yunpipe Documentation

Release 0.0.4

Yuxing Wang

September 09, 2016

Contents

yunpipe is an automatic setup tool for setting up the data analysis pipeline on the Amazon Web Services (AWS). It provides an easy to deploy, use and scale your data analysis algorithm and work flow on the cloud as well as sharing between colleges and institutions. It is developed in Python 3 using boto, the AWS SDK for Python.

Now yunpipe is on Pypi, and it supports pip. The latest version is v0.0.3.dev. To install yunpipe, *pip install yunpipe*. After install yunpipe:

To submit an algorithm or bring your analyze tool, use *wrap -ds*. For more options, check: *wrap –help*

To run single algorithm or deploy you analytical work flow, use *setup-pipe -f your-workflow-json*. For more options, check: *setup-pipe –help*

CHAPTER 1

General

1.1 General

This is an automatic setup tool for setting up the data analysis pipeline on the AWS. It provides utility functions to facilitate algorithm developer to build their analytical tools in docker container, published them for others to use and users to use those analytical tools to facilitate their analysis.

This tool is developed on Python 3.5.1. Using this on previous versions of python 3 or python 2 has not been tested.

Now yunpipe is on Pypi, and it supports pip. To install yunpipe, `pip install yunpipe`. After install yunpipe:

To submit an algorithm or bring your analyze tool, use `wrap -d`. For more options, check:

```
wrap --help
```

You will see the following options:

```
usage: wrap-script.py [-h] [-d] [-f FILES [FILES ...]] [-s] [-r REGISTRY]
                      [-u USER]

A tool to wrap your containers

optional arguments:
  -h, --help            show this help message and exit
  -d, --describe        use command line editor to describe your algorithm
  -f FILES [FILES ...], --files FILES [FILES ...]
                        List json files to describe your algorithms
  -s, --show            show described algorithm before generate new container
  -r REGISTRY, --registry REGISTRY
                        the registry where you want to upload the container,
                        default is Docker hub
  -u USER, --user USER  user name of docker hub account, default is wangyx2005
```

To run single algorithm or deploy you analytical work flow, use `setup-pipe -f your-workflow-json`. For more options, check:

```
setup-pipe --help
```

You will see the following options:

```
usage: setup-pipe-script.py [-h] [-uu] [-f FILES [FILES ...]]

A tool to set up your pipeline
```

```
optional arguments:
  -h, --help            show this help message and exit
  -uu, --use_user_credential
                        use user credential to run ecs task, we suggest using
                        a less privileged user for running ecs. For more
                        information, see our docs
  -f FILES [FILES ...], --files FILES [FILES ...]
                        json files to describe your work flow
```

To clean up your resources on the AWS after finishing your runs, use `clean-up`. If you are using a multiple step work flow, all your intermedia files stays for your future references. You do not need to do the clean up, as in our current setup, AWS will not charge you anything for the set up if you are not running anything except S3 storage for your results. Performing the clean up just to keep your AWS console neat.

CHAPTER 2

Quick Start

2.1 For Algorithm Developers

Using this automatic tool for algorithm developers is very easy. All you need to do is to have your algorithm containerized and run `wrap`

2.1.1 Using command line editor

yunpipe offers an command line editor to describe your algorithm. Use `wrap --describe --show --user your-docker-account` or `wrap -ds -u your-docker-account`. A interactive command shell will pop up to help you describe your algorithm. For detailed explanation of each entry, check detailed explanation of algorithm entries section.

After you finish describe your algorithm, a json file is showed to let you verify every thing is correct.

Once you confirm everything is correct, yunpipe will build a new image and upload to your docker hub account. Make sure you have already sign up and sign in you docker hub account. At the same time, a more detailed json file describe your algorithm will be generated and stored in ~/.yunpipe/algorithms folder under your algorithm name for algorithm users to use locally.

2.1.2 Use pre-prepared json file

yunpipe provides options using pre-prepared json file directly. use

```
wrap --files list-of-json-files --user your-docker-hub-account
```

yunpipe will build new images based on those files and upload to your docker hub account. At the same time, more detailed json files describe your algorithms will be generated and stored in ~/.yunpipe/algorithms folder under your algorithm name for algorithm users to use locally.

