HemeWeb: Blood flow simulation in the cloud using docker

Steven Steven

Master of Science
School of Informatics
University of Edinburgh
2016

Acknowledgements

I would like to thank Indonesian Government, specifically Indonesian Endowment Fund for Education (*Lembaga Pengelola Dana Pendidikan / LPDP*) of Ministy of Finance. With their help, I can study in University of Edinburgh from start to end without any problems.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Steven Steven*)

Table of Contents

List of Figures

Chapter 1

Introduction

Software is increasingly complex. Our everyday software is packed with features that make its usage difficult. To people without familiarity with the product, complex usage can be a barrier to use the software even when it can help them tremendously.

This also ties into the complexity of a research that use this software. Open science dictates that research should be reproducible or replicable for it to better validate the research. However, recent findings have shown that not many research in psychology are replicable.

1.1 Motivation

To study how blood flow in a given vessel, [2] developed a fluid dynamic simulation software named HemeLB. Currently, it is actively developed and used by researchers to help their study. For example, [3] used HemeLB for automated ensemble simulation of blood flow for a range of exercise intensities, [4] used it for detecting the difference of retinal hemodynamics with regards to diabetic retinopathy, and recently [5, 6] used it to understand branching pattern of blood vessel networks.

As I have written in the proposal for this dissertation [1], HemeLB works by calculating fluid flow in parallel by using lattice-Boltzmann method [2]. This calculation allows HemeLB to simulate blood flow within a given blood vessel structure. Unfortunately, the calculation part is only a small part of the workflow to run the simulation. There are multiple pre-processing and post-processing steps needed to run the simulation from start to the end. These includes of preparing the input so HemeLB can work on it, and also processing the output so it is ready to view.

These long pipelines of steps needed to run the simulation, coupled with the com-

plexity of the configuration of HemeLB created a high barrier of entry for scientists and doctors to use it. Furthermore, as observed previously [1], an interesting simulation will require parallel computing resources like ARCHER supercomputer which might be difficult to get access to by interested parties. While smaller simulation instances can run on a typical laptop, most of the problems will require more powerful machines.These facts might prevent usage of the software by interested parties. More importantly, it shows there are still improvements that can be done to lower the barrier of entry for users to use HemeLB. This is important for HemeLB, especially when it is envisioned to be an integral part of future medical decision [7].

Another aspect that HemeLB workflow can be improved is with regards to its reproducibility aspect. Researches that used HemeLB embrace reproducibility as one of its concern. As observed before [1], There are steps that are in place to make sure HemeLB and its simulation result are reproducible. First, the entire code base is publicly available on Github. Second, in running a simulation with HemeLB, version of the software is automatically recorded. Lastly, in addition to the version used, input files and configurations are also recorded automatically. These facts can be seen from the publications mentioned above [4, 3, 5, 6] that include all these information. This information allow researchers interested to replicate the simulation to do it manually. Automation of these steps could further improve HemeLB's reproducibility and allow peers to replicate, duplicate, and audit its simulation results quickly and easily. This automation will be important, in addition to being usable, for HemeLB to become an integral part of the medical decision in the future.

1.2 Objectives

Based on the needs to improve the usability. reproducibility, and auditability aspect of HemeLB project, I will develop a prototype web interface for HemeLB. This prototype web interface will lower barrier of entry in using HemeLB software compared to the current approach of using command line interface. In addition to that, using web interface will also allow features to be added to the simulation workflow which might not be essential to the HemeLB core itself. For example, automating packaging, sharing, and reproducing simulation result. These features are not essential for the HemeLB core, but definitely, help the overall workflow of blood flow simulation.

Using the dynamic capabilities of cloud computing vendor, the web backend should be able to dynamically scale without many efforts. On top of that, these infrastructures

are available to everyone with a cost, allowing its user to access it without having to get access to supercomputers. Its user should be able to run a blood flow simulation without having to deal with the complexity of running each step of the workflow manually.

In addition to the web interface, I will also develop deployment script so that peer could deploy its own instance of the web interface. This will ease up deployment process for individuals or organizations intending to use HemeLB for its own purposes. This script will be developed as part of the project.

1.3 Outline

I provide a brief introduction to the topic of this dissertation in this chapter. The rest of the chapters will be organized as follow:

- **Chapter 2, Background**. I will provide background information that is necessary for readers to understand the concepts, technology, and implementation that are done in this dissertation. HemeLB, containerization technology, cloud computing, High-Performance computing infrastructure, and other topics will be discussed in details in this chapter.
- **Chapter 3, Design and Implementation**. I will discuss the bulk of the work in this chapter. Implementation details and design of the proposed solutions will be provided and discussed in details.
- **Chapter 4, Evaluation**. In this chapter, I will discuss how the success of this project will be measured. I detailed how I conduct an interview and performance benchmark to measure how the proposed solution perform.
- **Chapter 5, Result and Analysis**. I will discuss my findings from the evaluation of the project.
- **Chapter 6, Conclusion**. I will conclude and summarize the whole project.
- **Chapter 7, Future work**. Some recommendations on the future of the project in the context of HemeLB simulation workflow.

Chapter 2

Background

In this chapter, I will discuss about various background information that are used as a basis for the work presented in this dissertation. I will discuss about how HemeLB currently works, High-Performance Computing (HPC) infrastructure, and docker.

2.1 Usability of complex applications

2.1.1 Current HemeLB workflow

Currently, running a blood flow simulation consists of multiple steps that needs to be run in sequence. These steps are done in a variety of interface, from command line to graphical user interface. Additionally, these steps also require various level of computing resources to work efficiently. In order to understand how the proposed work will improve the current conditions, I will elaborate on how HemeLB currently work. Also, discussion on computing resources and interface for each steps will be provided.

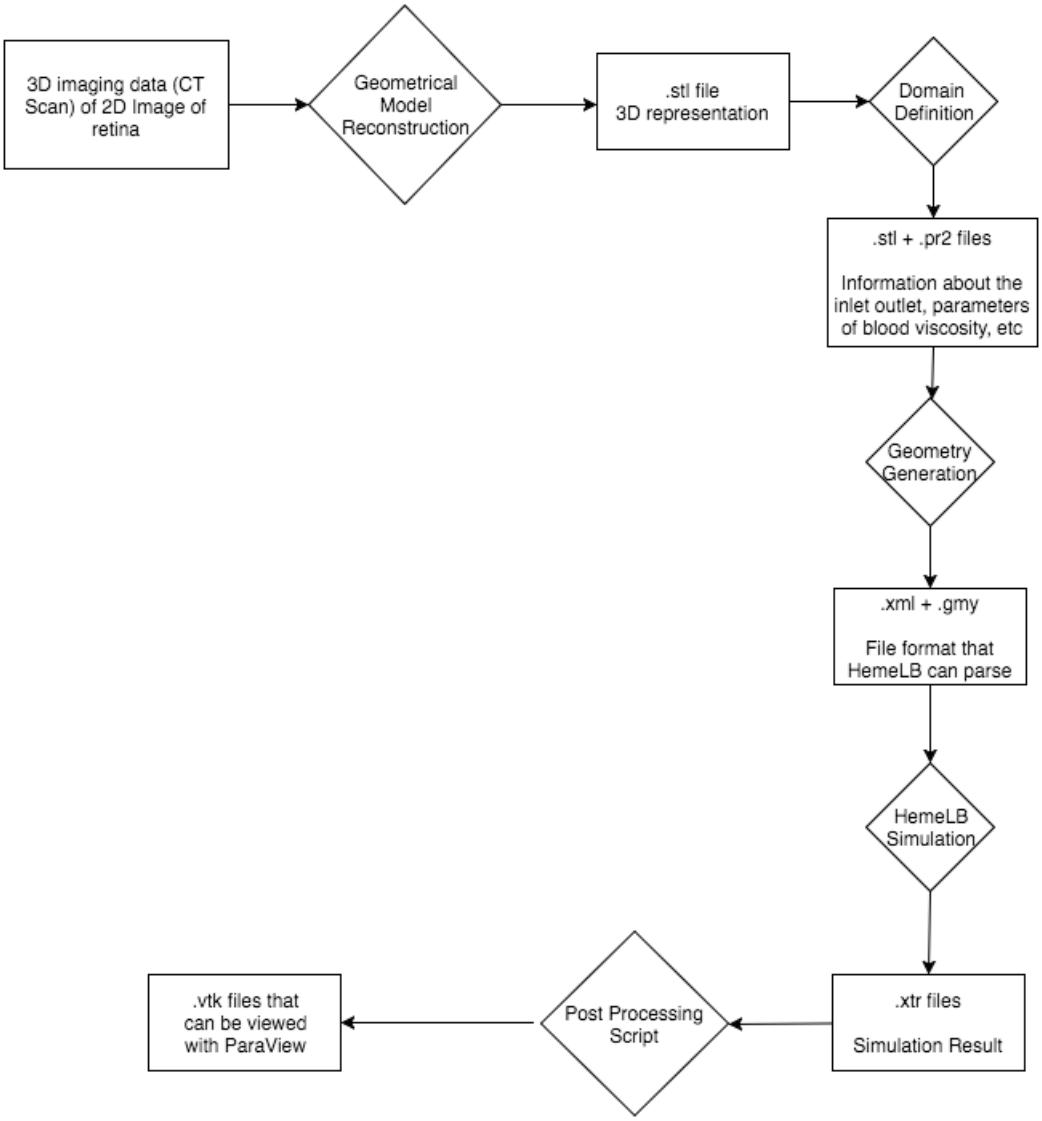

Figure 2.1: Current HemeLB workflow taken from [1]

Figure 2.1 illustrates steps involved in running HemeLB workflow. These steps will be discussed in details below:

1. **Geometrical model reconstruction**

   In this step, a 3D model of vascular system is constructed from the raw microscopic image of it. Alternatively, the 3D model can also be constructed from CT scan with its 3D imaging data. From this step, a 3D geometry file are generated in the form of .stl file. This process can run in a regular workstation just fine. However, it is highly problem dependent as the tools needed to parse and generate the 3D model are dependent to the problem experts try to simulate.

2. **Domain definition**

   3D geometry model generated from the previous step is now used as an input for the domain definition step. In this step, a graphical user interface is used to add domain information to the 3D model. Information like blood viscosity, inlet outlet placement, and blood pressure will determine how the simulation will run. The HemeLB setup tool was developed for this particular needs. The setup tool provides a graphical user interface for domain experts to add these parameters. All parameters are then saved in a profile file with .pr2 format. This step can run on standalone commodity hardware and should not require a highly parallel computing resources.

3. **Geometry generation**

   This step will take the encoded information from domain definition step and the 3D model of the vascular system to generate files that can be understood by the main HemeLB program. These files contain similar information with the previous 3D model and the profile file. However, both of them are now formatted in a HemeLB parseable format, an XML configuration file and a GMY geometry file. This geometry generation step can also run on a commodity hardware. However, it requires users to use command line interface to operate with the files. The process is done with piping the input files to a python script which is part of HemeLB setup tools.

4. **HemeLB simulation**

   The main heavy computations of the workflow are done in this step. Configuration and geometry files that are generated in the previous step are feed into the HemeLB binary as input files. HemeLB will then run calculations that govern how blood will flow inside the provided vascular system for a number of iterative steps. The number of steps is defined in the configuration file that is generated in the domain definition steps.

   As observed in the proposal of this work [1], HemeLB can scale up from 1 up to 32,000 cores in running the simulation [8]. This means that a typical problem could run in a commodity hardware with a small number of cores. However, bigger and scientifically challenging problems will require a higher number of cores that requires high-performance computing resources as portrayed in [6, 5] and [4]. In addition to that, users of HemeLB have to use command line interface

to configure, run HemeLB simulation, and interact with the output files.

Output files generated by this step are written in parallel into output directory which is set when running the simulation. These output files represent the state of blood flow in the vascular system at a given step count. The interval in which HemeLB writes an output is also set from the domain definition step.

5. **Post processing**

   The output files generated by the HemeLB simulations are not easily viewed by domain experts. The files are generated in a fashion that is efficient to write in parallel, however, it will need further conversion to make it easy to be interpreted. This is where the post-processing step comes in.

