NeuroCAAS Developer Guide

ta2507

April 2020

1 Introduction

Neuroscience Cloud Analysis As a Service welcomes external developers to deploy their existing analyses onto our platform. Once analyses are built, they can be loaded onto the website interface for NeuroCAAS seamlessly. We'll describe the process of developing analyses for NeuroCAAS in three steps: 1. Building a machine image. 2. Setting up your machine image to work as an immutable analysis environment. 3. Testing and Deployment. All steps are available via a python and shell script based API. Development will follow a principle of Infrastructure as Code (IaC), meaning that all of your development steps will be documented in code as you build.

2 Prerequisites

In order to follow the steps listed here, you will need the following:

- An AWS account (n.b. specific instructions below if developing on the Center for Theoretical Neuroscience account).
- An IAM user with programmatic access on that account (i.e. access key and secret access keys) (`https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users_create.html`)
- A local clone of the NeuroCAAS Repository.
- Custom resources for ssh-key management available at this repo: `https://github.com/binxio/cfn-secret-provider`
- jq 1.6 (a json parser)- *install via brew/apt-get*
- AWS CLI. - (`https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2.html`)
- AWS Sam CLI. Additional command line tools for serverless applications. `https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-sam-cli-install.html`

- Docker (for the AWS Sam CLI)- *install via Docker homepage- referenced in SAM CLI installation.*
- Anaconda (`https://www.anaconda.com`). *Not a strict requirement, but we will assume development in a conda virtual environment.*

First verify your aws cli installation by running:

```
aws configure
```

When prompted, enter the access key and secret access key associated with your IAM user account, as well as the AWS region you are closest to. IMPORTANT: If using the CTN AWS account to develop, please set your AWS region to be **us-east-1**.
Next verify your conda installation by running:

```
conda list
```

Then create a new environment as follows:

```
conda create -n neurocaas python=3.6
```

Note: the argument following the -n flag is the name of the virtual python environment you are creating, and can be whatever you wish. Now move into the root directory of the cloned neurocaas repo:

```
cd /path/to/local/neurocaas/
```

Activate your new environment, and install necessary packages by running:

```
conda activate neurocaas
conda install pip
pip install -r requirements.txt
```

3 Initializing NeuroCAAS

If you are developing within the CTN account, skip this first step, and start with the aws ssm get-parameter step.

To initialize NeuroCAAS, first follow the installation instructions for the binxio secret provider stack: `https://github.com/binxio/cfn-secret-provider`. Navigate within the repository to:

`neurocaas/ncap_iac/ncap_blueprints/utils_stack`.

Now run the following command:

```
bash initialize_neurocaas.sh
```

This will create the cloud resources necessary to deploy your resources regularly and handle the permissions necessary to manage and deploy cloud jobs, and ssh keys to access resources on the cloud. The results of initialization can be seen in the file `global_params_initialized.json`, with the names of resources listed. If you encounter an error, consult the contents of the file `neurocaas/ncap_iac/ncap_blueprints/utils_stack/init_log.txt`, and post it to the neurocaas issues page.

This process will also generate an ssh key, that is not printed to the repo for security reasons. In order to retrieve your ssh key, type the following command:

```
aws ssm get-parameter --name /testkeystack/neurocaas/private-dev-key
    --with-decryption
```

Paste the content of the "value" field (minus quotes, respecting line breaks) into a file **not under version control**, and call the file testkeystack-custom-dev-key-pair.pem. You will reference this key when developing a machine image later.

4 Initializing a blueprint

To start, we will need to build a computing environment where your analysis lives along with all of its required dependencies. To do this, we first need to initialize a blueprint for your stack. Navigate to the `ncap_blueprints` directory, and run the command:

```
bash iac_utils/configure.sh "name of your analysis algorithm"
```

Where the argument passed must be restricted to lowercase letters, numbers, and dash marks (−). This will create a folder with the specified name. If you navigate into this folder, you can see the blueprint that specifies an analysis pipeline. The parameters that you will most care about are the "Lambda.LambdaConfig.AMI", "Lambda.LambdaConfig.INSTANCE_TYPE" and "Lambda.LambdaConfig.COMMAND" parameters (although you will only have to specify an INSTANCE_TYPE to start with). AMI specifies the Amazon Machine Image where your software and dependencies are installed, and contains most of the analysis-specific configuration details that you must specify. INSTANCE_TYPE specifies the hardware configuration that is run by default (can be changed on demand) and is selected from a list of instance types available on AWS. Finally, COMMAND specifies a bash command that will be run on your remote instance [with parameters] to generate data analysis. Altering this will require you to alter further settings, which we will describe later.

For now, remove all Affiliates from the UXData area except for "debuggers" we will return to these later.

Once you have configured a blueprint, navigate to the "dev_utils" folder in the "ncap_blueprints" directory, and start up IPython. The bulk of building a

machine image will be done through an interactive, Python based API, described next.

5 Building a machine image

To handle image build and development, we have built a custom class to interface with the python AWS SDK, boto3. Make sure that your environment has the required dependencies listed in the Prerequisites section. Then import the "NeuroCaaSAMI" class from the develop_blueprint module, and give it the path to the folder you just configured:

```
from develop_blueprint import NeuroCAASAMI

devami = NeuroCaaSAMI("../path_to_configured_folder")
```

By calling methods on declared objects, you can create, destroy, develop and test machine images easily, without directly interfacing with the cloud. The declared object reads the blueprint that you have built, and intelligently uses the information there to streamline development.

