BigBInf Micro Cloud Platform

Brendan D. Ball
University of Cape Town

ABSTRACT

[Abstract]

Keywords

Big Data; BioInformatics; Clouds

1. BACKGROUND

Cloud infrastructure would go a long way in simplifying the processing of big data. Collaboration between researchers can potentially improve if they are able to process the same raw data while being in different locations. This would be done by keeping the data in a cloud solution and allowing researchers access to execute code on the raw data. There has been an explorative push towards cloud solutions from Amazon, Google etc, but there are significant drawbacks. The raw data needs to be uploaded initially and researchers often resort to mailing hard drives [Baker 2010].

An example of progress towards a cloud solution specifically for bioinformatics is Cloud BioLinux, which is a community driven project focussed on next generation sequencing. It is a toolkit which makes it easy to deploy virtual machines with bioinformatics infrastructure to a cloud platform. It bundles specific packages used in next generation sequence analysis, 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 [Krampis et al. 2012].

Micro clouds deployed on-site would overcome the challenge of uploading big data to a commercial cloud. However, since different research institutions would deploy their own micro clouds, a need for cloud interoperability arises to maintain the ability for researchers from different institutions to collaborate on the same data. The cloud interoperability will form a community cloud. A use case of a specific community cloud has some similar architectural properties to what we are looking for. These properties include autonomy (where each micro cloud will be managed independently), security, self management of nodes, and scalable [Jimenez et al. 2014].

The traditional approach to creating a cloud platform which allows users to run their own instances of operating systems (such as Amazon EC2) is using virtualisation technology. This includes both hardware level emulation support and the software needed to manage the virtualisation. These virtualisation schemes use machine level virtualisation [Fink 2014]. 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 compared to virtualisation emulating every machine instruction [Dua et al. 2014]. Of course this means that containers are only useful when complete virtualisation is not needed, instead containers allow isolated application deployment and portability.

2. DESIGN

2.1 Design Aims

The main aim of the micro cloud platform is to allow code to be uploaded by a user and execute it on local data. Users will make use of the micro cloud by interacting with a front end web interface. The web interface should be easy to use and provide useful feedback on a job that a user has submitted.

Given that multiple jobs can be submitted by multiple people, these jobs have to be scheduled properly to allow successful completion of each job. The default scheduler will rely on a first-in-first-out (FIFO) queue to schedule jobs. The whole platform should be fairly modular to allow pluggable components. This will allow possibilities such as adding a more advanced scheduler or completely replacing the back-end while keeping the front end web interface.

The final aim is to form a community cloud using a centralised discovery service. This will improve scalability compared to a full mesh network between micro clouds. The community cloud will allow sharing of data and collaboration of users between multiple micro clouds.

2.2 Constraints

2.2.1 Jobs

Jobs can only be run via docker containers. To create a job you have to create a Dockerfile.

ACM ISBN insert isbn here.
DOI: insertdoihere

2.2.2 Micro Cloud Deployment

Micro cloud deployments for evaluation are limited to UWC and UCT E-Research. This means that the community cloud that will be evaluated will only consist of two micro clouds.

2.3 Approach

Every software project needs a version control system (VCS) to store source code. Git will be used as the VCS for this micro cloud platform. Software also needs to be verified for correctness to decrease the chance of bugs occurring. This will be done by writing unit tests to allow continuous testing at all stages of the project.

2.3.1 Collaboration

This project is a collaboration between UCT and UWC. We met with the supervisors at UWC and continued communication with feedback was ongoing throughout the project lifetime. UCT E-Research was subsequently also brought onboard the project. Both UWC and UCT E-Research provided valuable hardware resources enabling thorough evaluation in a real world setting.

2.3.2 Kubernetes

Kubernetes has been chosen as the primary backend framework for managing the micro cloud. Kubernetes is specifically designed to run docker containers in a cluster. It is still a new system and Kubernetes version 1 has just recently been released which means it's ready for production use. Using containers instead of virtual machines has only recently become popular. The growth in popularity has resulted in a number of systems being developed which provide some overlapping functionality to Kubernetes. Kubernetes looks most promising as it is easy to use. Given that it is relatively new it is not cluttered and aims to solve a single problem of being a container manager.

2.3.3 OpenStack

OpenStack was the first system looked at as a base for a container manager. After much trouble trying to get it set up properly in a development environment it was decided that there are more suitable systems to solve the problem. OpenStack is very cluttered with many inter dependent components. The OpenStack Swift API will however be supported for storage as it is very likely that the micro cloud platform could be deployed on top of OpenStack.

2.4 Evaluation

2.4.1 Functionality

The functionality of the micro cloud will be evaluated by submitting sample analysis code as a job. The analysis code will be searching against a genome database from NCBI's NR database which is publicly available. The analysis code will make use of the BLAST+ search tool. The code itself will be custom queries. Different sized jobs can be tested by increasing or decreasing the size of the query which makes it easily scalable to have different sized test cases.

2.4.2 User Interface

The evaluation of the user interface will be qualitative in nature. It will be based on user feedback from the project supervisors, both from UWC and UCT. The main aim of the project is not to test user experience design, although it is acknowledged to be very important, which is why a few expert users will be testing the user interface.

2.5 Implementation

The micro cloud platform will make use of containers, more specifically Docker containers to execute code. Docker images are created using Dockerfiles which specify the execution environment and what files to execute. This means that submitting a job will require creating a Dockerfile and uploading this along with source code which will be executed. Docker containers will be managed using Kubernetes as specified above in the approach section.

