UNIVERSITY OF CAPE TOWN
DEPARTMENT OF COMPUTER SCIENCE

Computer Science Honours
Final Paper
2015

Title: **Micro Cloud Platform for Processing Big Data in Bioinformatics**

Author: **Brendan D. Ball**

Project Abbreviation: **BigBInf**

Supervisor: **Michelle Kuttel**

Category	**Min**	**Max**	**Chosen**
Requirement Analysis and Design	0	20	**20**
Theoretical Analysis	0	25	**0**
Experiment Design and Execution	0	20	**0**
System Development and Implementation	0	15	**15**
Results, Findings and Conclusion	10	20	**15**
Aim Formulation and Background Work	10	15	**10**
Quality of Paper Writing and Presentation	10		**10**
Adherence to Project Proposal and Quality of Deliverables	10		**10**
Overall General Project Evaluation *(this section allowed only with motivation letter from supervisor)*	0	10	**0**
Total marks		**80**	**80**

Micro Cloud Platform for Processing Big Data in Bioinformatics

Brendan D. Ball
University of Cape Town

ABSTRACT

Processing Big Data in Bioinformatics is often made difficult by the need to download the entire data set. The solution proposed is to create a micro cloud platform which would be deployed at various institutions. This micro cloud should allow users to process data without having to download it. Results are normally miniscule in size compared to the raw data, which makes them easy to download. A community cloud is formed by connecting micro clouds using a discovery service. A proof-of-concept is implemented and evaluated using sample Bioinformatics analysis code.

Keywords

Big Data; BioInformatics; Clouds; Docker; Kubernetes

1. INTRODUCTION

Cloud infrastructure is the combination of multiple compute and storage resources presented as a single system on the internet. Cloud computing has developed considerably over the last decade. Examples of commercial cloud offerings include Amazon Web Services and Google Compute Engine [24, 26]. Cloud infrastructure has the potential to simplify the processing of big data sets, as well as collaboration between remote researchers.

The first step in simplifying processing of big data sets is to minimize the need for transferring the data. This can be done by storing the data in a cloud and allow researchers access to process the data on the cloud, instead of downloading the data first before processing it. Minimizing the need for transfers of Big Data is important because limitations in current internet infrastructure makes collaboration difficult when Big Data is involved.

A commercial cloud such as Amazon EC2, still has the problem of uploading big data sets to the cloud infrastructure [14]. Micro clouds deployed on-site will overcome this challenge by keeping the data locally, while still reaping the benefits of a cloud platform. However, with different research institutions deploying their own micro clouds, cloud interoperability is required to allow researchers from different institutions to collaborate on the same data. Cloud interoperability creates a community cloud. A use-case of a specific community cloud by Jimenez et al. [21] contains core architectural properties needed for a successful community cloud. These properties include autonomy (where each micro cloud is managed independently), security, self management of nodes, and scalability.

The traditional approach to creating a cloud platform allows users to run their own instances of operating systems (such as Amazon EC2) via virtualisation technology. This includes both hardware level emulation support and the software needed to manage the virtualisation. These virtualisation schemes use machine level virtualisation [18]. Other commercial cloud platforms exist besides Amazon EC2, including Microsoft Azure, Google Cloud Platform, and Digital Ocean [19],[8],[2].

A new method, known as containerization, provides much of the same functionality, with added benefits of lower resource usage and better performance. Containers are able to run native machine instructions whereas virtualisation emulates every machine instruction [17]. Containers are only useful when complete virtualisation is not needed, but allow for isolated application deployment and portability. The use of containers instead of virtual machines has only recently become popular with the release of Docker [25]. The growth in popularity has resulted in a number of systems being developed including Rancher and Kubernetes.

Both virtualisation and containerization requires every instance to be configured by installing necessary software packages. Configuration can be time consuming, particularly in the context of bioinformatics. Cloud BioLinux is part of a cloud solution which solves this issue [23]. This toolkit makes it easy to deploy virtual machines to a cloud platform with bioinformatics infrastructure pre-configured. It bundles specific packages used in next generation sequence analysis (which is the processing of DNA data), thereby decreasing configuration time and increasing maintainability. Instances of Cloud BioLinux have been tested on the Amazon EC2 cloud platform and on a private Eucalyptus cloud. Eucalyptus is open source cloud software that enables you to create your own private cloud.

The source code for the micro cloud platform is available as a git repository at https://github.com/wraithy/bigbinf in the cloud branch.
The NCBI NR database used in this paper is available at ftp://ftp.ncbi.nlm.nih.gov/blast/db/ which takes 36GB of storage space.
The Blast search tool is available at ftp://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/LATEST.

Another project compatible with both Amazon EC2 and Eucalyptus is CloudMan [13]. CloudMan is designed as a workbench which manages cloud resources. A workbench relies on a pre-configured environment on which to execute code. It aims to improve biomedical data anaylsis and fully integrates with the Galaxy framework, a biomedical research platform. This cloud solution does not provide a user configurable execution environment as it relies on Galaxy tools to perform analyses.

