Amazon Distributed Runner Documentation

Release 0.1

Bas Hoonhout

November 11, 2016

Contents

The Amazon Distributed Runner (ADR) is a simple queueing system based on the Amazon SQS Message Queue and the Amazon S3 Buckets to handle batches of numerical model runs or other on-demand processes. Any node with the ADR installed can process jobs from the central queue. The ADR provides tools to launch and manage nodes at the Amazon EC2 computational facility, but the queueing system is not restricted to this facility.

The ADR is available through the OpenEarth GitHub repository: http://github.com/openearth/amazon-distributed-runner/

CHAPTER 1

Examples

```
adr config
adr create
adr launch -n 5
adr prepare
adr queue ~/GitHub/aeolis-models/nickling1995/nickling1995_*.txt
adr queue ~/GitHub/aeolis-models/dong2004/dong2004_*.txt
adr download ~/Downloads/
adr destroy
```

CHAPTER 2

Source code

2.1 Source code

The ADR package contains of four modules:

- adr: main module with all Amazon S3, SQS and EC2 functions
- fabfile: module for interaction with individual workers
- config: module for managing the package configuration
- console: module that defines the console commands

2.1.1 adr

adr.**cache_contents**(*path*)
Cache current contents of directory

Traverses a given directory and registers all contents in a hidden cache file.

Parameters **path** (*str*) – Path for which the contents need to be cached

Returns Cache filename

Return type str

See also:

restore_contents()

adr.**create**(*region_name='eu-central-1'*)
Create runner

Creates runner consisting of a unique runner ID, an Amazon S3 Bucket and an Amazon SQS Message Queue. The runner ID is a UUID4 string. The name of the bucket and queue is equal to the runner ID.

Parameters **region_name** (*str, optional*) – Amazon region identifier

Returns Runner ID

Return type str

See also:

launch(), queue(), process(), destroy(), create_queue(), create_bucket()

adr.**create_bucket**(*s3*, *runner_id*, *region_name='eu-central-1'*, *ACL='private'*)
Create Amazon S3 Bucket

Parameters

- **s3** (*boto3 resource*) – Boto3 S3 Resource object in specified region
- **runner_id** (*str*) – Runner ID and name of bucket to be created
- **region_name** (*str, optional*) – Amazon region idenifier that is used as constraint for inter-region data transfers
- **ACL** (*str, optional*) – Amazon access control list [default: private]

adr.**create_queue**(*sqs*, *runner_id*)
Create Amazon SQS Message Queue

Parameters

- **sqs** (*boto3 resource*) – Boto3 SQS Resource object in specified region
- **runner_id** (*str*) – Runner ID and name of queue to be created

adr.**deregister_workers**(*s3*, *runner_id*, *workers*, *prefix='_workers/'*)
Deregister workers

A worker is deregistered by deleting the empty file object with a prefix `prefix` and name equal to the public IP address of the worker in the Amazon S3 Bucket.

Parameters

- **s3** (*boto3 resource*) – Boto3 S3 resource object in specified region
- **runner_id** (*str*) – Runner ID
- **workers** (*str or list*) – String or list of strings with public IP addresses of workers to be deregistered
- **prefix** (*str, optional*) – Prefix for generated keys

See also:

`register_workers()`

adr.**destroy**(*runner_id*, *region_name='eu-central-1'*, *hosts=None*)
Destroy runner

Delete Amazon SQS Message Queue associated with specified runner, terminate its workers and clean its Amazon S3 Bucket (but keep the batch results).

Parameters

- **runner_id** (*str*) – Runner ID
- **region_name** (*str*) – Amazon region identifier
- **hosts** (*list, optional*) – List of specific hosts to destroy

Notes

No warnings, no undo!

adr.**download**(*runner_id*, *path*, *region_name='eu-central-1'*, *overwrite=False*)
Download batch results from Amazon S3 Bucket

Parameters

- **runner_id** (*str*) – Runner ID

- **path** (*str*) – Local download location
- **region_name** (*str, optional*) – Amazon region identifier
- **overwrite** (*bool, optional*) – Overwrite existing files

adr.**download_batch**(*s3*, *runner_id*, *batch_id*, *path*)
Download batch input from Amazon S3 Bucket

Download zipped batch input and unzip at specified location.

