COMP 6971

SEODIN: An Open Data Infrastructure

Concordia University

Name: Satnam Singh

ID: 40059599

Term: Summer 2018

Content

Introduction

This project is divided into three sections. Section 1 is about deploying and advancing the SEODIN application [1], which is a Web application to provide public access to research data. It involves building a monolithic application using JHipster, uploading data to the SQL database, configuring a platform on DigitalOcean, deploying the SEODIN application to DigitalOcean and enabling public access to the application by configuring DNS information on the Ptidej website.

Section 2 deals with the implementation of some design-patterns in Java which were taught in SOEN 6461 and upon successful implementation, could be used to illustrate the concepts in future iterations of the course.

Section 3 describes the first step undertaken to provide a hosting of Windows 2000 remotely to consolidate the resources of Ptidej team on the same server. The first step in doing this was to enable GUI access to a remote machine or droplet on DigitalOcean and installing VirtualBox on it which would be used to run a virtual instance of the Windows 2000 image.

Finally, Section 4 provides a conclusion to the activities done and the results achieved in the summer project.

Section 1

Process

The work done can be broadly classified in the following steps:

1. Making the local environment ready to build a Web application using JHipster [2] by installing the dependencies including but not limited to Yarn, Gradle, Node.js, Yeoman, Java8 and JHipster [2]. After this step, we have a local environment which is ready to build an application using JHipster.
2. Forking the existing SEODIN JHipster [3] application on GitHub to become familiar with the entities used for the SQL tables and the user interface. It was difficult to get the original code running because of the deprecated versions of the dependencies - a problem which was only augmented by the various property files of Gradle/Maven/Docker, which require understanding of the various components involved in them. At the end of this step, we would have the existing code of SEODIN on the local environment and we would be familiar with the entities and the motivation behind their creation.
3. Getting the research data from the archive in Google Drive, extract it and upload it to the application's SQL database using scripts prepared for each type of data/entities. After successful completion of this step, we would be able to navigate to the entities on the SEODIN application in the browser and view the uploaded data. Some basic CRUD operations can also be performed.
4. Downloading the research videos from an existing archive and uploading them to YouTube with the relevant license under the SEODIN channel for public access. For this purpose, creating a project on Google Cloud Platform to use the YouTube API. Running a script (in Python) to access all videos in a directory on the local machine, uploading them to YouTube and save the unique id generated. While Google provides a script which can be used for this purpose, it requires getting used how a project is made on GCP, how the security tokens are generated and used to authenticate the API. Once all the

videos are uploaded successfully, they can be viewed on the YouTube website with public access.

5. Pasting a link to the YouTube video on the related page of the SEODIN app. Also, pasting the link to the related page on SEODIN in the description of the YouTube video. This would help to cross reference videos from the SEODIN website to YouTube.
6. At this step, the application is running successfully on the local machine. We can start the application on any port, open it in a web browser, view or edit the entity data, and see the links to the videos on YouTube.
7. The next step entails deploying it to a cloud hosting service. Analysing the cost-value of the major cloud services and found Digital Ocean to be the most cost-effective.
8. Creating a droplet and setting up a similar environment on Digital Ocean with all the dependencies required to run the application with the same versions used to build the application. While reproducing this step in the future, it would be helpful to make a list of dependencies installed in Step 1 along with their versions to avoid any version compatibility related issues.
9. Deploying the application on the configured droplet by building a .war file of the application or using Docker image. A few difficulties were faced while performing this step, mostly as consequence of unfamiliarity with Gradle configuration files.
10. Configuring Apache Virtual Hosts [4] or any similar method to expose the SEODIN application to public using the existing Ptidej domain name. Successfully doing this step will result in a working link like “seodin.ptidej.net”.

Future Scope

There is a lot that can be done with the SEODIN application to ease the access to its data to potential users.