   In this step, the output files are piped into two python scripts that are included in HemeLB tools to convert them into a .VTU files. These .VTU files are viewable in a separate software called ParaView. With it, domain experts could visualize the result of the simulation in a graphical user interface which ParaView provided. This step can be run on commodity hardware without problems. However, to do this step, users will require to interact with command line interface.

All of these steps requires users to configure and install the tools required for each step by themselves. However, the target audience of HemeLB are biologists, clinician, researchers, and medical professionals, which might not have the capabilities and the technical know-how to do it. This is one of the motivating reasons for the proposed work in this thesis.

2.1.2 High performance computing infrastructure

Researchers increasingly use complex mathematical and computational approaches in doing their research. In understanding complex phenomenon, researchers use interdisciplinary approaches in providing insight into the problem [9]. Bioinformatics and computational biology are examples of this interdisciplinary discipline.

The problems that these disciplines tries to understand require computational approaches that are not cheap. The smaller size of these problems would probably run on a commodity hardware, more complex one will require highly parallel computing resources. These resources are often be found in the form of a supercomputer like ARCHER supercomputer. HemeLB software package is the prime example of these computational approach that requires highly parallel computing resources.

To tackle these type of problems that require large computing resources, two paradigms of computation is developed. These are High Performance Computing (HPC) and High Throughput Computing (HTC). While both of these disciplines are developed to solve problems that require large computing resources, they are different in the nature of the problems they are trying to solve.

High performance computing uses uniform computing nodes to perform a tightly coupled computation. They are often placed in one location and connected to high bandwidth network amongst them. This network connection allows these nodes to communicate with each other efficiently, thus, allowing them to coordinate computation across the nodes[10]. A message passing interface (MPI) library is often used to perform this type of computation and it allows every process to communicate with each other in parallel. As observed in the proposal, computer clusters, GPUs, and supercomputers are the prime example of computing resources to run this type of computation.

On the other hand, High throughput computing tries to treat computing resources like a utility line. Users should not have to worry about where the computing resources come from, they can just request it and it will be given. This type of paradigm uses a middleware that allows non-uniform computing resources to communicate and cooperate in order to solve common problems[10]. These differing resources will then do different works that are scheduled independently.

HemeLB uses MPI library to communicate between processes and run tightly coupled computations on each of these processes. Based on the definitions outlined above, we can safely categorize HemeLB as an HPC application that requires an HPC infrastructure to run its computation efficiently. These computations will simulate how the blood will flow in the blood vessel.

Running HPC application like HemeLB requires access to HPC infrastructure that is often managed by a university or a research facility. These institutions give access to HPC projects by computing hours on the basis of the merit of their proposal. For example, this is how the Partnership for Advanced Computing in Europe (PRACE)[1] and Engineering and Physical Science Research Council (EPSRC)[2] operate. They conduct a peer-review of proposals that indicates the need for their infrastructure and give access selectively.

The model of operation of this institute inherently discourages reproducibility of a research. Experts with reproducibility purposes have to compete for the limited amount

[1] http://www.prace-project.eu

[2] https://www.epsrc.ac.uk

of available computing hours with other projects. Most likely, reproducibility of past studies is not the top priority of this institutes, causing a barrier for reproducing computational research that relies on this type of infrastructure.

In addition to the above problem, most research that tackles biological and biomedical discipline often fall outside the scope of these institutions. Also at the time of writing, the institutions on these disciplines do not have access to HPC resources. These problems limit the capabilities of researchers to reproduce past studies easily.

2.1.3 Cloud computing

To answer the huge demand for computation powers by researchers and academics, a concept called grid computing was born in the 1990s [11, 12]. This concept treats computing resources like utility grid. Computing resources should be able to be acquired without users knowing how it was provided to them. This model caters mostly to the interest of researchers and academia that usually give CPU hours based on projects proposal [13]. An example of this is TeraGrid which ended its operation at 2011 [3].

Cloud computing paradigm was then developed based on the similar idea that computing resources should be available to the users without the user knowing where it came from. However, this is where the similarity ends. Cloud computing caters more towards the business aspect of these computing utilities. While grid computing prioritizes features and functionalities that researchers and academia would like, cloud computing vendors focus on features that business will pay. Cloud computing vendors are driven by economies of scale and it will not survive if businesses do not use their service [13].

Conditions outlined above have created a tight feedback loop between users and the cloud vendors. This led to the development of features that users need and will pay for. As observed in the proposal, Cloud vendor now is massively scalable, allows computing resources abstraction, configurable dynamically, and provisioned on-demand. This has led cloud computing vendors to be more relevant compared to grid computing.

Cloud vendors continue to grow significantly in the recent years. In 2013, it was reported that some cloud vendors reached had more than 90% growth per annum [14]. This growth enables them to keep more incentives for businesses and individuals to buy their services. In few instances [15, 16, 17], cloud vendors have cut their pricing for their service and fueled more demands. Renting computing resources is getting

[3] https://www.xsede.org/tg-archives

cheaper every year and could make more sense than building your own infrastructure.

On top of that, cloud vendors also do not vet projects based on their proposals. Projects could easily get access to computing resources as long as they can pay for it. The business mindset of this cloud vendors allows reproducibility to be a priority in research, unlike requesting resources from research institutes. This scenario is perfect for researchers and institutions that do not own their own HPC infrastructure. Instead of building their own, they can rent from the cloud vendors and does not have to worry about maintenance.

Conditions above are also capitalized by cloud vendors like Amazon by attracting customers that need computing resources for HPC applications [10]. While it is reported that running HPC applications on cloud platform will incur performance overhead, it is a viable alternative to having your own dedicated infrastructure. This is shown in various studies in the past, for example, Nekkloud project [18], NASA HPC applications [19], and an HPC application benchmark in public cloud [20]. In addition to that, the capabilities to massively scale your computing resources, limited by one's purchasing power, is an attractive feat for HPC application that needs scalability. This is also why HemeLB software package can rely on the cloud platform to scale up or down depending on the problems.

2.1.4 How other HPC applications tackle usability issues

In this section, I will discuss how similar HPC applications solve problems like HemeLB faces. Recently, some complex HPC software packages have been deployed to the cloud. Their experience in deploying the software and their approach to solving similar problems will be indispensable for the work that I proposed.

The first project that tackles a similar problem like HemeLB is Nekkloud. In this project, Nektar++ faces usability issue just like HemeLB. It is a complex high-order finite element code which mainly is operated using command line interface. The original workflow is complex and only limited number of people can operate it, barring people without computer expertise into actually taking advantage of the software package.

This usability problem coupled with the fact that users have to get access to HPC infrastructure, might not be a viable option for everyone. In order to answer these problems, Nekkloud project was hatched. Nekkloud was developed to make the software configuration invisible to the users. It uses a web application to provide an easy to use and learn interface for the domain experts. This allows people without computing ex-

pertise to actually run Nektar++ without having to be troubled with configurations. In addition to solving the usability problems, Nekkloud also uses public cloud vendors, alleviating concerns with regards to having dedicated HPC infrastructures from users.

Galaxy [21] is another project that is trying to tackle similar problems. It described itself as a web-based reproducible research platform and ran on public cloud infrastructure. With Galaxy, researchers can run various HPC applications that are compatible with an easy to use web interface. Users do not have to worry about the configurations and working in command lines, only worry about the software executions.

Developers of Galaxy developed a super-resolution spark (SRS) model to illustrate its use case. This model is an example of HPC application that requires a highly parallel computing resources like a supercomputer to run efficiently. However, running the SRS model in the cloud via Galaxy is observed to be a viable alternative. In addition to that, Galaxy provides an easy to use interface to run this model for the users. Making it easy for them to run their experiments and share it with their peers.

Nekkloud and Galaxy projects illustrate that web interface is a viable alternative interface for complex HPC applications. With the correct application and design, it will allow domain experts to operate the underlying HPC application with relative ease. However, the change of the underlying infrastructure from dedicated HPC infrastructure to the public cloud have some negative impacts.

One notable drawback is the lower performance of the HPC applications. It has been studied that running HPC applications on the public cloud means performance degradation compared to the dedicated HPC infrastructure. This performance penalty was observed in the mentioned project like Nekkloud, NASA HPC application, and Galaxy. However, this penalty is not so significant that it outweighs the benefit of making the HPC application more usable to the users.

2.2 Reproducibility and Auditability of software

In this section, I will discuss the problem of reproducibility and auditability and how the proposed components try to solve it

2.2.1 Reproducibility problem in computational research

American Physical Society [22] describes science as ”the systematic enterprise of gathering knowledge about the universe and organizing and condensing that knowledge

into testable laws and theories". For theories and experiments to be testable, it has to be independently reproducible or replicable by peers. However, a recent study [23] highlights that some published psychology studies are not replicable to the same significance as reported. In machine learning conferences [24], a similar sentiment is being shared. Being not replicable does not necessarily means that the result of those studies is wrong. However, it shows how much the academia do not prioritize verification of results. The novelty of idea is seen as better than verifying what we already know.

With the recent study highlighting a reproducibility problem of a research, previous pushes [25, 26] for reproducibility in studies become more relevant than ever. This is especially crucial in a discipline like computational research. Computational research discipline like bioinformatics and computational physics involves complex computation that requires huge computational power depending on the problem size. The complexity of configuration, algorithm, and execution process actually become a barrier for peers to replicate and reproduce works of studies of this discipline. Making results produced have less weight than it could be

This complexity is also one of the reasons why Galaxy project came into development [21]. While science values rigorous testing, replicable, and reproducible results, the complexity of HPC software package hinders that. Galaxy tries to solve that by producing automatic metadata that record the execution of an analysis done by tools on Galaxy platform. It records all the input files, configurations, and outputs of an analysis and makes it available for the users to view and copy. This metadata information enables users to share the analysis with their peers. Allowing peers, to replicate or reproduce the analysis results independent of the original researcher's. Galaxy allows the ease of reproducibility and replicability for the domain experts.

2.2.2 Containerization Technology and HPC application

One of the main problems with replicating or even reproducing results of studies with complex software package is configurations. A complex software package like HPC application often requires hands-on configuration by the users for it to run correctly. This complex configuration coupled with complex usage become barriers for an independent party to verify results of a study.

Containerization technology can help with the issue above. It is a technology that is developed on top of Linux Containers (LXC) which allow application virtualization. Containers, allow applications to be securely placed in its own environment that shares

the kernel with its host. Unlike full virtualization, containerization does not require a full operating system installed on the virtual environment so the application could run. Docker[4] is arguably the most popular containerization technology currently. It is built on LXC and added features that allow an application to be deployed easily. It handles container image creation, versioning, sharing and archiving in addition to running the image like LXC does. On top of that, there is also a web application called docker hub[5] that allows Dockerfile, a text file that contains the commands to build a container, to be inspected publicly if set correctly. It allows interested peers to audit a container and make changes to it for their own container.

Using containerization technology like docker can help with reproducing results of studies. Complex configuration process can be hidden with the containerization technology. Studies can package the complete software package in a docker container to be shared with peers for independent replication and reproduction of an analysis. In addition to that, the versioning feature of docker also allows the software package to continuously developed and for an analysis tied down to an older version of the package. An analysis can be replicated even if the software is currently way ahead compared to the time an analysis is made. It also creates a good opportunity for the study to be reproduced, an analysis could have different results if it is running with the new version of the package. Hence, containerization technology enabling reproducibility of results.

This is also the reason why Galaxy project supports docker in its toolsets. Galaxy ecosystem uses docker to create a secure, isolated environment for the tools and its dependencies [27]. Tools are versioned, archived, and shared with docker containers so peers can download and audit all the tools. This openness is also important for the tools to be trusted.

HemeLB as a complex HPC application is also actively developed. While development is ongoing, it is often used as part of a research. Docker can help with the recording of the version used for a study by its versioning feature. In addition to that, configuration work is also taken out of the hands of the peer because it is done initially during the packaging of the application. Containerization technology sounds appropriate to be used for HemeWeb to enable easy reproduction, replication, and audit for HemeLB simulation.

