DevOps Activity 1 – Part 3

Created by Steve 14-April-2018

In this version of the document, we will continue from Activity 1 – Part 2. In this guide we will install the Amazon Web Services Command Line Tools (aws cli) so that we can automate creating a Lambda Function (a serverless concept) that will allow each member of the class to add variables which will be displayed when the Lambda function invoked.

Overview

This document presumes you have followed DevOps Activity 1 – Part 2 guide and you have the following installed and configured correctly as instructed.

- Notepad++
- Git for Windows
- Oracle VM VirtualBox
- Hashicorp Vargrant
    - A vagrant managed VM

Objective

In this guide, we will be creating a simple AWS Lambda function which is a serverless concept explained below.

What is AWS Lambda

AWS Lambda is a compute service that lets you run code without provisioning or managing servers. AWS Lambda executes your code only when needed and scales automatically, from a few requests per day to thousands per second. You pay only for the compute time you consume - there is no charge when your code is not running.

More information:

https://docs.aws.amazon.com/lambda/latest/dg/welcome.html

We will not be developing the Lambda function, we will edit an existing code template, then deploy it to AWS using Hashicorp Terraform which will manage the code deployment for us.

The essential resources used, are as follows:

- An existing Linux VM
  - Created and Managed by Vagrant using a VirtualBox provider
- An appropriate AWS User (ACCESS ID, and SECRET KEY ID)
- Git
  - The local repo
- Github
  - The Remote repo (master)
- awscli
  - Command-line tools to call that allow scripting of AWS infrastructure and services.
- Code/Scripts
  - Provisioning Scripts
  - Infrastructure as Code

Installing AWS CLI on a Linux VM

Open a Git Bash terminal, and ensure that you in the correct location have started the Vagrant managed VM using vagrant up

```
cd ~/local-repos/devops-course-activity1
vagrant up
```


Result:

```
Steve Robinson@SurfaceLaptop MINGW64 ~/local-repos/devops-course-activity1 (master)
$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Checking if box 'ubuntu/xenial64' is up to date...
==> default: A newer version of the box 'ubuntu/xenial64' for provider 'virtualbox' is
==> default: available! You currently have version '20180410.0.0'. The latest is version
==> default: '20180413.0.0'. Run `vagrant box update` to update.
==> default: Clearing any previously set forwarded ports...
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
==> default: Forwarding ports...
    default: 22 (guest) => 2222 (host) (adapter 1)
==> default: Running 'pre-boot' VM customizations...
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2222
    default: SSH username: vagrant
    default: SSH auth method: private key
==> default: Machine booted and ready!
==> default: Checking for guest additions in VM...
    default: The guest additions on this VM do not match the installed version of
    default: VirtualBox! In most cases this is fine, but in rare cases it can
    default: prevent things such as shared folders from working properly. If you see
    default: shared folder errors, please make sure the guest additions within the
    default: virtual machine match the version of VirtualBox you have installed on
    default: your host and reload your VM.
    default:
    default: Guest Additions Version: 5.1.34
    default: VirtualBox Version: 5.2
==> default: Mounting shared folders...
    default: /vagrant => C:/Users/Steve Robinson/local-repos/devops-course-activity1
==> default: Machine already provisioned. Run `vagrant provision` or use the `--provision`
==> default: flag to force provisioning. Provisioners marked to run always will still run.
```

Once started, connected to the VM using ssh

`vagrant ssh`

Result:

```
Steve Robinson@SurfaceLaptop MINGW64 ~/local-repos/devops-course-activity1 (master)
$ vagrant ssh
Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.4.0-119-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

0 packages can be updated.
0 updates are security updates.


Last login: Fri Apr 13 18:54:59 2018 from 10.0.2.2
vagrant@ubuntu-xenial:~$
```

You are now logged in as the vagrant user, which as sudo rights. This will allow us to install the appropriate AWS Command Line Tools.

If we issue the command `pwd`, we will see we are in the vagrant user's home directory as seen below

We will first issue a command to clone the same Github repo which we used in Part 2, contains all the scripts required for this exercise.

Clone the Git Repo https://github.com/keepingbuildsgreen/devops-course-activity1 using the git clone command as shown below

```
sudo git clone https://github.com/keepingbuildsgreen/devops-course-activity1.git
```

List the folders contents using `ls -ltra` as shown below which confirms that we have indeed cloned the repo locally.

