Ardy Documentation

Release 0.0.4

avara1986

Nov 05, 2018

Contents

Ardy is a toolkit to work with AWS Lambda implementing Continuous Integration. AWS Lambda is a serverless compute service that runs your code in response to events and automatically manages the underlying compute resources for you. Alas, AWS Lambda has a very bad GUI interface, especially if you work with teams and releases. You can't easily see at a glance the active triggers you have, the resources of your AWS Lambda or have a version control.

With *Ardy* you can manage your AWS Lambda with a JSON config file stored in your VCS.

> **Warning:** If you want to work with AWS Lambda, it's recommended to read about it. *Ardy* helps and supports you to manage environments but doesn't performs "The black magic" for you. You can learn more about AWS Lambda in *this page*

CHAPTER 1

Content

1.1 Installation

Install the latest Ardy release via pip:

```
pip install ardy
```

You may also install a specific version:

```
pip install ardy==0.0.1
```

1.1.1 Credentials

Before you can deploy an application, be sure you have credentials configured. If you have previously configured your machine to run boto3 (the AWS SDK for Python) or the AWS CLI then you can skip this section.

If this is your first time configuring credentials for AWS you can follow these steps to quickly get started:

```
$ mkdir ~/.aws
$ cat >> ~/.aws/credentials
[default]
aws_access_key_id=YOUR_ACCESS_KEY_HERE
aws_secret_access_key=YOUR_SECRET_ACCESS_KEY
region=YOUR_REGION (such as us-west-2, us-west-1, etc)
```

1.2 Quickstart

Before start working with Ardy or AWS Lambda, if you don't know anything about AWS Lambda I recommend you the *AWS documentation*.

Suppose you have a project with multiple lambas with the following structure:

```
your-project
   ├ lambda1
   │ └ my_handler.py
   ├ lambda2
   │ └ main.py
   ├ lambda3
   │ └ main.py
```

To start working with *ardy*, the first step is to create the configuration file with JSON format. The default file name is **conf.jon**:

```
{
  "version": 1,
  "aws_credentials":{
    "aws_access_key_id": "YOUR-AWS-KEY-ID",
    "aws_secret_access_key": "YOUR-AWS-SECRET-KEY",
    "region": "eu-west-1"
  },
  "deploy": {
    "deploy_method": "FILE"
  },
  "Role": "arn:aws:iam::01234567890:role/service-role/LambdaTest",
  "Runtime": "python3.6",
  "lambdas": [
    {
      "FunctionName": "MyLambda",
      "Handler": "your-project.lambda1.my_handler.my_handler"
    },
    {
      "FunctionName": "MyOtherLambda",
      "Handler": "your-project.lambda2.main.main.my_handler"
    },
    {
      "FunctionName": "MyOtherOtherLambda",
      "Handler": "your-project.lambda3.main.main.my_handler"
    }
  ]
}
```

(See *more details about the configuration file*)

Now, you will have this structure in your project:

```
your-project
   ├ lambda1
   │ └ my_handler.py
   ├ lambda2
   │ └ main.py
   ├ lambda3
   │ └ main.py
   └ config.json
```

If you want to deploy your AWS Lambdas, you just must run this command in a shell:

```
ardy deploy
```

Or if you want to deploy a specific list of functions, you can deploy the AWS Lambdas with:

```
ardy deploy MyLambda MyOtherLambda
```

See *more details about how to deploy*

1.3 Configuration

All the behavior of *Ardy* toolkit is managed from the configuration file. This file is in **JSON format**.