We aim to demonstrate a proof-of-concept micro cloud platform that provides users with a configurable environment, taking advantage of open source cloud software and Docker containers to enable efficient processing of Big Data. The aim is to build a micro cloud platform, using Docker as the main technology, and form a community cloud which allows sharing of data and collaboration of users between multiple micro clouds. Users can access data and submit jobs to the micro cloud by interacting with a front end web interface. Evaluation is done by deploying two micro clouds at UWC. The functionality is evaluated by uploading sample code and executing it on data which already exists on the cloud.

UCT and UWC are collaborators on this project. UWC provided valuable hardware resources enabling thorough evaluation in a real world setting.

2. BACKGROUND

The setup of a micro cloud requires both middleware to enable installation of an execution environment, and cluster management software. We look at OpenStack and Kubernetes as possible systems for a cluster manager.

2.1 Docker

Docker is an implementation of a Linux container management tool. Docker functions similarly to virtualisation. It uses an image to quickly launch a pre-configured isolated environment. The key difference is that containers directly use the Linux kernel on the host machine to run native instructions, whereas virtual machines emulate the entire operating system, including the kernel. Every virtual machine launches its own kernel and is not aware of the host kernel. This key difference make containers lightweight, allowing them to be launched in a fraction of time compared to launching a virtual machine [22].

Docker builds on Linux containers by implementing an image format which makes use of an AUFS filesystem. This allows a Docker image to be built up from a number of intermediate layers. Layers can be added or removed without affecting another layer in the image [15]. Docker images contain all the data needed to launch the container, as well as any additional applications or data added to it by the user.

Docker images do not need to contain all the data that the container needs access to. Docker volumes provide a mechanism for mounting a directory from the host to inside the container. This is useful if the data is persistent, because it doesn't matter if the container gets destroyed when you can simply create a new container and mount the same data to a new container.

The layered approach to creating a Docker image allows images to be built using a script called a Dockerfile. Defining an image with a script makes Docker images much more portable and reproducible compared to an image from a running environment, as is the case with virtualisation [15].

The Docker group also hosts a publicly available image repository for upload or download of available Docker images [3]. These images provide full disclosure of its configuration via its Dockerfile. This provides great community support with images for specific use-cases. An example is CloudBioLinux which officially only provides virtual machine images, however third parties have uploaded Docker images of CloudBioLinux available for anyone to use [4].

2.2 OpenStack

Openstack is a full stack cloud platform which manages compute, storage, and networking resources. OpenStack can be used to orchestrate virtual machines and has recently been extended to support Docker containers as well. OpenStack aims to be a complete cloud solution which has resulted in it being cluttered, with many-inter dependent components [12]. The required functionality from a cluster manager for this project is limited compared to everything that OpenStack provides. A simpler cluster manager, specifically designed for use with docker is better suited for this project.

2.3 Kubernetes

Kubernetes is a pure container manager specifically designed to orchestrate Docker containers. Kubernetes version 1 has recently been released. Kubernetes is easy to use and not cluttered and aims to solve a single problem of being a container manager [28].

Google used Linux containers long before Docker was created. Google developed its own in-house container orchestration system called Borg. Google created Kubernetes after learning from mistakes made in the design of Borg. Borg was orginally designed as a monolithic system which makes it difficult to scale. Kubernetes is much more modular and is easy to deploy [28].

Kubernetes primarily aims to manage services, but still has the functionality to run a terminating container. Kubernetes abstracts away from containers with the notion of a pod. A pod is a group of one or more containers. Containers in a pod are guaranteed to be co-located on the same node. A replication controller is another abstraction above pods. A replication controller is used to quickly replicate pods with the main aim being load balancing. Replication controllers are used to always have a specified number of replicas running, and will automatically create another replica if one fails.

Kubernetes also implements its own volume abstraction separate from Docker volumes. It supports a variety of remote storage systems, most of which are a type of network file system or block storage [9].

2.4 Ceph Storage

Ceph storage is a distributed object storage system. Ceph provides flexibility and implements abstractions to use Ceph objects as block storage or as a network file system, in addition to using plain Ceph objects [29]. Ceph objects are based on RADOS (Reliable Autonomic Distributed Object Store). Ceph provides scalability and reliability through automatic distribution and replication of objects. An Object Storage Device (OSD) refers to a Ceph storage unit which can either be physical or logical. Ceph relies on hash algorithms to calculate an objects location using the object's identifier and the cluster layout. Objects are accessed via ceph monitors which behave like distributed servers. A ceph cluster always has more than one active monitor for redundancy and scalability.

2.5 Taverna

Taverna is a workflow tool designed to help scientists create reusable workflow tasks. Taverna aims to help the user create a single workflow by combining a number of tasks, often using distributed web services. Taverna has also been used in cloud solutions where all the necessary services are hosted in the same cloud. This has the benefit of only having to upload the raw data once to a different location and then being able to run the entire workflow on that cloud. Taverna only manages the workflow, and doesn't dictate which web services are available. It is used by the Bioinformatics community and researchers typically share their workflows with the community [30].

