An approach to Image Analysis on the OpenStack cloud service using Celery

Jonas Samuelsson, Jonas.Samuelsson.7503@student.uu.se, Uppsala University
Saim Mehmood, Department of Information Technology, Uppsala University
and Thomas Krannich, Department of Information Technology, Uppsala University

***Abstract*—This report will cover an approach to outsource a biologists software to an open source cloud service. Thereby we will consider the initialization process of the Virtual Machine (VM), how to provide and store the images, give insight to the automated process of the analysis pipeline and introduce our interface solution to receive the results, once they are computed.**

I. Introduction

The background of this project is derived from the biological analysis of growing Glioblastoma cells. The cancer cells in the related work are treated with target specific drugs to cause a denaturing effect on the cells natural growth. Once a laboratory issues images of these growing cancer cells, before and after drug treatment, it is of interest to compare the cells by an automated image analysis pipeline to make comprehensible and reproducible statements about their development and behaviour. For that purpose there are software solutions like CellProfiler and CellProfiler Analyst. The former tool is developed to compute phenotype measurements from an arbitrary amount of images automatically.

Now, the main aspect of this approach is to provide a Cloud based solution where the user can hand images to a storage, let these get processed and receive the output afterwards from the cloud. The processing here is transferred to a worker service within the Clouds VM.

II. Related Work

In 2012 Bednarz et al. [1] investigated a different approach on a similar challenge. They carried out the Image Analysis and Processing Toolbox by CSIRO to the NeCTAR cloud infrastructure and tried to give researchers access to biomedical image processing via a user interface. Their concept led to our assumption that this strategy of solving computing intensive tasks of biomedical image processing remotely on a cloud is in general of higher interest. Even though their concept exhibits major differences in the design pattern (PaaS), one can get an idea of how the user-cloud interaction can be modelled.

III. Container holding the images

The OpenStack Object Store project, known as Swift, offers cloud storage software so that data can be store and retrieve with a simple API. It is built for scale and optimized for durability, availability, and concurrency across the entire data set. Swift is preferred for storing unstructured data that can grow without bound.

To store and upload images in the cloud, Swift is used. The user can upload images according to his requirement and they will be processed by the VM.

To upload images, a user has to create a bucket inside Swift container using python api and then upload images inside that bucket. After uploading, the user is supposed to provide the name of bucket as an argument to main program. The program will then download all the images (using Swift API) inside the VM and use it as an input for processing inside CellProfiler. Important to note is that the container has to be public in order for the swift download command to work.

IV. Software

The cloud that is been used throughout this project is based on the OpenStack software. OpenStack provides tools for managing compute, storage and networking resources [2]. The software offers a Python Application Programming Interface (API) that is used for communication with the cloud and allows users to execute operations inside the cloud. Since one available API is implemented in Python, it is natural to implement the solution in Python as well. Examples of operations that can be executed with the mentioned API are to initialize a new VM, create a new storage container, or upload files to an already existing container.

When a VM is created, user has the option to choose between different flavors. Flavors denotes for example the amount of RAM, the number of CPU cores and the amount of available disk space that will be used with a chosen flavor. There are five different flavor types and the flavor that is used in this project is called m1.medium and is using 4GB of RAM, 2 CPU cores and 40GB of disk space [3].

Together with OpenStack there are a number of other open source softwares that are building blocks for this project.

A. RabbiMQ

RabbitMQ offers the functionality of a message broker. A message functions as a central point between different modules in an application. The broker sends and receives asynchronous messages from and to the different modules in the application. The messages are placed in a queue until they can be consumed and distributed by different parts of the application [4].

RabbitMQ can be installed on the Ubuntu operating system by typing ‘sudo apt-get install rabbitmq-server’in the terminal.

B. Celery

Celery is an asynchronous task queue system, and needs to be connected to a message broker. Among others, RabbitMQ is a supported message broker.

A object within Celery is often called a worker and the purpose of a worker is to consume possible message in the stored queue of the broker [5]. An important feature of Celery is that several workers can be connected to the same broker which allows the work to be parallelized. However, in this project only a single worker is currently used. Because of the fact that Celery is a Python package and can be installed using Pip, which is a tool widely used for installing such packages. Both Pip and Celery is installed by command-line in Ubuntu. The two commands for installing the two mentioned softwares are 'sudo apt-get install pip'and 'sudo pip install Celery'. Note that the installation of Pip needs to be complete before the installation of Celery is started.