[4]https://www.docker.com/
[5]https://hub.docker.com

Chapter 3

Design & Implementation

In this chapter, I will discuss HemeWeb's development and implementation. This will consist on how the HemeLB core docker container is developed, how it is deployed, and how the web application is developed.

3.1 HemeLB core docker container

HemeLB core is a docker container that consists of essential software and services needed to run HemeLB simulation. It is the main component that runs the HemeLB simulation on the cloud. Calculations will be done on the container which will be started on the compute nodes of the HemeWeb architecture.

Previously, there was an effort to make HemeLB software package portable by creating its own container package[1]. It has all the tools needed for HemeLB simulation workflow. Setup tools, post-processing scripts, and a VNC server that allows users to interact with the container's desktop using a browser. It is a step in a right direction towards making HemeLB portable and for peers to replicate and reproduce results of a simulation. However, HemeWeb will not require all those tools installed.

In this phase, I took the previous container and modify the dockerfile, the instructions file to build the container. It is a straightforward process of stripping out the tools not needed by the compute node to run HemeLB simulation. Once all the tools are removed, I modified the base image of the docker container from base ubuntu provided by docker, to base ubuntu provided by phusion. This change of base image allows the container to correctly start SSH service. It is needed to start SSH for the purpose of multi-container HemeLB simulation executions. The original base image has the prob-

[1] `https://github.com/mobernabeu/docker-hemelb`

lem in starting up services and the workaround is more troublesome than just changing the base image. The resulting container is a minimal container that contains only the HemeLB binary and SSH service running.

The modification of the dockerfile of the container is the initial part to make HemeLB core container created correctly. To correctly create the HemeLB container, this dockerfile needs to be integrated into the development workflow of HemeLB. Currently, the modified dockerfile still live under the HemeWeb codebase[2]. HemeLB development should trigger an automated build of HemeLB core containers with each version of the software it pushes to the public GitHub repositories. In addition, the development team also needs to create a consistent tag naming in order for the HemeLB core containers to be created correctly.

3.2 Deployment script

After creating the HemeLB core container, I proceed to create a deployment script to configure the overall software architecture. It contains many moving parts and configuring the architecture by hand will soon be unmanageable. Deployment script that I created tries to alleviate the pain of deployment by provisioning and configuring the architecture with minimal manual intervention. It is created using a configuration management and orchestration tools called ansible[3].

The basic goals of the script are as follow:

1. Provision the required master instance from cloud vendors
2. Configure the master instance with the correct security and network settings
3. Configure and install all the services needed by HemeWeb
4. Provision required compute instances from cloud vendors
5. Configure the compute instances with the correct security and network settings
6. Configure the compute instances to run HemeLB core docker container

The development of the script is straightforward. I used various modules that are available to ansible, including cloud vendors module, to automate the process as much

[2] https://github.com/SeiryuZ/HemeWeb/tree/master/hemelb_docker
[3] https://www.ansible.com

as possible. The script can provision instances easily, provided with the correct authentication credentials for each cloud vendors. After provisioning, the script will configure the instances until it is ready to be used for running HemeLB in the cloud.

Modularity is one of the concerns when the deployment script was developed. The deployment script should be able to be extended easily. That's why some common functionalities are gathered into its own module that can be called from specific script. These common functionalities are mostly the software installation and configuration part that have no difference between cloud vendors. However, for each specific cloud vendors, the deployment script have different entry path. This deals mainly with the platform-specific way to provision and configures the server instances from the cloud vendors. After this platform-specific deployment script is done, it will then call the common module to configure the instances as required. With this in mind, the deployment script has been developed to be able to be run for three cloud vendors. They are Digital Ocean[4], Amazon Web Service[5], and Google Cloud Platform[6].

The deployment script described in this section is available online at the HemeWeb repository under the deployment folder[7].

3.3 HemeWeb web application

In this section, I will discuss the bulk of the work of this project which is developing the web application component. The web application component will be the interface for users to interface with HemeLB simulation workflow and is an essential part of this project. It is developed using Python 2.7 and Django web framework[8]. I chose Django web framework due to my previous experience with the web framework and also the existing codebase have tools written in python. Using Django web framework is to make sure that the codebase in HemeLB software package is done mostly consistent with Python. Additionally, using Django also allow me to focus on the development instead of learning the framework due to my past experience with it.

All the source code for HemeWeb web application is available online at my GitHub's public repository[9].

[4] `https://www.digitalocean.com`
[5] `http://aws.amazon.com`
[6] `https://cloud.google.com`
[7] `https://github.com/SeiryuZ/HemeWeb/tree/master/deployment`
[8] `https://www.djangoproject.com`
[9] `https://github.com/SeiryuZ/HemeWeb/tree/master/src`

3.3.1 Architecture

3.3.1.1 Web application components

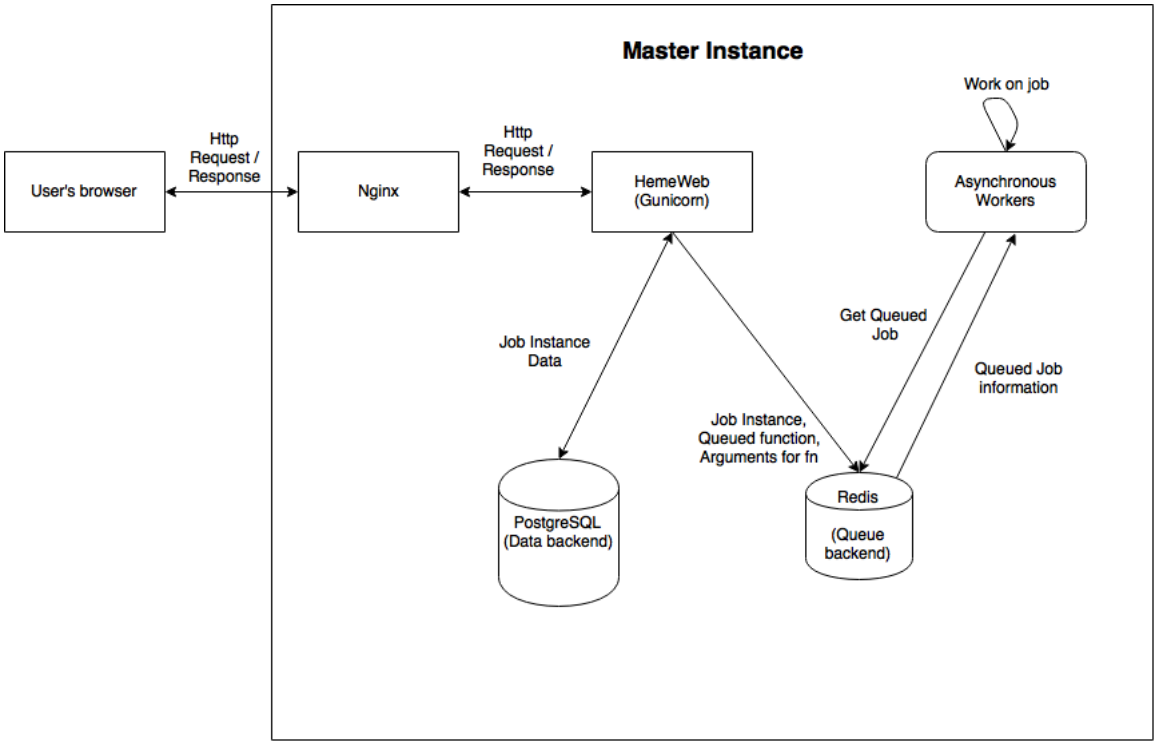

Figure 3.1: HemeWeb architecture

Figure 3.1 above illustrate how HemeWeb application process interacts with the other components inside the master instance. It also illustrates how user's HTTP request start a chain of events inside the master master instance that will eventually return a response to the user's browser.

To start, user's browser will send an HTTP request to the master instance. This request will be captured by Nginx process that will act as a reverse proxy. Nginx[10] will proxy the HTTP request towards the correct web application server process or serve static files depending on the requested URL. If the request is routed to the web application, HemeWeb web application which is handled by green unicorn[11] HTTP server will accept the request. This library will run the HemeWeb python code to process the HTTP request by the user. Depending on the type and path of the request, the web application will serve a static HTML as a response, or handle job-related logic that

[10] `https://www.nginx.com`
[11] `http://gunicorn.org`

might interact with another part of the system. One of the components the web application might interact with is the PostgreSQL database[12]. The database will persist job information locally on the instance to provide persistent information between HTTP request. However, it will be wiped out when the master instance is terminated and is not shared between HemeWeb instances.

Another part of the master instance the HemeWeb application can interact is with the queuing system. HemeWeb can submit a job into the queue which uses Redis datastore[13] as the queue backend. HemeWeb uses third party library called Django-rq that handles asynchronous background tasks handling using Redis backend. It uses the pub / sub mechanism of Redis to create a lightweight background job workers. HemeWeb process will store the function to be executed, the job instance and parameters to used by the function into the Redis backend. A background worker will look at the queue at an interval and work on a job if there's any in the queue. The worker will execute the function and update the instance with relevant job execution result. Finally, the worker will go back to being idle waiting for next job to be executed.

[12] `https://www.postgresql.org`
[13] `http://redis.io`

3.3.1.2 Docker components

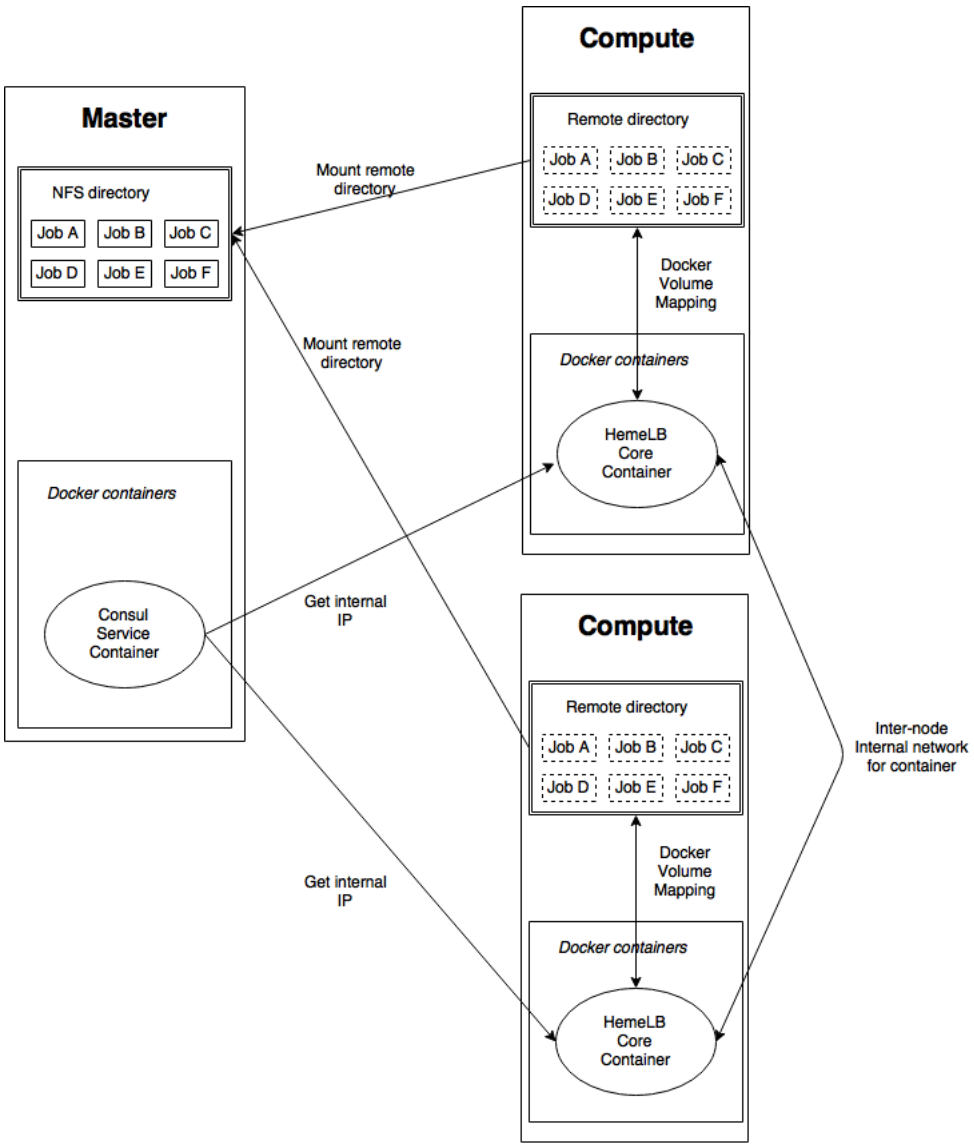

Figure 3.2: HemeWeb docker component

In this section, I will discuss the interaction between docker components and the hosts. As illustrated on figure 3.2, each host will run docker service to run specific docker container depending on their purpose.