2.6 Galaxy

Galaxy is a web application which provides tools for data retrieval and custom analysis tools specific to the field of Bioinformatics. Analyses on genomic data are often done via commandline tools. Galaxy aims to make such analyses simple by providing the functionality to the user through a web browser. Galaxy provides a multitude of tools and plugins to perform various tasks on genomic data. The web application is open-source and can be tested on the official hosted server or installed locally or on a private cloud[20].

3. DESIGN AND IMPLEMENTATION

Cloud infrastructure comprises one or more clusters. A cluster is a collection of servers (nodes). Data storage is also part of cloud infrastructure, but is typically separated from compute nodes and accessed over the network. Our platform consists of the following components: a web interface, a scheduler, an image builder and image repository, a cluster manager (the master node), multiple worker nodes, and a storage interface which allows reading from and writing to persistent storage. Figure 1 shows the connection between these components.

Docker is the main technology that we are using to build the micro cloud. We are focussing on the functionality of the platform, particularly on using a cluster manager together with Docker to build a Linux container based solution. User interface design is not considered and is outside the scope of this project.

3.1 System Architecture

Figure 1 gives an overview of how components interact. This section gives implementation-specific details for every component.

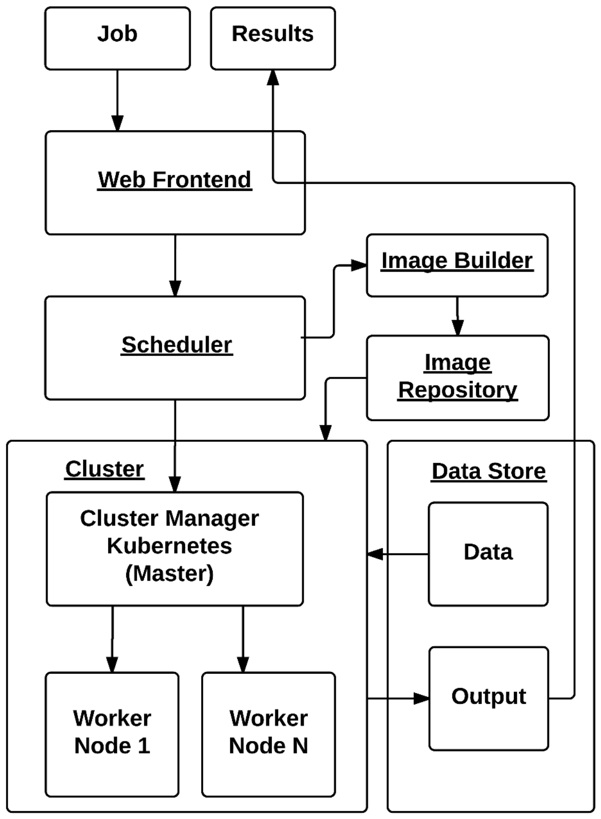


Figure 1: Our Micro Cloud Architecture showing the interaction between components of the cloud framework.

3.1.1 Job

A job describes a single instance of code uploaded for execution. The micro cloud platform uses Docker containers to execute code. Docker images are created with Dockerfiles, which specify the execution environment and what files to execute. Submitting a job requires a Dockerfile and source code to be executed, uploaded together packaged as a single tar archive.

3.1.2 Cluster Manager

To create a cloud platform with containers, a framework is needed to manage containers in a cluster which performs scheduling and resource allocation. OpenStack was the first system considered for use as a container manager. After much trouble setting up the development environment it was decided that there are more suitable systems to solve the problem. Kubernetes was chosen as the cluster manager framework for managing the micro cloud. Kubernetes handles scheduling a container onto a specific worker node. Kubernetes also notifies our scheduler when a container has terminated via a streaming http request that the scheduler initiates. Kubernetes' primary focus is to run non-terminating services. It provides replication and load balancing capabilities for services. We opted to make use of this by running the web interface, scheduler, image builder, private docker registry, and discovery service inside Kubernetes. Kubernetes aims to keep services running, which means that it will restart the service if it terminates for any reason.

3.1.3 Image Builder

A dedicated docker-in-docker container is run which is used to build docker images from dockerfiles. As mentioned in the background section, a Docker image should contain an operating system as well as the code and applications needed to perform the user's task. This is all specified in the Dockerfile. Once the image is built, it is pushed to a private docker repository which can be accessed by any node in the cluster. The docker image is pulled by a worker node when the job is assigned to that worker node.