2.5.1 Storage

The main problem that this project is trying to solve is processing Big Data. This means that we have to be able to access the data that sits on the micro cloud. The two most popular storage APIs are OpenStack Swift API and S3 API. By supporting both of these storage APIs it should be relatively easy to deploy the micro cloud on a large range of different clusters.

2.5.2 Web Interface

The backend web interface will be implemented using the python flask web framework. It is a minimalist framework which allows for rapid development of web interfaces. The front end web interface will be implemented using Angularjs to create a single page application.

3. REFERENCES

[Baker 2010] Monya Baker. 2010. Next-generation sequencing: adjusting to data overload. *nature methods* 7, 7 (2010), 495–499.

[Dua et al. 2014] Rajdeep Dua, Dharmesh Kakadia, and others. 2014. Virtualization vs Containerization to support PaaS. In *Cloud Engineering (IC2E), 2014 IEEE International Conference on*. IEEE, 610–614.

[Fink 2014] John Fink. 2014. Docker: a Software as a Service, Operating System-Level Virtualization Framework. *Code4Lib Journal* 25 (2014).

[Jimenez et al. 2014] Joaquin Jimenez, Pau Escrich, Roger Baig, Felix Freitag, and Leandro Navarro. 2014. Deploying PaaS for accelerating cloud uptake in the Guifi community network. In *Cloud Engineering (IC2E), 2014 IEEE International Conference on*. IEEE, 623–626.

[Krampis et al. 2012] Konstantinos Krampis, Tim Booth, Brad Chapman, Bela Tiwari, Mesude Bicak, Dawn Field, and Karen E Nelson. 2012. Cloud BioLinux: pre-configured and on-demand bioinformatics computing for the genomics community. *BMC bioinformatics* 13, 1 (2012), 42.

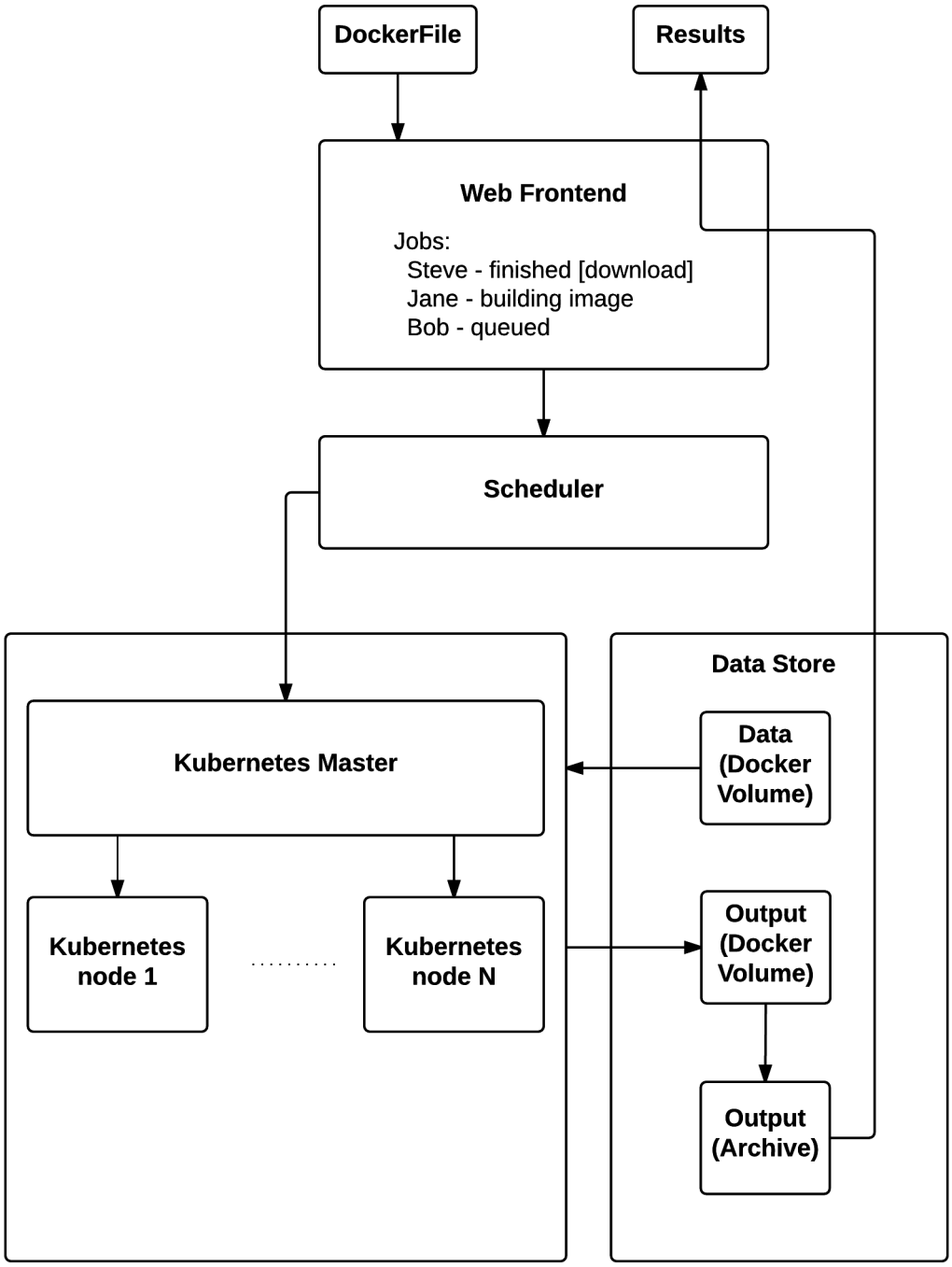

Figure 1: Micro Cloud Architecture