In the master instance, a consul container is constantly running to provide inter-container communication mechanism. Consul[14] is used by the docker containers to coordinate the internal networking communication between them. In a host, multiple

[14] https://www.consul.io

containers can be started without conflicting IP because they can communicate via the host. However, in HemeWeb the communication between containers will go beyond single hosts. Containers will need to communicate with other containers living on other hosts. Consul service is needed to coordinate this communication.

On the other hand, on the compute nodes, only HemeLB core container is run. The compute nodes will be started by a job submission to HemeWeb. After the nodes are started, HemeWeb process will instruct the compute nodes to pull the specified HemeLB core container version from docker hub. If the specified version is locally cached on the compute nodes, no network activity will be made. HemeLB core container is then started to accept simulation command.

To start the simulation, the HemeLB core containers requires the job directories to be exported via Networked File System interface. HemeWeb process in the master node will prepare the job directories with the correct folder and files locally in the master node. Every job submission to HemeWeb will create a specific folder for storing submitted job-related files. The exported job directories will then be mounted by the compute nodes provisioned for the simulation. Finally, HemeLB simulation can use the mounted job directories to read input from and write output to.

3.3.1.3 Job instance structure

As mentioned above, HemeWeb uses Django web framework to handle web functionalities. Django framework follows the object-oriented principle where everything is modeled as an object. Job information that is handled by HemeWeb is also modeled as an object that is derived from object class of Django. Literally named **Job**, is a class that represents simulation information. Each instance of this class will contains information specific to a simulation instance. **Job** class inherits from *django.db.models.Model* class that is included within Django framework. HemeWeb **Job** class extends this basic class and add functionalities specifically related to job information.

HemeWeb's Job class has the following attributes:

- id: This is a unique UUID field that represents the Job ID. UUID field was chosen because it is appropriate for the possibility of sharing the job simulation files between different deployment of HemeWeb. UUID can prevent clashes of job ID between these instances.

- input_file, stl_file, profile_file, output_file, configuration_file: These attributes keep track of the files that are used by the job. It is stored as the path to the

file in the local filesystem, but with Django functionalities, HemeWeb can work with the file as an object.

- container_image: This attribute determines which container of HemeLB core will be used in the simulation. Currently, the choice of the field's value is set manually in the codebase.
- instance_type: This determines which compute node type will be started for the simulation.
- instance_count: This attribute determines how many compute node will be started for the simulation. Currently, this is also set manually in the source code.
- status: The attribute to determine Job's status, whether it is *queued, added, done, failed, etc*
- created: Attribute to keep track when the job is created
- updated: Attribute to keep track when the job is updated

3.3.1.4 Job directory structure

Each simulation done with HemeWeb have its own job directory. These directories are located in the master instance and are shared with the compute node. To provide a clearer picture how the application package and work with the job's files, I will discuss how HemeWeb structure each job's files.

```
1 <UPLOAD_FOLDER_DIR>/<JOB_ID>
2 <UPLOAD_FOLDER_DIR>/<JOB_ID>/inputs/*
3 <UPLOAD_FOLDER_DIR>/<JOB_ID>/logs/*
4 <UPLOAD_FOLDER_DIR>/<JOB_ID>/outputs/*
5 <UPLOAD_FOLDER_DIR>/<JOB_ID>/metadata
```

Listing above illustrates the structure of a job instance's directory structure. The first component of all job instance's directory is the UPLOAD_FOLDER_DIR. This is the path to a directory in which HemeWeb will upload all files. To change this parameter, one should change the path in the HemeWeb settings file and restart the web application so the changes took place.

The next part of the directory structure is the job folder named with its ID. The ID will be generated by HemeWeb using Universally Unique Identifier (UUID) scheme.

More specifically, UUID Version 4 that depends on the random number. Using UUID is important for HemeWeb because it is accurately approximated to have a really low chance of producing duplicate ID. Hence, HemeWeb does not have to take care of the possibility of jobs having duplicate ID.

Next, we have the inputs folder inside the job folder. This is where all the inputs and configurations are stored by the web application. There's also logs folder, where the job stdout, stderr, and HemeLB logs are stored. The web application will read from this folder and make it available on the web interface. Outputs folder will be used by the HemeLB simulation to write output files in this folder. One final file is the metadata file. This file is used by the web application to store the state of the job. The job is pickled into this metadata file so when it is downloaded, the web application can unpickle the state of the job instance and it is preserved, ready to be used for another simulation.

A Job instance is preserved with this job directory structure. This allows job information to be preserved in an external storage that can be used for backup. HemeWeb currently supports uploading job directory to amazon simple storage service (S3). However, backup is not the only purpose for this functionalities. By making job directory files available to the public, independent peers can download these files and run the simulation with correct files and configurations. It allows peers to replicate a simulation exactly or with modifications.

3.3.2 Simulation workflow

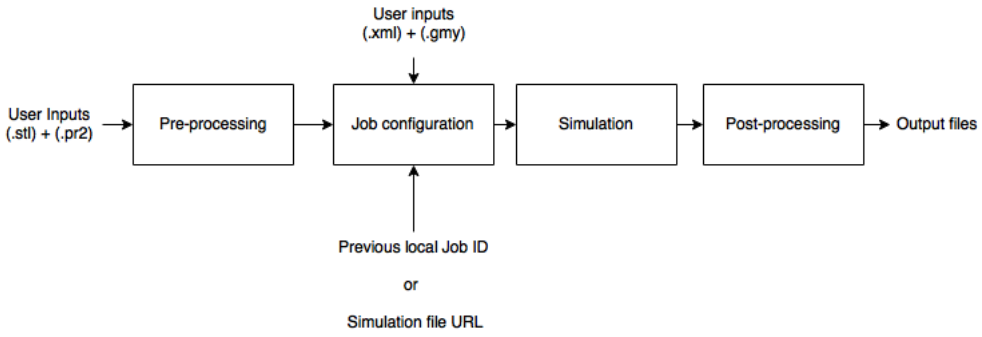

Figure 3.3: HemeWeb flow

Figure 3.3 illustrates how the HemeWeb web application works. HemeWeb currently consists of 4 core activities that will be discussed in details in the following

section.

3.3.2.1 Pre-processing

In this step, HemeWeb handles pre-processing of inputs that are needed so that HemeLB simulation can parse the files. The user provides a geometry file (.stl) and a profile file (.pr2) to the web application for processing. HemeWeb will then create a job instance with these two files, save them locally on master instances and queue the pre-processing job.

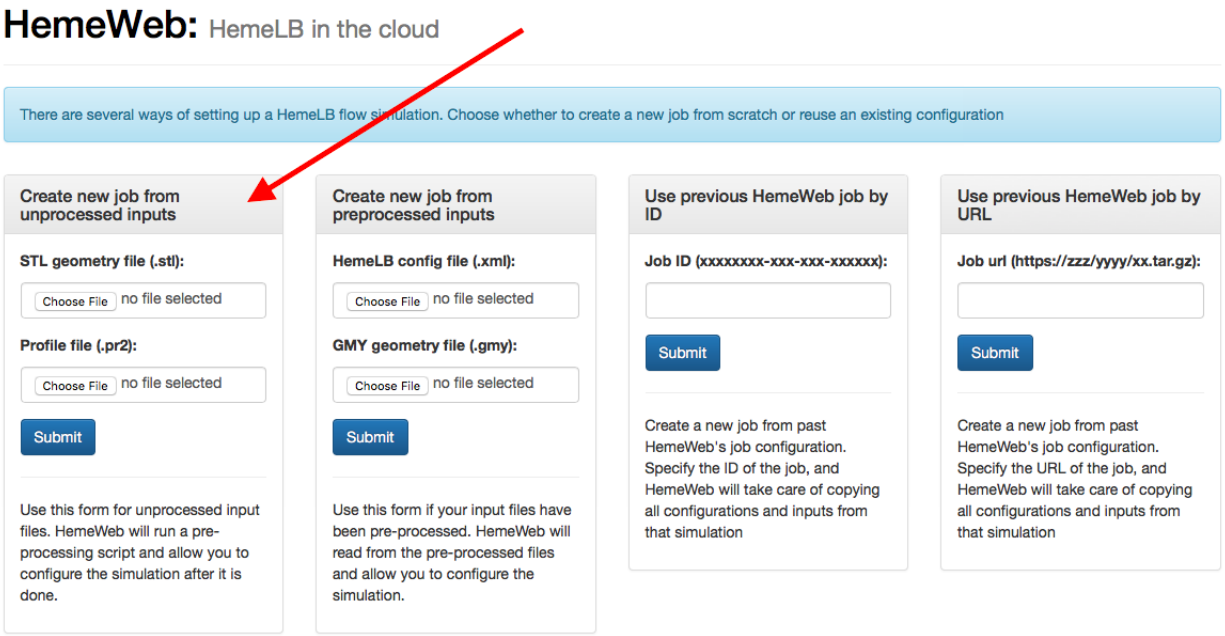

Figure 3.4: HemeWeb pre-processing form

Figure 3.4 shows HemeWeb user interface to upload the unprocessed files. After submission of both files and job being queued, the asynchronous worker on the master instance will work on the job whenever they are free. It will run the pre-processing python script to generate the geometry files and HemeLB configuration file. These files will then be saved to the master instance, and HemeWeb will track these files by recording the path to these files on the job instance. Now the job instance is ready for the next step. of the workflow.

3.3.2.2 Job configuration

In this step, HemeWeb application will take a job instance with correctly set geometry file (.gmy) and HemeLB configuration (.xml). However, there are multiple ways that HemeWeb can get this correctly set job instance. As illustrated on Figure 3.3 , there are 4 possible entry points for this step. The web interface for these 4 entry points are also shown on Figure 3.4 They are:

- **From the post-processing step.** These files are generated from the previous pre-processing step. The job instance is directly used in this step
- **User's provided geometry and configuration file.** User have pre-processed their own file locally, or have their own geometry and configuration files available. HemeWeb will create a new job instance, save both files and keep them tracked with the job instance.
- **User's provided previous job ID.** There are two possible case when user specify previous job ID. First, the previous job is available locally on the HemeWeb instance. Second, the previous job is cached on the persistent storage on the cloud vendor and are not available locally. HemeWeb will download the previous file from the persistent storage if it is not available locally. It will then create a new job instance that copy the previous job's geometry file and configuration file to be used for further configuration.
- **User's provided simulation file URL.** The last alternative is for user to provide the simulation file URL. Simulation files are uploaded to a persistent storage at the end of the workflow. These files, if made public, can be used by other instance of HemeWeb to download the simulation files and use it as a basis to create a new job instance. The way the system work is the same as using previous job ID, but its source is not its own persistent storage, but other people's simulation files.

After the job instance is created from one of the four way possible discussed above, HemeWeb will then ask users for the job configuration.