Tip: Before start, read the AWS Lambda Best practices

1.3.1 Base configuration

- **version:** [REQUIRED] The version of JSON format of *ardy* configuration file. Default 1.
- **aws_credentials:** The best practices are to store your credentials in your ~/.aws/credentials file , but, if it is mandatory, you could set your credentials in the configuration file.
  - **aws_access_key_id:**
  - **aws_secret_access_key:**
  - **region:** [REQUIRED]

1.3.2 Global configuration

You can set a global configuration for all your AWS Lambdas. Global configuration is like the keys used to deploy with API AWS Lambda

- **Role:** [REQUIRED] The Amazon Resource Name (ARN) of the IAM role that Lambda assumes when it executes your function to access any other Amazon Web Services (AWS) resources. For more information, see AWS Lambda: How it Works.
- **MemorySize:** The amount of memory, in MB, your Lambda function is given
- **Runtime:** The runtime environment for the Lambda function you are uploading
- **Timeout:** The function execution time at which Lambda should terminate the function
- **Publish:** This boolean parameter can be used to request AWS Lambda to create the Lambda function and publish a version as an atomic operation
- **Tags:** The list of tags (key-value pairs) assigned to the new function
- **VpcConfig:** If your Lambda function accesses resources in a VPC, you provide this parameter identifying the list of security group IDs and subnet IDs

1.3.3 Deploy configuration

- **deploy:**
  - **deploy_method:** [REQUIRED] String. Must be "*FILE*" or "*S3*". If deploy_method is *S3*, when *ardy* generate the artefact, it will be uploaded to S3. In that case, you must set *deploy_bucket*
  - **deploy_bucket:** String. The S3 bucket if *deploy_method* is S3

- **version_control:** Not implemented at this moment
- **deploy_file:** String. If you set a value to this key, *Ardy* doesn't build an artefact, instead, Lambas will be deployed with this file code.
- **deploy_environments:** List of strings. A map of environments. Each environment represents one possible deployment target. You could set a list of environments to filter. Each environment has a configuration defined for each lambda (see in details below)
- **use_alias:** [REQUIRED] Bool. This key change the behavior of deploy_environments. If it's False and *deploy_environments* is defined, a lambda will be deployed for each environment. If *use_alias* is True, each lambda will be deployed and an alias will be created for each environment. The alias will be a pointer to a specific Lambda function version

1.3.4 AWS Lambda configuration

You can set the same keys as in Global Configuration, and it will be overridden. See more here

- **lambdas: [REQUIRED] List of dictionaries. You can define the key value pair defined below for each AWS Lambda you v**
  - **FunctionName:** [REQUIRED] String. The name you want to assign to the function you are uploading
  - **Handler:** [REQUIRED] String. The function within your code that Lambda calls to start the execution
  - **Description:** A short, user-defined function description
  - **deploy_environments:** If *use_alias* is False, You can set the same keys as in Global Configuration and Lambda configuration, and it will be overridden. If use_alias is True, one AWS Lambda is deployed and Ardy create an alias pointed to the Lambda Version. Learn *more details about the alias*.
  - **triggers:** *more details about events and triggers*.

Tip: Learn more about Versions and Alias here

1.3.5 Examples

1.3.5.1 Basic S3

```
{
  "version": 1,
  "aws_credentials":{
    "region": "eu-west-1"
  },
  "deploy": {
    "deploy_method": "S3",
    "deploy_bucket": "lambdartefacts",
  },
  "Role": "arn:aws:iam::01234567890:role/service-role/LambdaTest",
  "Runtime": "python3.6",
  "lambdas": [
    {
      "FunctionName": "MyLambda",
```

(continues on next page)

(continued from previous page)

```
      "Handler": "your-project.lambda1.my_handler.my_handler"
    }
  ]
}
```

1.3.5.2 Basic FILE

```
{
  "version": 1,
  "aws_credentials":{
    "region": "eu-west-1"
  },
  "deploy": {
    "deploy_method": "FILE"
  },
  "Role": "arn:aws:iam::01234567890:role/service-role/LambdaTest",
  "Runtime": "python3.6",
  "lambdas": [
    {
      "FunctionName": "MyLambda",
      "Handler": "your-project.lambda1.my_handler.my_handler"
    }
  ]
}
```