Parameters

- **s3** (*boto3 resource*) – Boto3 S3 resource object in specified region
- **runner_id** (*str*) – Runner ID
- **batch_id** (*str*) – Batch ID
- **path** (*str*) – Local download directory

adr.**find_root**(*files*)
Returns the common root of a collection of file paths

Parameters

- **files** (*list*) – List of file paths
- **Returns** –
- **——** –
- **str** – Common file root

adr.**get_job**(*sqs*, *runner_id*, *delay=10*, *retry=30*)
Poll SQS Message Queue for job

Parameters

- **sqs** (*boto3 resource*) – Boto3 SQS resource object in specified region
- **runner_id** (*str*) – Runner ID
- **delay** (*int*) – Delay in seconds between polls
- **retry** (*int*) – Maximum number of polls

Returns Message received from queue parsed with `parse_message()`

Return type dict

See also:

`parse_message()`

adr.**get_runners**(*region_name='eu-central-1'*)
Get a list of valid runner ID's

A valid runner ID is an ID with an active Amazon SQS Message Queue and Amazon S3 Bucket.

Parameters **region_name** (*str*) – Amazon region identifier

Returns List of valid runner ID's

Return type list

See also:

`get_workers()`

adr.**get_workers**(*runner_id*, *region_name='eu-central-1'*, *prefix='_workers/'*)
Get list of public IP addresses of workers for specific runner

Parameters

- **runner_id** (*str*) – Runner ID
- **region_name** (*str, optional*) – Amazon region identifier
- **prefix** (*str, optional*) – Prefix used for registration of workers

Returns List of public IP addresses

Return type list

See also:

`get_runners()`, `register_workers()`, `deregister_workers()`

adr.**isiterable**(*lst*)
Checks if input is iterable

adr.**iterate_workers**(*runner_id*, *region_name='eu-central-1'*, *hosts=None*)
Iterator for Amazon EC2 instances associated with a given runner

Parameters

- **runner_id** (*str*) – Runner ID
- **region_name** (*str*) – Amazon region identifier
- **hosts** (*list*) – List of specific hosts to iterate

Returns Boto3 EC2 instance object associated with given runner

Return type boto3 EC2 instance

adr.**key_exists**(*s3*, *runner_id*, *key*)
Check if key exists in Amazon S3 Bucket

Parameters

- **s3** (*boto3 resource*) – Boto3 S3 resource object in specified region
- **runner_id** (*str*) – Runner ID
- **key** (*str*) – Key to be checked

Returns Flag indicating existence of key

Return type bool

adr.**launch**(*runner_id*, *n*, *region_name='eu-central-1'*, ***kwargs*)
Launch Amazon workers for specific runner

Launches Amazon instances for a given runner and registers the workers in the Amazon S3 Bucket. Each Amazon instance is tagged with the runner ID.

Parameters

- **runner_id** (*str*) – Runner ID
- **n** (*int*) – Number of instances to launch
- **region_name** (*str, optional*) – Amazon region identifier
- **kwargs** (*dict, optional*) – Keyword options to `launch_workers()`

Returns List with public IP addresses of workers

Return type list

See also:

`prepare()`, `stop()`, `launch_workers()`, `register_workers()`

`adr.launch_workers(ec2, runner_id, n=1, ami='ami-d09b6ebf', asg=['sg-13d17c7b'], akp='Amazon AeoLiS Test Key', ait='m3.medium')`