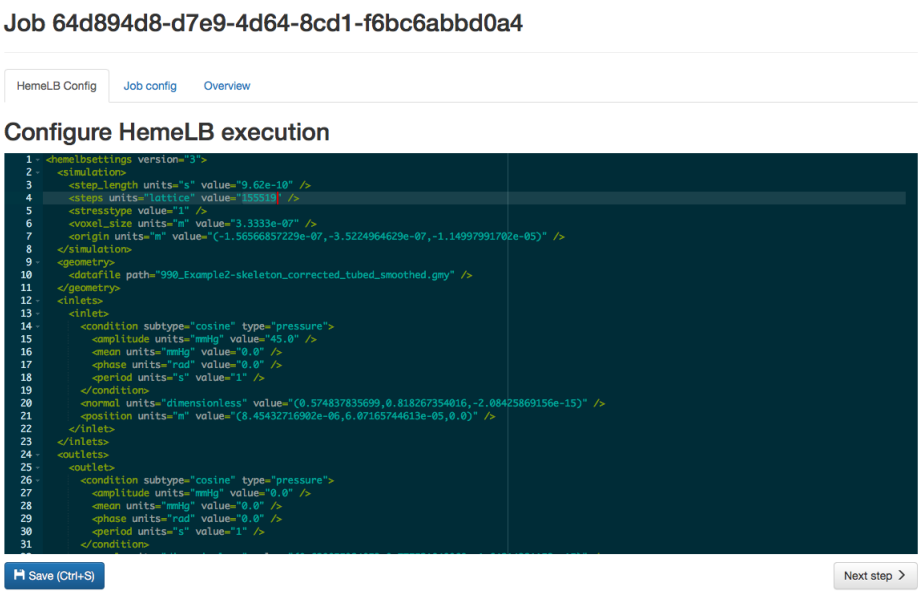

Figure 3.5: HemeWeb HemeLB configuration form

Figure 3.5 shows the interface where users are asked to configure the HemeLB simulation parameter. In this page, an online XML editor will be provided for the user to directly edit the .xml file that is provided by the users or from the pre-processing step. Users can directly edit values that affect simulation execution like inlet pressure, outlet pressure, blood viscosity, and etc. After configuring the simulation parameter it will then be redirected to job configuration page.

Figure 3.6: HemeWeb Job configuration form

Figure 3.6 shows the job configuration page. This page asks users about the parameter in which the simulation will be run with. These parameters are instance count, instance type, and HemeLB core container version. Instance count will determine how

many compute node will be provisioned for this simulation by HemeWeb. Instance type will determine what type of compute node will be started, and the HemeLB core container version will determine what version of the container the compute node will use to run HemeLB simulation. After all of these parameters are set, users will be asked to confirm the job execution in the overview page. In the page, the user can then finally queue the job into the queue system.

3.3.2.3 HemeLB simulation

Once job instance is queued into the simulation queue, a free asynchronous worker will pop the queue and run the job. The worker will start up the configured amount and type of server instance from the cloud provider. These instances will then be further reconfigured by an ansible script so that it points to the correct master instance address. Next, input files are shared via Networked File System(NFS), the compute units will mount the input folders to their instance.

The correct HemeLB core container version will be pulled from docker hub in the next step. This step will skip the download if the container asked are already cached in the image for compute units which are prepared on the deployment part. After all of these are done, then the simulation can finally begin. Master node will issue an MPI command to be run by the leader of compute nodes. The leader of compute node then will run this MPI command in the docker container. This command will be run on multiple compute node if it is configured as such in the previous steps.

The HemeLB simulation will run until outputs are produced. The output will be written back to the correct output folder in the shared folder. This means that the master instance will have access to the outputs file and can do further processing. This step ends with the termination of the instances.

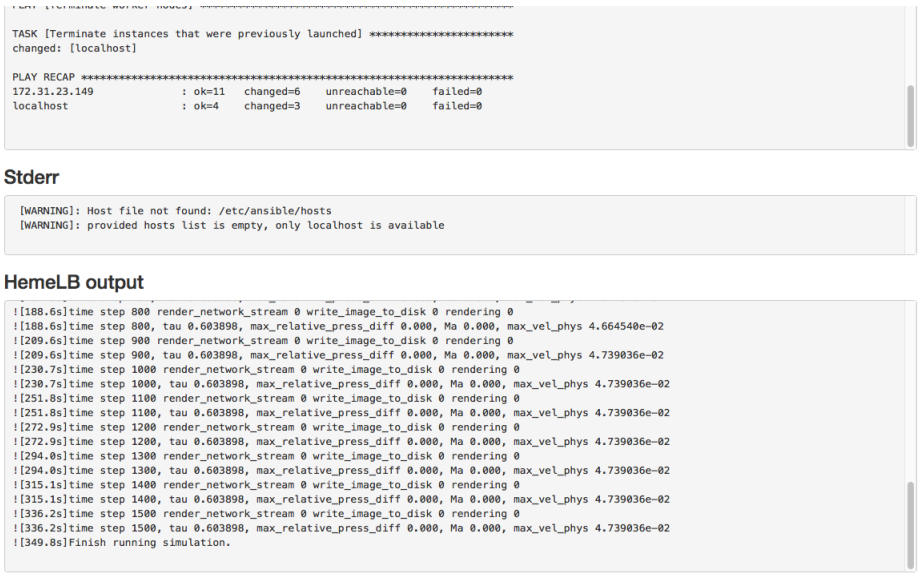

Figure 3.7: HemeWeb Job logs

Figure 3.7 shows how the Job execution will produce logs that can be viewed on HemeWeb web application. During simulation, logs are written to the respective job folder and are served to the browser by the HemeWeb app. Showing the logs allows users to easily view the progress of the job or even debug failed job.

3.3.2.4 Post-processing

After HemeLB simulation is finished, HemeWeb web app will do some post-processing steps to make sure the output files can be viewed easily. The outputs from HemeLB simulation are structured in such a way that makes it efficient to write in parallel. However, these outputs cannot be viewed by visualization system like Paraview. What HemeWeb will do is to pipe the output files to two python scripts that will format the output into a format that can be understood by ParaView.

However, the post-processing steps are not done yet. There are further steps that HemeWeb took to make sure that the simulation files, configurations, and results are preserved externally. HemeWeb will package the job directory, compressed it, and upload it into persistent storage that cloud vendors provide. As the time of writing, HemeWeb only supports amazon simple storage service. The simulation files are uploaded to this storage and made accessible to the public so other HemeWeb instance can use them. Also, with the job files persisted on persistent storage, the next HemeWeb instance deployed can take advantage of these files that it can use it as pre-

vious jobs to be used on current deployment

3.3.3 Implementation Challenge

In this section, I will try to outline and discuss the challenges in implementing this project, and if any, the solution that I choose.

3.3.3.1 Cloud vendors features and API difference

The challenge in developing the deployment script is the difference of cloud vendors' API and features. This has led to some problems when trying to create a common API to do a certain task. One notable problem is that the absence of image creation from running instance feature from one of the cloud vendors. Image creation feature is not an essential requirement of the project. However, with an image creation, the compute nodes that will be requested by the web application can be configured much quicker because all the pre-configuration that are done during the deployment phase. However, one of the cloud vendors does not have this feature. This creates a situation where there is no elegant way to create image with the deployment script and users are asked to manually created the image on the web interface

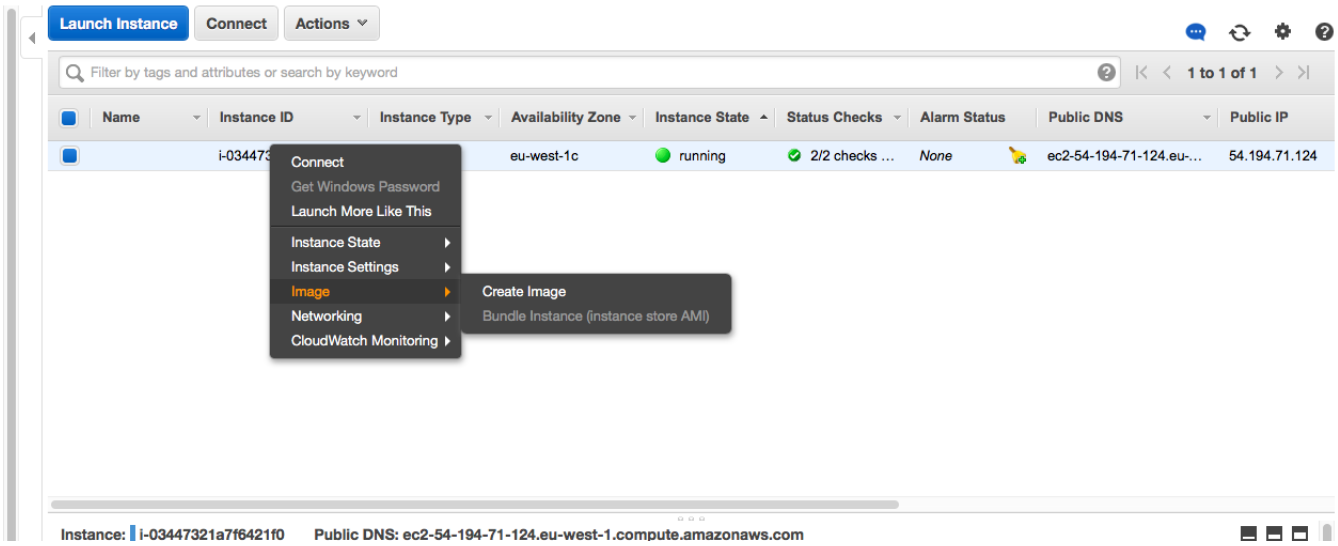

Figure 3.8: Manual image creation instead of automatic

Figure 3.8 shows how users are instructed to manually create an image from running instance. Users have to go the web interface of specific cloud vendors, right-click

on the running instance, and create the image from it. This is a simple workaround which is less complicated compared to accommodating different or missing features and API from different cloud vendors.

Another problem is the time constraint. Due to the time constraint, I cannot achieve full compatibility with all cloud vendors. The development time is mainly focused on amazon web service because it has all the features HemeWeb need. However, this means that the codebase is currently tied to one cloud vendors. Features like automatically reading past simulation files from cloud storage and uploading simulation files are tied to amazon infrastructure. It is possible to refactor these functionalities out to become more generic, however for the interest of time, I decided not to.

3.3.3.2 Security

Another challenge that I face during the development of HemeWeb is to handle the security of the application. However, security is not the main focus of this work and is apparent in the development of the application. I will still discuss the security issues so that I can give an objective assessment of the application.

The first security issues that I found is with regards to the compute node security in some cloud vendors. Digital ocean, for example, does not provide a ”real” private networking option within compute nodes. They have a ”shared” private networking options that allow other compute nodes, which are not even on your account have network connectivity to your node. Theoretically, this allow other people access to your private compute node if they have the credentials. In this case, I made sure that all the compute nodes have a sensible access policy to deter unauthorized access to the nodes. I only allow ssh with a public key and disabled password access to ssh. Also, it is also much better to choose cloud vendors that have real private networking like AWS. In which, the compute nodes are not accessible to other nodes that are not part of your own private network. This is much more secure and sensible.

Another security issue is on how compute and master node share simulation job’s files. It currently uses Network File System without any security measures towards the nodes that try to mount it via the private network. There is an opportunity to secure this communication by encrypting the job files, but it is not currently done.

3.4 Development process

The development process is divided into 5 phases. The planned phases are as follow:

1. Separate HemeLB core into its own container
2. Orchestrate the deployment of HemeLB cluster / infrastructure
3. Develop HemeWeb to accept user input
4. Extends HemeWeb to handle geometry generation workflow
5. Extends HemeWeb to handle domain definition step or Viewing of HemeLB simulation result

The development process loosely follows the agile method in which I regularly meet with the stakeholders every week to give an update and gather feedback on the project. The phases are designed in such a way to minimize the risk of having nothing at all during the end of the project phase. This is due to that HemeWeb can work on its own after finishing step 3. The HemeLB simulation can be done on its own. The rest of the steps are there to extend the functionalities of the HemeWeb to cover more functionalities.

During the first week of the development, I focused more on stripping the HemeLB core container into its own. I researched on how Docker and dockerfile work, and finding out what are the issues with the current container. After identifying the issues, which are ssh service and full of functionalities which are not essential, I stripped down the image and changed the base image so that the container could avoid the mentioned problems. I end up with smaller container size and it is available online on https://hub.docker.com.

However, one particular issue with what I have currently is that the Dockerfile is published as a part of the HemeWeb source code. It should be tied down to the HemeLB development instead of HemeWeb. Currently, the development model of HemeLB is that there is an internal private repository where the less than stable build is pushed to it, and there are public repositories where only stable builds are pushed into. The release process should include adding the dockerfile towards the core HemeLB source code repository and tagging the release correctly. HemeLB core containers should then be built automatically on docker hub with regards to additional tags being pushed to the public repository.