1.3.5.3 Multiple Environments

```
{
  "version": 1,
  "aws_credentials":{
    "region": "eu-west-1"
  },
  "deploy": {
    "deploy_method": "FILE",
    "deploy_environments": [
          "dev",
          "pre",
          "pro"
    ]
  },
  "Role": "arn:aws:iam::01234567890:role/service-role/LambdaTest",
  "Runtime": "python3.6",
  "Timeout": 30,
  "lambdas": [
    {
      "FunctionName": "MyLambda",
      "Handler": "your-project.lambda1.my_handler.my_handler",
      "Timeout": 45,
      "deploy_environments": {
        "dev": {
          "FunctionName": "MyLambda_dev",
          "Timeout": 10
        },
```

(continues on next page)

(continued from previous page)

```
        "pre": {
          "FunctionName": "MyLambda_pre"
        },
        "pro": {
          "FunctionName": "MyLambda_pro"
          "Timeout": 300
        }
      }
    }
  ]
}
```

1.3.5.4 Multiple Environments and multiple VPCS

```
{
  "version": 1,
  "aws_credentials":{
    "region": "eu-west-1"
  },
  "deploy": {
    "deploy_method": "FILE",
    "deploy_environments": [
          "dev",
          "pre",
          "pro"
    ]
  },
  "VpcConfig": {
    "SubnetIds": [
      "subnet-123",
      "subnet-456"
    ],
    "SecurityGroupIds": [
      "sg-789"
    ]
  },
  "Role": "arn:aws:iam::01234567890:role/service-role/LambdaTest",
  "Runtime": "python3.6",
  "lambdas": [
    {
      "FunctionName": "MyLambda",
      "Handler": "your-project.lambda1.my_handler.my_handler",
      "deploy_environments": {
        "dev": {
          "FunctionName": "MyLambda_dev",
        },
        "pre": {
          "FunctionName": "MyLambda_pre"
        },
        "pro": {
          "FunctionName": "MyLambda_pro"
          "Timeout": 300,
          "VpcConfig": {
            "SubnetIds": [
              "subnet-789"
```

(continues on next page)

(continued from previous page)

```
            ],
            "SecurityGroupIds": [
              "sg-123"
            ]
          },
        }
      }
    }
  ]
}
```

1.3.6 Advanced configuration

1.3.6.1 Alias and Versions

Environments

To use Alias with Ardy, you must set True "use_alias". ¿What is the difference?

If not use alias, when you deploy, for example, this configuration:

```
{
  "FunctionName": "LambdaExample1",
  "Handler": "myexamplelambdaproject.lambda1.main.my_handler",
  "deploy_environments": {
    "dev": {"FunctionName": "LambdaExample1_dev"},
    "pre": {"FunctionName": "LambdaExample1_pre",},
    "pro": {"FunctionName": "LambdaExample1_pro",}
  }
}
```

When you run this 3 commands:

```
ardy deploy LambdaExample1 dev
ardy deploy LambdaExample1 pre
ardy deploy LambdaExample1 pro
```

Ardy create a AWS Lambda for each environment ("LambdaExample1_dev", "LambdaExample1_pre", "LambdaExample1_pro"). Each AWS Lambda could has a specific configuration (I.E: Set different VPCS for each environment, different runtime...)

```
../_static/aws_environments.png
```

Alias

If use alias, with this configuration:

```
{
  "FunctionName": "LambdaExample1",
  "Handler": "myexamplelambdaproject.lambda1.main.my_handler",
  "deploy_environments": {
    "dev": {"Description": "AWS lambda LambdaExample1 DEV environment"},
    "pre": {},
    "pro": {"Description": "AWS lambda LambdaExample1 PRO environment"}
  }
}
```

When you run this 3 commands:

```
ardy deploy LambdaExample1 dev
ardy deploy LambdaExample1 pro
```

Ardy create just one AWS Lambda "LambdaExample1", increment its version and creates 2 alias pointed to diferents versions of the lambda.

