai
Default values for global configuration settings.

_key_is_yaml(*key*)
Test whether or not the specified S3 key is a YAML file.

Parameters **key** (*str*) – key in S3

Returns whether key is a YAML file

Return type bool

_load_config()
Check environment variables; call either *_load_config_s3()* or *_load_config_local()*.

Raises RuntimeError

_load_config_local(*conf_path*)
Load configuration from the local filesystem. Sets `self._raw_conf`.

Parameters **conf_path** (*str*) – path to configuration on local FS

_load_config_s3(*bucket_name*, *key_name*)
Retrieve and load configuration from S3. Sets `self._raw_conf`.

Parameters

- **bucket_name** (*str*) – Name of the S3 bucket to retrieve config from
- **key_name** (*str*) – config key or prefix in bucket

_load_yaml_from_disk(*path*)
Load a YAML file from disk and return the contents.

Parameters **path** (*str*) – path to load from

Returns deserialized YAML file contents

Return type dict

_make_jobs()
Reads `self._jobs_conf` and instantiates job classes, populating `self._jobs`.

_validate_config()
Validate the configuration in `self._raw_conf`. Writes `self._global_conf`.

get_global(*k*)
Return the value of the specified global configuration setting, from the global configuration (if present) or else from the global defaults.

Parameters **k** – configuration key to get

Returns value of global config setting

jobs
Return the list of *ecsjobs.jobs.base.Job* instances.

Returns list of jobs

Return type list

jobs_for_schedules(*schedule_names*)
Given one or more schedule names, return the list of jobs for those schedules (in order).

Parameters **schedule_names** (*list*) – schedule names to get jobs for

Returns list of Jobs for the specified schedules

Return type list

schedule_names
Return a list of all String schedule names defined in the config.

Returns all defined schedule names

Return type list

ecsjobs.reporter module

class ecsjobs.reporter.**Reporter**(*config*)
Bases: object

ECSJobs Report Generator and SES Sender

Initialize the Report generator.

Parameters **config** (ecsjobs.config.Config) – Configuration

_div_for_job(*job*, *exc=None*, *unfinished=False*)
Generate a div for the results email with the output or exception of a specific job.

Parameters

- **job** (ecsjobs.jobs.base.Job) – the Job to generate a div for
- **exc** (Exception or None) – Exception caught when running job, or None
- **unfinished** (*bool*) – whether or not the job was killed before being finished.

Returns HTML div for the report

Return type str

_make_report(*finished*, *unfinished*, *excs*, *start_dt*, *end_dt*)
Generate the HTML email report

Parameters

- **finished** (*list*) – Finished Job instances.
- **unfinished** (*list*) – Unfinished (timed-out) Job instances.
- **excs** (*dict*) – Dict of Jobs that generated an exception while running; keys are Job class instances and values are 2-tuples of the caught Exception objects and string formatted tracebacks.
- **start_dt** (*datetime.datetime*) – datetime instance when run was started
- **end_dt** (*datetime.datetime*) – datetime instance when run was finished

Returns HTML email report content

Return type str

_tr_for_job(*job*, *exc=None*, *unfinished=False*)
Generate a row in the results table for a specific job.

Parameters

- **job** (ecsjobs.jobs.base.Job) – the Job to generate a div for
- **exc** (2-tuple or None) – None or 2-tuple of Exception caught when running job and traceback formatted as a string.

- **unfinished** (*bool*) – whether or not the job was killed before being finished.

Returns Table row for the report

Return type str

run(*finished*, *unfinished*, *excs*, *start_dt*, *end_dt*, *only_email_if_problems=False*)
Generate and send the report.

Parameters

- **finished** (*list*) – Finished Job instances.
- **unfinished** (*list*) – Unfinished (timed-out) Job instances.
- **excs** (*dict*) – Dict of Jobs that generated an exception while running; keys are Job class instances and values are 2-tuples of the caught Exception objects and string formatted tracebacks.
- **start_dt** (*datetime.datetime*) – datetime instance when run was started
- **end_dt** (*datetime.datetime*) – datetime instance when run was finished
- **only_email_if_problems** (*bool*) – If True, only send email report if there were failures, exceptions, or unfinished jobs. Otherwise, always send email.

td(*s*)

th(*s*)

ecsjobs.runner module

class ecsjobs.runner.**EcsJobsRunner**(*config*, *only_email_if_problems=False*)
Bases: object

_poll_jobs()
Poll the jobs in `self._running`; if they're finished, move the Job to `self._finished`.

_report()
Generate and send email report.

_run_jobs(*jobs*, *force_run=False*)
Run the specified jobs.

Parameters

- **jobs** (*list*) – list of Job instances to run
- **force_run** (*bool*) – Run each job regardless of cron expression

run_job_names(*job_names*)
Run the named jobs, regardless of schedule.

Parameters **job_names** (*list*) – list of string job names to run

run_schedules(*schedule_names*)
Run the named schedules.

Parameters **schedule_names** (*list*) – names of the schedules to run

ecsjobs.runner.**main**(*argv=None*)

ecsjobs.runner.**parse_args**(*argv*)

ecsjobs.runner.**set_log_debug**(*logger*)
set logger level to DEBUG, and debug-level output format, via *set_log_level_format()*.

ecsjobs.runner.**set_log_info**(*logger*)
set logger level to INFO via *set_log_level_format()*.

ecsjobs.runner.**set_log_level_format**(*logger*, *level*, *format*)
Set logger level and format.

