BigBInf Micro Cloud Platform

Brendan D. Ball
University of Cape Town

ABSTRACT

[Abstract]

Keywords

Big Data; BioInformatics; Clouds

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 [Krishnan and Gonzalez 2015, Mathew 2014]. 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 transferring Big Data is important because limitations in current internet infrastructure makes collaboration difficult when Big Data is involved.

When using a commercial cloud such as Amazon EC2, the problem of uploading big data sets to their cloud infrastructure still exists [Baker 2010]. Micro clouds deployed on-site would 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. The cloud interoperability creates a community cloud. A use-case of a specific community cloud by Jimenez et al. [Jimenez et al. 2014] contains core architectural properties needed for a successful community cloud. These properties include autonomy (where each micro cloud is be 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 [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]. Containers are only useful when complete virtualisation is not needed, but allow for isolated application deployment and portability. Using containers instead of virtual machines has only recently become popular. 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 required 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 [Krampis et al. 2012]. This toolkit makes it easy to deploy virtual machines to a cloud platform with bioinformatics infrastructure preconfigured. 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.

Another project compatible with both Amazon EC2 and Eucalyptus is CloudMan [Afgan et al. 2015]. CloudMan is designed as a workbench which manages cloud resources. A workbench relies on a pre-configured environment on which to execute code. It is aimed at improving 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.

ACM ISBN insert isbn here.
DOI: insertdoihere

We aim to provide users with a configurable environment by taking advantage of open source cloud software and Linux containers to enable efficient processing of Big Data through the use of micro clouds. The aim is to build a micro cloud platform 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, one at UWC and one at UCT. 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, as is UCT E-Research. Both UWC and UCT E-Research provided valuable hardware resources enabling thorough evaluation in a real world setting.

2. BACKGROUND

Docker is built on top of Linux containers and is the main technology that we are using to build the micro cloud. To build a micro cloud platform a cluster manager is required. 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 uses the Linux kernel on the host machine to run native instructions. Containers are lightweight and launched in a fraction of time of a virtual machine [Joy 2015].

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 [Boettiger 2014].

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 creating an image from a running environment, as is the case with virtualisation [Boettiger 2014].

The Docker group also hosts a publicly available image repository where anyone can upload or download available Docker images. These images provide full disclosure of its configuration via its Dockerfile. This provides great community support as you will often find that an image already exists for specific use-cases. An example is of CloudBioLinux which officially only provides virtual machine images, however third parties have uploaded Docker images of CloudBioLinux available for anyone to use.

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 [Affetti et al. 2015]. 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.

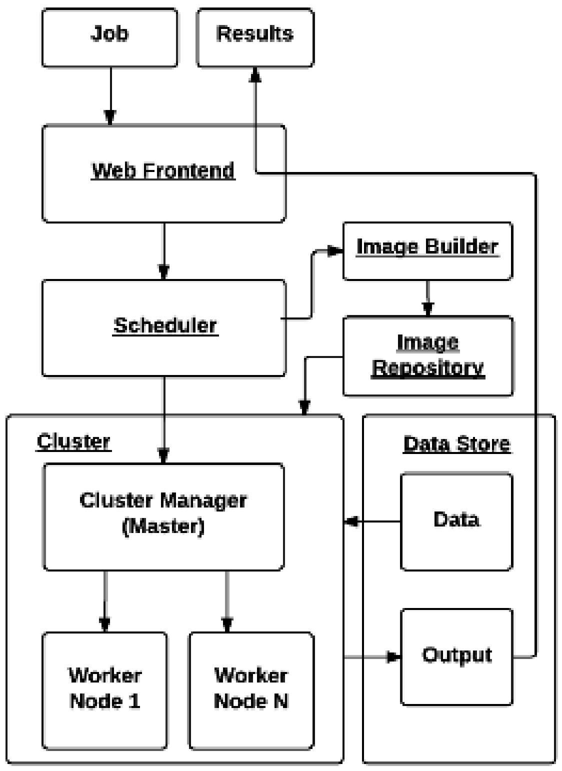

Figure 1: Micro Cloud Architecture

2.3 Kubernetes

Kubernetes is a pure container manager specifically designed to orchestrate docker containers. A new system, Kubernetes version 1 has recently been released. 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 [Verma et al. 2015].

Google has been using 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 [Verma et al. 2015].

3. DESIGN AND IMPLEMENTATION

Cloud infrastructure consists of one or more clusters. A cluster is a collection of servers (nodes). Data storage is also part of cloud infrastructure, but is typically seperated from computing 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 components.

The functionality of the platform is of primary concern, thus user interface design is not considered and is outside the scope of this project.

3.1 System Architecture

In Figure 1 you already saw an overview of how components interact. This section gives implementation specific details for every component.

3.1.1 Job

A job describes a single instance of code uploaded for execution. The micro cloud platform uses containers, more specifically 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 tarball.

3.1.2 Cluster Manager

To create a cloud platform which makes use of 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 has been chosen as the cluster manager framework for managing the micro cloud.

3.1.3 Docker Images

A dedicated docker-in-docker container is run which is used to build docker images from dockerfiles. The image is then 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

The default scheduler relies on a first-in-first-out (FIFO) queue to schedule jobs. There are two parts to the scheduler. The first part schedules new jobs' images to be built from the Dockerfiles. The second part schedules the jobs to be executed on one of the worker nodes.

3.1.5 Storage

The cluster at UWC makes use of Ceph storage which can be accessed by a RADOS Block Device (RBD). Kubernetes already provides support for mounting an RBD as a storage volume in a container.

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. The front end web interface is implemented using Angularjs to create a single page application.

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.

3.2 Constraints

Micro cloud deployments for evaluation are limited to UWC and UCT E-Research. The evaluation is limited to one community cloud containing two micro clouds.

3.3 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).

3.4 Evaluation

The functionality of the micro cloud is evaluated by submitting sample 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 [Pruitt et al. 2005]. The analysis code makes use of the Basic Local Alignment Search Tool (BLAST) [Camacho et al. 2009]. This tool is used by specifying a database and submitting a query containing specific search criteria. Different sized jobs are tested by increasing or decreasing the size of the query, which makes it easily scalable to different sized test cases.

4. REFERENCES

[Affetti et al. 2015] L Affetti, G Bresciani, and S Guinea. 2015. aDock: A Cloud Infrastructure Experimentation Environment Based on Open Stack and Docker. In *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on Cloud Computing.* IEEE, 203–210.

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

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

[Boettiger 2014] Carl Boettiger. 2014. An introduction to Docker for reproducible research, with examples from the R environment. *arXiv preprint arXiv:1410.0846* (2014).

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

[Dua et al. 2014] R Dua, A R Raja, and D Kakadia. 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] J Jimenez, P Escrich, R Baig, F Freitag, and L Navarro. 2014. Deploying PaaS for accelerating cloud uptake in the Guifi community network. In *2014 IEEE International Conference on Cloud Engineering*. IEEE, 623–626.

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

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

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

[Mathew 2014] S Mathew. 2014. Overview of Amazon Web Services. *Amazon Whitepapers* (2014).

[Pruitt et al. 2005] K D Pruitt, T Tatusova, and D R Maglott. 2005. NCBI Reference Sequence (RefSeq): a curated non-redundant sequence database of genomes, transcripts and proteins. *Nucleic acids research* 33, suppl 1 (2005), D501–D504.

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