3.1.4 Scheduler

We have implemented a scheduler based on first-in-first-out (FIFO) queues to schedule jobs. This scheduler is seperate from the Kubernetes scheduler, however it is possible to build all the functionality into the kubernetes scheduler as it allows pluggable schedulers. There are 3 FIFO queues: a build queue where dockerfiles are queued for the building process, a ready queue where Docker images are ready to be run, and an archiving queue where jobs are queued to have their results archived for downloading. The scheduler keeps track of running jobs using a list rather than a FIFO queue because running jobs are asynchronous. This means for example that job B can be started after job A and still still complete before job A. The number of concurrent jobs running is dependent on the number of worker nodes. Another two lists are used to keep track of completed jobs, one list for completed jobs with no results and another list for completed jobs with results ready to be downloaded. These two lists for completed jobs exist because it is possible for job to not have any results to download. Such a scenario might include writing results straight to Amazon S3 storage from within the container. S3 is Amazon's object storage service, accessed via http [1].

3.1.5 Storage

The cluster at UWC makes use of Ceph storage. We are using Ceph block storage because this can be mounted as a file system by using a RADOS Block Device (RBD). An RBD behaves like block storage. Kubernetes provides support for mounting an RBD as a storage volume in a container. However, due to difficulties getting Kubernetes to properly work with RBD during development, a different approach was taken. An RBD was mounted on the worker node outside of Kubernetes. It was then treated as a normal filesystem and mounted as host storage in the container. Using RBD in this way is still beneficial compared to using plain host storage as raw data doesn't have to be copied to each VM, which is infeasible given the size of most raw data in Bioinformatics (The raw data used during the evaluation of the system is 36GB).

3.1.6 Web Interface

The backend web interface is implemented using the Python flask web framework. It is a minimalist framework which allows for rapid development of web interfaces [7]. The front end web interface is implemented using Angularjs to create a single page application. The website provides a form to submit a job by uploading a tar archive. Two text boxes are provided to allow the user to specify mount points (by means of an absolute file system path such as "/db" and "/results") for the raw data and results folder inside the container. Below the form is a list of jobs which shows each job's current status. This list is updated by polling the server every few seconds. If a job is complete and the results folder contains files then the user will be able to download the results as a tar archive. On a different web page (titled 'regions') is a list of all available micro clouds. The user can click on a region to navigate to a different micro cloud.

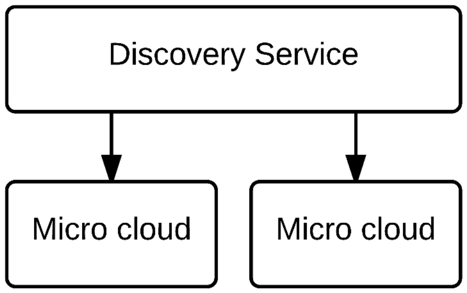


Figure 2: Community Cloud Architecture: Micro clouds are connected with a discovery service.

3.1.7 Community Cloud

A community cloud is formed by implementing a centralised discovery service. This improves scalability compared to a full mesh network between micro clouds. Figure 2 depicts how micro clouds are connected with the discovery service. We implemented the discovery service with a very simple web API which gives a list of connected micro clouds with their url (either an IP address or a domain) along with the name of the micro cloud. The discovery service has to be manually configured by adding a micro cloud's details to a configuration file. After pointing a micro cloud to the discovery service, it makes an http request to the service when the user visits the 'regions' web page on the front end web interface. The web page uses the list of connected micro clouds and creates a link for every micro cloud, allowing the user to click on it and be redirected to a different micro cloud.

3.2 Software Engineering

We implemented a proof-of-concept of the community cloud. The aim was not to focus on software engineering methodologies. However, to allow for reproducibility the source code is stored in a Git repository hosted on Github (Git is a version control system). All the components are containerized so a cluster can easily be deployed by running a few docker containers. This allows greater portability and reproducibility.

3.3 Evaluation

Micro cloud deployments for evaluation are limited to UWC. The evaluation was done on one community cloud containing two micro clouds. Each micro cloud was deployed on a single virtual machine (VM) as a single node cluster. Each VM has 1GB RAM, a single cpu, and 10GB of storage. Two 100Gb RBDs were provided, one for raw data and one for storing results. Due to both clusters residing on the same network, both clusters made use of the same storage. This would not be the case if each cluster was deployed at a different institution, but for the purposes of the evaluation it is not important.

The functionality of the micro cloud is evaluated by submitting sample Bioinformatics analysis code as a job. The analysis code searches against a genome database. We chose a database that is publicly hosted by the National Center for Biotechnology Information (NCBI), an institution dedicated to improving technologies used in genomic research [27]. The analysis code makes use of the Basic Local Alignment Search Tool (BLAST) [16]. This tool specifies a database and submits a query containing specific search criteria. The job runs successfully if the container exits and results are available for download. The results consists of a single text file output by the BLAST tool which gives details according to the query submitted. Different sized jobs are tested by increasing or decreasing the size of the query, which makes it easily scalable to different sized test cases.

