A1: Data Acquisition

Tyler Brown

The approach I used to complete Assignment 1 was to create a Python package, Okra, which comes with a command line tool to automate process.

Architecture

Using Okra

A command line tool is provided upon installing Okra for Assignment 1. Below is the help menu for this tool.

```
(okra) bash-3.2$ assn1 -h
usage: assn1 [-h] action repo_list directory_path logname

positional arguments:
  action          'get', 'update', 'extract_data' for git repos
  repo_list       file path to list of GitHub repos
  directory_path  file path to directory storing git repos
  logname         name of log file

optional arguments:
  -h, --help      show this help message and exit
```

We can see that three actions are available: 'get', 'update', and 'extract_data'. The 'get' action will perform `git clone` on all repositories in the "repos.list" input file we were assigned. The 'update' action will perform `git fetch` on all these repos. I didn't want to try to merge anything because upstream commits can require manual merging; we want this process to be automated. A repo is also not required to use `master` as it's main branch even though it would be the conventional practice. The 'extract_data' action will generate the three csv files which were requested for Assignment 1.

Deploying Okra

Okra stores all git repositories from `repo\_list` in `directory\_path`. In the case of Assignment 1, we're storing about 98GB of data. I used an AWS EC2 instance with 4GB of memory to process all of the repos. The repos are saved on an 150GB AWS EBS volume. This approach allowed me to bring down the EC2 instance when I'm not using it while persisting data in the EBS volume. I like the approach because I do not have to keep a compute server running. Gotchas for this approach are that the EBS volume and EC2 instance must share the same subnet in addition to the same region. Be sure to appropriately set the subnet if you're going to use this setup.

An IT Automation tool is used to automate some provisioning tasks for the EC2 instance (RedHat 2018). The Ansible playbook for this assignment is located under the `configuration` directory outside of the `01-data-acqu` directory. Ansible basically uses yaml configuration files to complete tasks after connecting to a server with SSH.

Unsuccessful Approaches

I attempted a number of unsuccessful approaches involving Data Pipeline Frameworks for this assignment. I found the frameworks to be overkill for this assignment. They're generally expensive to run, complicated to set up, and somewhat dependent on specific programming languages. The following list details what I tried and why it didn't work for this assignment.

- Apache Airflow (Apache 2018a)
  - Too many dependencies required to quickly launch application
  - Does not currently support Python 3.7, needs Python 3.6
  - Requires message broker like RabbitMQ to run locally or distributed which is more set up and dependencies
- Apache Nifi (Apache 2018b)
  - Uses own domain specific language (DSL) to manipulate data
  - Very dependent on Java
  - Too expensive and complicated to set up for the scope of this assignment
- Spring Cloud Data Flow (Pivotal 2018)
  - Too expensive to run and difficult to set up
  - Can use shell scripting for streams but tasks must be done in Java using the Spring Framework

If given more time, I'll continue using the Python package, "okra", that I wrote. It seems more important to focus on the analysis rather than the tools at this stage.

Performance

The internet connection speeds for an AWS instance probably helped performance quite a bit. Using a programming language besides Python would have most likely sped up data extraction.

Run Time

Table 1: Run Times by Process

Process	Log Start Time	Log End Time	Hours Estimate
Get repos	2019-01-21 02:54:16	2019-01-21 04:40:18	1.50
Update repos	2019-01-22 03:08:06	2019-01-22 03:11:34	0.05
Extract data	2019-01-22 07:31:02	2019-01-22 09:36:33	2.00

Table 1 shows us that the data collection from start to finish would be about 3.5 hours if the git repositories were not already stored. If the git repos are being stored then data collection from start to finish would be about 2.5 hours. These run times are well within the 24 hour maximum run time specified in Assignment 1.

Disk Space

Table 2: Disk Usage by Name

Name	Disk usage
commits.csv	881M
files.csv	1.9G
messages.csv	1.3G

Name	Disk usage
/repos/	98G

Data collection is efficient but it takes up about 102GB of memory. This seems fine given that we'll need to repeatedly reference this dataset throughout the semester.

Discussion

My approach for Assignment 1 was to create a Python package, Okra, to automate a bunch of shell commands, provide logging, and a command line interface. By deploying Okra on AWS, I was able to take advantage of good network speeds and collect data well under the maximum time limit. Data pipelines are overkill and take away from time which could be spent on the analysis at this stage of the project. I'll most likely continue adding to the Okra package for Assignment 2 unless requested to do otherwise.

References

Apache. 2018a. "Apache/Airflow: Apache Airflow." https://github.com/apache/airflow.

———. 2018b. "Apache Nifi." https://nifi.apache.org/.

Pivotal. 2018. "Spring Cloud Data Flow." https://cloud.spring.io/spring-cloud-dataflow/.

RedHat. 2018. "Ansible Is Simple It Automation." https://www.ansible.com/.