Launch Amazon workers, tag them and wait for them to be online

Parameters

- **ec2** (*boto3 resource*) – Boto3 EC2 resource object in specified region
- **runner_id** (*str*) – Runner ID
- **n** (*int, optional*) – Number of instances to launch [default: 1]
- **ait** (*str, optional*) – Amazon Instance Type
- **ami** (*str, optional*) – Amazon Machine Image
- **asg** (*list, optional*) – List of strings with Amazon Security Groups
- **akp** (*str, optional*) – Name of Amazon Key Pair

Returns List with public IP addresses of workers

Return type list

`adr.parse_message(message)`

Parses message from SQS Message Queue

Parameters **message** (*dict*) – Multi-level message from SQS Message Queue

Returns Flattened message

Return type dict

`adr.prepare(runner_id, region_name='eu-central-1', hosts=None, user='ubuntu', password=None, key_filename=None, warn_only=False, timeout=600)`

Prepare workers for specific runner

Install the Amazon Distributed Runner to all workers and start processing the queue. Installation procedure is defined in `fabfile`, which can be used as input to `fab` as well.

Parameters

- **runner_id** (*str*) – Runner ID
- **region_name** (*str, optional*) – Amazon region idenitifier
- **hosts** (*list, optional*) – List of specific hosts to prepare
- **user** (*str, optional*) – SSH username
- **password** (*str, optional*) – SSH password
- **key_filename** (*str, optional*) – Path to SSH key file
- **warn_only** (*bool, optional*) – Only warn on error, but attempt to continue [default: True]
- **timeout** (*int, optional*) – Maxumum duration in seconds of installation execution [default: 600]

See also:

`fabfile`

adr.**process**(*runner_id*, *workingdir='.'*, *region_name='eu-central-1'*, *stop_on_empty=False*)
Start processing loop for specific runner

Each worker needs to start this processing procedure. It polls the queue, processes jobs and uploads the result.

Parameters

- **runner_id** (*str*) – Runner ID
- **workingdir** (*str*) – Working dir at processing node
- **region_name** (*str, optional*) – Amazon region identifier
- **stop_on_empty** (*bool, optional*) – Flag to quit the processing loop if no messages are left [default: False]

See also:

`process_job()`

adr.**process_job**(*sqs*, *s3*, *runner_id*, *workingdir='.'*)
Process specific job

Parameters

- **sqs** (*boto3 resource*) – Boto3 SQS resource object in specified region
- **s3** (*boto3 resource*) – Boto3 S3 resource object in specified region
- **runner_id** (*str*) – Runner ID
- **workingdir** (*str, optional*) – Working directory at processing node

Returns Returns True if a job has been processed and False otherwise

Return type bool

See also:

`get_job()`, `download_batch()`, `upload_files()`

adr.**queue**(*runner_id, files, region_name='eu-central-1', command='aeolis {}', preprocessing='source ~/.envs/aeolis/bin/activate', postprocessing=None, store_patterns=['\\.nc$']*)
Queue job

Queues job to runner by zipping the root of all given input files, uploading the zipped input to the Amazon S3 Bucket and announcing the job to the Amazon SQS Message Queue.

Parameters

- **runner_id** (*str*) – Runner ID
- **files** (*list*) – List of file names used as input
- **region_name** (*str*) – Amazon region identifier
- **command** (*str*) – Command pattern to be executed. A single placeholder {} can be used to determine the location where the input file is inserted.
- **preprocessing** (*str*) – Command to be executed preceding the `command`.
- **postprocessing** (*str*) – Command to be executed following the `command`.
- **store_patterns** (*list*) – List of regular expressions to idenitfy files that need to be stored to the Amazon S3 Bucket after execution.