After the development of HemeLB core container, I then focused on how the architecture could be primed for the HemeLB simulation. It involves on configuring the servers and all supporting infrastructures on cloud vendors to be ready for HemeLB simulation. Network configuration, security configuration, docker configuration, and other should be handled automatically. I elect to choose ansible orchestration software because it is closely related to python language that I used.

In this phase, I successfully achieve the provision and deployment process that with the correct credentials and authorization, the script could provision and configure the architecture correctly so it is ready for HemeLB simulation. In addition to that, I successfully created the script so that it will be cloud vendors agnostic. I can deploy the architecture to google cloud vendors, amazon web service, and digital ocean.

However, it is to be noted that the deployment process that I achieve can only run HemeLB simulation from the command line. I have not considered the web application installation and configuration at this point of the deployment. I only considered the infrastructure being built and configured for HemeLB simulation.

Next, I started developing HemeWeb web application. I choose Django web application framework due to my experience with it. I created a basic interface, where job simulation is listed on the index page on the home interface. After that, I added a basic interface to add new job with a geometry file and HemeLB configuration file. The web application will then add those input files into a newly created job instance and configure the job instance in the job configuration step. The job will then be submitted.

Here I developed a separate ansible script that will be called when a queued job is being worked at. The ansible script is responsible for starting up compute nodes needed by the job and executing HemeLB simulation. After the simulation is done, the script is also responsible for correctly terminating the compute node.

After the basic HemeWeb web application is achieved, I extend it to handle preprocessing. I added an extra form in the adding new job form to handle a profile file and geometry file. These two files will be converted into a geometry file and HemeLb configuration file that the job configuration steps expect. In addition to adding the interface, I also added a new function on the job instance that will run this preprocessing step on the background.

Lastly, due to the limited time, I can only manage to run a small post-processing step on HemeWeb. What I did was adding post-processing step that converts the Extracted results from HemeLB output into a format that can be viewed by third party software, ParaView. The results are piped into two scripts that will output a .vtu that is

compatible with ParaView. This is done during the background activity after simulation result is outputted. However, currently, this is done on the master node.

In addition to that, I also managed to add persistence capability to HemeWeb application. What I did was to package the job simulation folder into a compressed archive and upload it to persistent storage service of cloud vendors. These archives can be queried by another instance of HemeWeb to get previous job IDs available for the particular cloud vendor account used to deploy HemeWeb. Also, the job simulation file URL is also showed on the web interface. Making it easy for the users to share the simulation file with their peers.

Chapter 4

Evaluation

In this chapter, I will discuss on the system's evaluation. Success will be measured by answering these questions:

- Can users run a simulation using our system ?
- Can users reproduce past simulations using our system ?
- Are users satisfied in using our system ?
- Are users more likely to run a simulation using our system compared to the command line ?
- How does it perform compared to the existing infrastructure ?

In answering questions above, I conduct two sets of evaluation. First, an online questionnaire was conducted to measure user's experience and system's usability. And secondly, a performance analysis comparing the performance of HemeLB between dedicated hardware against cloud vendors. The questionnaire will answer most of the questions related to user experience and usability above, while the performance benchmark will be the basis for performance evaluation and justification in using HemeWeb.

4.1 Questionnaire

In measuring user's experience, I created a questionnaire for HemeWeb using google form. The questionnaire was live for a week, from 3rd of August 2016 until 10th of August 2016.

At the start of the questionnaire, respondents are asked about their background information. This information will be used to provide demographic insight on the respondents and how it affect the type of answers the respondents will most likely give. Respondents will be asked to fill out their age, gender, job, discipline, and level of familiarity with various software tools. From their responses, I can determine whether the sample population is representative of the target audience.

The questionnaire is set to make tester run specific scenarios that HemeWeb is developed for. These scenarios are running a simulation using the web interface and reproducing past simulation with it. Testers running the scenario are given an option to skip the scenario and go straight to the questions below it to measure their experience if they find it too difficult.

The first scenario asks testers to run a simulation with given inputs. The questionnaires list two input files, a geometry file and a HemeLB configuration file that testers will need to download to their computers. After downloading the input files, testers are asked to open their browser and go to a specific URL where HemeWeb was deployed for the evaluation purposes. In that URL, testers will then add a new simulation job with the downloaded files, configure the job, and submit the job to the queue. The scenario ends when the job is submitted.

The second scenario asks testers to reproduce past simulation with a given URL that contains past simulation files. It asks testers to create a new job from the given URL instead of using files that are downloaded in the past scenario. After creating a new job, testers then will change some configuration and parameters from the past scenario and submit the job. The scenario also ends when the job is submitted.

Following each scenario are questions to measure whether users skip the scenario and an after-scenario questionnaires that users have to answer. The after-scenario questionnaire is based on the usability measurement at IBM developed by James R Lewis [28]. It measures users' usability satisfaction with the system with regards to given scenario. The questionnaire give 3 statements which testers should agree or disagree, they are:

- Overall, I am satisfied with the ease of completing the tasks in this scenario
- Overall, I am satisfied with the amount of time it took to complete the tasks in this scenario
- Overall, I am satisfied with the support information (online-line help, messages, documentation) when completing the tasks

In addition to above questions, there are some questions about the tester's willingness to do exactly the same tasks as the scenario, but with the command line. This question will measure user's willingness in using the command line interface compared to the web interface.

After running both scenarios, testers are then redirected to the final questionnaire. The Post Study System Usability Questionnaire which is based on the same work by James R Lewis [28]. In this questionnaire, testers are given 19 statements where they should agree or disagree, they are:

- Overall, I am satisfied with how easy it is to use this system
- It was simple to use this system
- I can effectively complete my work using this system
- I am able to complete my work quickly using this system
- I am able to efficiently complete my work using this system
- I feel comfortable using this system
- It was easy to learn to use this system
- I believe I became productive quickly using this system
- The system gives error messages that clearly tell me how to fix problems
- Whenever I make a mistake using the system, I recover easily and quickly
- The information (such as online help, on-screen messages, and other documentation) provided with this system is clear
- It is easy to find the information I needed
- The information provided for the system is easy to understand
- The information is effective in helping me complete the tasks and scenarios
- The organization of information on the system screens is clear
- The interface of this system is pleasant
- I like using the interface of this system

- This system has all the functions and capabilities I expect it to have
- Overall, I am satisfied with this system

In addition to the above questions, testers also will be asked to list the most negative aspects and positive aspects of the system if they have any. These questions will measure users satisfaction with the overall system, whether it is useful, whether the information given by the system is any good, and the interface quality.

4.2 Performance benchmarks

The second part of the evaluation is the performance benchmark. HemeWeb is running HemeLB simulation outside its original scope of being used on a highly parallel computing resources like a supercomputer. This could have an interesting impact on the performance of the system because cloud vendors, being much more easily accessible and provisioned, have an underlying infrastructure difference with the stand-alone infrastructure. This performance benchmarks will then measure whether the impact on the performance justify the claimed usability benefit that we measure on the first half of the evaluation.

The performance benchmark will be done by the internal tooling that is baked inside HemeLB. Every HemeLB simulation will produce a report file that measures the performance of said simulation. On this evaluation, I will compare the performance of HemeLB simulation on four different scenarios to measure the performance impact on having HemeLB simulation on the cloud with docker containers. They are:

- ARCHER supercomputer
- Highly parallel computing resources
- Cloud computing with docker
- Cloud computing without docker

The benchmark on ARCHER supercomputer will be the gold standard of the performance. It's a gold standard because the infrastructure has a clear purpose of being used for the HPC application. It has the necessary resources and components that are tailored for it. It should be the ideal performance scenario. Next, the local computing infrastructure. This paints a picture if how HemeLB will perform on a private local

infrastructure which is built to run highly parallel jobs. This will be the measure that can be used if HemeLB is to run on private infrastructure.

The next two benchmark will be done to measure the performance of HemeLB simulation compared to the local infrastructure. Cloud computing vendors have an underlying difference with regards towards how the infrastructures are structured and connected together. This could have an inherent performance different with a standalone infrastructure. However, an analysis will be made with regards to running HemeLB simulation with another overhead, which is docker. While docker is used for easily swapping of the HemeLB version, it might affect the performance of HemeLB simulation. With this benchmark, we can answer whether the flexibility of docker container can justify the performance that we get. It is also interesting to see whether there are performance penalties that we incur when using docker.

Chapter 5

Analysis

5.1 Usability result and analysis

5.2 Performance result and analysis

Chapter 6

Conclusion

Chapter 7

Future work

To better support HemeLB simulation workflow, HemeWeb can be further improved. I suggest the following areas to be further developed in the future:

1. Handle geometry generation step of HemeLB simulation workflow

   There are some steps which are not included in this project due to time constraints. One of them is the profile generation step. In this step, the domain experts should generate a profile file by pointing out how the simulation will run. They need to point where the blood will flow into the 3D model of the vascular system, where it will flow out, the blood viscosity, and other various parameters that will affect the simulation result. This process will most likely require a graphical user interface for the domain experts to interact with.

2. Viewing simulation result on the browser.

   HemeLB simulation outputs are in a format that is viewable by a third party tool, ParaView. It will be ideal if HemeWeb can be one stop solution for HemeLB simulation that domain experts do not have to bother with all other tools to view the output of it. A ParaView integration can be done in the next step of the development so that simulation result can be directly viewed on the browser so users do not have to bother with an extra tool to configure and install.

3. HemeLB simulation security

   As outlined in the implementation challenge of the project, security was not the main focus of this project. However, if this project is to be an essential part of the future medical decision, security will need to be addressed seriously. After

all, the patient's private health information will be used for the simulation. A system using such highly private information should be better secured.

Also, in line with the simulation security. HemeWeb instance should be better protected. Currently, HemeWeb does not have protected user area. Individuals without correct credentials can just run the simulation by entering the address of the application on their browser. This is not ideal because this simulation costs money. It should be better protected by having appropriate security measures against unauthorized access.

4. HemeWeb notification

   Currently, users are asked to wait until simulation is done to get the output. The user had to keep the page open or checking back in an interval to see whether it is finished. It would be a better experience for users to be notified when a simulation is finished. The user can leave the web application and get back to it when notification is sent.

5. Better interface to configure HemeLB parameters.

   Echoing the suggestions that are collected during the evaluation of HemeWeb, there should be a better interface to configure HemeLB parameters. Currently, HemeWeb provides an online XML editor for users to update the HemeLB parameter directly. However, this process is prone to error and might not be the best interface for new users to configure HemeLB simulation. A better solution can be in the form of an automatic web form building from the XML file. With it, users can use the easier web form to select and specify the parameters for HemeLB simulation. This is currently not done because the active development cycle of HemeLB means that the format of XML is also actively developed and could create complications in the building in the form if the format is changed.

6. Cloud vendor abstraction on web application

   One challenge of the project was the difference between cloud vendors. Due to the time constraint, the developed web application is tied down to amazon web services only. It would be ideal if HemeWeb could work on any cloud vendors with minimal changes. This is going to be more of a reconciling the difference between cloud vendors and make an abstraction layer that HemeWeb will need to call whenever it needs to interact with the cloud vendors' feature.

The project did achieve cloud vendor abstraction for the deployment scripts. The infrastructure can be deployed to three different cloud vendors easily. They are google cloud platform, amazon web service, and digital ocean. However, the web application needs more work to achieve the similar feat. Infrastructure can be deployed on these infrastructures, but HemeWeb still cannot work.