The Celery worker in this project is used to start the CellProfiler software, which is described in Section IV-D.

C. Flask

Flask is a microframework implemented in Python. In short, it provides a user with a web-interface and, with some programming and settings, the VM can be accessible from a users web browser. The framework functions as a Representational State Transfer (REST)-ful service. A RESTful service is the way of handling machine-to-machine communication [6]. The user can enter the IP-address of the VM in the browser and a state of the program will be displayed to the user. In this project, Flask is used to display the output files that are created as a result of CellProfiler. The files that are displayed in the are clickable and if that actions is taken, the files will be downloaded to the local computer.

Since Flask is a part of python it can be installed with pip. The command for installing it is 'sudo pip install Flask'.

D. CellProfiler

As mentioned above, CellProfiler is used to analyze images of stem cells to see how they react on different kind of drugs. The software can be run either with a graphical user interface or by command line. In a cloud environment with automated processes it is often preferable to run a software by command line if the option is present. The installation process of the software is a bit longer compared to the other mentioned software therefor is shown in the Appendix B. When the installation of the software is complete the software can be started with the following command: 'python CellProfiler.py -c -i input_folder -o output_folder -p pipeline_file.cppipe'. Note that there are three arguments that need to be passed to the command. The first argument is the input folder containing all the images that are supposed to be processed by the program. The second argument is an output folder which will contain the output files of the program when the process is complete. The third argument is the pipeline that is to be used for the process. The -c parameter denotes that the program should run without a graphical user interface.

E. Other Packages needed

Keystone is an OpenStack project that provides Identity, Token, Catalog and Policy services for use specifically by projects in the OpenStack family. For the python-keystoneclient theres a Python API (the keystoneclient module), and a command-line script (keystone).

For installing the above two libraries, you need to run the following two commands: 'sudo apt-get install python-keystoneclient'and 'sudo pip install python-swiftclient'

To connect with OpenStack Object store (Swift) we need python-swiftclient. It is a python client for Swift API. Theres also a Python API (the swiftclient module), and a command-line script (swift).

V. Contextualization of the Solution

A user that accesses the VM which is supposed to run the software does not want to care about installation. As said earlier, CellProfiler is a software used by biologists and it is not fair to assume that every person that will use the program has experience with Ubuntu. If the software mentioned above has to be installed every time a user wants to run CellProfiler in the cloud, large time overhead is created. Because of this overhead, it is better if the VM is contextualized. Contextualization means that the VM spawns with all the software pre-installed, so that the user can log into the machine and directly start the program without having to think about required software.

Contextualization is a feature that exists in the OpenStack API and is called Cloud-init [7]. A bash script containing the commands that are used for installing the different software can be passed as an argument when the VM is created.

VI. Results

In order to produce some output results a new VM should be started, and this can be done locally from your own computer using the file start-instance.py that is available in the GIT repository. For this file to work the credentials to the Cloud need to be sourced. The credential file is not available from the GIT repository since every individual has a unique credentials file. An example is shown in Figure 1.

```
Jonas-Air:cancer-project-ACC Jonas$ python start-instance.py
Launching a new instance, may take some time.
The new server IP is: 130.238.29.144
```

Fig. 1: Example of how to initiate a new VM.

When the new VM have been initiated the user should use an ssh-connection and log in to the VM and can be seen in Figure 2.

```
Jonas-Air:cancer-project-ACC Jonas$ ssh -i jonasacckey.pem ubuntu@130.238.29.144
```

Fig. 2: Example that shows an SSH-connection to the VM.

After the connection is established a celery worker has to be started. This can be done with the command 'celery –loglevel=INFO worker -A celery_tasks'and is shown in Figure

3. Note that the credentials once again have to be sourced in the current terminal window.

```
ubuntu@jonas-for-cellprofiler:~/cancer-project-ACC$ celery --loglevel=INFO worker -A celery_tasks
```

Fig. 3: Example of starting a new Celery worker.

Once the worker is up and running the solution is ready to start. Figure 4 shows an example of how the main program can be started and Figure 5 shows the result that is displayed to the user after the main program is done with execution. The files that are shown in Figure 5 can be clicked on which will start a download of the files to the local computer.

```
ubuntu@jonas-for-cellprofiler:~/cancer-project-ACC$ python main_program.py jonas-cp-bucket output_folder/ simpletransproject.cppipe
Starting CellProfiler
```

Fig. 4: Example of starting the main program.

