Amazon S3 integration using (i|o)m_python

NXLog can both send and receive events to and from an Amazon S3 cloud object storage. The NXLog python modules for input and output are used (`im_python` and `om_python`) as well as boto3 the AWS SDK for Python. More information about `boto3` can be found at https://aws.amazon.com/sdk-for-python/

Configuring boto3

The first step is to install and configure boto3 into your system. Boto3 can be installed can using `pip` or your package manager respectively.

Using pip

```
$ pip install boto3
```

Using the package manager of a Debian based distro

```
# apt-get install python-boto3
```

After creating an AWS service account, your local setup requires some configuration. Two files responsible for selecting the default region as well as your credentials must be created. This can be done interactively, if you have the AWS CLI installed or manually, by editing the files shown bellow. Credentials for your AWS account can be found in the IAM Console. You can create or use an existing user. Go to manage access keys and generate a new set of keys.

~/.aws/config

```
[default]
region=eu-central-1
```

~/.aws/credentials

```
[default]
aws_access_key_id = YOUR_ACCESS_KEY
aws_secret_access_key = YOUR_SECRET_KEY
```

More information about the initial setup and the credentials can be found at https://boto3.readthedocs.io/en/latest/guide/quickstart.html and https://boto3.readthedocs.io/en/latest/guide/configuration.html

NOTE The region and credential configuration can also be hardcoded in the code however, this is not considered a good practice.

AWS S3 Buckets, objects, keys and structure

Both the input and output python scripts interact with a bucket on Amazon S3. The scripts will not create, delete or alter the bucket and any of its properties, permissions or management options. It is the responsibility of the user to create the bucket, provide the appropriate permissions (ACL) and further configure any options like lifecycle options, replication options, encryption, etc. Similarly, the scripts do not alter the storage class of the objects stored or any other properties or permissions. General information about Amazon S3 can be found at https://docs.aws.amazon.com/AmazonS3/latest/gsg/GetStartedWithS3.html and the Amazon S3 console at https://docs.aws.amazon.com/AmazonS3/latest/gsg/GetStartedWithS3.html

Amazon S3 stores objects inside containers called buckets. There is a finite number of buckets that you can have and an infinite number of objects that you can store. We selected a schema where we store events on a single bucket and each object has a key that references the server (or service) name, the date and the event received time. Even though Amazon S3 uses a flat structure to store object, objects with similar key prefixes are grouped together resembling the structure of a file system. The following is a visual representation of the naming scheme used. Note that the key name in the deepest level represent time, however Amazon S3 uses the `:` character as a special character and to avoid escaping we selected the `.` character to substitute it.

- MYBUCKET/
  - SERVER01/
    - 2018-05-17/
      - 12.36.34.1
      - 12.36.35.1
    - 2018-05-18/
      - 10.46.34.1
      - 10.46.35.1
      - 10.46.35.2
      - 10.46.36.1
  - SERVER02/
    - 2018-05-16/
      - 14.23.12.1
    - 2018-05-17/
      - 17.03.52.1
      - 17.03.52.2
      - 17.03.52.3

Storing events to Amazon S3

This section explains the python script and NXLog configuration needed to store events to an

Amazon S3 cloud object storage.

Configure NXLog similarly to the following. For simplicity we are reading events from a file.

nxlog.conf

```
<Input in>
    Module      im_file
    File        "input.log"
    SavePos FALSE
    ReadFromLast    FALSE
</Input>

<Output out>
    Module      om_python
    PythonCode  s3_write.py
</Output>

<Route exec_to_file>
    Path    in => out
</Route>
```

The python script used to store events on Amazon S3.

s3_write.py

```
#!/usr/bin/python
import nxlog
import boto3
from botocore.exceptions import ClientError
from datetime import datetime

BUCKET = 'MYBUCKET'
SERVER = 'MYSERVER'

lastdt = datetime.now()

def counter(rdtime, v=[0]):
        global lastdt
        if lastdt < rdtime:
             lastdt = rdtime
             v[0]=0
        v[0]+=1
        return v[0]

def write_data(event):
    nxlog.log_debug('Python alerter received event')


    raw = event.get_field('raw_event')
    rtime = event.get_field('EventReceivedTime')
    dt = datetime.strptime(rtime, "%Y-%m-%d %H:%M:%S")
    key = SERVER + '/' + rtime.replace(' ', '/').replace(':', '.',3) + '.' +
str(counter(dt))


    #Insert credentials at ~/.aws/credentials
    client = boto3.client('s3')
    #You can hardcode but not advisable
    #client = boto3.client('s3', aws_access_key_id='XXXXX',
aws_secret_access_key='XXXXXXXX')
    try:
         client.put_object(Body=raw, Bucket=BUCKET, Key=key)
    except ClientError as e:
    nxlog.log_error("Error: {0}".format(e))
```

