Final Project Writeup
{AWSHalide.py}

Norman Ponte
nponte@andrew.cmu.edu

December 14, 2016

1 Summary

For my project I setup a python library which automatically configures an AWS backend to process Halide projects given to it by the API. This project was meant to provide the user with the ability to setup and run code in complicated build environments on the cloud without having to spend time focusing on how to interface with AWS.

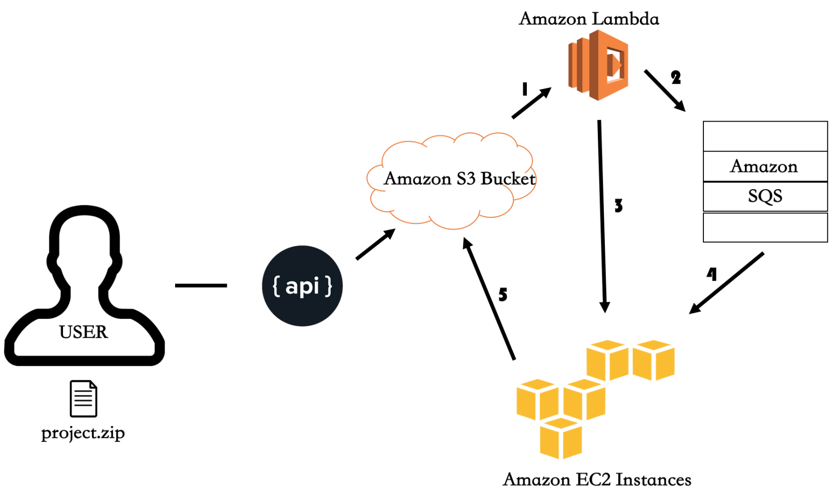
2 Background/Approach

2.1 Current Problem

Often when coders write code using Halide and languages of the sort setting up the build environment can be a pain and this limits the users ability to run code on a variety of machines or many machines. In this project I wanted to remove that part of the process. An example that came to mind was in assignment 2 when we developed the neural network using Halide. Often when I made changes to the NN I had a long wait time before I could see the changes. With the system I propose you would be able to upload the code to a queue and then AWS would use the principles of elasticity to run the tasks you commission allowing you to go through many iterations of code. For the cloud I decided to use AWS as it offers a large amount of services already in the cloud.

2.2 AWS Backend

In setting up the backend I followed this AWS guide `https://aws.amazon.com/blogs/compute/better-together-amazon-ecs-and-aws-lambda/` to connect my ECS, Lambda, and SQS instances. In total I use four amazon AWS services.



1. EC2 Container Service (ECS)
   I used ECS for a couple reasons. The first I mentioned in my summary. One of the core ideas is that often build environment for dsl languages are difficult to build and update. ECS is a service which interacts with Docker to build and stop Docker containers on the EC2 instances it spawns. The second reason I wanted ECS is that ECS has built in scalability and load balancing and other services that are essential to running a multiple node system and this allows the user to benefit from these services without worrying about implementation.

2. AWS Lambda
   AWS lambda is a service released about two years ago that yet again makes the coders life easier in term on managing a cloud. AWS works by creating a function in Node.js when specific events happen (In this case the trigger was uploading files to S3 bucket) it will create a task definition in the SQS queue with the required information. Then it will communicate with ECS that it has to process items from the queue. Lambda almost acts like the manager of the backend placing items in the SQS queue and notifying the ECS service of work.

3. Amazon Simple Queue Service (SQS)
   The lambda function places tasks in this queue for the ECS instances to operate on. This is just a standard queue which provides FIFO processing. SQS is used as a simple queue here to store tasks which can be processed by the ECS.

4. Amazon Simple Storage Service (S3)
   Perhaps amazons best service is the S3 cloud storage. This is a simple service to store and retrieve any data. This is used in this project to store images the zip files and the result of the input code. Another great feature

of S3 is the public libraries it contains with an image library of 100M images. The ongoing work on the project is focused on using deduplication to avoid costs of storing the zip files on the cloud. One of the problems with the current solution is that you may be are uploading the same code zipped up repeatedly to the cloud and the system does not recognize this and this incurs additional cloud costs where they are unnecessary,

2.3 API Examination

1. config.py
   Even though this is not part of the API this might be the most essential part of the library. One of the goals of the project was that the developer only has to initialize all the required variables and then they don't have to worry about them. In this file the user can specific what the output of their program/system is and how to run it on the cloud. They also set variables such as their cloud instances and IAM permissions. I take all these variables and place them in the required places when calling the API.

