WHITE PAPER

Enabling Agile Data Analysis with AWS and Greenplum

By XXXXXX

Pivotal

Table of Contents

Business Problem

At Pivotal, we work with many customers who are looking to adopt an agile approach to how they develop and deploy applications. For applications, this is done by utilizing continuous integration with products like Pivotal Cloud Foundry and methodologies of extreme programming and test-driven development. A common request is, “How can we apply these types of methodologies for data when many of our users are still struggling to just get access to data that they need to analyze? The time it takes to get data loaded just to determine if there is any value in the data presents a problem towards a ‘fail fast’ model of development (i.e., nothing useful in this data, let’s throw this dataset out and try a new one).” These customers and their data scientists really need a way to get data into a database like Greenplum quickly so they can iterate on the data, perform the necessary analysis, and provide data “rules” for database designers and developers to incorporate this data into modeled data warehouses.

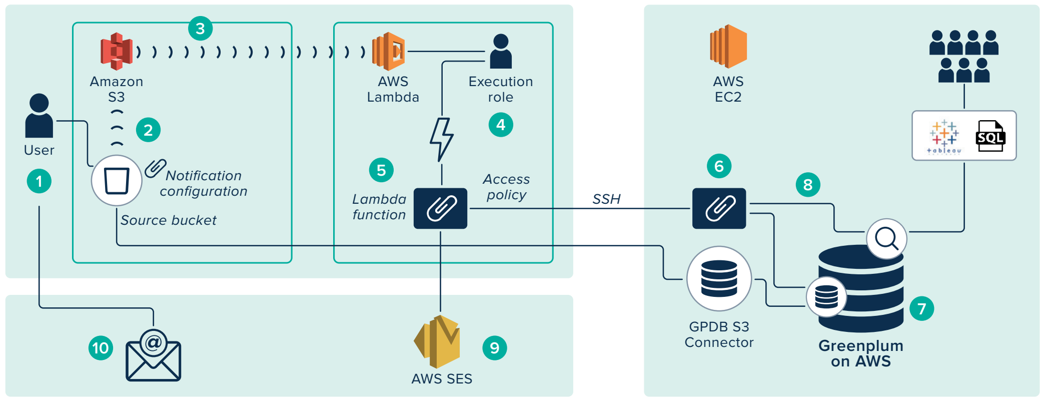
The solution presented in this paper utilizes Greenplum on Amazon Web Services (AWS)—along with several AWS components—to automate a “no-IT touch” mechanism to upload data to AWS S3 and be able to quickly access this data using standard reporting and SQL-based tools from Greenplum.

The following components are highlighted in this paper:

- AWS S3
- AWS S3 notifications and Lambda handler
- AWS SES for email notifications
- Greenplum on AWS
- Greenplum S3 connector
- Custom scripts

High-Level Flow

An extended version of the AWS Lamdba tutorial is available here.



Analytic Sandbox

1. A user uploads an object to the source bucket in Amazon S3.
2. Amazon S3 detects the object-created event.
3. Amazon S3 publishes the s3:ObjectCreated:* event to AWS Lambda by invoking the Lambda function and passing event data as a function parameter.
4. AWS Lambda executes under the role that you specified at the time you created the Lambda function.
5. AWS Lambda executes the Lambda function.
6. The Lambda function connects via SSH to EC2 Greenplum master instance and executes the script for generating the objects in Greenplum. The function passes the S3 bucket, key, and (optionally) the table name to create (from S3 metadata).
7. Script determines the external table headings by parsing the "header" row from the S3 key in the bucket (assumed to be the first row). The Greenplum S3 connector is used in the creation of the external table. Admin permissions are optionally granted to the generated external table to the specified admin_role from the scripts configuration file.
8. Script generates a user view on the external table as a level of abstraction between the user and the underlying table. Admin and Viewer permissions are optionally granted to the generated view to the specified admin_role and viewer_roles from the scripts configuration file.
9. Lambda function receives a response from the script, determines if the response was successful or failed, and sends an email to the configured email recipients.
10. Configured users receive email that file is ready for analysis

At this point, users can start querying the views from their tool of choice connected to Greenplum using fully supported ANSI SQL.

Implementation Steps

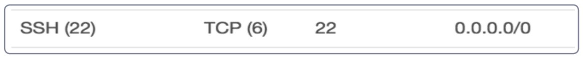
The name of the project is greenplumSailfish. The name was chosen because the sailfish is the fastest fish in the sea and this project enables you to quickly "fish" for data. Below are the implementation steps to complete the project:

Greenplum

1. Install Greenplum on AWS (if it's not already installed). For this paper, I've used the Pivotal Greenplum marketplace offering. You can watch the how-to video here.