The cloud platform was tested by submitting three jobs containing analysis code as described above. Each job tested a different sized query. After successful completion of each job the results were available for download, archived in a tar file. Each job's results contained a single text file created by the BLAST tool, which contained details about the query and the database and outputs generated based on evaluating the query with the database. The exact results output from the job do not give any insights regarding the evaluation of the cloud platform because we are testing the platform itself and not the analysis code.

4. RESULTS AND DISCUSSION

We have implemented the micro cloud proof-of-concept and evaluated it using Bioinformatics analysis code. Figure 4 shows the web interface available to the user.

The implemented micro cloud platform enables a user to submit a job, which it then automatically builds and schedules onto a worker node. Once the job is complete it allows the user to download the results. The user is able to view the job queue and the status of each job. Together with the discovery service, this makes up the core functionality of the micro cloud platform and provides a proof-of-concept that focuses on using Docker containers for running batch jobs. A useful function that the implementation does not provide, is the ability to stop and remove a running container. Once a user submits a job it can not be cancelled and has to run to completion. Completed jobs stay in the queue to allow the user to download the results and can be removed by the user.

4.1 Submit a Job

Figure 4a shows the form for submitting a new job. To submit a job the user needs to specify a tar file to upload. The tar file must contain a Dockerfile in its root directory and any other source code required to run the job. Documentation on how to write a Dockerfile is available in the Docker documentation [6]. Figure 3 shows the Dockerfile which was used in the evaluation above. A Dockerfile starts with specifying a base Docker image. The base image used in Figure 3 is identified by simonalpha/ncbi-blast-docker. Docker images can either be official or third party (created by the community), and users can search for images on the official Docker Hub [3]. The image used in this case is a third party image which already contains the NCBI Blast application [5].

```
FROM simonalpha/ncbi-blast-docker

ADD query.fasta /blast/

CMD blastp -db /db/ncbi_nr/nr -query query.fasta
          -out /results/results.out
```

Figure 3: An example Dockerfile used in evaluation.

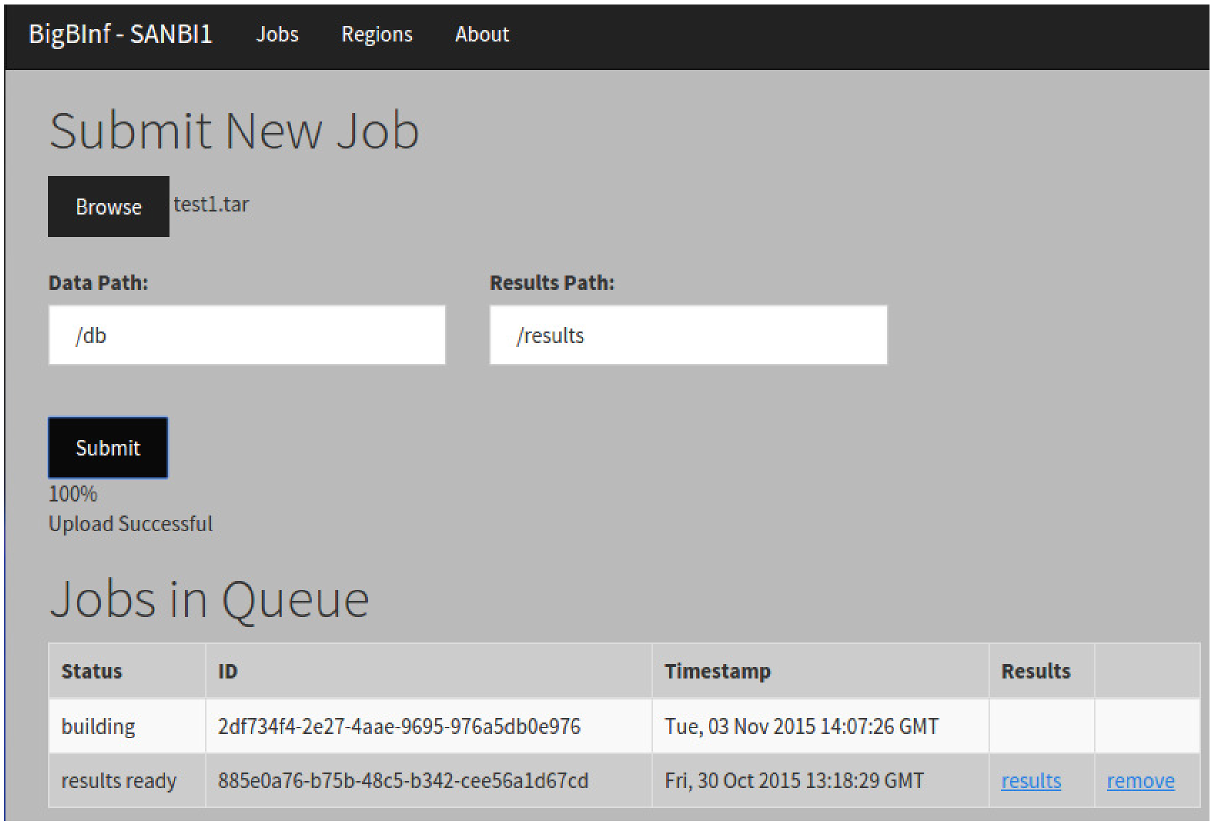
After selecting the tar file, the user has the option of specifying two mount points for the docker container, one mount point specifies the raw data path and the other specifies the directory where results will be stored. If the data path is not specified by the user then the raw data will not be mounted in the Docker container. If the results path is not specified then the user will not be able to download the results once the job has completed running.

