kappa Documentation

Release 0.4.0

Mitch Garnaat

March 13, 2016

Contents

Contents:

CHAPTER 1

Why kappa?

You can do everything kappa does by using the AWS Management Console so why use kappa? Basically, because using GUI interfaces to drive your production environment is a really bad idea. You can't really automate GUI interfaces, you can't debug GUI interfaces, and you can't easily share techniques and best practices with a GUI.

The goal of kappa is to put everything about your AWS Lambda function into files on a filesystem which can be easily versioned and shared. Once your files are in git, people on your team can create pull requests to merge new changes in and those pull requests can be reviewed, commented on, and eventually approved. This is a tried and true approach that has worked for more traditional deployment methodologies and will also work for AWS Lambda.

CHAPTER 2

The Config File

The config file is at the heart of kappa. It is what describes your functions and drives your deployments. This section provides a reference for all of the elements of the kappa config file.

2.1 Example

Here is an example config file showing all possible sections.

```
---
name: kappa-python-sample
environments:
  env1:
    profile: profile1
    region: us-west-2
    policy:
      resources:
        - arn: arn:aws:dynamodb:us-west-2:123456789012:table/foo
          actions:
          - "*"
        - arn: arn:aws:logs:*:*:*
          actions:
          - "*"
   event_sources:
     -
       arn: arn:aws:kinesis:us-west-2:123456789012:stream/foo
       starting_position: LATEST
       batch_size: 100
  env2:
    profile: profile2
    region: us-west-2
    policy_resources:
      - arn: arn:aws:dynamodb:us-west-2:234567890123:table/foo
        actions:
        - "*"
      - arn: arn:aws:logs:*:*:*
        actions:
        - "*"
   event_sources:
     -
       arn: arn:aws:kinesis:us-west-2:234567890123:stream/foo
       starting_position: LATEST
       batch_size: 100
```

```
35 lambda:
36   description: A simple Python sample
37   handler: simple.handler
38   runtime: python2.7
39   memory_size: 256
40   timeout: 3
41   vpc_config:
42     security_group_ids:
43       - sg-12345678
44       - sg-23456789
45     subnet_ids:
46       - subnet-12345678
47       - subnet-23456789
```

Explanations:

Line Number	Description
2	This name will be used to name the function itself as well as any policies and roles created for use by the function.
3	A map of environments. Each environment represents one possible deployment target. For example, you might have a dev and a prod. The names can be whatever you want but the environment names are specified using the –env option when you deploy.
5	The profile name associated with this environment. This refers to a profile in your AWS credential file.
6	The AWS region associated with this environment.
7	This section defines the elements of the IAM policy that will be created for this function in this environment.
9	Each resource your function needs access to needs to be listed here. Provide the ARN of the resource as well as a list of actions. This could be wildcarded to allow all actions but preferably should list the specific actions you want to allow.
15	If your Lambda function has any event sources, this would be where you list them. Here, the example shows a Kinesis stream but this could also be a DynamoDB stream, an SNS topic, or an S3 bucket.
18	For Kinesis streams and DynamoDB streams, you can specify the starting position (one of LATEST or TRIM_HORIZON) and the batch size.
35	This section contains settings specify to your Lambda function. See the Lambda docs for details on these.

CHAPTER 3

Commands

Kappa is a command line tool. The basic command format is:

```
kappa [options] <command> [optional command args]
```

Available `options` are:

- –config <config_file> to specify where to find the kappa config file. The default is to look in `kappa.yml`.
- –env <environment> to specify which environment in your config file you are using. The default is `dev`.
- –debug/–no-debug to turn on/off the debug logging.
- –help to access command line help.

And `command` is one of:

- deploy
- delete
- invoke
- tag
- tail
- event_sources
- status

Details of each command are provided below.

3.1 deploy

The `deploy` command does whatever is required to deploy the current version of your Lambda function such as creating/updating policies and roles, creating or updating the function itself, and adding any event sources specified in your config file.

When the command is run the first time, it creates all of the relevant resources required. On subsequent invocations, it will attempt to determine what, if anything, has changed in the project and only update those resources.

3.2 delete

The `delete` command deletes the Lambda function, remove any event sources, delete the IAM policy and role.

3.3 invoke

The `invoke` command makes a synchronous call to your Lambda function, passing test data and display the resulting log data and any response returned from your Lambda function.

The `invoke` command takes one positional argument, the `data_file`. This should be the path to a JSON data file that will be sent to the function as data.

3.4 tag

The `tag` command tags the current version of the Lambda function with a symbolic tag. In Lambda terms, this creates an `alias`.

The `tag` command requires two additional positional arguments:

- name - the name of tag or alias
- description - the description of the alias

3.5 tail

The `tail` command displays the most recent log events for the function (remember that it can take several minutes before log events are available from CloudWatch)

3.6 test

The `test` command provides a way to run unit tests of code in your Lambda function. By default, it uses the `nose` Python testrunner but this can be overridden my specifying an alternative value using the `unit_test_runner` attribute in the kappa config file.

When using nose, it expects to find standard Python unit tests in the `_tests/unit` directory of your project. It will then run those tests in an environment that also makes any python modules in your `_src` directory available to the tests.

3.7 event_sources

The `event_sources` command provides access the commands available for dealing with event sources. This command takes an additional positional argument, `command`.

- command - the command to run (list|enable|disable)

3.8 status

The `status` command displays summary information about functions, stacks, and event sources related to your project.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search