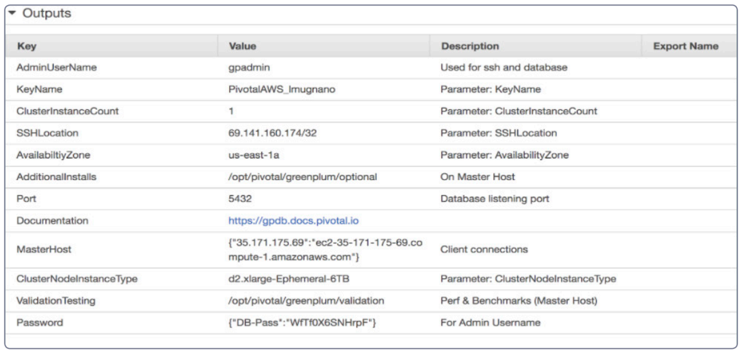
   Note: You have the option through the Marketplace to deploy Greenplum under hourly pricing or use an existing Pivotal data license for deploying.

2. For ease of deployment, for this paper I've set an inbound rule on the AWS Greenplum Marketplace security group to fully open up SSH. This will allow the Lambda function to invoke the code on the EC2 instance without the added complexity of custom AWS VPCs, subnets, etc.



SSH (22)	TCP (6)	22	0.0.0.0/0

3. Note the Greenplum master's public IP address, which you'll use later in the process. If you use the Greenplum Marketplace and provided CloudFormation scripts, you'll find the master IP address in the output of the CloudFormation (MasterHost line below).



Outputs

Key	Value	Description	Export Name
AdminUserName	gpadmin	Used for ssh and database	
KeyName	PivotalAWS_lmugnano	Parameter: KeyName	
ClusterInstanceCount	1	Parameter: ClusterInstanceCount	
SSHLocation	69.141.160.174/32	Parameter: SSHLocation	
AvailabiltiyZone	us-east-1a	Parameter: AvailabilityZone	
AdditionalInstalls	/opt/pivotal/greenplum/optional	On Master Host	
Port	5432	Database listening port	
Documentation	https://gpdb.docs.pivotal.io		
MasterHost	{"35.171.175.69":"ec2-35-171-175-69.compute-1.amazonaws.com"}	Client connections	
ClusterNodeInstanceType	d2.xlarge-Ephemeral-6TB	Parameter: ClusterNodeInstanceType	
ValidationTesting	/opt/pivotal/greenplum/validation	Perf & Benchmarks (Master Host)	
Password	{"DB-Pass":"WfTf0X6SNHrpF"}	For Admin Username	

4. Git clone the greenplumSailfish repo onto the Greenplum master. You can put the code under any directory owned by gpadmin. If you can't clone, then copy greenplumSailfish.py, greenplumSailfish_cfg.py, and argparse.py. Note the full path to the scripts; you'll use this later in the process. The code is available in a GitHub repository to download onto your Greenplum master.

```
git clone https://github.com/lmugnano/greenplum-sailfish
```

Pivotal

5. Edit the greenplumSailfish_cfg.py script to set the static parameters for the script using these sample settings:

```
import sys
sys.dont_write_bytecode = True
# Operation
action = 'genS3externalTable'
''' Valid values are

genS3externalTable
'''

# Connection Configuration
host = 'localhost'
port = 6432
db = 'gpadmin'
user = 'gpadmin'

# Greenplum S3 Connector Configuration file
s3_config = '/home/gpadmin/s3/s3.conf'

# Naming convention for generated objects
ext_tbl_prefix = 'ext_'
ext_tbl_suffix = ''
view_prefix = 'vw_'
view_suffix = ''

# Permissions of generated objects
admin_role = 'admin_role'
viewer_role = 'ro_role'
```

Amazon Web Services (AWS)

1. If you aren't already using AWS SES, validate the email address you want to use to send email from the Lamdba function. Please see the AWS documentation on this step.

2. Create a new S3 bucket if you don't already have one. In your S3 bucket, create a folder called "sandbox." This will be the folder that we'll trigger off of later in the process. (No special properties or permissions are set on the example bucket.) As the owner of the bucket, my AWS login has permissions to "list objects," "write objects," "read bucket permissions," and "write bucket permissions." The required permissions for the functionality in this paper are "list objects" and "write objects" for users who would be putting files into the bucket.

3. In your sandbox folder, create a new folder called "config."

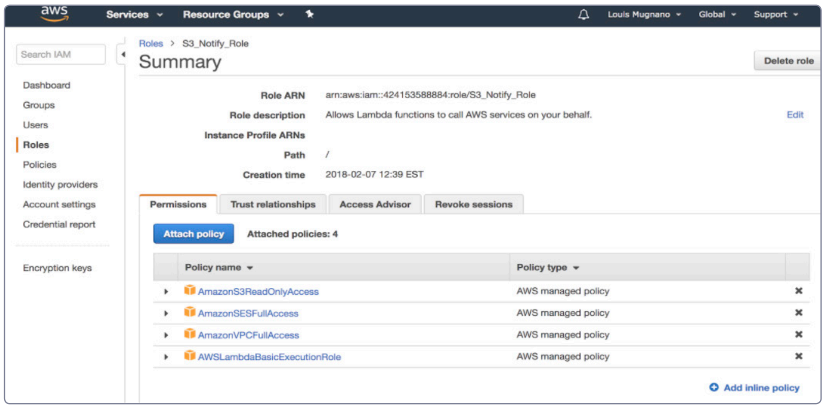
4. Put the following files into the sandbox/config folder on your S3 bucket:

a. The PEM file for connecting via SSH to your Greenplum master EC2 instance. Name this file "PivotalGreenplum_master.pem."