2. def init()
   The init function initializes most of the backend. First it updates the script that will be uploaded to the cloud with your set variables in config. Next it creates the necessary structures in AWS. These being the buckets, the queues, the lambda functions, and the EC2 Cloud Service.

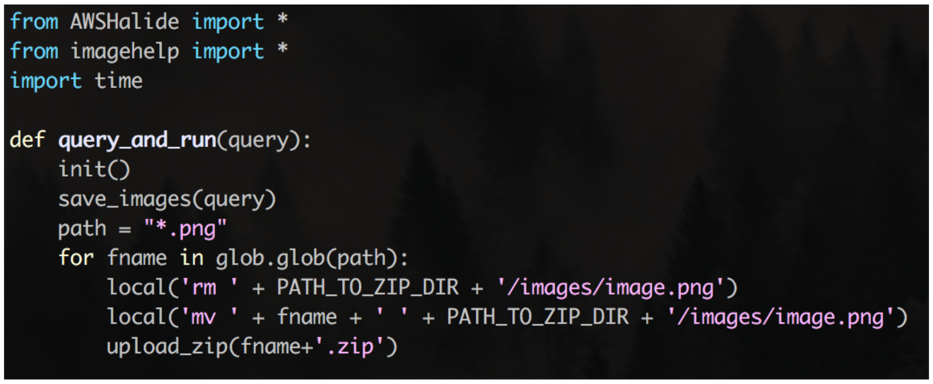
3. def upload_zip(zip_name)
   The parameter here is the name of the zip file you want to upload to the cloud. This file will take the parameter from config which states what to zip and creates the zip. This then adds the zip file to the s3 bucket which then triggers an event which causes the EC2 instances to perform the desired script on the zip. This function also removes the zip file from the users local file system.

4. def list_bucket()
   List the contents of the bucket

5. def download_output(output_dir, zip_name)
   Downloads the file name into the desired directory. The name is the same as the zip name put into upload$_zip$.

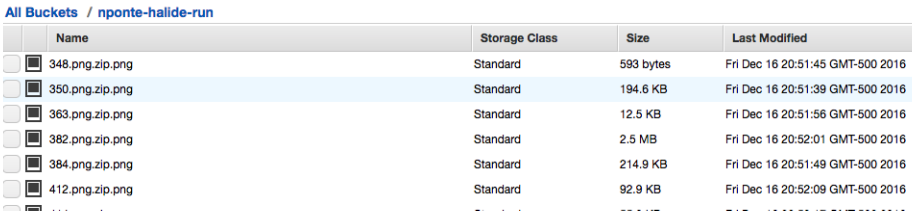
3 Results

For the result section I will focus on a piece of code which I believe shows the potential upside of this API.



```
from AWSHalide import *
from imagehelp import *
import time

def query_and_run(query):
    init()
    save_images(query)
    path = "*.png"
    for fname in glob.glob(path):
        local('rm ' + PATH_TO_ZIP_DIR + '/images/image.png')
        local('mv ' + fname + ' ' + PATH_TO_ZIP_DIR + '/images/image.png')
        upload_zip(fname+'.zip')
```



All Buckets / nponte-halide-run

Name	Storage Class	Size	Last Modified
348.png.zip.png	Standard	593 bytes	Fri Dec 16 20:51:45 GMT-500 2016
350.png.zip.png	Standard	194.6 KB	Fri Dec 16 20:51:39 GMT-500 2016
363.png.zip.png	Standard	12.5 KB	Fri Dec 16 20:51:56 GMT-500 2016
382.png.zip.png	Standard	2.5 MB	Fri Dec 16 20:52:01 GMT-500 2016
384.png.zip.png	Standard	214.9 KB	Fri Dec 16 20:51:49 GMT-500 2016
412.png.zip.png	Standard	92.9 KB	Fri Dec 16 20:52:09 GMT-500 2016

In this code I combine images queried from google on a search and halide code into a zip which I then upload to my bucket using the API. One thing to note is the removal of the zip from the local file system after the upload to the cloud. This means that I could infinitely run this code and there is no limit to how many images I want to process.