4.2 Ceph Storage

Originally, one of the aims of the micro cloud was to allow the user to view the raw data that is available on the micro cloud and specify data to be mounted in the job's container. This would have been done by using Kubernetes to mount an RBD in the container. A single dataset would then reside on an RBD and a new dataset would be copied to its own RBD. Unfortunately we were unable to get this functionality of Kubernetes to work. We resorted to mounting a single RBD for all raw data on the host node, outside of Kubernetes and use it as host storage inside Kubernetes. This was not a problem for our single node cluster, but would become a problem for a multi-node cluster. This means that a user does not have the option of selecting raw data to mount inside the container, instead, all available raw data is mounted in every container. This was not a problem for the evaluation as we only dealt with a single dataset, it is however a major shortcoming in the functionality of the micro cloud.

4.3 Image Builder Bottleneck

Docker saves on storage space and bandwidth with regards to caching docker images. For example if one container uses an Ubuntu base image for the first time, the Ubuntu image must be downloaded. If a second container also uses the same Ubuntu image then it uses the cached copy instead of downloading the image again. This benefit comes from the layered filesystem that Docker uses. The base image used in the evaluation is about 600MB in size. This should only have resulted in a longer build process for the first job submitted (which was using that base image). The subsequent jobs should have been built in a few seconds. This unfortunately was not the case. Every job took extremely long to move from the build queue to the ready queue. A bottleneck was found after a rigorous investigation. This bottleneck occurs between the Docker builder and the image repository. This means that building the image from the Dockerfile is indeed quick, given that the base image is already cached. The problem occurs when pushing the image to the repository. Docker appears to be pushing the entire image to the repository and spends a lot of time buffering the image to disk, before starting the upload to the repository. Poor performance with pushing a large Docker image seems to be a result of Docker implementation decisions specific to push-



Submit New Job

test1.tar

Data Path: /db

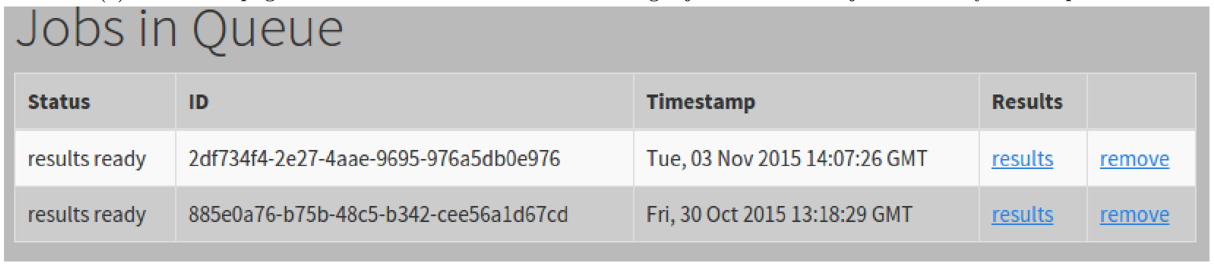
Results Path: /results

100%
Upload Successful

Jobs in Queue

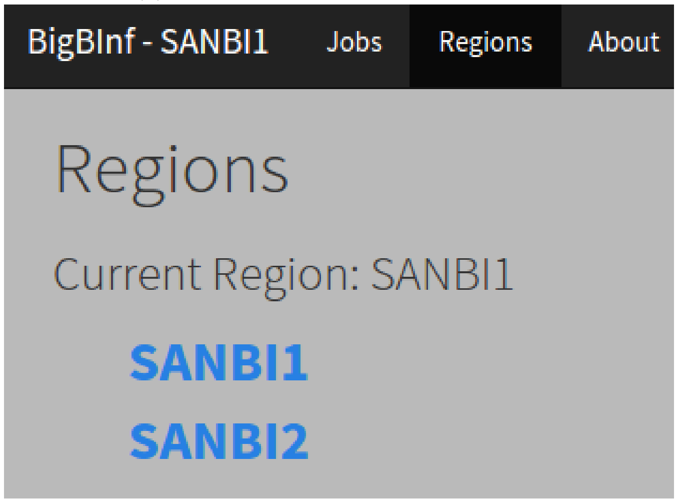
Status	ID	Timestamp	Results	
building	2df734f4-2e27-4aae-9695-976a5db0e976	Tue, 03 Nov 2015 14:07:26 GMT		
results ready	885e0a76-b75b-48c5-b342-cee56a1d67cd	Fri, 30 Oct 2015 13:18:29 GMT	results	remove