b. A file named sandbox_config.json. You can edit this example file for your specific implementation:

```
{
"email_sender":"<email address validated via SES>",
"email_recipient":"<email recipients for email notification>",
"aws_region":"us-east-1",
"gpdb_master_host":"<Greenplum master host IP>",
"script":"source .bashrc; cd <dir where you put python code on master>; python
greenplumSailfish.py -cfg greenplumSailfish_cfg"
}
```

5. Create a new role for the Lambda function and choose Lambda as a trusted resource.



Summary

Role ARN arn:aws:iam::424153588884:role/S3_Notify_Role
Role description Allows Lambda functions to call AWS services on your behalf.
Instance Profile ARNs
Path /
Creation time 2018-02-07 12:39 EST

Attached policies: 4

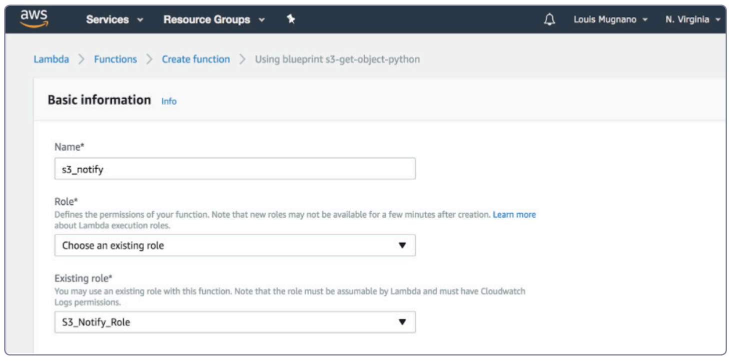
Policy name	Policy type
AmazonS3ReadOnlyAccess	AWS managed policy
AmazonSESFullAccess	AWS managed policy
AmazonVPCFullAccess	AWS managed policy
AWSLambdaBasicExecutionRole	AWS managed policy

The following policies are required:

AWSLambdaBasicExecution AmazonS3ReadOnlyAccess	Required for Lambda functions using S3.
AmazonSESFullAccess	This is specifically needed because our Lambda function sends email using the SES service.
AmazonVPCFullAccess	Gives permissions for Lambda to access your EC2 instance.

6. Create a Lambda function (I used python for language) and chose blueprint s3-get-object-python from this page.

Pivotal


aws
Services
Resource Groups
Louis Mugnano
N. Virginia
Lambda > Functions > Create function > Using blueprint s3-get-object-python
Basic information
Info

Name*

s3_notify

Role*

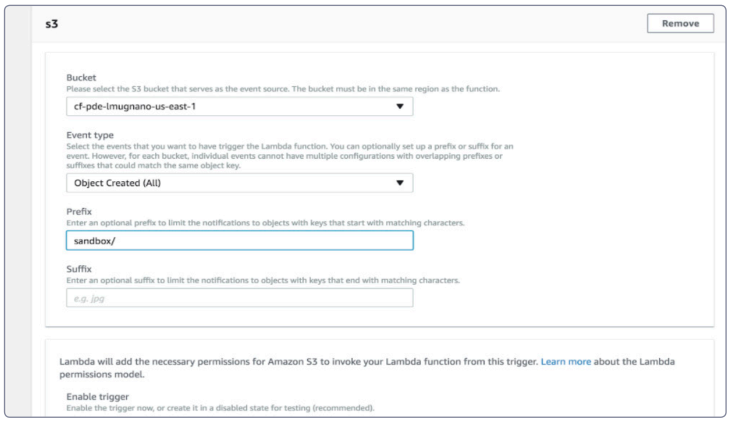
Defines the permissions of your function. Note that new roles may not be available for a few minutes after creation. Learn more about Lambda execution roles.

Choose an existing role

Existing role*

You may use an existing role with this function. Note that the role must be assumable by Lambda and must have Cloudwatch Logs permissions.

S3_Notify_Role



Bucket

Please select the S3 bucket that serves as the event source. The bucket must be in the same region as the function.

cf-pde-lmugnano-us-east-1

Event type

Select the events that you want to have trigger the Lambda function. You can optionally set up a prefix or suffix for an event. However, for each bucket, individual events cannot have multiple configurations with overlapping prefixes or suffixes that could match the same object key.

