**­**

**Lambda Automation Basics Activity Guide**

We’ll be using AWS remote environments for our labs. Every student MUST HAVE AN AWS ACCOUNT CREATED IF YOU’D LIKE TO FOLLOW ALONG IN ACTIVITIES DURING CLASS.

*NOTE: The code you’ll use in this class is located at .*[*https://github.com/rich-morrow/autom8d-foundations*](https://github.com/rich-morrow/autom8d-foundations)

*You can install with:*

git clone https://github.com/rich-morrow/autom8d-foundations.git

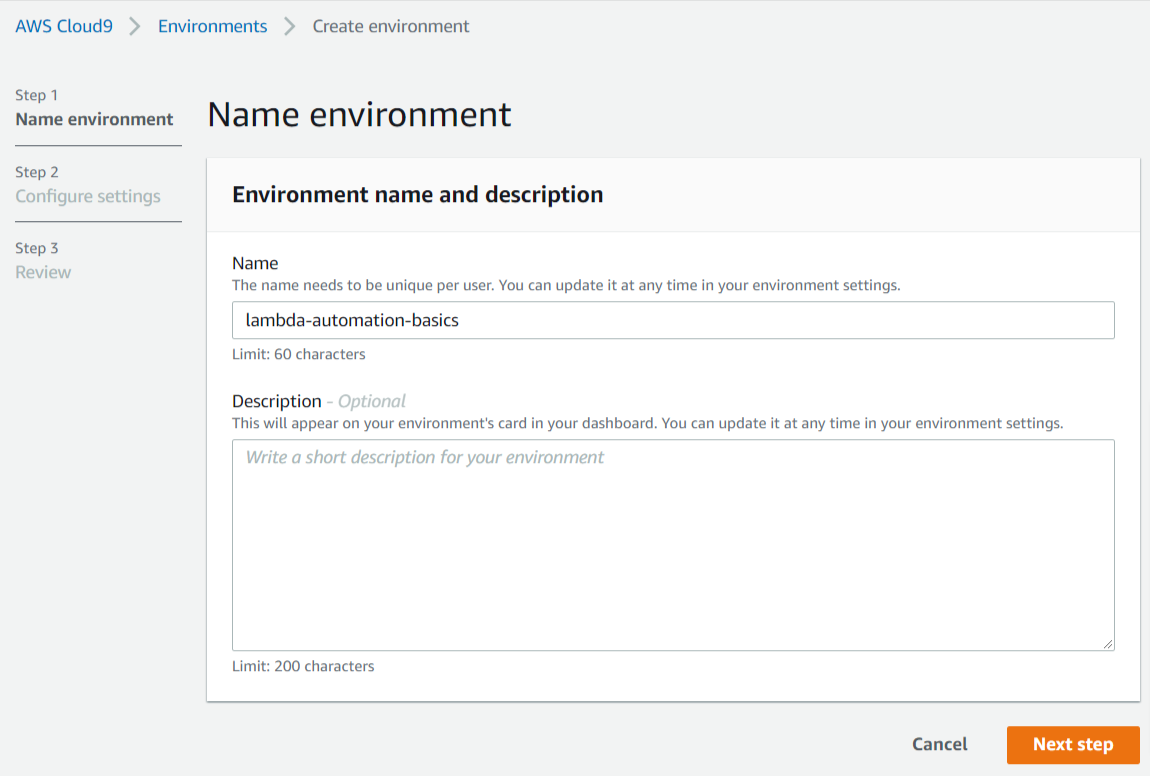
*Also helpful to know:*

* *In Linux (which we’ll use exclusively), you can “Tab Complete” long filenames. For example, if I had a directory named “i-am-a-super-long-hard-to-type-directory”, I can cd into it by typing “cd i-am-a” then just press tab. If there are files or directories that match those first few characters, linux will list them out for you, and if there’s only a single match, it will automatically fill in the remaining characters for you. Carpal Tunnel BEGONE!!!*
* *Commands you’re asked to enter on the terminal are formatted as so in this document:*

cd ~/Desktop

**Activity #1 (Real world Lambda function in Cloud9 IDE)**

Log into your AWS console and browse to the [Cloud9 area](https://console.aws.amazon.com/cloud9/home/create) (we use us-east-1, but you can switch Regions if you like):



NOTE: As of 2 May, 2022, these are all the default settings.