Prepare your json file

If you prefer, your algorithm json file directly. Here is an example of the json file describing the same algorithm in Using command line editor section.

```
{
    "container_name": "wangyx2005/change123",
    "system": "ubuntu",
    "input_file_path": "/",
```

```
    "output_file_path": "/",
    "executable_path": "/change123",
    "run_command": "sh /change123.sh $input $output",
    "name": "change123-s3",
    "instance_type": "",
    "memory":
    {
        "minimal": 50,
        "suggested": 128
    },
    "CPU": 1,
    "user_specified_environment_variables":
    [
        {
            "name": "TEST_ENV",
            "required": true
        }
    ],
    "port":
    [
        {
            "port": 9090,
            "protocol": "tcp"
        }
    ]
}
```

detailed explanation of algorithm entries:

- **container_name**: your containerized algorithm image. should be reachable from `docker pull`
- **system**: the system from which your image is built on. We currently support only ubuntu
- **run_command**: the command to run your algorithm. please substitute your input file with `$input`, output file/folder with `$output` and using the executable with the full path.
- **input_file_path**: the folder where input file should be
- **output_file_path**: the folder where output file should be
- **executable_path**: the full path of the executable
- **name**: A name which other algorithm user will refer this algorithm as. Need to be unique.
- **instance_type**: As algorithm developer, we believe you have a better understanding of your algorithm than anyone else. please suggest a instance type where this algorithm preferably running on on AWS.
- **memory**: the minimal and suggested memory requirement for running this algorithm container. You can omit minimal.
- **CPU**: the number of CPU used for using algorithm
- **user_specified_environment_variables**: this is the list of variable you allow other algorithm user to use, such as seed.
- **port**: the port number your algorithm exposed.

We will add a registry option to allow people upload images to other registries like amazon container registry.

Right now, sharing algorithm between users requires sharing corresponding json files in ~/.yunpipe/algorithms as well. We are working on to set up a database to enable developers to save the algorithm json file remotely to make it easy for users to use their algorithms

2.2 For Algorithm User

2.2.1 Prerequisition on AWS

Make sure you have full access to **Lambda**, **Simple Queue Services**, **S3**, **EC2 Container Services**, **CloudWatch** and **EC2**. If you are using aws container registry to host container image, **EC2ContainerRegistry** access will also be needed.

Here is a screen shot of the policies.

Additionally, make sure you have the following two roles in your IAM: **EC2ActionsAccess** role with **CloudWatchActionsEC2Access** policy and **ecsInstanceRole** role with **AmazonEC2ContainerServiceforEC2Role** policy.

The former one allows AWS, more specifically, cloud watch to stop/terminate ec2 instance on your behave. The later one allows ec2 instance register to ecs cluster, poll images for ecr and write logs to CloudWatch log. These two are will be automatically generated when you first use CloudWatch alarm and ecs if using the aws console. Here is a link about how to sent these up.

Lastly, create a role called **lambda_exec_role** with the following policy:

```
{
    "Statement": [
        {
            "Action": [
                "logs:*",
                "cloudwatch:*",
                "lambda:invokeFunction",
                "sqs:SendMessage",
                "ec2:Describe*",
                "ec2:StartInsatnces",
                "ec2:RunInstances",
                "iam:PassRole",
                "ecs:StartTask",
                "ecs:ListContainerInstances",
                "ecs:DescribeContainerInstances"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:logs:*:*:*",
                "arn:aws:lambda:*:*:*",
                "arn:aws:sqs:*:*:*",
                "arn:aws:ec2:*:*:*",
                "arn:aws:cloudwatch:*:*:*",
                "arn:aws:ecs:*:*:*",
                "*"
            ]
        }
    ],
    "Version": "2012-10-17"
}
```

This role will allow lambda function to send input file information to message queue, check resources to start ecs task, launch ec2 instance into ecs cluster if needed and register ec2 on cloudwatch for shutdown.

You can added manually on aws console or run the following command.

```
create-lambda-exec-role
```

We suggest creating a iam user with permissions only to access sqs and s3 for ecs task run.

2.2.2 Prepare your work flow json file

Right now you need to prepare your work flow json file directly. We will integrate common workflow language (cwl) to prepare work flow in the near future.