Object Created (All)

Prefix

Enter an optional prefix to limit the notifications to objects with keys that start with matching characters.

sandbox/

Suffix

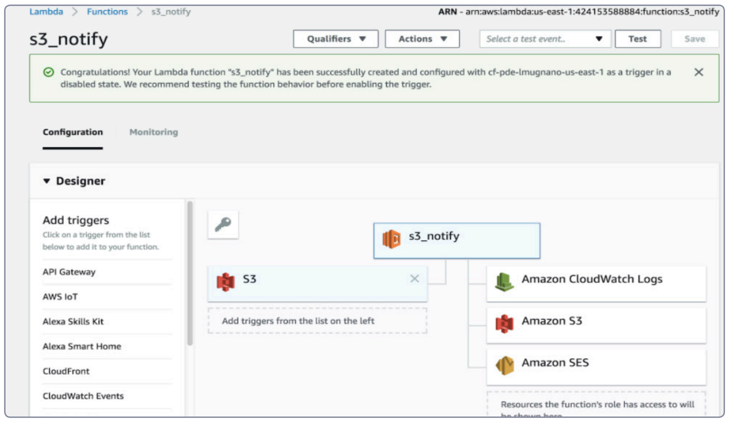
Enter an optional suffix to limit the notifications to objects with keys that end with matching characters.

e.g. jpg

Lambda will add the necessary permissions for Amazon S3 to invoke your Lambda function from this trigger. Learn more about the Lambda permissions model.

Enable trigger

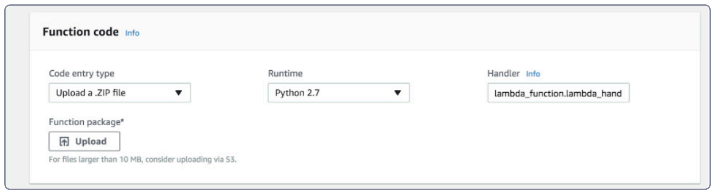
Enable the trigger now, or create it in a disabled state for testing (recommended).



ARN - arn:aws:lambda:us-east-1:424153588884:function:s3_notify

Congratulations! Your Lambda function "s3_notify" has been successfully created and configured with cf-pde-lmugnano-us-east-1 as a trigger in a disabled state. We recommend testing the function behavior before enabling the trigger.

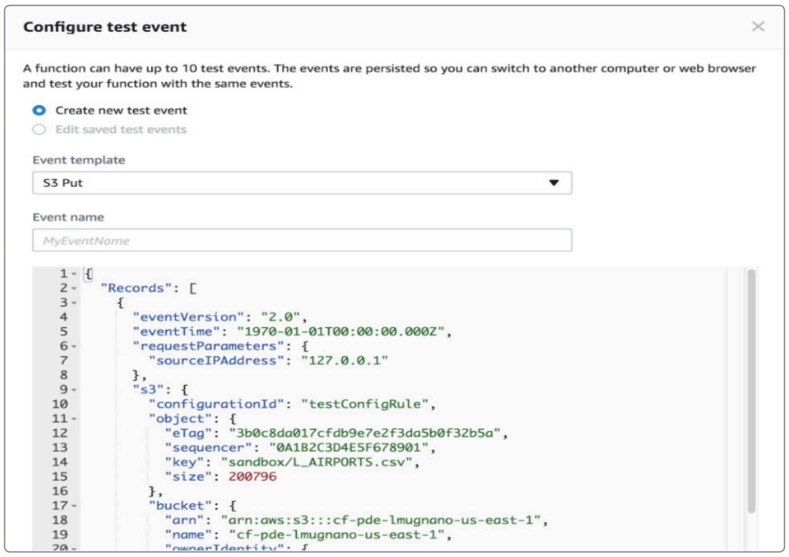
7. Manually edit and package up the Lambda function:
    a. Building dependency package for Lambda function is in Appendix 1.
    b. The Lambda function itself is in Appendix 2.
8. Upload ZIP package to Lambda function and save.


Function code Info

Code entry type	Runtime	Handler
Upload a .ZIP file	Python 2.7	lambda_function.lambda_hand

Function package*
Upload
For files larger than 10 MB, consider uploading via S3.

9. Copy a CSV file into your S3 bucket under the sandbox directory. The first line should be the header for the columns in the file (you can pick a file from the Greenplum sandbox tutorial faa dataset; the test in this paper uses L_AIRPORTS.csv). Configure the test event (I named it s3notifytest). If you copy in the test from Appendix 3, you just need to change the items in bold to match the specifics of the test file you have under your S3 bucket.


Configure test event

A function can have up to 10 test events. The events are persisted so you can switch to another computer or web browser and test your function with the same events.