Examples

```
{
  "version": 1,
  "aws_credentials":{
    "region": "eu-west-1"
  },
  "deploy": {
    "deploy_method": "S3",
    "deploy_bucket": "lambdartefacts",
    "deploy_environments": [
      "dev",
      "pre",
      "pro"
    ],
    "use_alias": true
```

(continues on next page)

(continued from previous page)

```
  },
  "Role": "arn:aws:iam::01234567890:role/service-role/LambdaTest",
  "Runtime": "python3.6",
  "lambdas": [
    {
      "FunctionName": "LambdaExample1",
      "Handler": "myexamplelambdaproject.lambda1.main.my_handler",
      "deploy_environments": {
        "dev": {},
        "pre": {},
        "pro": {}
      }
    },
    {
      "FunctionName": "LambdaExample2",
      "Handler": "myexamplelambdaproject.lambda2.main.my_handler",
      "deploy_environments": {
        "dev": {},
        "pre": {},
        "pro": {}
      }
    },
    {
      "FunctionName": "LambdaExample3",
      "Handler": "myexamplelambdaproject.lambda3.main.my_handler",
      "deploy_environments": {
        "dev": {},
        "pre": {},
        "pro": {}
      }
    }
  ]
}
```

1.3.6.2 Triggers

AWS Lambda support AWS services that you can configure as event sources:

- Amazon S3
- Amazon DynamoDB
- Amazon Kinesis Streams
- Amazon Simple Notification Service
- Amazon Simple Email Service
- Amazon Cognito
- AWS CloudFormation
- Amazon CloudWatch Logs
- Amazon CloudWatch Events
- AWS CodeCommit
- Scheduled Events (powered by Amazon CloudWatch Events)

- AWS Config
- Amazon Alexa
- Amazon Lex
- Amazon API Gateway
- Other Event Sources: Invoking a Lambda Function On Demand
- Sample Events Published by Event Sources

Ardy actually support integration with S3, SNS and loudWatch Events. The worst integration of AWS Lambda is the trigger configuration. You can't see all triggers as a glance in your lambdas configuration and, if you use whe AWS Cli, each trigger is configured outside AWS Lambda, it's mean, a trigger of S3 is a Event of a S3 bucket; a trigger of SNS is a subscription of a SNS Topic.

Tip: Learn more about Versions and Triggers here

Examples

```
{
  "version": 1,
  "aws_credentials":{
    "region": "eu-west-1"
  },
  "deploy": {
    "deploy_method": "S3",
    "deploy_bucket": "lambdartefacts",
    "deploy_environments": [
      "dev",
      "pre",
      "pro"
    ],
    "use_alias": false
  },
  "Role": "arn:aws:iam::01234567890:role/service-role/LambdaTest",
  "Runtime": "python3.6",
  "lambdas": [
    {
      "FunctionName": "LambdaExample_S3_1",
      "Handler": "myexamplelambdaproject.lambda1.main.my_handler",
      "Description": "string1",
      "triggers": {
        "s3": [
          {
            "Id": "trigger_from_LambdaExample_S3_7",
            "bucket_name": "lambdatriggers",
            "Events": [
              "s3:ObjectCreated:*"
            ],
            "Filter": {
              "Key": {
                "FilterRules": [
                  {
                    "Name": "Prefix",
```

(continues on next page)

(continued from previous page)

```
                    "Value": "test_"
                  },
                  {
                    "Name": "Suffix",
                    "Value": ""
                  }
                ]
              }
            }
          }
        ]
      }
    },
    {
      "FunctionName": "LambdaExample_SNS_2",
      "Handler": "myexamplelambdaproject.lambda2.main.my_handler",
      "triggers": {
        "sns": [
          {
            "TopicArn": "arn:aws:sns:eu-west-1:123456789012:TestLambdas"
          }
        ]
      }
    },
    {
      "FunctionName": "LambdaExample_CWE_3",
      "Handler": "myexamplelambdaproject.lambda3.main.my_handler",
      "triggers": {
        "cloudwatchevent": [
          {
            "Name": "Raise1minute",
            "ScheduleExpression": "cron(* * * * ? *)",
            "State": "DISABLED",
            "Description": "Run every 1 minute"
          }
        ]
      }
    }
  ]
}
```

1.4 Code Examples

This section provides code examples that demonstrate common Amazon Web Services scenarios using Ardy.