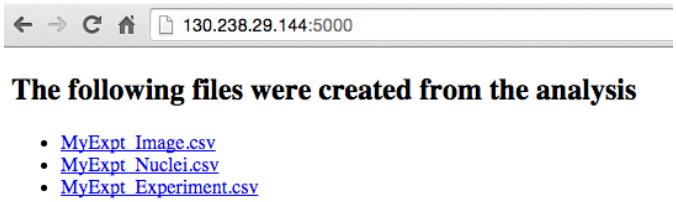

Fig. 5: Showing what the user is met with when he or she enters the IP of the VM in the web browser.

VII. Conclusion and Future Work

Implementing the mentioned features, we faced a lot of minor and major pitfalls in software design and cloud computing. One of the first major decisions we had to make was choosing a suitable solution to store and handle a larger amount of images if the data exceeds the storage capacity of the VM. This solution should also possibly be suitable for the output of the program and the access of the flask API. This way we learned to make use of the properties of a clouds bucket storage system. Generally speaking, the proposed solution fulfills all relevant aspects of the given challenge. We provide a vertical scalable storage solution which can easily be loaded with data to be analysed. The automated pipeline runs the image analysis with a celery worker once the input and a pipeline is specified. The output is easily accessible and downloadable just by visiting a webpage once the processing is done.

So, the main aspects and conditions of uploading data, processing and providing the output are complied. Lets take a fair equal look on things that are still a little messy and provide open potential for improvement and future upgrades. The probably most obvious bottleneck regarding runtime of our approach is the serial workflow of the image processing. One future work could be splitting the input data into a equal partitioning and sent it via the broker to several celery worker.

Considering the fact that we tried to provide a solution that aims for convenient handling, our solution suffers a little from two minor improvable aspects: A) the user still has to be more or less familiar with the swiftclient API to upload data and B) the user has to start multiple python scripts manually on the VM since the whole project is not running as a one-click-application after the upload so far.

The current solution uses a snapshot when a new VM is created. This snapshot does already have all the needed software installed. The reason why real contextualization is not used is because there were several problems with cloud-init and is possibly something that can be improved in the solution. It would be preferable to install the different software with cloud-init instead so that the user can actually change the script file if needed.

Appendix A
Git repository

Link to the Git repository: https://github.com/JonasSam/cancer-project-ACC.git

Appendix B
Installation of CellProfiler by command line

To start CellProfiler inside a VM you need to install couple of python libraries in advance. The following is the list of commands that you have to run on your VM to get it ready for your processing of images:

- sudo apt-get update
- sudo apt-get install -y git python-h5py python-zmq python-matplotlib cython openjdk-7-jdk python-wxgtk2.8 python-scipy python-mysqldb python-vigra –fix-missing
- export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
- export LD_LIBRARY_PATH=/usr/lib/jvm/java-7-openjdk-amd64/jre/lib/amd64/server:/usr/lib/jvm/java-7-openjdk-amd64:/usr/lib/jvm/java-7-openjdk-amd64/include
- git clone https://github.com/CellProfiler/CellProfiler.git
- cd CellProfiler/
- git checkout 2.1.1

References

[1] Bednarz, t., et al. 'cloud-based image analysis and processing toolbox for biomedical applications.' 8th ieee international conference on escience. 2012.

[2] OpenStack. Openstack, open source cloud computing software. http://www.openstack.org/. Accessed: 2015-11-01.

[3] OpenStack. Openstack, flavors. http://docs.openstack.org/openstack-ops/content/flavors.html. Accessed: 2015-11-01.

[4] RabbitMQ. Rabbitmq, messaging that just works. https://www.rabbitmq.com/. Accessed: 2015-11-01.

[5] Celery. Celery, distributed task queue. http://www.celeryproject.org/. Accessed: 2015-11-01.

[6] Flask. A python mircoframework. http://flask.pocoo.org/. Accessed: 2015-11-01.

[7] Cloud-init. Openstack, provide user data to instances. http://docs.openstack.org/user-guide/cli_provide_user_data_to_instances.html. Accessed: 2015-11-01.