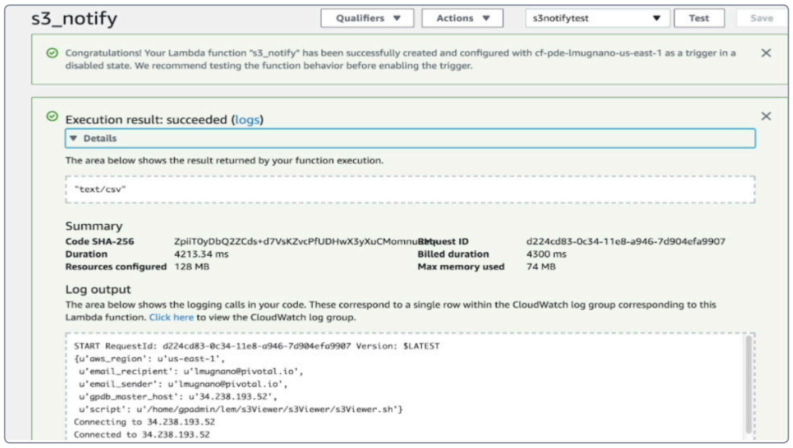
[✓ Create new test event]
Edit saved test events

Event template: S3 Put

Event name: MyEventName

```
{
  "Records": [
    {
      "eventVersion": "2.0",
      "eventTime": "1970-01-01T00:00:00.000Z",
      "requestParameters": {
        "sourceIPAddress": "127.0.0.1"
      },
      "s3": {
        "configurationId": "testConfigRule",
        "object": {
          "eTag": "3b0c8da017cfdb9e7e2f3da5b0f32b5a",
          "sequencer": "0A1B2C3D4E5F678901",
          "key": "sandbox/L_AIRPORTS.csv",
          "size": 200796
        },
        "bucket": {
          "arn": "arn:aws:s3:::cf-pde-lmugnano-us-east-1",
          "name": "cf-pde-lmugnano-us-east-1",
          "ownerIdentity": {
```

10. Run the test by choosing the s3notifytest test and pressing the Test button.



Congratulations! Your Lambda function "s3_notify" has been successfully created and configured with cf-pde-lmugnano-us-east-1 as a trigger in a disabled state. We recommend testing the function behavior before enabling the trigger.

Execution result: succeeded (logs)

Details

The area below shows the result returned by your function execution.

```
"text/csv"
```

Summary

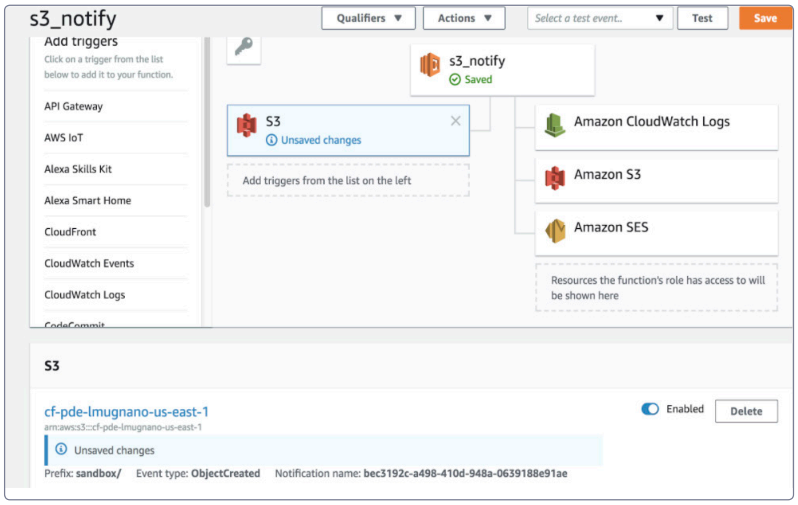
Code SHA-256	ZpiiT0yDbQ2ZCds+d7VsKZvcPfUDHwX3yXuCMomnu	**Request ID**	d224cd83-0c34-11e8-a946-7d904efa9907
Duration	4213.34 ms	**Billed duration**	4300 ms
Resources configured	128 MB	**Max memory used**	74 MB

Log output

The area below shows the logging calls in your code. These correspond to a single row within the CloudWatch log group corresponding to this Lambda function. Click here to view the CloudWatch log group.

```
START RequestId: d224cd83-0c34-11e8-a946-7d904efa9907 Version: $LATEST
{u'aws_region': u'us-east-1',
 u'email_recipient': u'lmugnano@pivotal.io',
 u'email_sender': u'lmugnano@pivotal.io',
 u'gpdb_master_host': u'34.238.193.52',
 u'script': u'/home/gpadmin/lem/s3Viewer/s3Viewer/s3Viewer.sh'}
Connecting to 34.238.193.52
Connected to 34.238.193.52
```

11. Double-check and then enable the S3 notification.



cf-pde-lmugnano-us-east-1
arn:aws:s3:::cf-pde-lmugnano-us-east-1
Unsaved changes
Prefix: sandbox/ Event type: ObjectCreated Notification name: bec3192c-a498-410d-948a-0639188e91ae

12. Save the function.

13. Test by uploading a file to the S3 bucket. When a file is uploaded, you should get an email with this subject:

```
SUCCESS: Amazon S3 file upload on bucket cf-pde-lmugnano-us-east-1
```