Appendix A

Deployment Script

A.1 Cloud vendor specific provisioning scripts

Listing A.1: Amazon web service specific provisioning script

```
1
2 - hosts: localhost
3   connection: local
4   gather_facts: false
5   vars:
6     - region: eu-west-1
7     - master_node_count: 1
8     - worker_node_count: 0
9     - master_instance_type: m3.large
10    - worker_instance_type: c4.xlarge
11
12  tasks:
13
14    - name: Create ssh security group
15      ec2_group:
16        name: ssh
17        description: HemeWeb - Security group that allow
            SSH from public IP
18        region: "{{ region }}"
19        rules:
```

```
20          - proto: tcp
21            from_port: 22
22            to_port: 22
23            cidr_ip: 0.0.0.0/0
24
25    - name: Create web security group
26      ec2_group:
27        name: web
28        description: HemeWeb - Security group that allow
             web traffic
29        region: "{{ region }}"
30        rules:
31          - proto: tcp
32            from_port: 80
33            to_port: 80
34            cidr_ip: 0.0.0.0/0
35
36    - name: Provision master node
37      ec2:
38        region: "{{ region }}"
39        image: ami-f95ef58a
40        instance_type: "{{ master_instance_type }}"
41        key_name: hemeweb
42        groups:
43          - default
44          - ssh
45          - web
46        wait: true
47        count_tag:
48          name: master
49        exact_count: "{{ master_node_count }}"
50        instance_tags:
51          name: master
52
53    - name: Provision worker nodes
```

```
      ec2:
        region: "{{ region }}"
        image: ami-f95ef58a
        instance_type: "{{ worker_instance_type }}"
        key_name: hemeweb
        groups:
          - default
          - ssh
        wait: true
        count_tag:
          name: worker
        exact_count: "{{ worker_node_count }}"
        instance_tags:
          name: worker
```

Listing A.2: Digital ocean specific provisioning script

```

- hosts: localhost
  connection: local
  gather_facts: false
  vars:
    - image_slug: ubuntu-14-04-x64
    - droplet_size: 8gb
    - region_id: lon1
    - ssh_key_id:  1732394 # Enter your ssh key id from
      digital ocean API


  tasks:

    - name: Provision master node
      digital_ocean:
        id: 9999
        name: master
        unique_name: true
```

```
19          state: present
20          command: droplet
21          size_id: "{{ droplet_size }}"
22          region_id: "{{ region_id }}"
23          image_id: "{{ image_slug }}"
24          private_networking: yes
25          wait: false
26          ssh_key_ids: "{{ ssh_key_id }}"
27
28
29      # TODO: Loop these
30      - name: Provision worker nodes
31        digital_ocean:
32          id: 1
33          name: worker1
34          unique_name: true
35          state: present
36          command: droplet
37          region_id: "{{ region_id }}"
38          size_id: "{{ droplet_size }}"
39          image_id: "{{ image_slug }}"
40          private_networking: yes
41          wait: false
42          ssh_key_ids: "{{ ssh_key_id }}"
43
44      - name: Provision worker nodes
45        digital_ocean:
46          id: 2
47          name: worker2
48          unique_name: true
49          state: present
50          command: droplet
51          region_id: "{{ region_id }}"
52          size_id: "{{ droplet_size }}"
53          image_id: "{{ image_slug }}"
```

```
54          private_networking: yes
55          wait: false
56          ssh_key_ids: "{{ ssh_key_id }}"
```

Listing A.3: Google cloud platform specific provisioning script

```
1
2  - hosts: localhost
3    connection: local
4    gather_facts: false
5    vars:
6      - hemeweb_project_id: 127280593056
7      - image: "ubuntu-1404-trusty-v20160610"  # https://
         console.cloud.google.com/compute/images?_ga
         =1.221098584.1390118076.1465823354&project=hemeweb&
         filter=name:ubuntu*
8      - region: "europe-west1-b"
9      - master_node_count: 1
10     - worker_node_count: 2
11     # https://cloud.google.com/compute/docs/machine-types
          #standard_machine_types
12     - master_instance_type: n1-standard-1
13     - worker_instance_type: n1-standard-1
14
15   tasks:
16
17
18     - name: Provision master node
19       gce:
20         image: "{{ image }}"
21         machine_type: "{{ master_instance_type }}"
22         instance_names: master
23         project_id: "{{ hemeweb_project_id  }}"
24         state: active
25         tags: master,http
26         zone: "{{ region }}"
```

```
27
28      - name: Provision worker nodes
29        gce:
30          image: "hemeweb-worker-node"
31          machine_type: "{{ worker_instance_type }}"
32          instance_names: worker1
33          project_id: "{{ hemeweb_project_id  }}"
34          state: active
35          tags: workers
36          zone: "{{ region }}"
```

A.2 Cloud vendor specific deployment scripts

Listing A.4: Amazon web service specific deployment script

```
1  ---
2
3  - include: ../deploy.yml
4    vars:
5      master_hosts_group: "tag_name_master"
6      worker_hosts_group: "tag_name_worker"
7      preferred_network_interface: "eth0"
8      preferred_ansible_interface: "ansible_default_ipv4"
```

Listing A.5: Digital ocean specific deployment script

```
1  ---
2
3  - include: ../deploy.yml
4    vars:
5      master_hosts_group: "master"
6      worker_hosts_group: "workers"
7      preferred_network_interface: "eth1"
8      preferred_ansible_interface: "ansible_default_ipv4"
```

Listing A.6: Google cloud platform specific deployment script

```
1  ---
```

```
2
3 - include: ../deploy.yml
4   vars:
5     master_hosts_group: "tag_master"
6     worker_hosts_group: "tag_workers"
7     preferred_network_interface: "eth0"
8     preferred_ansible_interface: "ansible_default_ipv4"
```

A.3 Common deployment script

Listing A.7: deploy.yml

```
1 ---
2 # Configure all nodes
3 - hosts: all
4   become: true  # allow priviledge escalation
5   roles:
6     - common
7
8
9 # Specific to master node
10 - hosts: "{{ master_hosts_group }}"
11   become: true
12   roles:
13     - nginx
14     - redis
15     - postgresql
16     - hemeweb_master
17
18
19 # Specific to worker nodes
20 - hosts: "{{ worker_hosts_group }}"
21   become: true
22   vars:
23     master_ip: "{{ hostvars[groups[master_hosts_group
         ][0]][ preferred_ansible_interface ]['address'] }}
```

```
      "
  roles:
    - hemeweb_worker
```

A.3.1 Common role

Listing A.8: roles/common/tasks/main.yml

```
#
# Common tasks to be run on the entire architecture
# Configuration tasks
#

---

- include: ssh.yml
- include: apt.yml
- include: hosts.yml

- name: Create shared Folder
  file: path=/shared state=directory owner={{
    ansible_ssh_user }} group={{ ansible_ssh_user }}
```

A.3.1.1 SSH specific common role

Listing A.9: roles/common/tasks/ssh.yml

```
#
# Configure SSH config and keys on nodes
#

---

- name: Copy hosts private key
  copy: src=insecure_key dest=~/.ssh/id_rsa force=yes
    mode=0600
  become: false

```

```
11 - name: Copy containers private key
12   copy: src=container_key dest=~/.ssh/container_key force
       =yes mode=0600
13   become: false
14
15 - name: Copy insecure key pub to authorized key
16   become: false
17   authorized_key: user={{ ansible_ssh_user }} key={{
       lookup('file', 'insecure_key.pub') }}
18
19 - name: Configure SSH hosts config
20   become: false
21   template: src=ssh_config.j2 dest=~/.ssh/config force=
       yes
```

A.3.1.2 Package manager specific common role

Listing A.10: roles/common/tasks/apt.yml

```
1  ##########################################################
2  # Configure APT, install packages and upgrade all
     packages
3  ##########################################################
4
5  ---
6
7  - name: Configure apt repository
8    apt_repository: repo={{ item.name }} state=present
9    with_items:
10     - name: "deb␣http://us.archive.ubuntu.com/ubuntu/␣
         trusty␣universe"
11     - name: "deb␣http://us.archive.ubuntu.com/ubuntu/␣
         trusty␣multiverse"
12
13 - name: Add docker keyserver to apt
14   command: apt-key adv --keyserver hkp://p80.pool.sks-
       keyservers.net:80 --recv-keys 58118
```

```
      E89F3A912897C070ADBF76221572C52609D
15
16 - name: Add docker repository to apt
17   shell: sh -c "echo 'deb https://apt.dockerproject.org/
       repo ubuntu-$(lsb_release -sc) main' | cat > /etc/
       apt/sources.list.d/docker.list"
18
19 - name: Update apt cache
20   apt: update_cache=yes
21
22
23 ############################################################
24 # Update kernel and restart if necessary
25 ############################################################
26 - name: Update kernel
27   apt: name=linux-image-generic-lts-vivid
28   register: kernel_install
29
30 - name: Restart all nodes
31   shell: sleep 2 && shutdown -r now "Ansible updates
       triggered"
32   async: 1
33   poll: 0
34   ignore_errors: true
35   when: kernel_install.changed
36
37 - name: Waiting for all nodes to come back
38   local_action: wait_for host={{ inventory_hostname }}
       port=22 state=started delay=5 timeout=300
39   become: false
40   when: kernel_install.changed
41
42
43 ############################################################
44 # Install softwares
```

```
45 ############################################################
46 - name: Install needed software
47   apt: name={{ item.name }} state=latest
48   with_items:
49     - name: python
50     - name: openmpi-bin
51     - name: docker-engine
52     - name: libffi-dev
53
54 - name: Upgrade packages
55   apt: upgrade=yes
```

A.3.1.3 Host specific common role

Listing A.11: roles/common/tasks/hosts.yml

```
1 ############################################################
2 # Update /etc/hosts file
3 # We need separate task for this, because this task
4 # can be run independently of configuration tasks
5 ############################################################
6
7 ---
8 # From https://gist.github.com/rothgar/8793800
9 - name: Update /etc/hosts file
10   template: src=hosts.j2 dest=/etc/hosts
```

Listing A.12: roles/common/templates/hosts.j2

```
1 # {{ ansible_managed }}
2 127.0.0.1    localhost localhost.localdomain localhost4
    localhost4.localdomain4
3 ::1          localhost localhost.localdomain localhost6
    localhost6.localdomain6
4
5
6 {{ hostvars[groups[ master_hosts_group ][0]][
    preferred_ansible_interface ]['address'] }}  master
```

```
7
8
9  {% if worker_hosts_group in groups %}
10   {% for host in groups[ worker_hosts_group ] %}
11     {{ hostvars[host][ preferred_ansible_interface ]['
         address'] }} hemelb-node-{{ loop.index }}
12   {% endfor %}
13 {% endif %}
```

A.3.2 Redis role

Listing A.13: roles/redis/tasks/main.yml

```
1  #
2  # Configuration specific for Redis
3  #
4
5  ---
6
7  - name: Add redis ppa
8    apt_repository: repo='ppa:chris-lea/redis-server'
9
10 - name: Update apt cache
11   apt: update_cache=yes
12
13 - name: Install redis-server
14   apt: name=redis-server state=latest
15   register: install_result
16
17 - name: Restart redis
18   service: name=redis-server state=restarted
19   when: install_result.changed
```

A.3.3 Nginx role

Listing A.14: roles/nginx/tasks/main.yml

```
1  #
```

```
2  # Configuration specific for Nginx
3  #
4
5  ---
6
7  - name: Add nginx ppa
8    apt_repository: repo='ppa:nginx/stable'
9
10 - name: Update apt cache
11   apt: update_cache=yes
12
13 - name: Install nginx
14   apt: name=nginx state=latest
15   register: install_result
16
17 - name: Configure nginx
18   template: src=nginx.j2 dest=/etc/nginx/nginx.conf force
       =True
19   register: nginx_configuration
20
21 - name: Configure reverse proxy for HemeWeb
22   template: src=default.j2 dest=/etc/nginx/sites-
       available/default force=True
23   register: sites_available
24
25 - name: Restart nginx
26   service: name=nginx state=restarted
27   when: install_result.changed or nginx_configuration.
       changed or sites_available.changed
```

Listing A.15: roles/nginx/templates/default.j2

```
1  server {
2          listen 80;
3          server_name  default_server ;
4
```

```
5         root /var/www/hemeweb/src;
6
7         location / {
8                 try_files $uri @proxy_to_app;
9         }
10
11   location /shared {
12     root /;
13                try_files $uri @proxy_to_app;
14   }
15
16        location @proxy_to_app {
17                proxy_set_header X-Forwarded-For
                    $proxy_add_x_forwarded_for;
18                proxy_set_header Host $http_host;
19                proxy_redirect off;
20                proxy_pass http://127.0.0.1:8000;
21        }
22 }
```