1. The application is currently built using a monolithic architecture, which is difficult to scale, modify and debug. A transition to service oriented architecture, using microservices that do only one task can provide an answer to the problems posed by the monolithic architecture.
2. The present setup allows the users to view the data in a very structured way, reducing the ways in which it can be downloaded. This makes it difficult to fetch only custom fields, process and store them. A better approach could be to provide a URI which takes input fields and provide them in a standardized JSON format. For example, if all data for a developer with developer id “dev1” is required, a URI should be able to handle a POST request with dev_id = “dev1” and return a payload with the relevant data. If multiple developer ids are provided in the request, then a JSON should be prepared with a list of data fields for all the dev_ids requested.

   To achieve this, the use-cases must be identified for which the data can be requested from the application. Then, seperate APIs could be built to fetch the data from SQL and return in a format which confirms with a standard like “JSON API” [5].
3. While deploying the application on a droplet on DigitalOcean, the version of dependencies must match with the version that was used on the local machine where the application was built. At times, it is cumbersome to debug issues due to version mismatches. A utility/script could be built to automate the process.
4. While there is an option to manually resize the Digital Ocean droplets to allow the application to scale in the future, it is not automatic and must be commissioned manually. If such is the requirement, a transition can be made to PAAS providers, which have the option to scale up/down the application automatically based on the volume of traffic.

Section 2

Design Patterns from SOEN 6461

Along with working on the SEODIN application, some design patterns, taught in SOEN 6461 were also implemented in Java to be reused in future installment of the course. The following is a list of the design patterns implemented with the real world analogy from which the implementation is motivated.

1. Association
    a. Aggregation DP: A team of players.
    b. Composition DP: Human anatomy.
2. Decorator DP: Optional Extras while purchasing cars.
3. Abstract Factory DP: Selection of different sorting techniques based on choice.
4. Iterator DP: Customer support tickets.
5. Observer DP: Messaging in a Slack group.
6. Singleton DP: Initialization of a shared behaviour.
7. Template DP: (Similar) steps in message transfer in multiple messaging services.

URL: https://github.com/singhsatnam/design-patterns

Section 3

Enabling GUI access to Digital Ocean droplet

This composes the first step in deploying a Windows 2000 VM on Digital Ocean to consolidate all the virtual assets of the Ptidej Team on the same servers. A Windows OS image is not officially provided or supported by DigitalOcean. In order to have a working Windows platform, the following two steps were considered and tried. Eventually, the second approach was found to be feasible and was implemented.

1. Installing Windows on DigitalOcean:- DigitalOcean does not yet support droplets running Windows OS. This approach, though not officially supported by Digital Ocean, is a work-around to run Windows on DigitalOcean. However, based on the requirement of Windows 2000, this approach has to be abandoned as only Windows 8 and 10 were viable options.
2. Running Windows 2000 image on VirtualBox:- A linux droplet was created on Digital Ocean and GUI access to the droplet was enabled using Virtual Network Computing [6]. Hence, it was possible to install and run VirtualBox on the droplet with the end purpose to host a Windows 2000 image on it.

Section 4

Conclusion

Over the summer, the three activities were done as described above. There is still work which requires to be done to further the tasks undertook in the three activities. For example, SEODIN application is still being worked upon, the implementation of the design-patterns is being improved and the logistic problems related to the deployment of a working Windows 2000 image virtually are being evaluated and resolved.

Besides these activities, a quiz based on concurrent programming in Erlang was also hosted on HackerRank as a weekly activity for the Ptidej team.

References

1. SEODIN application Background: http://www.ptidej.net/publications/documents/170919+Seodin.doc.pdf
2. JHipster: https://www.jhipster.tech/production/
3. https://github.com/ptidejteam/seodin
4. Virtual Hosts on Digital Ocean: https://www.digitalocean.com/community/tutorials/how-to-set-up-apache-virtual-hosts-on-ubuntu-16-04
5. JSON.API: http://jsonapi.org/
6. Configuring VNC on Digital Ocean: https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-vnc-on-ubuntu-16-04