Email body example:

```
File sandbox/otp200912.gz was uploaded to S3 bucket cf-pde-lmugnano-us-east-1

Analytics auto enablement process returned with:
SUCCESS: File uploaded and is available for reporting with view s3_sandbox.
vw_otp200912
```

14. You can view logs and monitor invocations for Lambda functions from CloudWatch.

Additional Tasks/Future Considerations

1. This paper has intentionally left out topics around VPC and security groups because those would differ from customer to customer. As part of the Lambda function configuration, you can specify that the Lambda function run under your specific VPC and security groups. See AWS documentation for details on this.
2. The initial minimum custom code built for the Greenplum portion of the implementation is being provided via a static public GitHub at the time of this writing. This code is a minimum viable product for purposes of this paper and the code will continue to add functionality as we work with our customers and iterate on the development. Greenplum customers can gain access to the newest code by emailing the author and being added to the private Git repository. To just get an idea of what the code generates, I included examples of log output from a sample run in Appendix 4.
3. Accessing data from S3 directly via Greenplum enables quick data analysis and allows you to exploit the full capabilities of the database, but for performance reasons, it would often make sense to load the data directly into Greenplum internal tables. This is left out of the paper, but one idea to enable this is to utilize user-defined metadata on the S3 object where the user can decide to "cache" the data in Greenplum. Based on that, the greenplumSailfish.py script would generate a permanent table and load the data directly into Greenplum. The main reason for the view abstraction is so the user can make this decision later simply by repointing the view to the internal table instead of the external table without impacting the users accessing the data.
4. With implementation of #3, a data lifecycle management policy also needs to be defined with the users so that space on the database can be managed properly (i.e., at some point, we'll "flush the cache").
5. Heavy usage of a mechanism like this will bloat the catalog due to the creating and dropping of multiple objects throughout the process. Catalog maintenance is critical for performance of any Greenplum database where users are allowed to directly create objects in the database.

Conclusion

This paper includes the “long version” of the steps involved to set up an end-to-end solution like this. Most of the steps can and will ultimately be scripted, and Pivotal will look at how to simplify the configuration of a solution like this within our marketplace offerings. Even with these simplifications we create, it’s still beneficial to understand what’s “under the hood,” which is what this paper provides.

Appendix 1
(Lamdba Function Dependency ZIP Package)

Below are the steps I used to build the lambda_function.zip package for uploading to AWS Lamdba. This package is needed because Lambda doesn't include some of the python dependencies required by the script.

Note: The built package (lambda_function.zip) is available on the greenplumSailfish Git repo if you don't want to build it yourself.

1. Spin up an AWS Redhat linux EC2 instance (t2micro is fine).
2. SSH to AWS instance.
3. Sudo to root: sudo su - root
4. Install pip:
    a. curl "https://bootstrap.pypa.io/get-pip.py" -o "get-pip.py"
    b. python get-pip.py
    c. pip -V
5. Install virtualenv: pip install virtualenv
6. Install python in virtualenv: virtualenv -p /usr/bin/python2.7 /root/code
7. Activate environment: source /root/code/bin/activate
8. Install pycrypto: pip install pycrypto
9. Install paramiko: pip install paramiko
10. Install zip: yum install zip -y
11. Install unzip: yum install unzip -y
12. Add libraries to zip:
    a. cd path/to/my/helloworld-env/lib/python2.7/site-packages
    b. zip -r ~/lambda_function.zip .
    c. cd path/to/my/helloworld-env/lib64/python2.7/site-packages
    d. zip -r ~/lambda_function.zip .
13. Download lambda_function.zip and add any handwritten python code to the ZIP file for deployment to AWS. The custom lambda_function is in Appendix 2 below.

Appendix 2 (Lambda Function)

Create a file called lambda_function.py and put the following code into the file. Include this file in the lambda_function.zip package you created in Appendix 1.