See also:

`download()`, `queue_job()`, `upload_batch()`

Notes

Be aware that the common root of all input files is zipped and uploaded. If the input files are located in very different locations, these files may have a very shallow common root that is potentially very large.

adr.**queue_job**(*sqs*, *runner_id*, *batch_id*, *command*, *store_patterns=None*, *preprocessing=None*, *postprocessing=None*)
Construct and send message to the Amazon SQS Message Queue

Parameters

- **sqs** (*boto3 resource*) – Boto3 SQS resource object in specified region
- **runner_id** (*str*) – Runner ID
- **batch_id** (*str*) – Batch ID
- **command** (*str*) – Command pattern to be executed. A single placeholder {} can be used to determine the location where the input file is inserted.
- **preprocessing** (*str*) – Command to be executed preceding the `command`.
- **postprocessing** (*str*) – Command to be executed following the `command`.
- **store_patterns** (*list*) – List of regular expressions to idenitfy files that need to be stored to the Amazon S3 Bucket after execution.

adr.**register_workers**(*s3*, *runner_id*, *workers*, *prefix='_workers/'*)
Register workers

A worker is registered by creating an empty file object with a prefix `prefix` and name equal to the public IP address of the worker in the Amazon S3 Bucket.

Parameters

- **s3** (*boto3 resource*) – Boto3 S3 resource object in specified region
- **runner_id** (*str*) – Runner ID
- **workers** (*str or list*) – String or list of strings with public IP addresses of workers to be registered
- **prefix** (*str, optional*) – Prefix for generated keys

See also:

`deregister_workers()`

adr.**restore_contents**(*path*)
Restore contents of directory based on cache file

Compares the current contents of a directory with the previously cached contents of that directory and removes any files and directories that have been added.

Parameters **path** (*str*) – Path for which the contents need to be restored

See also:

`cache_contents()`

adr.**start**(*runner_id*, *region_name='eu-central-1'*, *hosts=None*)
Start stopped workers for specific runner

Parameters

- **runner_id** (*str*) – Runner ID

- **region_name** (*str, optional*) – Amazon region identifier
- **hosts** (*list, optional*) – List of specific hosts to start

See also:

`stop()`, `register_workers()`

`adr.`**`stop`**(*runner_id*, *region_name='eu-central-1'*, *hosts=None*)
Stop running workers for specific runner

Parameters

- **runner_id** (*str*) – Runner ID
- **region_name** (*str, optional*) – Amazon region identifier
- **hosts** (*list, optional*) – List of specific hosts to stop

See also:

`start()`, `deregister_workers()`

`adr.`**`upload_batch`**(*s3, runner_id, path, exclude_patterns=['\\.log$', '\\.nc$', '\\.pyc$']*)
Upload batch input to Amazon S3 Bucket

Creates a unique batch ID and uploads the batch input under that id to the Amazon S3 Bucket.

Parameters **s3** (*boto3 resource*) – Boto3 S3 resource object in specified region

Returns Batch ID

Return type str

`adr.`**`upload_files`**(*s3, runner_id, batch_id, path, include_patterns=['\\.nc$'], overwrite=False*)
Upload batch results to Amazon S3 Bucket

Traverses directory tree and upload all files that match one or more regular expressions.

Parameters

- **s3** (*boto3 resource*) – Boto3 S3 resource object in specified region
- **runner_id** (*str*) – Runner ID
- **batch_id** (*str*) – Batch ID
- **path** (*str*) – Root directory for traversal
- **include_patterns** (*list, optional*) – List of regular expressions from which at least one should match for a file to be uploaded
- **overwrite** (*bool, optional*) – Flag to enable overwriting remote files [default: False]

2.1.2 fabfile

`fabfile.`**`install`**
Prepare node for processing queued ADR jobs

Installs the following packages:

•dtach

•virtualenv

•boto3

•fabric

- docopt
- amazon-dsitributed-runner

Creates a virtual environment `adr` and copies the local AWS credentials.

Parameters **required_packages** (*list, optional*) – Additional Python packages to install

`fabfile.`**`runv`**(*cmd*, *env='~/.envs/adr'*, *socket=None*)
Run command in virtual environment

Parameters

- **cmd** (*str*) – Shell command
- **env** (*str*) – Path to virtual environent
- **socket** (*str, optional*) – Name of socket for running command detached

Returns Command return value (if not detached)

Return type str

Notes

Detachting the process is done with the `dtach` command, which should be available at the node.