Environment Name: lambda-automation-basics

Environment type: “Create a new EC2 instance for environment (direct access)

Instance type: “t2.micro”

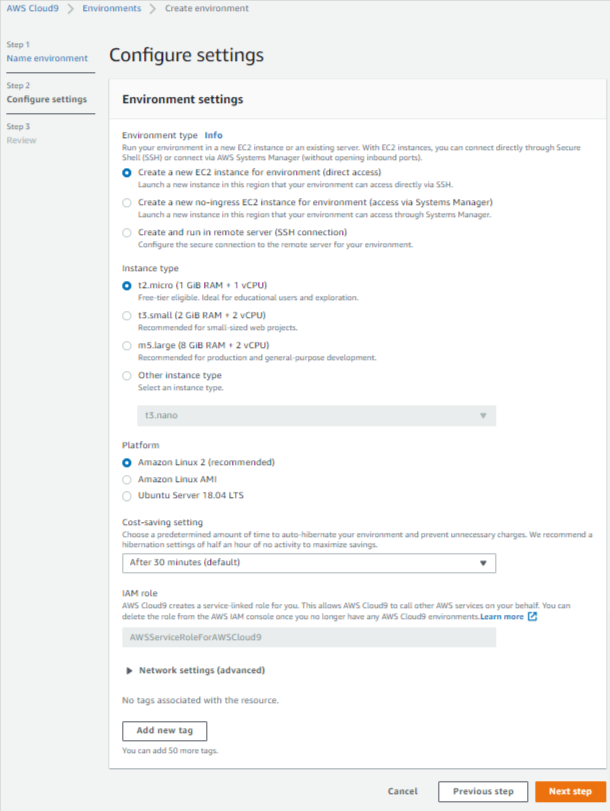
Platform: “Amazon Linux 2”

Cost-saving setting: “After 30 minutes”

IAM Role: “AWSServiceRoleForAWSCloud9”

Click “Next Steps” until you get to “Review” page,

then click “Create Environment”. You will see messages like “We are creating your AWS Cloud9 environment. This can take a few minutes”, then “Connecting”. When your environment is complete, you’ll see a page that says “Welcome to your development environment”.



**Git clone the code locally**

At the bottom of the window, you’ll see a terminal. In the terminal, enter:

git clone <https://github.com/rich-morrow/autom8d-foundations.git>

#then cd into the codebase, activity1

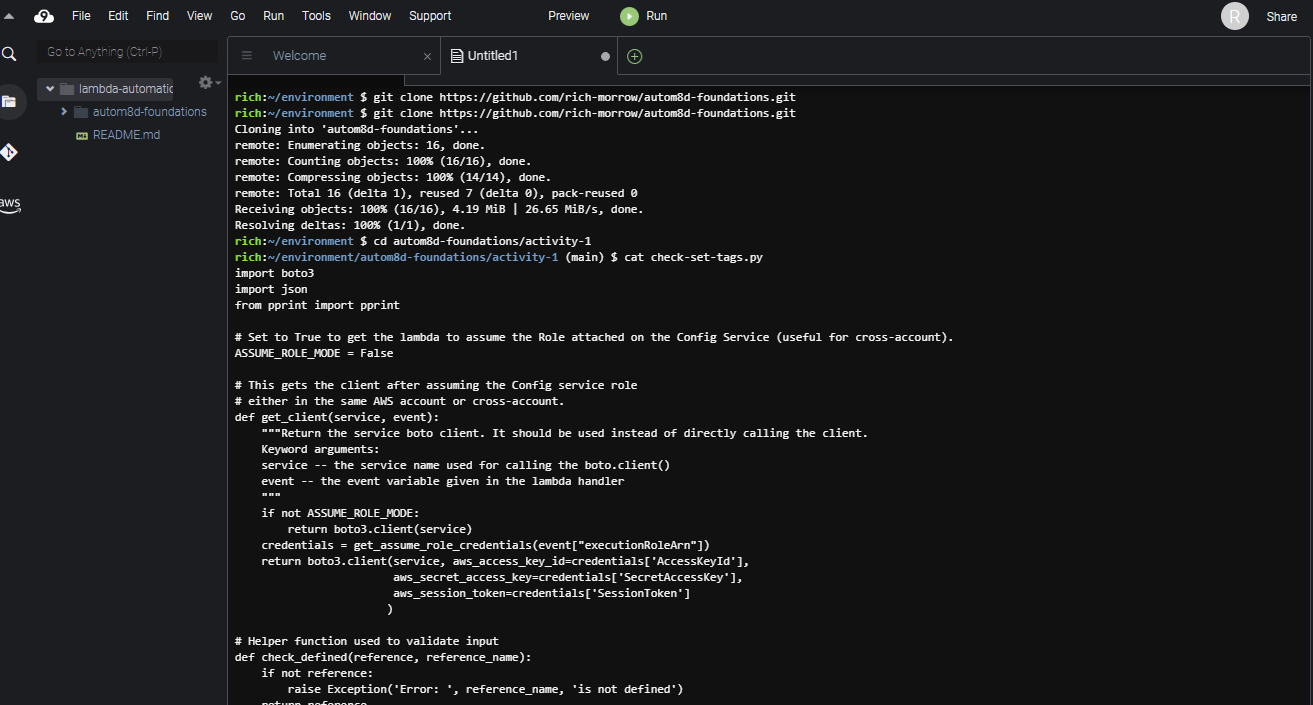
cd autom8d-foundations/activity-1

#finally, cat the code out:

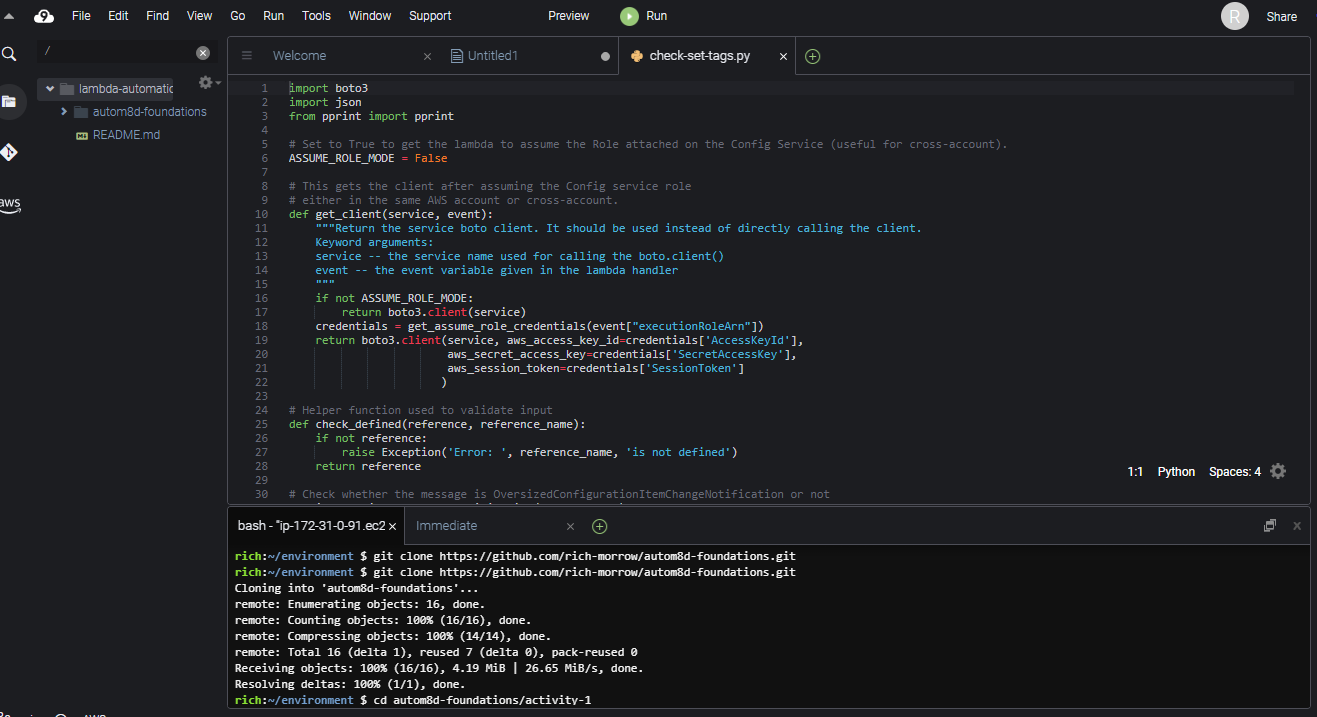
cat check-set-tags.py

**Browse the code**

Grab the terminal divider (double lines just above the terminal) and drag it to a larger view so it looks something like this:

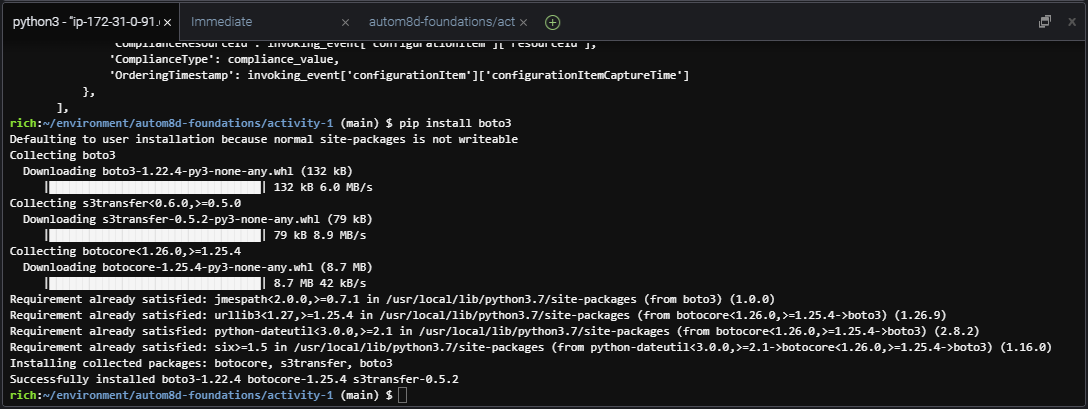


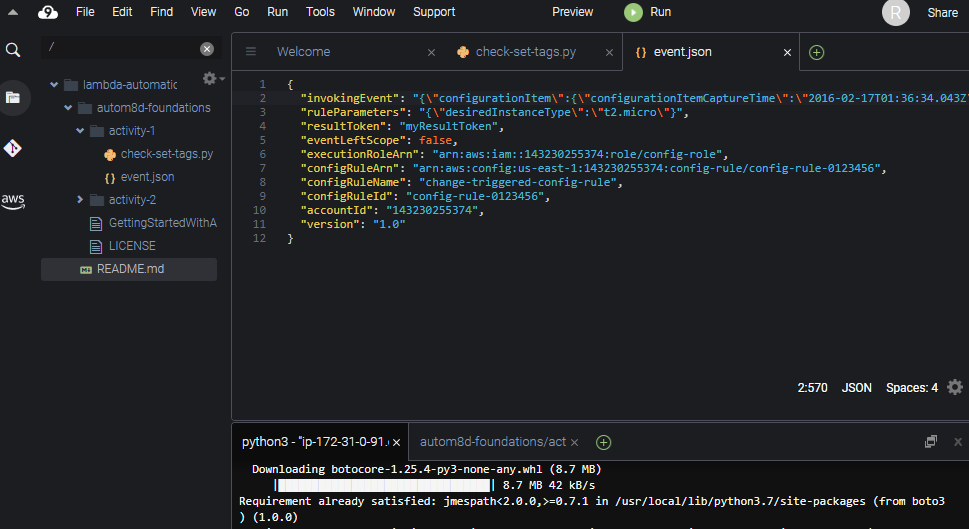
This is basically the code we’ll use for activity #2. For now, don’t worry too much about what it does… let’s open it in Cloud9 by choosing “File→Open”, then selecting “activity-1/check-set-tags.py”. Drag your terminal view down smaller so you can better see the code. As shown below:



You can attempt to run the code directly by pushing the green “play” button named “Run” at page top, but it will fail as boto3 has not been loaded into our Cloud9 environment. Let’s fix that by doing a pip install in the terminal at page bottom:

pip install boto3

Your install should look something like this:



Expand autom8d-foundations→activity-1 in the left hand navigation. Double click the “event.json” object to open it in a new tab. This event object is a mock of what would be passed to the Lambda function in a real, live execution. We can alter it later on if we choose

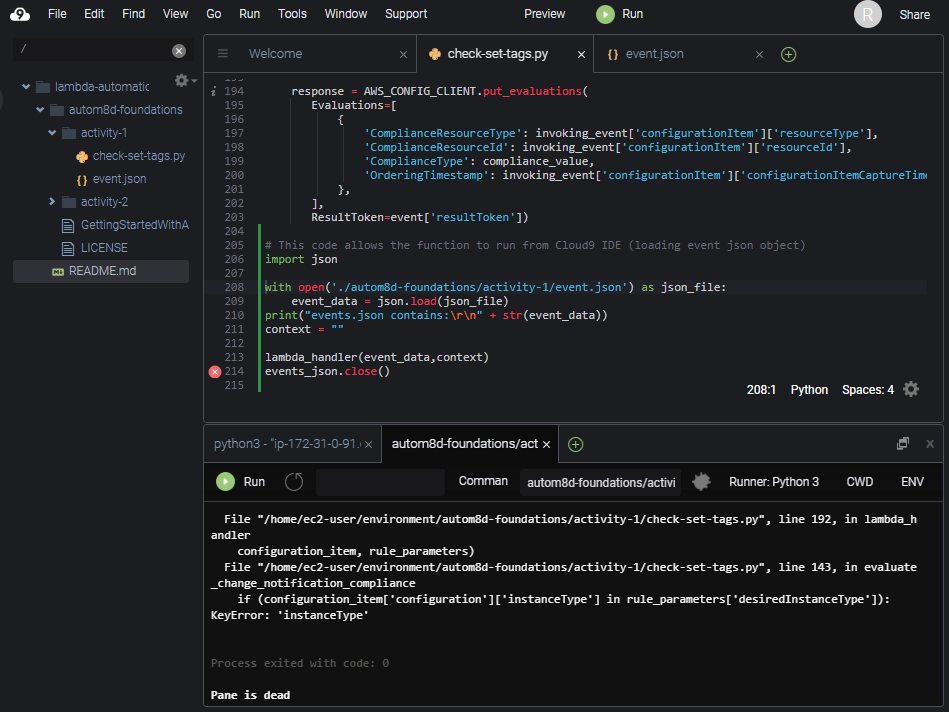
select the “check-set-tags.py” tab to look at our code again. Notice how there are several “def” functions defined. Your instructor will walk through each, describing at a high level what’s going on.

Note, also that since we’re running this code in the IDE, we have to:

1. Manually load our events.json code (lines 208,209)
2. Directly invoke our “lambda\_handler” method (line 213). When this function runs natively in Lambda (as you’ll see in Activity #2), you’ll see that “lambda\_handler” is what is called automatically for us by the Lambda service.

Also, we should point out that if you’re doing a lot of local development and/or dealing with event objects frequently, there is a more elegant way to do this – with the SAM CLI. Although what we’ve done here will work just fine for our demo purposes (and it also nicely highlights some python operations like file open, read, etc), if you want to learn the “right way” to deal with Lambda in Cloud9, check out this AWS user guide: [Working with AWS serverless applications using the AWS Toolkit](https://docs.aws.amazon.com/cloud9/latest/user-guide/serverless-apps-toolkit.html).

Now that boto is installed, we should be all ready to run our code. With our focus on the “check-set-tags.py” tab, let’s hit “Run” again at the page top!