Note: The built package (lambda_function.zip) is available on the greenplumSailfish Git repo if you don't want to build it yourself. This code is included into the lambda_function.zip package.

```
from __future__ import print_function

import json
from pprint import pprint
import urllib
import boto3
from botocore.exceptions import ClientError
import paramiko
import ntpath

print('Loading function')

# ---------------------------------------------------------------------------
# Global Variables
# ---------------------------------------------------------------------------
s3 = boto3.client('s3')
gpdb_host = ''
email_sender = ''
email_recipient = ''
aws_region = ''
config_dir = 'sandbox/config/'
script = ''

# ---------------------------------------------------------------------------
# Init
# ---------------------------------------------------------------------------
def initialize(bucket):

    global gpdb_host
    global email_sender
    global email_recipient
    global aws_region
global script

try:

   data = s3.get_object(Bucket=bucket, Key=config_dir + "sandbox_config.json")
   json_config = json.loads(data['Body'].read())
   pprint(json_config)
   gpdb_host = json_config["gpdb_master_host"]
   email_sender = json_config["email_sender"]
   email_recipient = json_config["email_recipient"]
   aws_region = json_config["aws_region"]
   script = json_config["script"]

except Exception as e:
   print(e)
   print('Error getting sandbox_config object from bucket {}. Make sure it exists and
   your bucket is in the same region as this function.'.format(bucket))
   raise e

# ---------------------------------------------------------------------------
# Function to send mail
#
# How to send email is from:
# https://docs.aws.amazon.com/ses/latest/DeveloperGuide/send-using-sdk-python.html
# Had to add AmazonSESFullAccess to the role and had to verify email address from SES
# ---------------------------------------------------------------------------
def send_mail(bucket, key, rc, response):

   SENDER = email_sender
   RECIPIENT = email_recipient
   AWS_REGION = aws_region

   # The subject line for the email.
   if rc==0:
     SUBJECT = "SUCCESS: "
   else:
     SUBJECT = "FAILED: "
     SUBJECT += "Amazon S3 file upload on bucket " + bucket

# The email body for recipients with non-HTML email clients.
BODY_TEXT = ("File " + key + " was uploaded to S3 bucket " + bucket + "\r\n\r\n"

"Analytics auto enablement process returned with:\r\n" +
```

Pivotal

```
      response)
    # The character encoding for the email.
    CHARSET = "UTF-8"

    # Create a new SES resource and specify a region.
    client = boto3.client('ses',region_name=AWS_REGION)

    # Try to send the email.
    try:
    #Provide the contents of the email.
    mail_response = client.send_email (
      Destination={
        'ToAddresses': [
          RECIPIENT,
        ],
      },
      Message={
        'Body': {
          'Text': {
            'Charset': CHARSET,
            'Data': BODY_TEXT,
          },
        },
        'Subject': {
          'Charset': CHARSET,
          'Data': SUBJECT,
        },
      },
      Source=SENDER,
    )

    # Display an error if something goes wrong.
    except ClientError as e:
      print(e.mail_response['Error']['Message'])
      return -1
    else:
      print("Email sent! Message ID:"),
      print(mail_response['ResponseMetadata']['RequestId'])
      return 0

 # ----------------------------------------------------------------------------
```

Pivotal

```
# Function to invoke remote script
#
# Code primarily from:
# https://aws.amazon.com/blogs/compute/scheduling-ssh-jobs-using-aws-lambda/
# ---------------------------------------------------------------------------
def invoke_script(bucket, key, timeout=30, bg_run=False):

   s3.download_file(bucket, config_dir + 'PivotalGreenplum_master.pem', '/tmp/keyname.pem')
   head = s3.head_object(Bucket=bucket, Key=key)

     if 'dbtable' in head['Metadata']:
       dbtable = head['Metadata']['dbtable']
     else:
       dbtable = ''

     host=gpdb_host
     user='gpadmin'

   k = paramiko.RSAKey.from_private_key_file("/tmp/keyname.pem")
   c = paramiko.SSHClient()
   c.set_missing_host_key_policy(paramiko.AutoAddPolicy())

   print ("Connecting to " + host)
   c.connect( hostname = host, username = "gpadmin", pkey = k, timeout = 10 )
   print ("Connected to " + host)

   commands = [
     script
     ]
   for command in commands:
     print ("Executing {}".format(command))
     stdin , stdout, stderr = c.exec_command(command + " -s3_bucket '%s' -s3
     key '%s'" % (bucket, key))
     rc = stdout.channel.recv_exit_status()
     stderr = stderr.read()
     vw_nm = stdout.read()
     if rc != 0:
       return rc, "FAILED: File upload failed with error:\r\n    " + stderr

   return rc, "SUCCESS: File uploaded and is available for reporting with view
   " + vw_nm

# ---------------------------------------------------------------------------
# Event Handler for S3 notification
# ---------------------------------------------------------------------------
def lambda_handler(event, context):
   #print("Received event: " + json.dumps(event, indent=2))

   # Get the object from the event
   bucket = event['Records'][0]['s3']['bucket']['name']
   key = urllib.unquote_plus(event['Records'][0]['s3']['object']['key'].
   encode('utf8'))

   # Exit if file is part of config
   if key.startswith(config_dir):
     return ''

   try:
     response = s3.get_object(Bucket=bucket, Key=key)
   except Exception as e:
     print(e)
     print('Error getting object {} from bucket {}. Make sure they exist and your bucket
     is in the same region as this function.'.format(key, bucket
     raise e

   initialize(bucket)
   rc, script_response = invoke_script(bucket,key)
   send_mail(bucket,key,rc,script_response)

   # Return response
   return response['ContentType']
```