5.1 Launching a machine image

In general, if you have an AMI in your blueprint, you can simply call :

```
devami.launch_ami()
```

Which will launch the AMI listed in your blueprint. When starting with a new blueprint, call:

```
devami.launch_ami(ami = string)
```

Where string is the id of an AMI you like (you can find many on the AWS marketplace), or one of the following special codes:

- "ubuntu18": ubuntu version 18.
- "ubuntu16": ubuntu version 16.
- "dlami18": AWS deep learning ami, with pre-installed deep learning libraries on ubuntu 18.
- "dlami16": as with dlami18, but on ubuntu16.

Calling this method will initialize an instance for you, and then provide you with the ip address of that initialized instance. You can then ssh into the instance with the key that you retrieved when initializing NeuroCAAS, like so:

```
ssh -i /path/to/your/local/sshkey ubuntu@{ip address}
```

Please note that if you use a custom ami, you may be prompted to log in as root, or ec2-user instead of ubuntu.

5.2 Developing a machine image into an immutable analysis environment

After connecting to your remote instance via ssh, you can download your code repositories and dependencies to it, and test basic functionality. Once this is done, you will have to clone the repository neurocaas_remote into the user's remote directory. This directory ensures that live logging of analysis progress to users proceeds smoothly. The basic logic of all NeuroCAAS analyses is that when users run jobs on neurocaas, they deposit a special file called a "submit" file. This file contains the path to the primary dataset (or datasets) that they would like to analyze, the path to a corresponding configuration file for parameters, and a timestamp of when the job was submitted. This information is parsed by a serverless lambda function, which finally provides the instance with four parameters: 1) the S3 bucket we will be communicating with. 2) the path to a single dataset in S3 that we will be analyzing (multiple datasets are parallelized to different instances). 3) the path to the results folder where we will deposit results, and 4) the path to the configuration file that we will use to analyze data (these are the four curly braces in the COMMAND field listed above). Then, a job manager will call a script on this instance (via the COMMAND parameter), which can then pull data and configuration files from user storage, analyze that data according to provided configuration files, and then push the results back to the user. We provide basic template scripts that will streamline the process of setting up logging, pulling data to and from user storage, etc. If you would like to write your own routines to do so (e.g., if you want to specify a different COMMAND) structure in your blueprints we encourage you to do so, although messing with parallel processing in communication with AWS can be thorny. For users who just want to get started, we recommend looking at the "run_main.sh" script in the "neurocaas_remote" repo, and pasting their own bash commands into the indicated area therein. IMPORTANT: Note that you must perform a shutdown of the instance in your commands to make sure your instance is terminated. We recommend you include this in the COMMAND value as shown in the above example, as it is most reliable there. IMPORTANT: this script will not be run by the a user with the same permissions or root directory that you have when developing. Rather, it will be run by a user with admin priviledges and a different home directory. This makes it necessary to source user environment files and alter the PATH variable explicitly (for a more realistic simulation of how the script will be run with these permissions, try testing commands as admin by running "sudo -i" first).

We recommend testing your script locally by writing the COMMAND that you have in your blueprint directly into the CLI, with the four parameters in brackets specified as above.

5.3 Saving your machine image

After you have written a script and tested it locally, you should save your machine image. In order to do so, return to your IPython console, and run the command:

```
devami.create_image(name)
```

where the name is an identifier you will provide to your newly created image. You can update your blueprint with this new image by running:

```
devami.update_blueprint(ami_id,message=None)
```

Where ami_id is the id of the ami provided as output to the create command. Not providing an ami_id will update with the last image that you have created. Updating the blueprint will also automatically generate a two git commits for the repo, documenting the state of the blueprint before and after you performed this update for reproducibility purposes. The message command, if provided, will be a message associated with this pair of git commits for readability.

5.4 Cleaning up

After you have saved your machine image and updated your blueprint, you can terminate it by running:

```
devami.terminate_devinstance()
```

If you have not created an image before doing so, you will be prompted for confirmation. If you would like to step away from developing for a while, you can run:

```
devami.stop_devinstance()
```

And conversely,

```
devami.start_devinstance()
```

Note that you can launch new development images, but you can only do so after terminating your current one to prevent losing track of development.

5.5 Testing a machine image

NOTE: this step can only be done AFTER initially deploying a blueprint. Our Python development API has the capacity to *mock* the job managers that parse user input. In order to test your machine image including the inputs and outputs that a user would see, follow these steps: 1) you upload data and configuration files to the s3 bucket, just as a user would. 2) you manually write a submit.json file, like below:

```
{
    "dataname":"path/inputs/data.zip",
    "configname":"path/configs/config.json",
    "timestamp": "debugging_identifier"
}
```

Where the dataname and configname values point to the data that you uploaded in step 1.

Then, run

```
devami.submit_job_log(submitpath)
```

Where submitpath is the path to the submit file you wrote. This will trigger processing in your development instance as a background process (you can observe it with top). Make sure to cancel or remove the instance shutdown command when you are running this test, otherwise your instance will stop after the processing finishes. You can monitor the status and output of this job as it proceeds directly from python with:

```
devami.job_status(index)
devami.job_output(index)
```

Where index gives the number of job you would like to analyze (default is -1, the most recent). The results themselves will be returned to AWS S3

6 Deploying your blueprint

Once you have a working image, it is useful to deploy it as a NeuroCAAS analysis, to perform further testing using the access configuration a user would have (see "Testing a machine image"). To do so, navigate to the "neurocaas_blueprints" directory, and run the following command:

```
bash iac_utils/fulldeploy.sh "name of your pipeline here"
```

This will run all the steps necessary to build the cloud resources corresponding to your blueprint, and you can test it further from the python API after adding some test users:

6.1 Adding users

Once your blueprint has successfully been deployed, you can authorize some users to access it.