(a) The home page which contains a form for submitting a job and a list of jobs currently in the queue


Jobs in Queue

Status	ID	Timestamp	Results	
results ready	2df734f4-2e27-4aae-9695-976a5db0e976	Tue, 03 Nov 2015 14:07:26 GMT	results	remove
results ready	885e0a76-b75b-48c5-b342-cee56a1d67cd	Fri, 30 Oct 2015 13:18:29 GMT	results	remove

(b) The job queue after a job is complete



Regions

Current Region: SANBI1

SANBI1

SANBI2

(c) A list of available micro clouds

Figure 4: Micro Cloud Screenshots: The front end web interface available to users.

ing an image to a remote repository. There have been issues opened on this topic on the official Docker Github repository [11],[10].

A change in core architecture of the cluster could fix this bottleneck. The proposed design would be to remove the single dedicated docker builder and instead make every worker node a docker builder. This would mean that the scheduling of building and running jobs would be more tightly integrated, as the image must be built on the worker node that it is scheduled to run on. This would eliminate the need for a private image repository and thus eliminate the process of pushing an image to a repository. Docker is already a dependency for every worker node to be able to run Docker containers. The only problem with this architecture is that it has to be supported by the cluster manager used. In the case of this cloud platform, Kubernetes unfortunately doesn't currently support it.

4.4 Reflection

We started this project with the aim of implementing a micro cloud platform that allows users to submit jobs and process raw data. We successfully implemented a proof-of-concept micro cloud platform and successfully evaluated it with sample Bioinformatics code. We had hoped to perform the evaluation on two multi-node deployments, one at UWC and one at UCT. This was unfortunately not feasible and thus we resorted to deploying two single node clusters at UWC. The storage solution for the micro cloud was not entirely satisfactory, as users could not browse the available raw data. The data was still available for processing from within the container though, which means that we partially met that goal.

There was originally more emphasis placed on creating a community cloud, however this part of the project was mostly neglected due to the complexity of creating the micro cloud itself. Although a simple discovery service was implemented, it is not what we originally envisioned.

The aim was to focus on the core functionality of processing a job, which means we didn't take security into account. The implementation entirely neglects security and does not provide any sort of authentication mechanism. This is not a problem because it wasn't part of the original aims, but rather was considered an add-on or next step to this project.

The micro cloud platform is completely dockerized, which means it can be deployed by running a few Docker containers. The Kubernetes cluster can also be deployed using Docker containers. This makes the project reproducible, with one caveat being the Ceph storage required to complete the micro cloud deployment.

Implementing the micro cloud turned out to be more difficult than expected. The main reason being a lack of prior knowledge and experience with Docker. This made it difficult to make a good choice regarding the cluster manager. There may be a better suited cluster manager than Kubernetes, however it provided most of the funcionality required. The easiest part of implementing the micro cloud platform was the web interface, using Python and Flask for the backend and Angularjs for the front end. These parts were fairly straightforward and prior experience made it easier.

5. CONCLUSIONS

The implemented micro cloud platform is proof that using Docker containers as a lightweight alternative to virtual machines is a good way to run jobs in an isolated environment, while making the execution environment flexible and user configurable. This is important as it does not restrict a user to a limited set of configured tools, and is thus more maintainable because the administrator is not required to keep the tools up to date. A user configurable environment does not mean that it is time consuming, because Dockerfiles allow re-using a specific environment and simply adding to it the extra requirements. The public Docker repository allows people to share their configured environment with the community. We made use of the public Docker repository in the evaluation and found an image configured with the NCBI Blast application. The image worked perfectly and eliminated the need for us to create our own image containing the Blast application.

While Docker allows sharing of configured execution environments, sharing micro cloud deployments between institutions is the real focus for improving collaboration. The discovery service creates a loosely coupled community cloud. It allows the user to see a list of the connected micro clouds and choose to redirect to a different micro cloud. This is simple but effective in promoting collaboration between micro clouds.

5.1 Future Work

The micro cloud platform provides basic functionality, but entirely neglects aspects such as security. The discovery service needs to be extended to include a central authentication system which allows users from any micro cloud to login and use any other connected micro cloud. Security would also include managing access to raw data. There might be privacy concerns with certain data and more granular access control to specific data would be required to meet security standards.

There is also the issue of the docker builder bottleneck. The proposed architecture change discussed in the findings would fix this issue, however implementing this architecture would mean adding core functionality to the Kubernetes framework itself. This would not be an easy task, but would be well worth the effort. It is also possible that the Kubernetes team might add this functionality in the future, as kubernetes is still new and limited in functionality. The alternative would be to test other container manager frameworks to find a cluster manager that is better equipped compared to Kubernetes.

6. ACKNOWLEDGEMENTS

Thanks to Mr. Peter van Heusden and his colleagues (South African National Bioinformatics Institute) for enabling the evaluation of this project by providing two VMs as well as Ceph RBD storage. They were extremely helpful by providing timeous support when problems occurred. In addition, Peter also provided the analysis code used in the evaluation and helped with making the 36GB database available on the RBD storage.