Appendix 3
(Test)

```
{
  "Records": [
    {
      "eventVersion": "2.0",
      "eventTime": "1970-01-01T00:00:00.000Z",
      "requestParameters": {
      "sourceIPAddress": "127.0.0.1"
      },
      "s3": {
        "configurationId": "testConfigRule",
        "object": {

        "eTag": "3b0c8da017cfdb9e7e2f3da5b0f32b5a",
        "sequencer": "0A1B2C3D4E5F678901",
        "key": "sandbox/L_AIRPORTS.csv",
        "size": 200796
      },
      "bucket": {
        "arn": "arn:aws:s3:::cf-pde-lmugnano-us-east-1",
        "name": "cf-pde-lmugnano-us-east-1",
        "ownerIdentity": {
          "principalId": "EXAMPLE"
        }
        },
        "s3SchemaVersion": "1.0"
      },
      "responseElements": {
        "x-amz-id-2": "EXAMPLE123/5678abcdefghijklambdaisawesome/
        mnopqrstuvwxyzABCDEFGH",
        "x-amz-request-id": "EXAMPLE123456789"
      },

      "awsRegion": "us-east-1",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "EXAMPLE"
      },
        "eventSource": "aws:s3"
      }
    ]
  }
```

Appendix 4
(greenplumSailfish.py Sample Log Output)

The following shows sample log output of an execution of the greenplumSailfish.py script. This shows the Greenplum SQL generated to build the objects in the database when the script is invoked by the S3 notification Lamdba handler.

Invocation of script:
python ./greenplumSailfish.py -cfg greenplumSailfish_cfg -s3_bucket='cf-pde-lmugnano-us-east-1' -s3_key='sandbox/otp200912.gz'

```
-------------  CONNECTION CONFIGURATION  -----------------
Host: localhost
Port: 6432
Database: gpadmin
----------------------------------------------------------
Generate External table for S3 Object
determine_headings-1:
   drop external table if exists s3_sandbox.ext_otp200912_head

determine_headings-2:
   create external table s3_sandbox.ext_otp200912_head ( header text ) location
('s3://s3.amazonaws.com/cf-pde-lmugnano-us-east-1/sandbox/otp200912.gz config=/home/
gpadmin/s3/s3.conf') format 'text'

determine_headings-3:
   select replace(unnest(header),'"','') from (select string_to_array(header,',') as header from
s3_sandbox.ext_otp200912_head limit 1) a

determine_headings-4:
   drop external table if exists s3_sandbox.ext_otp200912_head

create_ext_table-1:
   drop external table if exists s3_sandbox.ext_otp200912 cascade

create_ext_table-2:
   create external table s3_sandbox.ext_otp200912 ( year text, quarter text, month
text, dayofmonth text, dayofweek text, flightdate text, uniquecarrier text, airlineid
text, carrier text, tailnum text, flightnum text, origin text, origincityname text,
originstate text, originstatefips text, originstatename text, originwac text, dest
text, destcityname text, deststate text, deststatefips text, deststatename text, destwac
text, crsdeptime text, deptime text, depdelay text, depdelayminutes text, depdel15 text,
departuredelaygroups text, deptimeblk text, taxiout text, wheelsoff text, wheelson text,
taxiin text, crsarrtime text, arrtime text, arrdelay text, arrdelayminutes text, arrdel15
text, arrivaldelaygroups text, arrtimeblk text, cancelled text, cancellationcode text,
diverted text, crselapsedtime text, actualelapsedtime text, airtime text, flights text,
distance text, distancegroup text, carrierdelay text, weatherdelay text, nasdelay text,
securitydelay text, lateaircraftdelay text, firstdeptime text, totaladdgtime text,
longestaddgtime text, divairportlandings text, divreacheddest text, divactualelapsedtime
text, divarrdelay text, divdistance text, div1airport text, div1wheelson text,
div1totalgtime text, div1longestgtime text, div1wheelsoff text, div1tailnum text,
div2airport text, div2wheelson text, div2totalgtime text, div2longestgtime text,
div2wheelsoff text, div2tailnum text, div3airport text, div3wheelson text, div3totalgtime
text, div3longestgtime text, div3wheelsoff text, div3tailnum text, div4airport text,
div4wheelson text, div4totalgtime text, div4longestgtime text, div4wheelsoff text,
div4tailnum text, div5airport text, div5wheelson text, div5totalgtime text,
div5longestgtime text, div5wheelsoff text, div5tailnum text, dummy_1 text ) location
('s3://s3.amazonaws.com/cf-pde-lmugnano-us-east-1/sandbox/otp200912.gz config=/home/
gpadmin/s3/s3.conf') format 'csv' (header)

grant_to_role:
   grant ALL on s3_sandbox.ext_otp200912 to admin_role

create_user_view-1:
   drop view if exists s3_sandbox.vw_otp200912

create_user_view-2:
   create view s3_sandbox.vw_otp200912 as select * from s3_sandbox.ext_otp200912

grant_to_role:
   grant ALL on s3_sandbox.vw_otp200912 to admin_role

grant_to_role:
   grant SELECT on s3_sandbox.vw_otp200912 to ro_role
```