Implementing Python/PDF-to-text/NLTK via an AWS Lambda Function

This document describes a AWS Lambda function-based approach to perform the PDF-to-text plus NLP process on AWS (instead of, say, on Spark).

Assumptions:

- PDF file names do not contain spaces.

Table of Contents:

Setup Summary

1. Create the lambda function
    a. ~~Start an **EC2 instance** using the official Amazon Linux AMI (based on Red Hat Enterprise Linux)~~
    b. ~~On the EC2 instance, Build any **shared libraries** from source.~~
    c. Create a **virtualenv** with all your python dependencies.
    d. Write a python **handler** function to respond to events and interact with other parts of AWS
    e. Write a python **worker**, as a command line interface, to process the data
    f. **Bundle** the virtualenv, code and binary libs into a zip file
    g. **Publish** the zip file to AWS Lambda
2. Deploy the lambda function

Part 1 - Lambda function code (for step 2.9)

```
#!/usr/bin/env python

import os
os.environ["NLTK_DATA"] = "."       # Location of nltk_data (means "current directory")

import nltk
import urllib
import boto3

# PDF Conversion function (PDF miner)

from pdfminer.pdfinterp import PDFResourceManager, PDFPageInterpreter#process_pdf
from pdfminer.pdfpage import PDFPage
from pdfminer.converter import TextConverter
from pdfminer.layout import LAParams

from cStringIO import StringIO

def pdf_to_text(pdfname):

    # PDFMiner boilerplate
    rsrcmgr = PDFResourceManager()
    sio = StringIO()
    codec = 'utf-8'
    laparams = LAParams()
    laparams = None
    device = TextConverter(rsrcmgr, sio, codec=codec, laparams=laparams)
    interpreter = PDFPageInterpreter(rsrcmgr, device)

    # Extract text (catch exceptions, such as password protected files)
    try:
        fp = file(pdfname, 'rb')
        for page in PDFPage.get_pages(fp, check_extractable=False):
            interpreter.process_page(page)
        fp.close()
    except:
        return ""

    # Get text from StringIO
    text = sio.getvalue()

    # Cleanup
    device.close()
    sio.close()

    return text


# This is the main function (the event handler)
# It will be called every time 'S3 event' we defined takes place
# We will set it up in such a way that every time a new PDF is uploaded to our S3 bucket,
# an event is triggered and dispatched to this lambda function

def lambda_handler(event, context):

    s3r = boto3.resource('s3')

    # Get the bucket name and object key from the event details
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = urllib.unquote_plus(event['Records'][0]['s3']['object']['key']).decode('utf8')
    try:
        # Extract name of the pdf file from the event
        pdf_name = event['Records'][0]['s3']['object']['key']

        # download pdf file to our local environment (to the container in which lambda is running)
        s3r.Object(bucket, pdf_name).download_file( "/tmp/" + pdf_name)

        # convert pdf to text, keep the extracted text in 'txt' variable
```

```
    txt = pdf_to_text("/tmp/" + pdf_name)

    # Perform NLP
    words = nltk.word_tokenize(txt.decode('utf-8'))
    pos_word = nltk.pos_tag(words)
    # … sentence spltiting ; syntactiv parsing, NER, ...

    # Compose new file name - replace extension from .pdf to .txt
    txt_file = pdf_name.replace('.pdf','.nlp')

    # Store final results as a temp file, again to the local /tmp
    temp_file = open("/tmp/" + txt_file, "w")
    temp_file.write(str(pos_word))

    # Upload results back to s3, the same bucket, just new file extension (txt instead of pdf)
    s3r.Object(bucket, txt_file).upload_file("/tmp/" + txt_file)

    print "Processed " + pdf_name
    return "success"
except Exception as e:
    print(e)
    print('Error while processing {0} in {1}'.format(key, bucket))
    raise e
```

Part 2 - Packaging Lambda function

1. Create working directory (can be called anything, we will call it `'lambda'`)
```
[ec2 ~]$ mkdir lambda
[ec2 ~]$ cd lambda
```

2. Create python virtual environment (can be called anything, we will call it 'venv')
```
[ec2 lambda]$ virtualenv -p /usr/bin/python2.7 venv
Already using interpreter /usr/bin/python2.7
New python executable in venv/bin/python2.7
Also creating executable in venv/bin/python
Installing setuptools, pip...done.
```