- Simple project
- Project with Alias and versions
- Project with Triggers
- Project with multiple environments

1.5 Deploy

To deploy your project, you can create a script or add a command in a shell (See *more details about the command line*)

```
from ardy.core.deploy import Deploy

if __name__ == '__main__':
    deploy = Deploy(path=os.path.dirname(os.path.abspath(__file__)))

    deploy.run("myexamplelambdaproject")
```

1.6 Command Line

Ardy's command line, by default, search a *config.json* at the same path that the command is running. But you can set a different path with the argument *-p*.

optional arguments:

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

-f CONFFILE, --conffile CONFFILE Name to the project config file

-p PROJECT, --project PROJECT Project path

Commands:

- *deploy:* Upload functions to AWS Lambda
- *invoke:* Invoke functions from AWS Lambda
- *build:* Create an artefact

If you want to deploy *all your AWS Lambdas* defined in your *config.json* file

```
ardy deploy
```

Or if you want to deploy a specific list of functions, you can deploy the AWS Lambdas with:

```
ardy deploy MyLambda MyOtherLambda
```

You can deploy only an environment:

```
ardy deploy MyLambda MyOtherLambda dev
ardy deploy MyLambda MyOtherLambda pre
ardy deploy MyLambda pro
```

1.6.1 Example Scenario

You have a project with this structure:

```
main-project
├ lambda-subproject
│  ├ lambda1
│  │  └ my_handler.py
│  ├ lambda2
│  │  └ main.py
```

(continues on next page)

(continued from previous page)

```
│  ├ lambda3
│  │ └ main.py
└ config.json
```

The path of your project is */var/www/main-project/lambda-subproject* and a config.json like that:

```
{
  "version": 1,
  "aws_credentials":{
    "region": "eu-west-1"
  },
  "deploy": {
    "deploy_method": "FILE"
  },
  "Role": "arn:aws:iam::01234567890:role/service-role/LambdaTest",
  "Runtime": "python3.6",
  "lambdas": [
    {
      "FunctionName": "MyLambda",
      "Handler": "your-project.lambda1.my_handler.my_handler"
    },
    {
      "FunctionName": "MyOtherLambda",
      "Handler": "your-project.lambda2.main.main.my_handler"
    }
  ]
}
```

You're in */var/www/main-project/*, and want to deploy *MyLambda*:

```
ardy -p lambda-subproject deploy MyLambda
```

But, if you're in */home/Caerbannog_user/*, and want to deploy *MyLambda*:

```
ardy -f /var/www/main-project/config.json -p /var/www/main-project/lambda-subproject␣
↪deploy MyLambda
```

1.7 Code Examples

To start working with AWS Lambda I recommend reading these pages or doing some labs:

1.7.1 Documentation

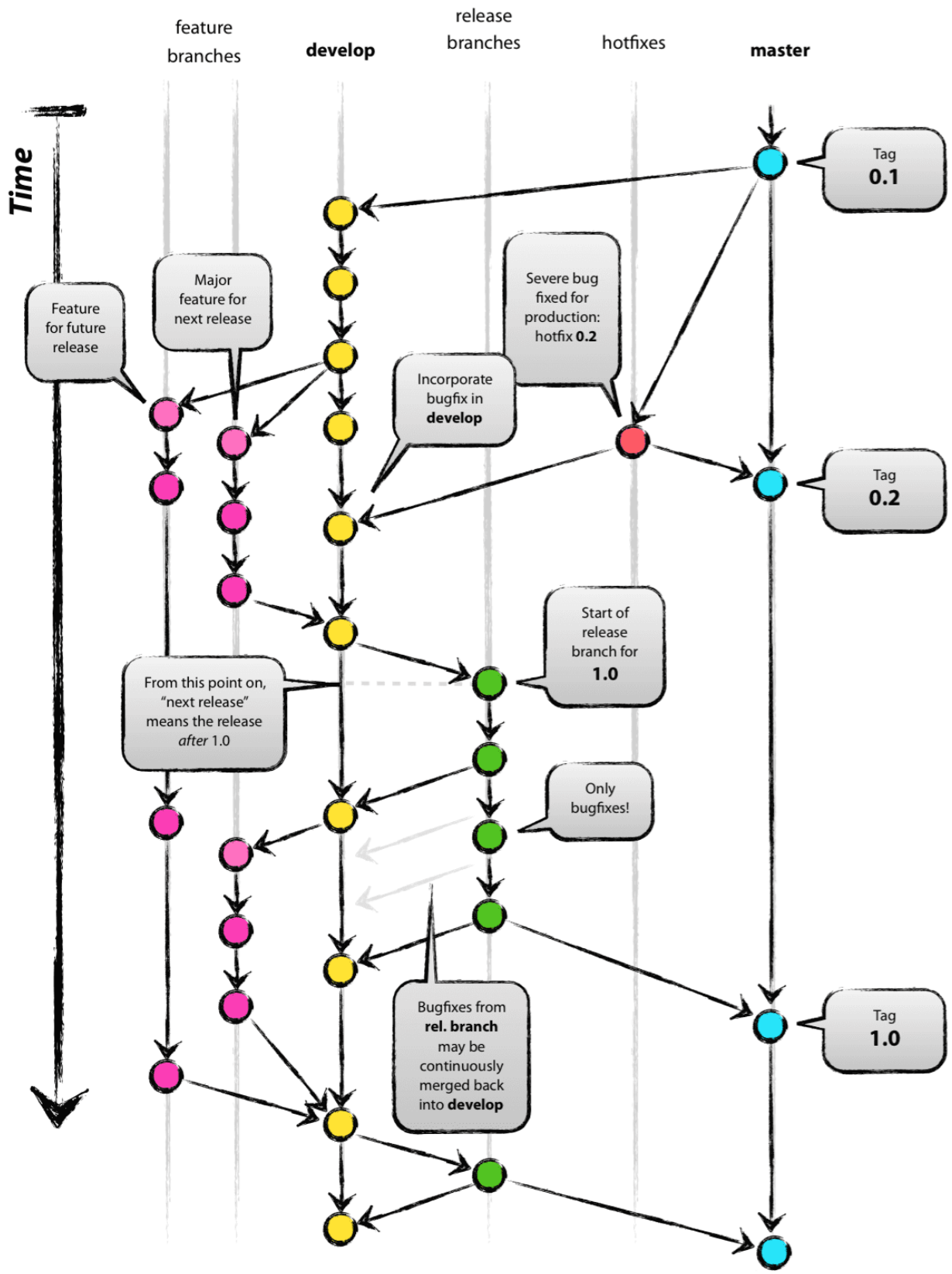
- AWS Lambda official doc
- AWS Lambda Blog
- AWS Lambda Forum
- Project with Triggers

1.7.2 Labs

- Qwiklabs

1.8 How to contrib

This project is built with Git Flow. If you want to commit some code, please use this pattern:


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

1.9 Core References

1.9.1 Subpackages

1.9.1.1 Ardy Build package

class ardy.core.build.build.**Build**(**args*, ***kwargs*)
Bases: ardy.config.ConfigMixin

copytree(*src*, *dst*, *symlinks=False*, *ignore=None*)

create_artefact(*src*, *dest*, *filename*)

get_src_path()

mkdir(*path*)

pip_install_to_target(*path*, *requirements=u''*, *local_package=None*)
For a given active virtualenv, gather all installed pip packages then copy (re-install) them to the path provided. :param str path:

Path to copy installed pip packages to.

Parameters

- **requirements** (*str*) – If set, only the packages in the requirements.txt file are installed. The requirements.txt file needs to be in the same directory as the project which shall be deployed. Defaults to false and installs all pacakges found via pip freeze if not set.
- **local_package** (*str*) – The path to a local package with should be included in the deploy as well (and/or is not available on PyPi)

static read(*path*, *loader=None*)

run(*src_folder*, *requirements=u'requirements.txt'*, *local_package=None*)
Builds the file bundle. :param str src:

The path to your Lambda ready project (folder must contain a valid config.yaml and handler module (e.g.: service.py).