Result:

All the script required for this part of the guide are in a folder called `aws-cli-setup` to get there we use the following command:

```
cd ~/devops-course-activity1/scripts/aws-cli-setup
```

List the folders contents using the command below

```
ls -ltra
```

Result:

```
drwxr-xr-x 4 vagrant vagrant 4096 Apr 14 13:24 ..
-rwxrwxr-x 1 vagrant vagrant  751 Apr 14 14:05 install-awscli-pip3.sh
drwxrwxr-x 2 vagrant vagrant 4096 Apr 14 14:06 .
-rwxrwxr-x 1 vagrant vagrant   22 Apr 14 14:08 uninstall-awscli-pip3.sh
vagrant@ubuntu-xenial:~/devops-course-activity1/scripts/aws-cli-setup$
```

Install AWS CLI

We are going to run the installer script which will do all the heavy lifting.

During the process many things will happen, then it will wait and prompt you for some localised settings.

For the AWS_ACEESS_KEY_ID use the following (copy/paste)

AKIAJJ5NU64KDT2A5XTQ

For the AWS_SECRET_ACCES_KEY use the following (copy/paste)

EhxzVk9P9iJeopOJmoosCA7/QV1wKwF+m2Zm7V8U

Example running the custom awscli installer script

Run the following command:

./install-awscli-pip3.sh

```
Collecting python-dateutil<2.7.0,>=2.1 (from botocore==1.10.4->awscli)
  Downloading https://files.pythonhosted.org/packages/4b/0d/7ed381ab4fe80b8ebf34411d14f253e1cf3e56e2820ffa1d8844b23859a2/python_dat
eutil-2.6.1-py2.py3-none-any.whl (194kB)
    100% |████████████████████████████████| 194kB 2.5MB/s
Collecting jmespath<1.0.0,>=0.7.1 (from botocore==1.10.4->awscli)
  Downloading https://files.pythonhosted.org/packages/b7/31/05c8d001f7f87f0f07289a5fc0fc3832e9a57f2dbd4d3b0fee70e0d51365/jmespath-0
.9.3-py2.py3-none-any.whl
Collecting six>=1.5 (from python-dateutil<2.7.0,>=2.1->botocore==1.10.4->awscli)
  Downloading https://files.pythonhosted.org/packages/67/4b/141a581104b1f6397bfa78ac9d43d8ad29a7ca43ea90a2d863fe3056e86a/six-1.11.0
-py2.py3-none-any.whl
Building wheels for collected packages: PyYAML
  Running setup.py bdist_wheel for PyYAML ... done
  Stored in directory: /home/vagrant/.cache/pip/wheels/03/05/65/bdc14f2c6e09e82ae3e0f13d021e1b6b2481437ea2f207df3f
Successfully built PyYAML
Installing collected packages: pyasn1, rsa, docutils, colorama, six, python-dateutil, jmespath, botocore, s3transfer, PyYAML, awscl
i
Successfully installed PyYAML-3.12 awscli-1.15.4 botocore-1.10.4 colorama-0.3.7 docutils-0.14 jmespath-0.9.3 pyasn1-0.4.2 python-da
teutil-2.6.1 rsa-3.4.2 s3transfer-0.1.13 six-1.11.0
You are using pip version 8.1.1, however version 10.0.0 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.

Installed awscli.
AWS Access Key ID [****************5XTQ]: AKIAJJ5NU64KDT2A5XTQ
AWS Secret Access Key [****************7V8U]: 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
Default region name [eu-west-1]:
Default output format [None]:
Updating awscli
Collecting awscli
  Downloading https://files.pythonhosted.org/packages/8e/ec/9f0fd5bb3c954742eb37545297bf62bfdb46ab80e9f5beea0c1ea263fc5f/awscli-1.1
5.4-py2.py3-none-any.whl (1.3MB)
    100% |████████████████████████████████| 1.3MB 801kB/s
Collecting colorama<=0.3.7,>=0.2.5 (from awscli)
  Downloading https://files.pythonhosted.org/packages/b7/8e/ddb32ddaabd431813e180ca224e844bab8ad42fbb47ee07553f0ec44cd86/colorama-0
.3.7-py2.py3-none-any.whl
Collecting rsa<=3.5.0,>=3.1.2 (from awscli)
  Downloading https://files.pythonhosted.org/packages/e1/ae/baedc9cb175552e95f3395c43055a6a5e125ae4d48a1d7a924baca83e92e/rsa-3.4.2-
py2.py3-none-any.whl (46kB)
    100% |████████████████████████████████| 51kB 4.6MB/s
Collecting botocore==1.10.4 (from awscli)
  Downloading https://files.pythonhosted.org/packages/22/66/9dcf61c1756a0de1ac86547a0fdb8c8641e345049cbf6f18c2ad6dab69d0/botocore-1
.10.4-py2.py3-none-any.whl (4.2MB)
    44% |██████████████                  | 1.9MB 2.3MB/s eta 0:00:02
```
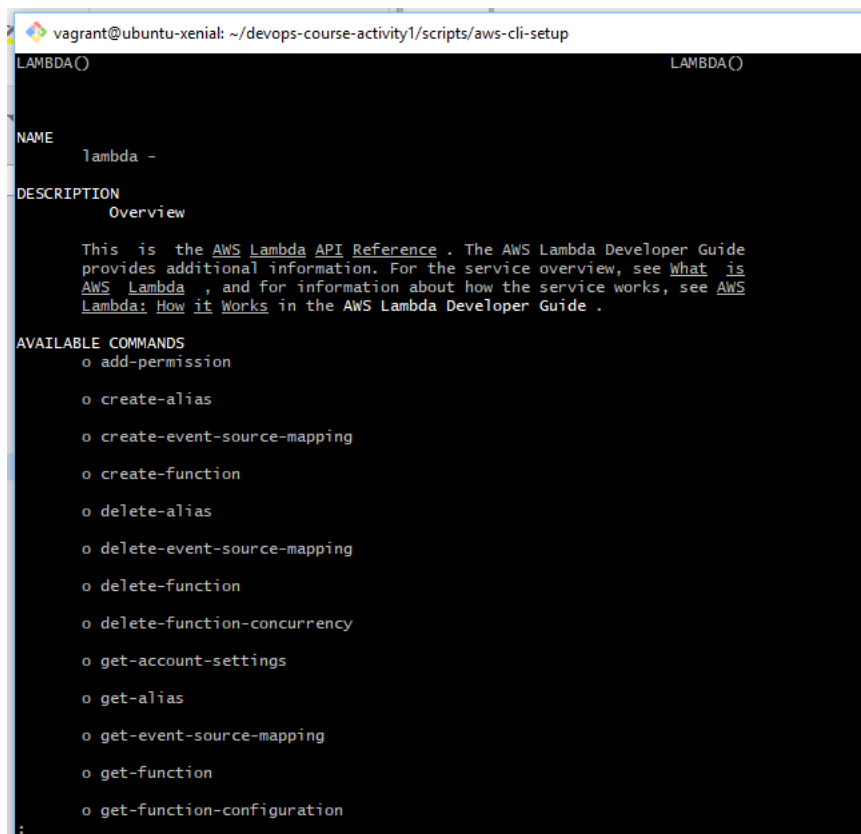
Once the awscli has been installed, we can test it by running some commands.