3. Activate virtual environment 'venv'
```
[ec2 lambda]$ source venv/bin/activate
(venv)[ec2-user@ip-10-10-1-55 lambda]$
```

4. Install boto3, pdfminer, nltk
```
(venv)[ec2 lambda]$ pip install boto3
[skipping output]

(venv)[ec2 lambda]$ pip install nltk
[skipping output]

(venv)[ec2 lambda]$ pip install pdfminer
[skipping output]
```

5. Install nltk data library 'punkt' (into current directory, hence `os.environ["NLTK_DATA"] = "."` in the script)
```
(venv)[ec2 lambda]$ python -m nltk.downloader -d ./ punkt
[nltk_data] Downloading package punkt to ./…
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

6. Install nltk data library 'average_perceptron_tagger' (also into current directory)
```
(venv)[ec2 lambda]$ python -m nltk.downloader -d ./ averaged_perceptron_tagger
[nltk_data] Downloading package averaged_perceptron_tagger to ./…
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
```

7. Start creating lambda package (we will call it `pdfnlp.zip`).
   First step - add all python libraries (it's important to first 'cd' to where they are):
```
(venv)[ec2 lambda]$ cd venv/lib/python2.7/site-packages/
(venv)[ec2 site-packages]$ zip -r ../../../../pdfnlp.zip *
[skipping output]
```

8. Add nltk_data libraries 'taggers' and 'tokenizers'. Since we installed them to the 'root' of our working directory, we will 'cd' back there first.
```
(venv)[ec2 site-packages]$ cd ../../../../
(venv)[ec2 lambda]$ zip -r pdfnlp.zip taggers tokenizers
[skipping output]
```

9. Finally, let's add the main function. It should be placed in the 'root' of our main working directory (into 'lambda' directory, where we are right now). Either copy it into here, or vi and cut/paste. Let's call it converter.py ('vi' and then copy/paste the code from Part 1)
```
(venv)[ec2 lambda]$ vi converter.py
[save it once done editing]
```

10. Add the lambda function (`converter.py`) to the package:

```
(venv)[ec2-user@ip-10-10-1-55 lambda]$  zip pdfnlp.zip converter.py
  adding: converter.py (deflated 56%)
```

Any time changes are made to the function, you can just re-run this step, re-adding updated function to the zip, nothing else needs to be done.

11. Quick checkpoint, this is what our current working directory looks like:

```
(venv)[ec2 lambda]$  ls -lt
-rw-rw-r-- 1 ec2-user ec2-user 43271908 Jul 23 02:45 pdfnlp.zip
-rwxrwxr-x 1 ec2-user ec2-user     2849 Jul 23 02:38 converter.py
drwxrwxr-x 3 ec2-user ec2-user     4096 Jul 23 02:25 taggers
drwxrwxr-x 3 ec2-user ec2-user     4096 Jul 23 02:25 tokenizers
drwxrwxr-x 7 ec2-user ec2-user     4096 Jul 23 02:19 venv
```

12. And if we peeked inside our zip file, it's directory structure should look something like this:

```
(venv)[ec2 lambda]$  zipinfo pdfnlp.zip | grep "/$" | grep -v "/."
drwxrwxr-x  3.0 unx        0 bx stor 16-Jul-23 02:19 boto3/
drwxrwxr-x  3.0 unx        0 bx stor 16-Jul-23 02:19 boto3-1.3.1.dist-info/
drwxrwxr-x  3.0 unx        0 bx stor 16-Jul-23 02:19 botocore/
drwxrwxr-x  3.0 unx        0 bx stor 16-Jul-23 02:19 botocore-1.4.39.dist-info/
drwxrwxr-x  3.0 unx        0 bx stor 16-Jul-23 02:19 concurrent/
drwxrwxr-x  3.0 unx        0 bx stor 16-Jul-23 02:19 dateutil/
drwxrwxr-x  3.0 unx        0 bx stor 16-Jul-23 02:19 docutils/
drwxrwxr-x  3.0 unx        0 bx stor 16-Jul-23 02:19 docutils-0.12-py2.7.egg-info/
drwxrwxr-x  3.0 unx        0 bx stor 16-Jul-23 02:19 futures-3.0.5.dist-info/
drwxrwxr-x  3.0 unx        0 bx stor 16-Jul-23 02:19 jmespath/
drwxrwxr-x  3.0 unx        0 bx stor 16-Jul-23 02:19 jmespath-0.9.0.dist-info/
drwxrwxr-x  3.0 unx        0 bx stor 16-Jul-23 02:10 _markerlib/
drwxrwxr-x  3.0 unx        0 bx stor 16-Jul-23 02:19 nltk/
drwxrwxr-x  3.0 unx        0 bx stor 16-Jul-23 02:19 nltk-3.2.1-py2.7.egg-info/
drwxrwxr-x  3.0 unx        0 bx stor 16-Jul-23 02:19 pdfminer/
drwxrwxr-x  3.0 unx        0 bx stor 16-Jul-23 02:19 pdfminer-20140328-py2.7.egg-info/
drwxrwxr-x  3.0 unx        0 bx stor 16-Jul-23 02:10 _markerlib/
drwxrwxr-x  3.0 unx        0 bx stor 16-Jul-23 02:10 pip-6.0.8.dist-info/
drwxrwxr-x  3.0 unx        0 bx stor 16-Jul-23 02:10 pkg_resources/
drwxrwxr-x  3.0 unx        0 bx stor 16-Jul-23 02:19 python_dateutil-2.5.3.dist-info/
drwxrwxr-x  3.0 unx        0 bx stor 16-Jul-23 02:10 setuptools/
drwxrwxr-x  3.0 unx        0 bx stor 16-Jul-23 02:10 setuptools-12.0.5.dist-info/
drwxrwxr-x  3.0 unx        0 bx stor 16-Jul-23 02:19 six-1.10.0.dist-info/
drwxrwxr-x  3.0 unx        0 bx stor 16-Jul-23 02:25 taggers/
drwxrwxr-x  3.0 unx        0 bx stor 16-Jul-23 02:25 tokenizers/
```

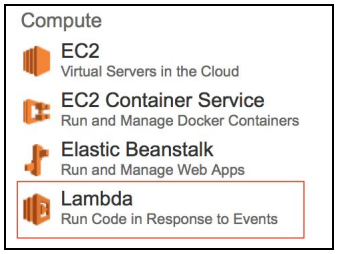
13. Copy pdfnlp.zip to an S3 bucket (in this example we will use bucket 'ipythonv01'):

