Accretion

Jun 18, 2019

Accretion

Important: The Accretion builders and infrastructure should be stable now, but the CLI interfaces may change.

Important: Accretion relies on AWS Lambda and AWS Step Functions, and so will NOT work in the Osaka-Local (ap-northeast-3) region.

Tooling for building AWS Lambda Layers containing desired dependencies.

Find our source code on GitHub.

CHAPTER 1

Overview

1.1 Concept

The idea of Accretion is to make building and consuming dependencies in AWS Lambda simple.

Historically, in order to consume dependencies in Lambda, you had to build your dependencies locally, hoping that everything was compatible with the Lambda runtime, zip everything up, upload it to S3, and finally update your Lambda definition.

When Lambda launched Lambda Layers, this changed. Now, you can separate the contents of your eventual Lambda deployable into two categories:

1. Things that you depend on.
2. Things that you author.

When you build your Lambda, rather than needing to include everything in one zip, you can define dependencies on one or more Layers. This means that you can control and iterate on your dependencies and your code independently.

However, you still need to build those dependencies and create those Layers. This is where Accretion comes in.

1.2 Problem

There are a few fundamental problems that people encounter when building dependencies:

1. Dependencies need to be built in as close as possible to the Lambda runtime. This can be difficult to programmatically ensure.
2. Building dependencies is likely to take longer than building your code.
3. Dependencies are unlikely to change as frequently as your code, especially during development. This together with #2 means that your builds probably take a lot longer than they could and repeat a lot of work.
4. In the AWS parlance, this is "undifferentiated heavy lifting". You probably don't care how it happens, you just want to have your dependencies there when you need them.

1.3 Goals

- It should be very simple to define what dependencies should be built.
- Dependencies should be defined as broadly or as rigidly as the native tools allow.
- The user should not need to do anything more than provide a listing that native tools will understand and resolve.
- Users should be able to determine what is actually installed in a Layer version that they are consuming.
- Notifications should be available to tell a consumer that there is a new Layer version available.
- For the end user, building their Lambda deployment artifact should be as simple as possible. For Python, this should ideally simply mean uploading the project wheel file as the Lambda zip.

1.3.1 Assumptions

These are initial assumptions for simplicity. As Accretion evolves, these should be revisited and addressed.

1. Only the primary public artifact repositories will be considered as dependency sources.
2. All artifacts and Layers will be publicly consumable.
3. The license info for the Layer cannot be determined.
4. Project names must not exceed 70 characters. This is to save space to add runtime information to the Layer name. Initially, we use the language specified at the start to build a separate artifact for every runtime. A better approach might be to require specified runtimes from the start, but this will have its own issues because the artifacts for one language version might not always be compatible with the artifacts for other language versions.
5. Given the above, we are not caring about SSE anywhere. Once support for private resources is added, SSE support should also be added everywhere.

1.4 Solution

There are several components needed to solve this:

1. **Artifact Builder.** Given a dependency definition, build a dependency artifact.
2. **Layer Creator.** Given a dependency artifact and details, determine whether it differs from the previous Layer version. If it does, publish a new Layer version.
3. **Simple CLI.** In addition to the lower-level resources that will build and orchestrate all of the above, a simple CLI should be defined that provides access to important information about all managed resources as well as the ability to create new resources.

1.5 Moving Forward

Moving forward, some additional features that should be considered:

- Support for additional languages. The system architecture is designed to support arbitrary languages, but initially only Python builders are defined.
- Support for private artifacts and Layers. This will require some way to define the permissions for a Layer.
- Support for private artifact repositories.
- Support for custom builders. This could cover both additional languages and non-standard artifact repositories.

- Scheduled re-creation of Layers.
- Event source for updates to public package repositories.
- Global artifact replication. The original plan was to have a single artifact builder with an artifact replication layer that would send those artifacts to all target regions. This plan was abandoned because the full regional isolation is both much simpler to manage and probably a better idea for isolation anyway. However, the artifact and layer builders are decoupled in such a way that a replication layer could still be added between them. Depending on demand, we can reconsider this.
- An API that provides all of the capabilities of the CLI.

CHAPTER 2

Manifests

All manifests are JSON.

2.1 Artifact Manifest

When a Layer artifact is created, the artifact builder also creates a manifest file that describes what that artifact contains.

- **ProjectName** : Name of the project.
- **ArtifactS3Key** : S3 key in the regional artifacts bucket that contains the artifact.
- **Runtimes** : List of Lambda runtimes that are compatible with this artifact.
- **Requirements** : List of requirements strings as they were requested.
- **Installed** : List of package version structures for each package that was actually installed.
  - **Name** : Name of package.
  - **Version** : Version of package.

```
{
    "ProjectName": "example layer",
    "ArtifactS3Key": "accretion/artifacts/exampleLayer/4b14d8bf-61ff-4514-9f9a-
↪ebb59dba08fe.zip",
    "Requirements": [
        "cryptography",
        "requests"
    ],
    "Installed": [
        {
            "Name": "asn1crypto",
            "Version": "0.24.0"
        },
        {
            "Name": "certifi",
```

(continues on next page)

(continued from previous page)

```
            "Version": "2019.3.9"
        },
        {
            "Name": "cffi",
            "Version": "1.12.3"
        },
        {
            "Name": "chardet",
            "Version": "3.0.4"
        },
        {
            "Name": "cryptography",
            "Version": "2.6.1"
        },
        {
            "Name": "idna",
            "Version": "2.8"
        },
        {
            "Name": "pycparser",
            "Version": "2.19"
        },
        {
            "Name": "requests",
            "Version": "2.21.0"
        },
        {
            "Name": "six",
            "Version": "1.12.0"
        },
        {
            "Name": "urllib3",
            "Version": "1.24.2"
        }
    ],
    "Runtimes": [
        "python3.6"
    ]
}
```

2.2 Layer Manifest

When a Layer version is created, the layer builder creates a manifest file that describes that Layer.