`fabfile.`**`start`**
Start ADR instance on specific runner

ADR instance is started detached under socket `adr` and in virtual environment `adr`.

Parameters **runner_id** (*str*) – Runner ID

`fabfile.`**`stop`**
Stop all ADR instances

2.1.3 config

`config.`**`ask_question`**(*cfg*, *keys*, *display*, *masked=False*, *split=False*)
Helper function to ask wizard question and alter config structure

Parameters

- **cfg** (*dict*) – Config structure to be altered
- **keys** (*tuple*) – Key traversal for config structure that localizes the value that is addressed in the question
- **display** (*str*) – The question that is displayed to the user
- **masked** (*bool, optional*) – Flag to mask the current config value (used for passwords)
- **split** (*bool, optional*) – Flag to split the user input on comma's

Returns Updated config structure

Return type dict

See also:

`disp_item()`, `get_item()`, `set_item()`

config.**disp_item**(*val*, *masked=False*)
Convert config value in display value

Joins lists by comma's and masks secret value for the first 80%.

Parameters

- **val** (*str or list*) – Config value
- **masked** (*bool, optional*) – Flag to enable masking

Returns Display value

Return type str

config.**get_item**(*cfg*, *keys*)
Gets item from config structure by key traversal

Parameters

- **cfg** (*dict*) – Config structure
- **keys** (*tuple*) – Key traversal for config structure

Returns Remaining part of config structure after traversal

Return type dict or config value

config.**load_config**(**keys*)
Load specific part of config file

Parameters **keys** (*tuple*) – Key traversal of config structure

Returns Part of config structure

Return type dict or config value

Examples

```
>>> config.load_config()
>>> config.load_config('aws', 'credentials')
```

See also:

`update_config()`, `write_config()`, `get_item()`

config.**set_item**(*cfg*, *keys*, *val*)
Sets item in config structure by key traversal

Parameters

- **cfg** (*dict*) – Config structure
- **keys** (*tuple*) – Key traversal for config structure
- **val** (*any*) – Config value to be set

Returns Updated config structure

Return type dict

config.**update_config**(**keys*)
Update specific part of config file

Parameters **keys** (*tuple*) – Key traversal of config structure. The last value is the value that will be set.

Examples

```
>>> config.write_config('aws', 'credentials', 'access_key_id', 'ABCDEFGHIJKLMNOPQRSTUVWXYZ')
```

See also:

load_config(), write_config(), set_item()

config.**wizard**()
Configuration wizard

Loads current configuration values and asks a sequence of questions to allow altering the current values. If no input is given, the current value is not changed.

See also:

load_config(), write_config(), write_aws_config(), ask_question()

config.**write_aws_config**(*cfg*)
Write relevant parts of config structure to private files in AWSCLI format

Parameters **cfg** (*dict*) – Config structure following JSON_DEFAULT

See also:

write_config()

config.**write_config**(*cfg*)
Write config structure to private file

Parameters **cfg** (*dict*) – Config structure following JSON_DEFAULT

See also:

load_config(), update_config(), write_aws_config()

2.1.4 console

console.**adr_config**()
adr_config : Configure Amazon Distributed Runner

Usage: adr config [options]

Options:

-h, --help	Show this help message and exit

console.**adr_console**()
adr : Amazon Distributed Runner

Creates a queue for handling batches, launches workers to process batches or queues batches. Also contains the processor script that runs on the workers.