For example:

To get help on awscli commands we can run the following:

```
aws lambda help
```

Result:

Type q <enter> to exit the help screen.

Now run the following command:

```
aws lambda list-functions
```

Result:

Depending on how many Lambda function you have it will list them in json format and might look like…

```
{
    "Functions": [
        {
            "Handler": "lambda-echo.handler",
            "TracingConfig": {
                "Mode": "PassThrough"
            },
            "Timeout": 2,
            "Runtime": "nodejs6.10",
            "LastModified": "2018-04-16T23:30:03.282+0000",
            "MemorySize": 128,
            "FunctionName": "lambda-echo",
            "FunctionArn": "arn:aws:lambda:eu-west-
1:948957106729:function:lambda-echo",
            "Version": "$LATEST",
            "Role": "arn:aws:iam::948957106729:role/lambda-echo-role",
            "Description": "Echo Static String Responder",
            "CodeSize": 472,
            "Environment": {
                "Variables": {
                    "LAST_NAME": "Robinson1",
                    "FIRST_NAME": "Steve1"
                }
            },
            "CodeSha256": "226FctpGgvy6SoIouAofxkqMAgCZbEOSXmhCI5DKitE=",
            "RevisionId": "269a34bd-46d2-4dfe-84b0-c5c8b7114c35"
        }
    ]
}
```

Now we are ready to deploy our Lambda function.

Deploying the Lambda function

navigate to the `devops-course-activity1/scripts/aws-lambda/func1` folder

List the contents:

```
ls -ltra
```

Result:

We will now edit the `variables.sh` file.

Please figure out together by communicating with each other how to edit a file on Linux, and change the variables as asked in the variables file.

<mark>Once you have added the required information, please inform the instructor.</mark>

<mark>When it is Ok to do so as guided by the instructor, issue the following command to deploy the lambda function to the cloud.</mark>

PLEASE WAIT HERE, until given approval to run the command!

Run the deployment script as follows:

```
./deploy.sh
```

The Lambda function will be deployed.

Once deployed run the `test.sh` script to verify that it works.

Discussion

Discussion on what has just transpired, and what we built, and where it went. The instructor will review the code with the class, some key insights and will walk the class through the AWS Console, so we can see the Lambda Function in action live on the internet.