Uh oh. We see errors again: “KeyError: ‘instanceType’”. This time, however, the error is expected (our “Configuration” object doesn’t have an expected “instanceType” for the mock. We can correct this, but then we’ll see other errors for “tags”, etc. For our purposes, this is just fine. We’ve validated that our code works… it’s just a bad “events” object being passed in. In Activity 2, we’ll see proper operation of this code when we come back and watch it run in a live scenario with real “Configuration” data passed into it from AWS Config.

\*\*\*\*\*\*\*\*\* END OF ACTIVITY #1 STOP HERE!!! \*\*\*\*\*\*\*\*\*

You now know basic operation (env create, terminal usage, code loading/running/debugging) of Cloud9, and you’ve gotten some experience with a fairly advanced Lambda function.

**Activity #2: Automating tag enforcement with AWS Lambda and AWS Config**

In this lab, we’re going to set up our ‘check-set-tags’ Lambda function to automatically be invoked by AWS Config when it detects any configuration change on our EC2 instances. The pseudocode for this function looks like:

foreach ec2 instance {

if(instance is approved instanceType) {

if(instance is tagged with “env=prod”) {

if(instance is not tagged with “owner”) {

tag instance with “[owner=rich@quicloud.com](mailto:owner%3Drich@quicloud.com)”

}

mark COMPLIANT and EXIT

} else {

stop instance

}

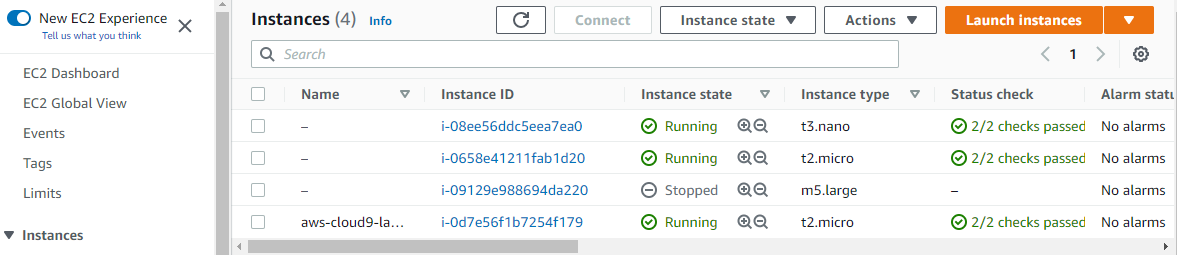
mark NON\_COMPLIANT

}

COMPLIANT OR NON\_COMPLIANT will appear in the AWS Config Dashboard, as well

**Start up two (or more) EC2 instances**

Before we start our function running, we’ll need some test instances for the script to check. In the AWS Console, browse to EC2, and start up two instances. One of size t2.micro, and one of size t3.nano. You may still have a t2.micro running from activity 1 where we played with Cloud9. If desired, you may also start instances of other sizes as well (just be careful… larger instance types can incur significant costs in AWS). Once your instances are started, you should see something like this below (we still have our Cloud9 instance running).



**Create appropriate Lambda role**

The Lambda function we’ll use needs proper execution permissions to perform all of it’s interactions with other AWS Services. We’ll use our pre-created policy document to create a role.

**Create our Lambda function**

The Lambda function we’ll use needs proper execution permissions to perform all of it’s interactions with other AWS Services. We’ll use our pre-created policy document to create a role.

**Create custom AWS Config rule using our Lambda function**

The Lambda function we’ll use needs proper execution permissions to perform all of it’s interactions with other AWS Services. We’ll use our pre-created policy document to create a role.

**Test our Config Rule by tweaking EC2 tags**

The Lambda function we’ll use needs proper execution permissions to perform all of it’s interactions with other AWS Services. We’ll use o

\*\*\*\*\*\*\*\*\* END OF ACTIVITY #2 STOP HERE!!! \*\*\*\*\*\*\*\*\*

You now have a good understanding of everything involved with using AWS Config with custom Lambda functions to programatically and proactively enforce tagging policies.