Usage: adr create Create runner adr launch Launch workers adr prepare Prepare workers adr start Start workers adr stop Stop workers adr destroy Destroy runner and workers adr queue Queue batch to runner adr process Process batches from queue adr download Download batch results adr list List available runners adr set Set current runner adr config Configuration wizard

Options:

-h, --help	Show this help message and exit
--verbose=LEVEL	Write logging messages [default: 30]

`console.`**`adr_create`**`()`
adr_create : Create runner

Usage: adr create [options]

Options:

-h, --help	Show this help message and exit
--region=REGION	Amazon region
--verbose=LEVEL	Write logging messages [default: 30]

`console.`**`adr_destroy`**`()`
adr_destroy : Destroy runner and workers

Usage: adr destroy [<runner>] [options]

Positional arguments: runner Runner ID

Options:

-h, --help	Show this help message and exit
--hosts=HOSTS	Comma-separated list of hostnames
--region=REGION	Amazon region
--verbose=LEVEL	Write logging messages [default: 30]

`console.`**`adr_download`**`()`
adr_download : Download batch results

Usage: adr download <path> [<runner>] [options]

Positional arguments: path Download location runner Runner ID

Options:

-h, --help	Show this help message and exit
--region=REGION	Amazon region
--overwrite	Overwrite existing files
--verbose=LEVEL	Write logging messages [default: 30]

`console.`**`adr_launch`**`()`
adr_launch : Launch workers

Usage: adr launch [<runner>] [options]

Positional arguments: runner Runner ID

Options:

-h, --help	Show this help message and exit
-n N	Number of workers [default: 1]
--region=REGION	Amazon region
--ami=AMI	Amazon Machine Image (AMI)
--asg=SG	Comma-separated list of Amazon Security Groups
--akp=KEY	Amazon Key Pair
--ait=TYPE	Amazon Instance Type

--verbose=LEVEL Write logging messages [default: 30]

`console.`**`adr_list`**`()`

adr_list : List available runners and hosts

Usage: adr list [<runner>] [options]

Positional arguments: runner Runner ID

Options:

-h, --help	Show this help message and exit
--region=REGION	Amazon region
--verbose=LEVEL	Write logging messages [default: 30]

`console.`**`adr_prepare`**`()`

adr_prepare : Prepare workers

Usage: adr prepare [<runner>] [options]

Positional arguments: runner Runner ID

Options:

-h, --help	Show this help message and exit
--hosts=HOSTS	Comma-separated list of hostnames
--user=USER	SSH username
--password=PW	SSH password
--key=KEY	SSH key filename
--region=REGION	Amazon region
--verbose=LEVEL	Write logging messages [default: 30]

`console.`**`adr_process`**`()`

adr_process : Process batches from queue

Usage: adr process [<runner>] [options]

Positional arguments: runner Runner ID

Options:

-h, --help	Show this help message and exit
--workingdir=PATH	Working directory [default: .]
--region=REGION	Amazon region
--verbose=LEVEL	Write logging messages [default: 30]

`console.`**`adr_queue`**`()`

adr_queue : Queue batch

Usage: adr queue <file>... [<runner>] [options]

Positional arguments: file Input files to queue runner Runner ID

Options:

-h, --help	Show this help message and exit
--command=CMD	Shell command

--verbose=LEVEL Write logging messages [default: 30]

`console.`**`adr_set`**`()`

adr_set : Set current runner

Usage: adr set [<runner>]

Positional arguments: runner Runner ID

Options:

-h, --help Show this help message and exit

`console.`**`adr_start`**`()`

adr_start : Start workers

Usage: adr start [<runner>] [options]

Positional arguments: runner Runner ID

Options:

-h, --help Show this help message and exit

--hosts=HOSTS Comma-separated list of hostnames

--region=REGION Amazon region

--verbose=LEVEL Write logging messages [default: 30]

`console.`**`adr_stop`**`()`

adr_stop : Stop workers

Usage: adr stop [<runner>] [options]

Positional arguments: runner Runner ID

Options:

-h, --help Show this help message and exit

--hosts=HOSTS Comma-separated list of hostnames

--region=REGION Amazon region

--verbose=LEVEL Write logging messages [default: 30]

CHAPTER 3

Indices and tables

- *genindex*
- *modindex*
- *search*

a

c

f

W