7. REFERENCES

[1] Amazon web services: S3. https://aws.amazon.com/s3/, 2015.

[2] Digital ocean. https://www.digitalocean.com/, 2015.

[3] Docker hub. https://hub.docker.com, 2015.

[4] Docker image biolinux. https://hub.docker.com/r/gawbul/docker-bio-linux8/, 2015.

[5] Docker image ncbi blast+ search tool. https://hub.docker.com/r/simonalpha/ncbi-blast-docker, 2015.

[6] Dockerfile documentation. http://docs.docker.com/engine/reference/builder, 2015.

[7] Flask: Python web framework. http://flask.pocoo.org/, 2015.

[8] Google cloud platform. https://cloud.google.com/compute, 2015.

[9] Kubernetes concepts. http://kubernetes.io/v1.0/docs/user-guide/README.html, 2015.

[10] Proposal: Cross repository push. https://github.com/docker/distribution/issues/634, 2015.

[11] Proposal: push performance improvements. https://github.com/docker/docker/issues/14018, 2015.

[12] L. Affetti, G. Bresciani, and S. Guinea. adock: A cloud infrastructure experimentation environment based on open stack and docker. In *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on Cloud Computing*, pages 203–210. IEEE, 2015.

[13] E. Afgan, K. Krampis, N. Goonasekera, K. Skala, and J. Taylor. Building and provisioning bioinformatics environments on public and private clouds. In *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 223–228. IEEE, 2015.

[14] M. Baker. Next-generation sequencing: adjusting to data overload. *Nature Methods*, 7(7):495–499, 2010.

[15] C. Boettiger. An introduction to docker for reproducible research, with examples from the r environment. *arXiv preprint arXiv:1410.0846*, 2014.

[16] C. Camacho, G. Coulouris, V. Avagyan, N. Ma, J. Papadopoulos, K. Bealer, and T. L. Madden. Blast+: architecture and applications. *BMC Bioinformatics*, 10(1):421, 2009.

[17] R. Dua, A. R. Raja, and D. Kakadia. Virtualization vs containerization to support paas. In *Cloud Engineering (IC2E), 2014 IEEE International Conference on*, pages 610–614. IEEE, 2014.

[18] J. Fink. Docker: a software as a service, operating system-level virtualization framework. *Code4Lib Journal*, 25, 2014.

[19] D. Gannon, D. Fay, D. Green, K. Takeda, and W. Yi. Science in the cloud: lessons from three years of research projects on microsoft azure. In *Proceedings of the 5th ACM workshop on Scientific cloud computing*, pages 1–8. ACM, 2014.

[20] J. Hillman-Jackson, D. Clements, D. Blankenberg, J. Taylor, and A. Nekrutenko. Using galaxy to perform large-scale interactive data analyses. *Current protocols in Bioinformatics*, pages 10–5, 2012.

[21] J. Jimenez, P. Escrich, R. Baig, F. Freitag, and L. Navarro. Deploying paas for accelerating cloud uptake in the guifi community network. In *2014 IEEE International Conference on Cloud Engineering*, pages 623–626. IEEE, 2014.

[22] A. M. Joy. Performance comparison between linux containers and virtual machines. In *2015 International Conference on Advances in Computer Engineering and Applications (ICACEA)*, pages 342–346. IEEE, 2015.

[23] K. Krampis, T. Booth, B. Chapman, B. Tiwari, M. Bicak, D. Field, and K. E. Nelson. Cloud biolinux: pre-configured and on-demand bioinformatics computing for the genomics community. *BMC Bioinformatics*, 13:42, 2012.

[24] S. P. T. Krishnan and J. L. U. Gonzalez. Google compute engine. In *Building Your Next Big Thing with Google Cloud Platform*, pages 53–81. Springer, 2015.

[25] V. Marmol, R. Jnagal, and T. Hockin. Networking in containers and container clusters. *Proceedings of netdev 0.1, Feb 14-17, 2015, Ottawa, On, Canada.*

[26] S. Mathew. Overview of amazon web services. *Amazon Whitepapers*, 2014.

[27] K. D. Pruitt, T. Tatusova, and D. R. Maglott. Ncbi reference sequence (refseq): a curated non-redundant sequence database of genomes, transcripts and proteins. *Nucleic acids research*, 33(suppl 1):D501–D504, 2005.

[28] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes. Large-scale cluster management at Google with Borg. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, Bordeaux, France, 2015.

[29] F. Wang, M. Nelson, S. Oral, S. Atchley, S. Weil, B. W. Settlemyer, B. Caldwell, and J. Hill. Performance and scalability evaluation of the ceph parallel file system. In *Proceedings of the 8th Parallel Data Storage Workshop*, pages 14–19. ACM, 2013.

[30] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, and P. Fisher. The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. *Nucleic acids research*, page gkt328, 2013.