AWS saving

Release 1.1.3

Feb 22, 2022

Contents:

This package contains the classes for managing the saving of the AWS services.

CHAPTER 1

Getting started

AWS saving package is implemented for deploying a lambda that you can invoke for stopping, deleting or starting each RDS, EC2, bucket, Sagemaker domain and Stack instance.

The goal is to implement this package for each AWS service can stop, delete and start its instances.

It is part of the educational repositories to learn how to write stardard code and common uses of the TDD.

1.1 Prerequisites

You can use Serverless framework for deploying the lambda function: if you want to use the guide below, you have to install Serverless framework before

```
npm install -g serverless
```

If you want to use another AWS tool, you can see the repository aws-tool-comparison before to implement your version.

1.2 Installation

The package is not self-consistent. So you have to download the package by github and to install the requirements before to deploy on AWS:

```
git clone https://github.com/bilardi/aws-saving
cd aws-saving/
pip3 install --upgrade -r requirements.txt
npm install
export AWS_PROFILE=your-account
SLS_DEBUG=* sls deploy --stage production
```

Or if you want to use this package into your code, you can install by python3-pip:

```
pip3 install aws_saving
python3
>>> import aws_saving.saving as Saving
>>> help(Saving)
```

Read the documentation on readthedocs for

- Usage
- Development

1.3 Change Log

See CHANGELOG.md for details.

1.4 License

This package is released under the MIT license. See LICENSE for details.

CHAPTER 2

Usage

The lambda reads each tag of each RDS, EC2, Bucket and Stack instance, and when it finds the tag named

- **Saving**, if the value is `Enabled`, it checks the following points
- **Stop**, if the instance status, hour and crontab value matches, it will stop the instance
- **Delete**, if the instance status, hour and crontab value matches, it will delete the instance
- **Start**, if the instance status, hour and crontab value matches, it will start the instance

You can define 4 tags types on your instace:

- Key=Saving,Value=Enabled
- Key=Stop,Value='`0 18 . . .`'
- Key=Delete,Value='`0 18 . . .`'
- Key=Start,Value='`0 8 . . .`'

and

- if you have added deleted protection on your object, you can force the deletion by **force** property
- if you need to split the load by one lambda per service, you can uncommented the lambda functions and **invoke_lambda** property
- you can decide which AWS services manage by **services_name** property
- the default **timezone** is `Etc/GMT`

Search the **bold word** in the serverless.yaml file, or see below for an example.

2.1 Example

You need an infrastructure with

- a bucket for loading your data

- a RDS instance where loading the bucket data
- an EC2 instance where installing your software
- a RDS cluster where loading the ETL data
- a domain with an user of SageMaker Studio
- a lambda for managing the saving costs

This system is not necessary 24H but only 2 hours in the morning from 8:00 AM and the developers work until 6:00 PM

- the EC2 and RDS instances in production can be started at 7:30 AM and stopped at 10:30 AM
- the stacks in staging (*) can be deleted at 6:00 PM
- the apps created by user of SageMaker Studio can be deleted at 6:00PM

The tags that you have to add to your objects are

- Key=Saving,Value=Enabled for all stacks, EC2 and RDS instances
- Key=Start,Value='`30 7 . . .`' for EC2 and RDS instances in production
- Key=Stop,Value='`30 10 . . .`' for EC2 and RDS instances in production
- Key=Delete,Value='`0 18 . . .`' for all stacks in staging (*)
- Key=Stop,Value='`0 18 . . .`' for all apps in that domain of SageMaker Studio

(*) or any other non-production environment that you can re-deploy as needed.

CHAPTER 3

Development

The environments for development can be many: you can organize a **CI/CD system** with your favorite software. The primary features of your CI/CD are: having a **complete environment for**

- **development** for each developer, to implement something and for running unit tests
- **staging** for running unit and integration tests, to check everything before release
- **production**

If you want to use AWS CDK and AWS CodePipeline, you can see these repositories before to implement your version

- aws-simple-pipeline for using a library ready
- aws-tool-comparison for seeing its implementation

3.1 Run tests

```
cd aws-saving/
pip3 install --upgrade -r requirements.txt
python3 -m unittest discover -v
```

3.2 Get services with start/stop features

The goal is to implement this package for each AWS service can start and stop its instances.

```
cd aws-saving/
bash first.skimming.sh
ls tmp/
```

3.3 Deploy on AWS

```
cd aws-saving/
export AWS_PROFILE=your-account
SLS_DEBUG=* sls deploy --stage development
```

3.4 Remove on AWS

The stack has the tags necessary for being deleted itself by the lambda deployed in production. Or you can run the integration tests passing the name of the STAGE deployed (ie, staging):

```
cd aws-saving/
export AWS_PROFILE=your-account
# integration test for removing the stack you have created
STAGE=staging python3 -m unittest discover -v -p integration_test_only_cloudformation.
↪py
```

For testing all services, you already have to deploy the stack before run the integration tests below. There is something missing: with the commands below, the stack deletion reaches the StackStatus value `DELETE_FAILED`. So you will have to skip manually RDS resources for deleting the stack (see **TODO** in cloudformation.py).

```
cd aws-saving/
export AWS_PROFILE=your-account
# integration test for removing before s3, ec2, rds instances and then it tries to␣
↪remove the stack you have created
STAGE=staging python3 -m unittest discover -v -p integration_test_all_services.py
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search