Parameters

- **logger** (*logging.Logger*) – the logger object to set on
- **level** (*int*) – logging level; see the logging constants.
- **format** (*str*) – logging formatter format string

ecsjobs.schema module

class ecsjobs.schema.**Schema**
Bases: object

base_schema = {'$schema': 'http://json-schema.org/schema#', 'additionalProperties':

schema_dict
Return the full generated schema dict.

Returns generated JSONSchema

Return type dict

validate(*config_dict*)
Validate the specified configuration dict against the schema.

Parameters **config_dict** (*dict*) – configuration to validate

ecsjobs.version module

1.4 Changelog

1.4.1 1.0.0 (2021-08-23)

- Bump to 1.0.0 since I've been using this for years
- Build and test against Python 3.9; build Docker image off of Python 3.9
- Add `rsnapshot` apk to Docker image
- Updates for new pytest version

1.4.2 0.4.3 (2021-01-06)

- Fix deprecated PyYAML load calls.

1.4.3 0.4.2 (2021-01-06)

- Relax PyYAML version in order to work with Python 3.9.

1.4.4 0.4.1 (2018-08-11)

- Add leading newlines when passing report to `failure_command`
- In the `failure_html_path` global config setting, the string `{date}` will be replaced with the current datetime (at time of config load) in `%Y-%m-%dT%H-%M-%S` format.

1.4.5 0.4.0 (2018-02-25)

- Add `awscli` to Docker image
- Add new global configuration options:
  - **failure_html_path** - *(optional)* a string absolute path to write the HTML email report to on disk, if sending via SES fails. If not specified, a temporary file will be used (via Python's `tempfile.mkstemp`) and its path included in the output.
  - **failure_command** - *(optional)* Array. A command to call if sending via SES fails. This should be an array beginning with the absolute path to the executable, suitable for passing to Python's `subprocess.Popen()`. The content of the HTML report will be passed to the process on STDIN.

1.4.6 0.3.0 (2017-12-01)

- Document release process
- Document how to run in ECS as a Scheduled Task
- `LocalCommand` - if `script_source` parameter is specified, instead of ignoring `command` send it as arguments to the downloaded script.
- `LocalCommand` bugfix - Handle when retrieved script_source is bytes instead of string.
- Add `-m` / `--only-email-if-problems` command line argument to allow suppressing email reports if all jobs ran successfully.
- Make report email subject configurable via `email_subject` global configuration option.

1.4.7 0.2.0 (2017-11-30)

- Initial mostly-complete release

1.5 Development

Any and all contributions are welcome.

1.5.1 Installing for Development

To setup ecsjobs for development:

1. Fork the ecsjobs repository on GitHub and clone it locally; `cd ecsjobs`.
2. Create a `virtualenv` to run the code in and install it:

```
$ virtualenv venv
$ source venv/bin/activate
$ python setup.py develop
```

3. Check out a new git branch. If you're working on a GitHub issue you opened, your branch should be called "issues/N" where N is the issue number.

1.5.2 Release Checklist

1. Ensure that Travis tests are passing in all environments.
2. Ensure that test coverage is no less than the last release (ideally, 100%).
3. Build docs for the branch (locally) and ensure they look correct (`tox -e docs`). Commit any changes.
4. Increment the version number in ecsjobs/version.py and add version and release date to CHANGES.rst. `export ECSJOBS_VER=x.y.z`. Ensure that there are CHANGES.rst entries for all major changes since the last release, and that any new required IAM permissions are explicitly mentioned. Commit changes and push to GitHub. Wait for builds to pass.
5. Confirm that README.rst renders correctly on GitHub.
6. Upload package to testpypi, confirm that README.rst renders correctly.
   - Make sure your ~/.pypirc file is correct (a repo called `test` for https://testpypi.python.org/pypi).
   - `rm -Rf dist`
   - `python setup.py sdist bdist_wheel`
   - `twine upload -r test dist/*`
   - Check that the README renders at https://testpypi.python.org/pypi/ecsjobs
7. Tag the release in Git, push tag to GitHub:
   - tag the release with a signed tag: `git tag -s -a $ECSJOBS_VER -m "$ECSJOBS_VER released $(date +%Y-%m-%d)"`
   - Verify the signature on the tag, just to be sure: `git tag -v $ECSJOBS_VER`
   - push the tag to GitHub: `git push origin $ECSJOBS_VER`
8. Upload package to live pypi:
   - `twine upload dist/*`
9. Run `./build_docker.sh` to build the Docker image. Take note of the generated (timestamp) tag and `export TIMESTAMP=<generated timestamp tag>`.
10. Re-tag the generated Docker image with the version and "latest" and then push to Docker Hub:

- `docker tag jantman/ecsjobs:$TIMESTAMP jantman/ecsjobs:$ECSJOBS_VER`
- `docker push jantman/ecsjobs:$ECSJOBS_VER`
- `docker tag jantman/ecsjobs:$TIMESTAMP jantman/ecsjobs:latest`

- `docker push jantman/ecsjobs:latest`

11. On GitHub, create a release for the tag. Run `pandoc -f rst -t markdown_github CHANGES.rst` to convert CHANGES.rst to Markdown, and use the appropriate section for the GitHub release description.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

e

Index

Symbols

B

C

D

E