The following is a sample work flow json file. It puts input files in container-clouds-input s3 bucket and expects output files in container-clouds-output. It runs single algorithm "change123-s3".

```
{
    "input_s3_name": "container-clouds-input",
    "output_s3_name": "container-clouds-output",
    "key_pair": "wyx-cci",
    "account_id": "your-aws-account-id",
    "region":"us-east-1",
    "process":
    {
        "type": "single_run",
        "algorithms":
        [
            {
                "name": "change123-s3",
                "port": [],
                "variables":
                {
                    "TEST_ENV": "blah blah blah"
                }
            }
        ]
    }
}
```

yunpipe also supports sequential work flow. All you need to do is to change value **"single_run"** in **"process"."type"** entry to **"sequence_run"** and describe the list of algorithms in the sequence you want to run in the **"process"."algorithms"** field.

2.2.3 Run your work flow

Once a JSon file of your work flow description is generated, you can run your work flow on the cloud simply by running the following command.

```
setup-pipe -f work-flow-json
```

yunpipe will try to find your AWS login info from environment variables, then shared credential file ~/.aws folder, which is the aws sdk default location. If AWS login is not found, yunpipe will ask your aws credentials and configs then stored in ~/.aws folder.

We suggested not using **-uu** or **–use_user_credential** flag. If using such flag, ecs tasks will use user's credential to run and user's credential will be displayed as plaintext in ecs task definition, which might cause security problem. We suggest creating a iam user with permissions only to access sqs and s3 and using it's credential.

If you are not using **-uu** or **–use_user_credential** flag, yunpipe will check ~/.yunpipe/task file for credentials, if yunpipe failed to find it, it will ask you for such informations and stored in ~/.yunpipe/task file.

When the set up is finished, you will receive the following message. You can start uploading your input files and letting cloud handle the rest.

```
You can start upload files at some_S3_bucket_name
You will get your result at some_S3_bucket_name
```

Enjoy pipelining!!

CHAPTER 3

How it works

3.1 How it works

3.1.1 Overview

This automatic toolbox for image processing is current build on AWS. The main computational work is done on Amazon EC2 Container Service (Amazon ECS). In our case, each image processing job (processing one file/image using one algorithm) is launched into separate ECS tasks on the ECS cluster, which can be scaled perfectly.

Normally, an image processing work flow contains multiple algorithms and intermediate data transfer from one algorithm to another. This tool will also handle that for the users. All the intermediate data are stored in the S3 bucket for transfer and possible later use of users.

This tools also facilitates researchers to bring their own algorithms. To use your own algorithm, simply prepare it inside a Docker container, and push the image to Docker Hub or locally. Then, by using the container wrapper tool, you can describe all the details of your algorithm: including its command line options, user defined variables and required resources to run it.

The following is the Amazon services we use to construct this automation:

- **Amazon EC2 Container Service (ECS)**: In our case, this is the platform of all computational work.
- **AWS Lambda** : AWS Lambda is a compute service that runs code in response to events. We use this to trigger the actual image processing tasks.
- **AWS S3**:
- **AWS Simple Queue Service (SQS)**: SQS is a fast, reliable, scalable, fully managed message queuing service.
- **AWS CloudWatch**: Amazon CloudWatch is a monitoring service for AWS cloud resources and the applications you run on AWS. We use this to collect running logs, monitor and terminate idle EC2 instances.

3.1.2 Wrap

In the area of imaging processing, researchers use various different algorithms to processing images. Therefore it is crucial to allow users to bring their own algorithm into the work flow. This tool make it very easy to integrating customer own tools, simply installed inside Docker containers and described their behavior precisely using our command line editor or, if user prefer, via a JSon document that can easily imported.

With the information of the algorithm and the containerized algorithm, a new container with a running script is generated. This container is the actual container when the algorithm is used. The running script handles all the hassles to run the processing job: retrieving input information from SQS, downloading input object from S3 bucket, putting input

object to correct location, running the processing job, uploading the result object to designed location and managing logs throughout the whole process.

3.1.3 Run

To achieve high scalability, we disassemble complex work flow into two part: single algorithm processing and information transfers between algorithms.

For each single algorithm processing, a microservice is built around it on AWS:

- a lambda function is used to monitor the input event. It can be triggered by a file upload to the input file s3 bucket event or invoked by called from other lambda functions. Once it is triggered, it send the information of the input file to a message queue and start a ECS task to start algorithm processing.
- a message queue, we use Amazon Simple Queue Service (SQS) in the current version, is used to hold the information about the input files.
- a ecs task is used to perform the actual algorithm processing on the cloud. It retrieve the input file information from the message queue, download from sources and process the files. Once it is finished, the resulting file is upload to another s3 bucket and the ecs task is terminated.