Edit the `BUCKET` and `SERVER` variables on the code. Events are stored in the Amazon S3 bucket with object key names comprised from the server name, date in YYYY-MM-DD format, time in HH.MM.SS format, plus a counter since events can be received on the same second.

Retreving events from Amazon S3

This section explains the python script and NXLog configuration needed to retrieve events from an

Amazon S3 cloud object storage.

Configure NXLog similarly to the following. For simplicity we are saving events to a file.

nxlog.conf

```
<Input in>
    Module      im_python
    PythonCode  s3_read.py
</Input>

<Output out>
    Module      om_file
    File        "output.log"
</Output>

<Route exec_to_file>
    Path    in => out
</Route>
```

s3_read.py

```
#!/usr/bin/python

import boto3
import nxlog

BUCKET = 'MYBUCKET'
SERVER = 'MYSERVER'
# 1000 Keys is the maximum allowed from the AWS API
MAXKEYS = 1000

POLL_INTERVAL = 30

def save_key(key):
    fo = open('lastkey.log', 'wb', 0)
    fo.write(key)
    fo.close()

def load_key():
    try:
         fo = open('lastkey.log', 'rb', 0)
         key = fo.read()
         fo.close()
         return key
    except IOError:
         return ''

def read_event(module):
    nxlog.log_debug('Checking for new archives')
    #Insert credentials at ~/.aws/credentials
    client = boto3.client('s3')
    #You can hardcode but not advisable
    #client = boto3.client('s3', aws_access_key_id='XXXXX',
aws_secret_access_key='XXXXXXXX')
    lastkey = load_key()
    keycount = MAXKEYS

    while MAXKEYS == keycount:
         if lastkey == '':
              data = client.list_objects_v2(Bucket=BUCKET, Prefix=SERVER,
MaxKeys=MAXKEYS);
         else:
              data = client.list_objects_v2(Bucket=BUCKET, Prefix=SERVER,
MaxKeys=MAXKEYS, StartAfter=lastkey);

         keycount = data['KeyCount']

         if keycount > 0:
              for obj in data['Contents']:
                   lastkey = obj['Key']
                   raw = client.get_object(Bucket=BUCKET, Key=lastkey)
                   line = raw['Body'].read().decode('utf-8')
                   event = module.new_logdata()
                   event.set_field('raw_event', line)
                   event.post()
                   save_key(lastkey)

    module.add_read_event(POLL_INTERVAL)
```

Edit the `BUCKET` and `SERVER` variables on the code. The `POLL_INTERVAL` is the time the script will wait before checking again for new events. The `MAXKEYS` variable should be fine in all cases as the default value of 1000 keys. The script keeps track of the last object retrieved from Amazon S3 by means of a file called `lastkey.log`, stored locally. Even in the event of an abnormal termination the script will continue from where it stopped. You can delete (or even edit) the `lastkey.log` file to reset that behavior.

Serialization and compression

In the previous examples only the `raw_event` field was stored in the objects. An easy way to store more than one field is to "pickle" (better known as serialize or marshal) them together. The following lines of python code show how to do so for all the fields of an event.

Pickle events

```
import pickle



all = {}
for field in event.get_names():
     all.update({field: event.get_field(field)})

newraw = pickle.dumps(all)

client.put_object(Body=newraw, Bucket=BUCKET, Key=key)
```

Compressing the events with gzip is also possible. The following python code explains how to do so.

Gzip events

```
import StringIO
import gzip

out = StringIO.StringIO()
with gzip.GzipFile(fileobj=out, mode="w") as f:
     f.write(newraw)

gzallraw = out.getvalue()

client.put_object(Body=gzallraw, Bucket=BUCKET, Key=key)
```