- **LayerArn** : Layer Amazon Resource Name (Arn).
- **LayerVersion** : Layer version that was created.
- **ArtifactManifest** : Structure identifying location of artifact manifest.
  - **S3Bucket** : S3 regional artifacts bucket name.
  - **S3Key** : S3 key in the regional artifacts bucket that contains the artifact manifest.

```
{
    "Layer": {
```

(continues on next page)

(continued from previous page)

```
        "Arn": "arn:aws:states:region:account-id:stateMachine:stateMachineName",
        "Version": 3
    },
    "ArtifactManifest": {
        "S3Bucket": "accretion-regional-bucket",
        "S3Key": "accretion/manifests/exampleLayer/4b14d8bf-61ff-4514-9f9a-
↪ebb59dba08fe.manifest"
    }
}
```

CHAPTER 3

Accretion CLI

The Accretion CLI is the primary point for controlling Accretion resources.

The Accretion CLI maintains configuration state in a "Deployment File".

Warning: Accretion is under active development and is not yet stable. The below reflects the target interface for the Accretion CLI. Not all commands will work yet.

3.1 Usage

3.1.1 init

Initialize the `DEPLOYMENT_FILE` for deployments to `REGIONS`.

This does NOT deploy to those regions.

Run `accretion update` to update and fill all regions in a deployment file.

```
accretion init DEPLOYMENT_FILE REGIONS...
```

3.1.2 update

Update deployments in all regions described in `DEPLOYMENT_FILE`.

This will also initialize any empty deployment regions and complete any partial deployments.

```
accretion update all DEPLOYMENT_FILE
```

3.1.3 add regions

Add more `REGIONS` to an existing deployment description in `DEPLOYMENT_FILE`.

This does NOT deploy to those regions.

Run `accretion update` to update and fill all regions in a deployment file.

```
accretion add regions DEPLOYMENT_FILE REGIONS...
```

3.1.4 destroy

Destroy all resources for an Accretion deployment described in `DEPLOYMENT_FILE`.

Warning: This will destroy ALL resources in ALL regions. Be sure that is what you want to do before running this.

```
accretion destroy DEPLOYMENT_FILE
```

3.1.5 request

Request a new layer version build.

Important: These operations are currently completely asynchronous with no way of tracking a layer build through the CLI. I plan to add tooling around this later, but the exact form it will take is still TBD. mattsb42/accretion#27

raw

Request a new layer in every region in `DEPLOYMENT_FILE`. The Layer must be described in the Accretion format in `REQUEST_FILE`.

```
{
    "Name": "layer name",
    "Language": "Language to target",
    "Requirements": {
        "Type": "accretion",
        "Requirements": [
            {
                "Name": "Requirement Name",
                "Details": "Requirement version or other identifying details"
            }
        ]
    },
    "Requirements": {
        "Type": "requirements.txt",
        "Requirements": "Raw contents of requirements.txt file format"
    }
}
```

Note: The only supported language at this time is `python`.

```
accretion request raw DEPLOYMENT_FILE REQUEST_FILE
```

requirements

Request a new layer named `LAYER_NAME` in every region in `DEPLOYMENT_FILE`. The Layer requirements must be defined in the Python requirements.txt format in `REQUIREMENTS_FILE`.

```
accretion request DEPLOYMENT_FILE REQUIREMENTS_FILE
```

3.1.6 list

layers

Important: This command has not yet been implemented.

List all Accretion-managed Lambda Layers and their versions in the specified region.

```
accretion list layers DEPLOYMENT_FILE REGION_NAME
```

3.1.7 describe

layer-version

Important: This command has not yet been implemented.

Describe a Layer version, listing the contents of that Layer.

```
accretion describe layer-version DEPLOYMENT_FILE REGION_NAME LAYER_NAME LAYER_VERSION
```

3.1.8 check

Important: This command has not yet been implemented.

Check a "Request File" for correctness.

```
accretion check REQUEST_FILE
```

3.2 Deployment File

> **Warning:** Deployment files MUST NOT be modified by anything other than Accretion tooling.

An Accretion deployment file describes the stacks associated with a single Accretion deployment.

It is a JSON file with the following structure:

```
{
    "Deployments": {
        "AWS_REGION": {
            "Core": "STACK_NAME",
            "ArtifactBuilder": "STACK_NAME",
            "LayerBuilder": "STACK_NAME"
        }
    }
}
```

3.3 Request File

An Accretion require file describes the Layer that is being requested.

It is a JSON file with the following structure:

```
{
    "Name": "layer name",
    "Language": "Language to target",
    "Requirements": {
        "Type": "accretion",
        "Requirements": [
            {
                "Name": "Requirement Name",
                "Details": "Requirement version or identifying details"
            }
        ]
    },
    "Requirements": {
        "Type": "requirements.txt",
        "Requirements": "Raw contents of requirements.txt file format"
    }
}
```