Yunpipe also supports sequential analysis pipeline. The intermediate data is also stored in s3 bucket, which is also used as the launch event for the next analysis step.

As each task is run on specific type of EC2 instance, the default ECS scheduler does not fit our need. In our current version, lambda function is also in charge of checking resources to start ecs task, launch ec2 instance into ecs cluster if needed and register ec2 on cloudwatch for shutdown. This is not very efficient. In our future version, a customer scheduler will be added the substitute that part in lambda function.

CHAPTER 4

Demos

CHAPTER 5

API References

5.1 yunpipe

PACKAGE CONTENTS: pipeline (package) scripts (package) utils wrapper (package)

5.1.1 yunpipe.wrapper

PACKAGE CONTENTS container_wrapper

yunpipe.wrapper.container_wrapper

describe_algorithm(): command line editor of the detailed information of algorithm container.

```
:rtype: json
```

generate_all(alg, args): generate dockerfile, build new image, upload to registry and generate detailed information of the new image :para alg: algorthm information user provided :type: json :para agrs: command line argument from script.wrap. args.user and args.registry

```
:type: argparser object
```

generate_dockerfile(system_name, container_name): generate the dockerfile content. Based on the system which user's prebuild image, generate dockerfile including adding run enviroment of runscript.py and add runscript. :para system_name: the system name in which the docker image is built on :tpye: string :para container_name: user's algorithm container :tpye: string :return: the dockerfile content. :rtype: string

generate_image(name, folder_path, args): build new docker image and upload. giver new docker image name and dockerfile, build new image, tagged with user account and pushed to desired registry. Default registry is docker hub, will support other registry soon. :para name: new docker image name. Without tag and registry. :type: string :para folder_path: the path to tmp folder where stores dockerfiles. path is ~/.cloud_pipe/tmp/name :typr: string :para args: command line arguments passed in from scripts.wrap, currently only useful entry is user, will using registry soon :type: argparser object :rtpye: docker image with repo name generate_image_info(alg_info, container_name) generate wrapped image information for ecs task :para alg_info: algorthm information user provided :type: json :para container_name: access name of the wrapped container :type string rtype: json

generate_runscript(input_path, output_path, name, command): generate runscript that fetch information from sqs, handling download/upload file and run script. :para input_path: input folder :type: string :para output_path: output folder :type: string :para name: new docker image name :type: string :para command: run com-

mand of user's algorithm script :type: string :return: the runscript for runscript.py. Include fetching information, download / upload file and run script. :rtype: string

get_instance_type(alg_info): Based on the algorithm developer provided information, choose an apporperate ec2 instance_type :para alg_info: a json object contains necessory information about algorithm :type: json :rtype: sting of ec2 instance type show_dockerfile(system_name, container_name) show_runscript(input_path, output_path, name, command) wrapper(alg_info) automatic generate dockerfile according to the information user provided. :para alg_info: a json object contains necessory information about algorithm :type: json

5.1.2 yunpipe.pipeline

5.1.3 yunpipe.utils

Utility functions for yunpipe

create_folder(folder): create folder if not existed

get_full_path(path): convert a relative path to absolute path.

get_int(message, default): transfer user input to int numbers. Continue asking unless valid input. If user omit the input and default is set to non-None, get default number instand.

```
:para message: input message should to user
:type: string

:para default: default value

:rtype: int
```

get_true_or_false(message, default=False): transfer user input Y/n into True or False

```
:para message: input message should to user
:type: string

:para default: default value

:rtype: boolean
```

5.1.4 yunpipe.script

command line scripts for yunpipe.

yunpipe.script.wrap

command line script for wrapping up user algorithm

yunpipe.script.setup_pipe

command line script for setting up pipeline on the AWS

CHAPTER 6

Roadmap

6.1 Road Maps

6.1.1 Recent work

- one to all / all to one algorithm setup
- ecs customer scheduler to substitute lambda part
- add support to upload container images to places other than Docker hub
- add user being able to override instance_type
- work flow language for processing work flow
- replace sqs with overrides environment variables, namely input s3, bucket and keys, mainly change lambda part

6.1.2 Future work

- GUI for generating workflow

CHAPTER 7

Indices and tables

- genindex
- modindex
- search