```
(venv)[ec2 lambda]$  aws s3 cp pdfnlp.zip s3://ipythonv01/
upload: ./pdfnlp.zip to s3://ipythonv01/pdfnlp.zip
```

Alternatively, `pdfnlp.zip` can be uploaded to S3 via AWS console.

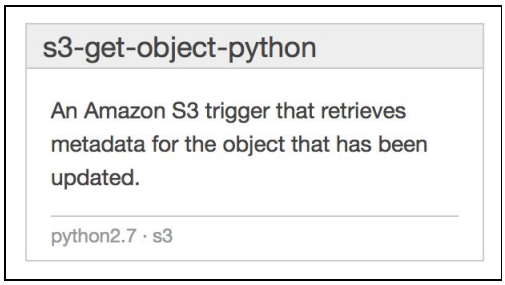
Part 3 - Deploying Lambda function

1. Navigate to Lambda function control panel of AWS Dashboard:

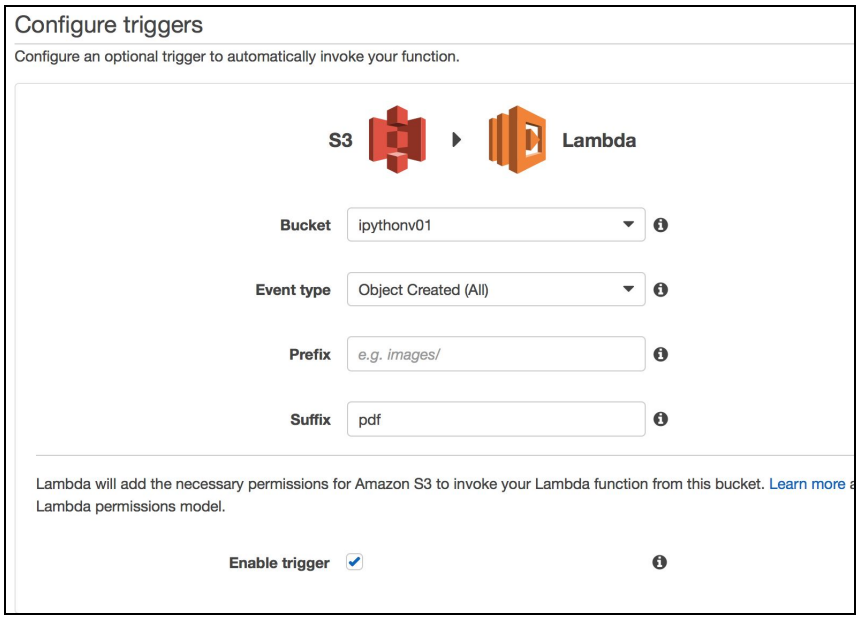


2. Click “Create Lambda Function”

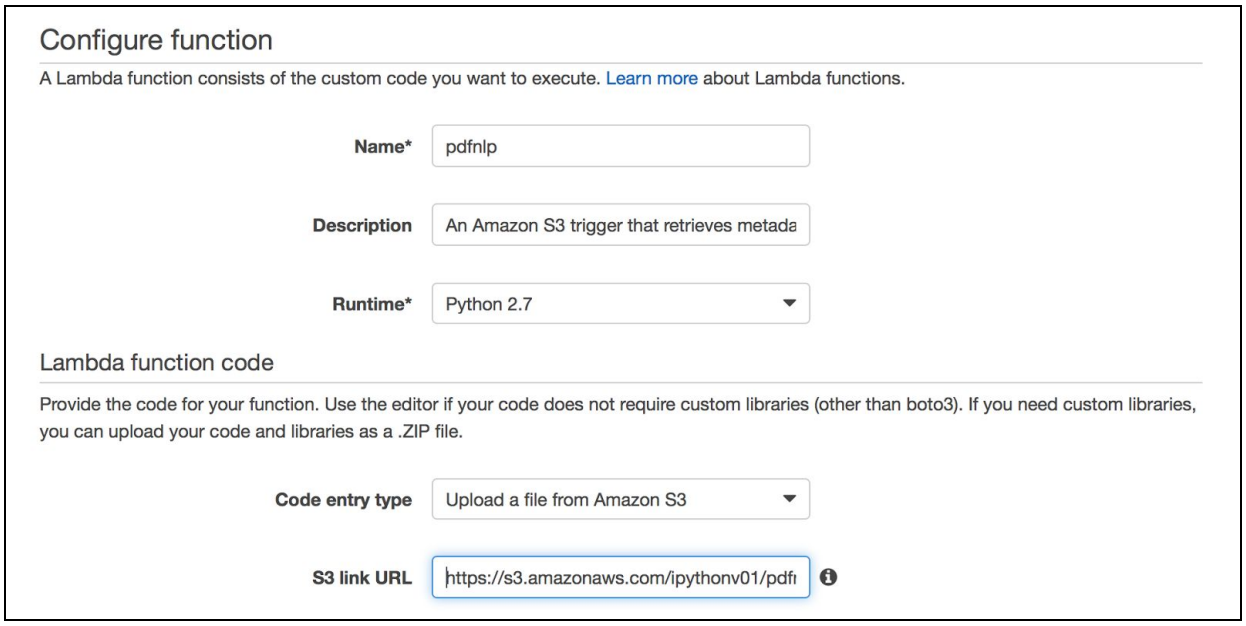
3. In the “Select blueprint”, choose “S3-get-object-python” blueprint



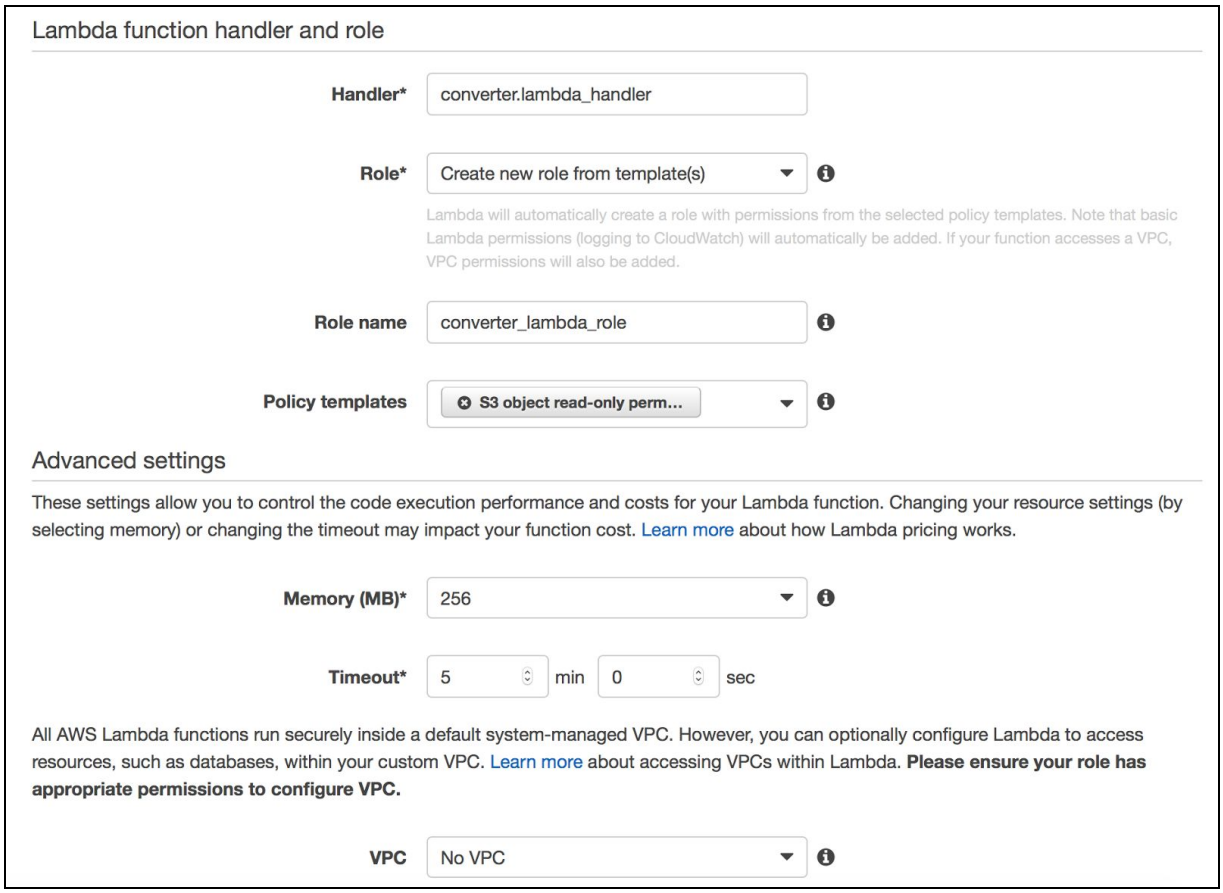
4. Select Bucket where PDFs will be placed, set “Event type” to “Object Created (All)”, suffix to “pdf” and check “Enable trigger” and hit “Next”