Parameters **local_package** (*str*) – The path to a local package with should be included in the deploy as well (and/or is not available on PyPi)

set_src_path(*src_folder*)

src_path = u''

static timestamp(*fmt=u'%Y-%m-%d-%H%M%S'*)

1.9.1.2 Ardy Cmd package

class ardy.core.cmd.main.**Command**(**args*, ***kwargs*)
Bases: object

args = []

config = None

exit_ok(*msg=u''*)

exit_with_error(*msg=u”*)

init_config(*arguments*)

parse_commandline()

parser = None

parser_base

static print_error(*msg=u”*)

static print_ok(*msg=u”*)

1.9.1.3 Ardy Deploy package

class ardy.core.deploy.deploy.**Deploy**(**args*, ***kwargs*)

Bases: ardy.config.ConfigMixin

build = None

build_artefact(*src_project=None*)

Run deploy the lambdas defined in our project. Steps: * Build Artefact * Read file or deploy to S3. It's defined in config[“deploy”][“deploy_method”]

Parameters **src_project** – str. Name of the folder or path of the project where our code lives

Returns bool

deploy()

Upload code to AWS Lambda. To use this method, first, must set the zip file with code with *self.set_artefact(code=code)*. Check all lambdas in our config file or the functions passed in command line and exist in our config file. If the function is upload correctly, update/create versions, alias and triggers

Returns True

static is_client_result_ok(*result*)

lambdas_to_deploy = []

remote_create_lambada(***kwargs*)

remote_get_lambda(***kwargs*)

remote_list_lambdas()

remote_publish_version(***kwargs*)

remote_update_alias(***kwargs*)

remote_update_code_lambada(***kwargs*)

remote_update_conf_lambada(***kwargs*)

run(*src_project=None*, *path_to_zip_file=None*)

Run deploy the lambdas defined in our project. Steps: * Build Artefact * Read file or deploy to S3. It's defined in config[“deploy”][“deploy_method”] * Reload conf with deploy changes * check lambda if exist

- Create Lambda
- Update Lambda

Parameters

- **src_project** – str. Name of the folder or path of the project where our code lives
- **path_to_zip_file** – str.

Returns bool

set_artefact(*code*)

Parameters **code** – dic. it must be with this shape

{'ZipFile': } or {'S3Bucket': deploy_bucket, 'S3Key': s3_keyfile, } :return:

set_artefact_path(*path_to_zip_file*)
Set the route to the local file to deploy :param path_to_zip_file: :return:

1.9.1.4 Ardy Invoke package

class ardy.core.invoke.invoke.**Invoke**(**args*, ***kwargs*)
Bases: ardy.config.ConfigMixin

import_function(*name*)

run(*lambda_name*, *local=True*)

1.9.1.5 Ardy Triggers package

ardy.core.triggers.**get_trigger**(*trigger*, *lambda_conf*, *lambda_function_arn*)

Subpackages

Ardy Triggers Cloudwatch Event package

class ardy.core.triggers.cloudwatchevent.**Driver**(**args*, ***kwargs*)
Bases: ardy.core.triggers.driver.Trigger

get_awsservice_method = u'get_cloudwatchevent_client'

put(**args*, ***kwargs*)

trigget_type = u'cloudwatchevent'

Ardy Triggers S3 package

class ardy.core.triggers.s3.**Driver**(**args*, ***kwargs*)
Bases: ardy.core.triggers.driver.Trigger

get_awsservice_method = u'get_s3_resource'

put(**args*, ***kwargs*)

trigget_type = u's3'

Ardy Triggers SNS package

class ardy.core.triggers.sns.**Driver**(**args*, ***kwargs*)

Bases: ardy.core.triggers.driver.Trigger

get_awsservice_method = u'get_sns_client'

put(**args*, ***kwargs*)

trigget_type = u'sns'

1.9.2 Module contents

Python Module Index

a

Index

S

T