5. Give the function a name (pdfnlp in this case), make sure Runtime is Python 2.7, choose code location to “Upload a file from Amazon S3” and populate S3 link URL with http URL to the lambda function zip file (note, has to be “http://”, not “s3://”)



6. On the same screen, specify hander name (name of the file containing lambda function separated with a dot from the name of the lambda handler function)

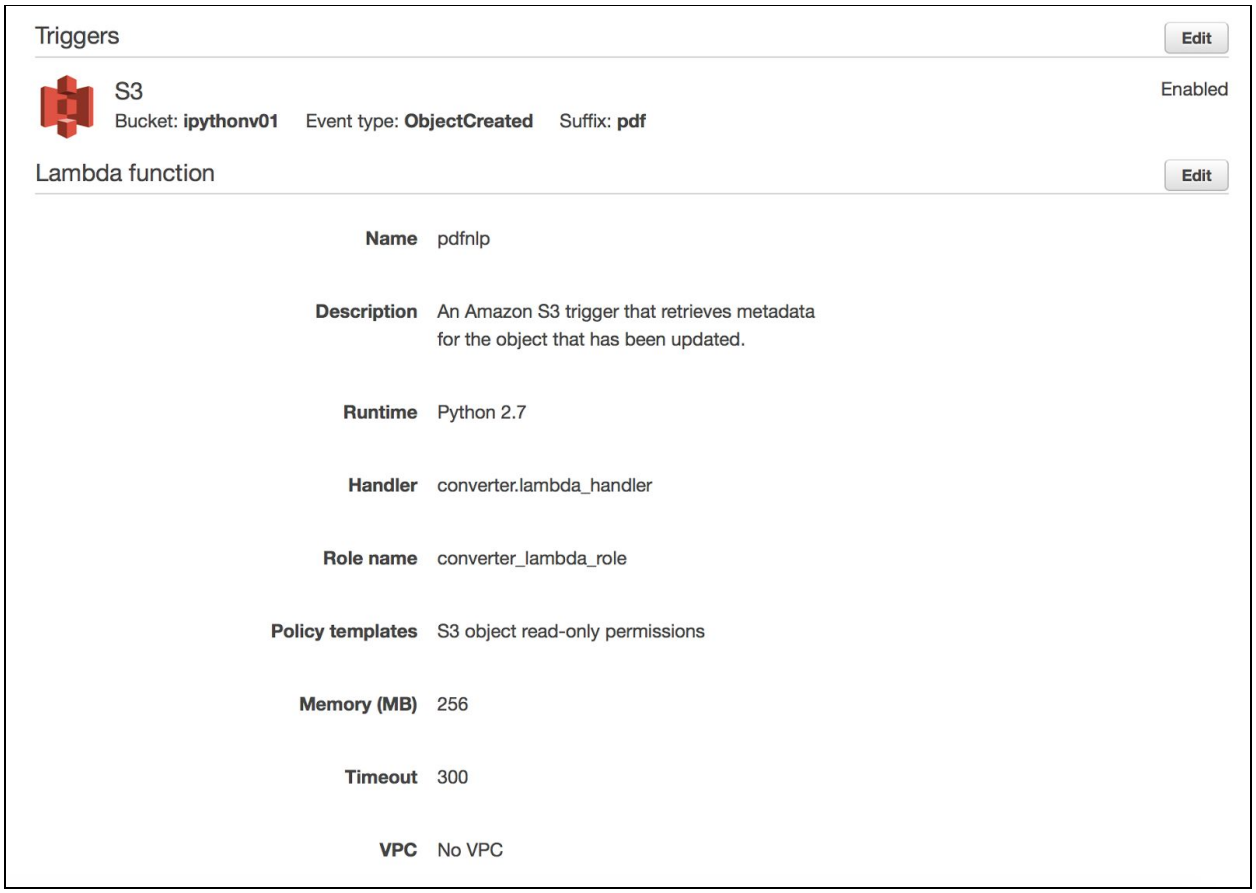


Allow Role to be set to “Create new role from templates”, give it a unique role name, increase memory from 128 to 256Mb and increase timeout to the max available, 5 mins in this case.

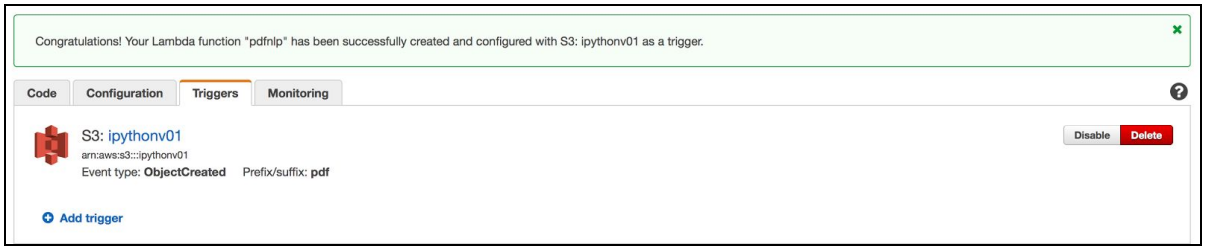
Note on memory: PDF wiles will have to be loaded to memory for parsing/processing, therefore by choosing memory limit here we’re also choosing limit on how big of a PDF file we will be able to process. On one hand it’s tempting to choose maximum available, on the other hand, it will cost more $$ (charges for lambda functions are composed from both memory allocations and runtime).

Hit “Next” to go to the next screen.

7. Review definitions:

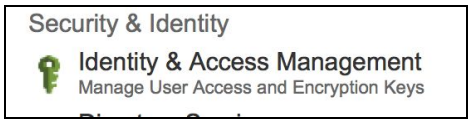


Hit "**`Create function`**". You should see confirmation screen:



8. The function is now ready, however, it has "Read only" permissions for S3. In order to be able to save converted files it needs write permissions as well. Let's correct that.

Navigate to Identity and Access Management console

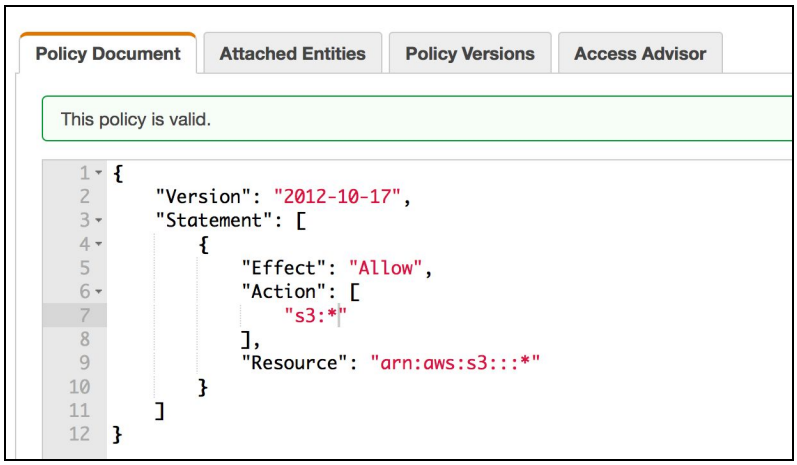


Choose "**`Roles`**" on the left and then search for **`converter_lambda_role`** (or whatever name was given in previous step):

Click on the role
and screen with details will open:

Click on the role which name begins with "`AWSLambdaS3…`", the content will look similar to:

Replace s3:GetObject with "*", which should look like this:



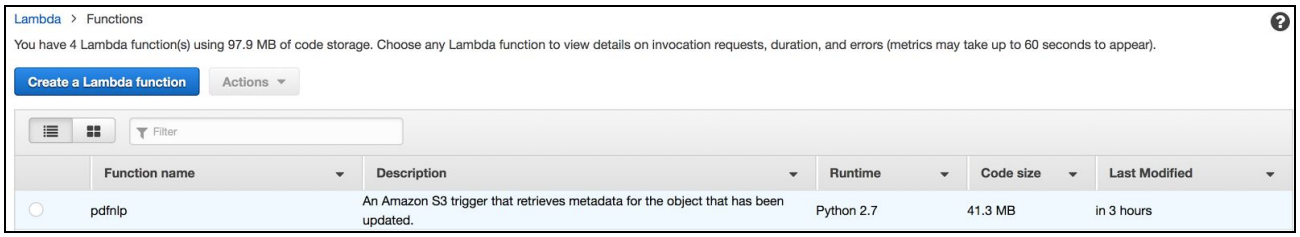
Validate and Save.

Part 4 - Running and monitoring

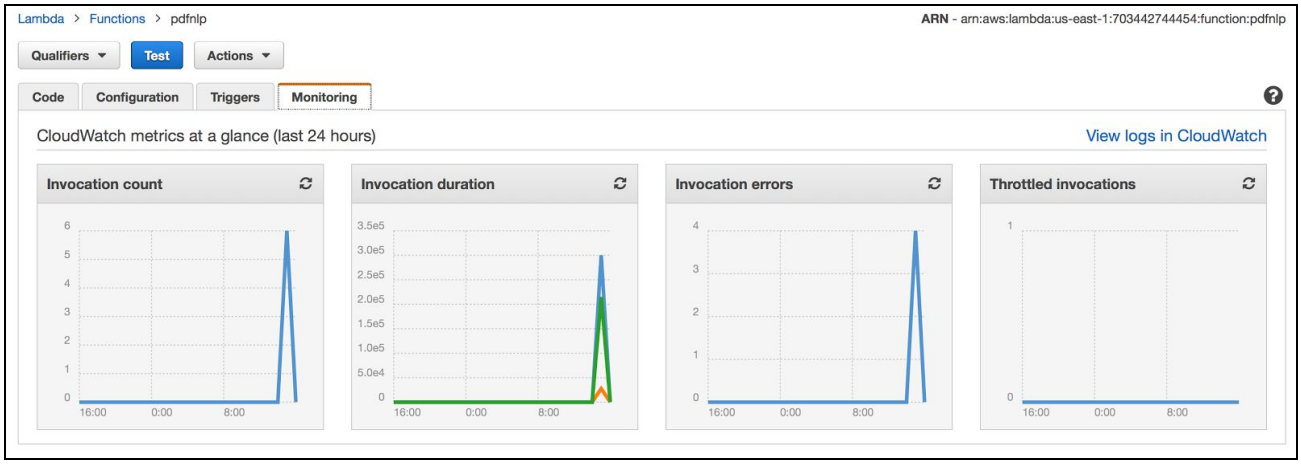
1. To trigger lambda function invocation all we have to do is drop a pdf file into the S3 bucket:

```
[ec2-user@ip-... lambda]$  aws s3 cp s3://og-data-public/budgets/unorganized/aberdeen_SC_13.pdf ./
download: s3://og-data-public/budgets/unorganized/aberdeen_SC_13.pdf to ./aberdeen_SC_13.pdf
[ec2-user@ip-... lambda]$  aws s3 cp aberdeen_SC_13.pdf s3://ipythonv01/
upload: ./aberdeen_SC_13.pdf to s3://ipythonv01/aberdeen_SC_13.pdf
```

2. Next, navigate to the Lambda functions screen:

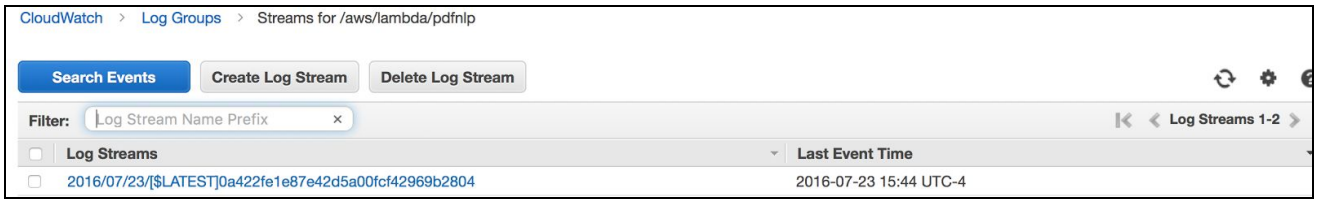


3. Click on the function and you should see the following screen:



Click on the “View logs in CloudWatch” in the upper right corner (just above the “Throttled invocations” chart)

4. Next screen shows this lambda function’s log streams:



Most likely there will be just one, click on it.

5. Finally we will see details of the invocation:

```
▾Processed aberdeen_SC_13.pdf
▾END RequestId: e7757d47-510d-11e6-94dd-d7632898bcd9
▾REPORT RequestId: e7757d47-510d-11e6-94dd-d7632898bcd9 Duration: 61012.43 ms Billed Duration: 61100 ms Memory Size: 256 MB Max Memory Used:
178 MB
```

It took one minute (~61k ms) to convert the file. Also, 178Mb of memory was used (out of 256)
(not very promising performance in terms of speed considering that it was a small ~1Mb PDF)

6. S3 bucked should now contain converted file called aberdeen_SC_13.nlp

aberdeen_SC_13.nlp	Standard	424 KB	Sat Jul 23 15:45:46 GMT-400 2016
aberdeen_SC_13.pdf	Standard	883.4 KB	Sat Jul 23 15:44:44 GMT-400 2016