Listing A.16: roles/nginx/templates/nginx.j2

```
1  user www-data;
2  worker_processes auto;
3  pid /run/nginx.pid;
4
5  events {
6         worker_connections 768;
7         # multi_accept on;
8  }
9
10 http {
11
12        ##
13        # Basic Settings
14        ##
```

```
15
16          sendfile on;
17          tcp_nopush on;
18          tcp_nodelay on;
19          keepalive_timeout 65;
20          types_hash_max_size 2048;
21          # server_tokens off;
22
23  client_body_timeout 6000s;
24  client_max_body_size 5G;
25
26          # server_names_hash_bucket_size 64;
27          # server_name_in_redirect off;
28
29          include /etc/nginx/mime.types;
30          default_type application/octet-stream;
31
32          ##
33          # SSL Settings
34          ##
35
36          ssl_protocols TLSv1 TLSv1.1 TLSv1.2; # Dropping
              SSLv3, ref: POODLE
37          ssl_prefer_server_ciphers on;
38
39          ##
40          # Logging Settings
41          ##
42
43  log_format upstreamlog '[$time_local] $remote_addr -
      $remote_user - $server_name  to: $upstream_addr:
      $request upstream_response_time
      $upstream_response_time msec $msec request_time
      $request_time';
44
```

```
45          access_log /var/log/nginx/access.log upstreamlog;
46          error_log /var/log/nginx/error.log;
47
48          ##
49          # Gzip Settings
50          ##
51
52          gzip on;
53          gzip_disable "msie6";
54
55          # gzip_vary on;
56          # gzip_proxied any;
57          # gzip_comp_level 6;
58          # gzip_buffers 16 8k;
59          # gzip_http_version 1.1;
60          # gzip_types text/plain text/css application/json
                application/javascript text/xml application/
               xml application/xml+rss text/javascript;
61
62          ##
63          # Virtual Host Configs
64          ##
65
66          include /etc/nginx/conf.d/*.conf;
67          include /etc/nginx/sites-enabled/*;
68  }
69
70
71  #mail {
72  #       # See sample authentication script at:
73  #       # http://wiki.nginx.org/
       ImapAuthenticateWithApachePhpScript
74  #
75  #       # auth_http localhost/auth.php;
76  #       # pop3_capabilities "TOP" "USER";
```

```
77 #        # imap_capabilities "IMAP4rev1" "UIDPLUS";
78 #
79 #        server {
80 #                listen      localhost:110;
81 #                protocol    pop3;
82 #                proxy       on;
83 #        }
84 #
85 #        server {
86 #                listen      localhost:143;
87 #                protocol    imap;
88 #                proxy       on;
89 #        }
90 #}
```

A.3.4 Postgresql role

Listing A.17: roles/postgresql/tasks/main.yml

```
1  #
2  # Configuration specific for PostgreSQL
3  #
4
5  ---
6
7  - name: Install postgresql
8    apt: name={{ item.name }} state=latest
9    with_items:
10     - name: postgresql
11     - name: postgresql-contrib
12     - name: postgresql-server-dev-all
13   register: install_result
14
15 - name: Allow postgresql to trust 127.0.0.1
16   template: src=pg_hba.j2 dest=/etc/postgresql/9.3/main/
       pg_hba.conf force=True
```

```
17    register: config_update
18
19  - name: Restart postgreql
20    service: name=postgresql state=restarted
21    when: install_result.changed or config_update.changed
22
23  # This is ugly because becoming an unpriviledged user '
      postgres' is problematic
24  # http://docs.ansible.com/ansible/become.html
25  - name: Create superuser
26    command: sudo -u postgres createuser --superuser {{
        ansible_ssh_user }}
27    ignore_errors: true
28
29  - name: Create database
30    become: false
31    command: createdb hemeweb
32    ignore_errors: true
```

Listing A.18: roles/postgresql/templates/pg_hba.j2

```
1  # PostgreSQL Client Authentication Configuration File
2  # ===================================================
3  #
4  # Refer to the "Client Authentication" section in the
     PostgreSQL
5  # documentation for a complete description of this file.
      A short
6  # synopsis follows.
7  #
8  # This file controls: which hosts are allowed to connect,
      how clients
9  # are authenticated, which PostgreSQL user names they can
      use, which
10 # databases they can access.  Records take one of these
     forms:
```

```
11 #
12 # local      DATABASE  USER  METHOD  [OPTIONS]
13 # host       DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
14 # hostssl    DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
15 # hostnossl  DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
16 #
17 # (The uppercase items must be replaced by actual values
   .)
18 #
19 # The first field is the connection type: "local" is a
   Unix-domain
20 # socket, "host" is either a plain or SSL-encrypted TCP/
   IP socket,
21 # "hostssl" is an SSL-encrypted TCP/IP socket, and "
   hostnossl" is a
22 # plain TCP/IP socket.
23 #
24 # DATABASE can be "all", "sameuser", "samerole", "
   replication", a
25 # database name, or a comma-separated list thereof. The "
   all"
26 # keyword does not match "replication". Access to
   replication
27 # must be enabled in a separate record (see example below
   ).
28 #
29 # USER can be "all", a user name, a group name prefixed
   with "+", or a
30 # comma-separated list thereof.  In both the DATABASE and
    USER fields
31 # you can also write a file name prefixed with "@" to
   include names
32 # from a separate file.
33 #
34 # ADDRESS specifies the set of hosts the record matches.
```

```
    It can be a
35 # host name, or it is made up of an IP address and a CIDR
    mask that is
36 # an integer (between 0 and 32 (IPv4) or 128 (IPv6)
    inclusive) that
37 # specifies the number of significant bits in the mask.
    A host name
38 # that starts with a dot (.) matches a suffix of the
    actual host name.
39 # Alternatively, you can write an IP address and netmask
    in separate
40 # columns to specify the set of hosts. Instead of a CIDR
    -address, you
41 # can write "samehost" to match any of the server's own
    IP addresses,
42 # or "samenet" to match any address in any subnet that
    the server is
43 # directly connected to.
44 #
45 # METHOD can be "trust", "reject", "md5", "password", "
    gss", "sspi",
46 # "krb5", "ident", "peer", "pam", "ldap", "radius" or "
    cert". Note that
47 # "password" sends passwords in clear text; "md5" is
    preferred since
48 # it sends encrypted passwords.
49 #
50 # OPTIONS are a set of options for the authentication in
    the format
51 # NAME=VALUE. The available options depend on the
    different
52 # authentication methods -- refer to the "Client
    Authentication"
53 # section in the documentation for a list of which
    options are
```

```
54 # available for which authentication methods.
55 #
56 # Database and user names containing spaces, commas,
     quotes and other
57 # special characters must be quoted.  Quoting one of the
     keywords
58 # "all", "sameuser", "samerole" or "replication" makes
     the name lose
59 # its special character, and just match a database or
     username with
60 # that name.
61 #
62 # This file is read on server startup and when the
     postmaster receives
63 # a SIGHUP signal.  If you edit the file on a running
     system, you have
64 # to SIGHUP the postmaster for the changes to take effect
     .  You can
65 # use "pg_ctl reload" to do that.
66
67 # Put your actual configuration here
68 # ----------------------------------
69 #
70 # If you want to allow non-local connections, you need to
      add more
71 # "host" records.  In that case you will also need to
     make PostgreSQL
72 # listen on a non-local interface via the
     listen_addresses
73 # configuration parameter, or via the -i or -h command
     line switches.
74
75
76
77
```

```
78 # DO NOT DISABLE!
79 # If you change this first entry you will need to make
      sure that the
80 # database superuser can access the database using some
      other method.
81 # Noninteractive access to all databases is required
      during automatic
82 # maintenance (custom daily cronjobs, replication, and
      similar tasks).
83 #
84 # Database administrative login by Unix domain socket
85 local   all             postgres
                                        peer
86
87 # TYPE  DATABASE        USER            ADDRESS
                    METHOD
88
89 # "local" is for Unix domain socket connections only
90 local   all             all
                                      peer
91 # IPv4 local connections:
92 host    all             all             127.0.0.1/32
              trust
93 # IPv6 local connections:
94 host    all             all             ::1/128
                    md5
95 # Allow replication connections from localhost, by a user
     with the
96 # replication privilege.
97 #local   replication     postgres
                                        peer
98 #host    replication     postgres        127.0.0.1/32
              md5
99 #host    replication     postgres        ::1/128
                    md5
```

Bibliography

[1] S. Steven, "Hemeweb: Container based high performance computing scenario in cloud infrastructure for hemelb," informatics research proposal, University of Edinburgh, April 2016.

[2] M. D. Mazzeo and P. V. Coveney, "Hemelb: A high performance parallel lattice-boltzmann code for large scale fluid flow in complex geometries," *Computer Physics Communications*, vol. 178, no. 12, pp. 894–914, 2008.

[3] M. A. Itani, U. D. Schiller, S. Schmieschek, J. Hetherington, M. O. Bernabeu, H. Chandrashekar, F. Robertson, P. V. Coveney, and D. Groen, "An automated multiscale ensemble simulation approach for vascular blood flow," *Journal of Computational Science*, vol. 9, pp. 150–155, 2015.

[4] M. O. Bernabeu, Y. Lu, J. Lammer, L. P. Aiello, P. V. Coveney, and J. K. Sun, "Characterization of parafoveal hemodynamics associated with diabetic retinopathy with adaptive optics scanning laser ophthalmoscopy and computational fluid dynamics," in *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 8070–8073, IEEE, 2015.

[5] C. A. Franco, M. L. Jones, M. O. Bernabeu, I. Geudens, T. Mathivet, A. Rosa, F. M. Lopes, A. P. Lima, A. Ragab, R. T. Collins, *et al.*, "Dynamic endothelial cell rearrangements drive developmental vessel regression," *PLoS Biol*, vol. 13, no. 4, p. e1002125, 2015.

[6] C. A. Franco, M. L. Jones, M. O. Bernabeu, A.-C. Vion, P. Barbacena, J. Fan, T. Mathivet, C. G. Fonseca, A. Ragab, T. P. Yamaguchi, *et al.*, "Non-canonical wnt signalling modulates the endothelial shear stress flow sensor in vascular remodelling," *Elife*, vol. 5, p. e07727, 2016.

[7] C. Green, "Computer simulation could become 'integral' in the diagnosis of," 2014.

[8] D. Groen, J. Hetherington, H. B. Carver, R. W. Nash, M. O. Bernabeu, and P. V. Coveney, "Analysing and modelling the performance of the hemelb lattice-boltzmann simulation environment," *Journal of Computational Science*, vol. 4, no. 5, pp. 412–422, 2013.

[9] M. Huerta, G. Downing, F. Haseltine, B. Seto, and Y. Liu, "Nih working definition of bioinformatics and computational biology," *US National Institute of Health*, 2000.

[10] A. W. Service, "Whitepaper: Intro to hpc on aws." `https://d0.awsstatic.com/whitepapers/Intro_to_HPC_on_AWS.pdf`, August 2015. (Accessed on 08/05/2016).

[11] F. Berman, G. Fox, and A. J. Hey, *Grid computing: making the global infrastructure a reality*, vol. 2. John Wiley and sons, 2003.

[12] I. Foster and C. Kesselman, *The Grid 2: Blueprint for a new computing infrastructure*. Elsevier, 2003.

[13] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *2008 Grid Computing Environments Workshop*, pp. 1–10, Ieee, 2008.

[14] fsn.co.uk, "The economy is flat so why are financials cloud vendors growing at more than 90 percent per annum?." `http://www.fsn.co.uk/channel_outsourcing/the_economy_is_flat_so_why_are_financials_cloud_vendors_growing_at_more_than_90_percent_per_annum#.UbmtsPlJPGA/`, March 2013. (Accessed on 08/06/2016).

[15] J. Barr, "Aws price reduction 42 – ec2, s3, rds, elasticache, and elastic mapreduce — aws blog." `https://aws.amazon.com/blogs/aws/aws-price-reduction-42-ec2-s3-rds-elasticache-and-elastic-mapreduce/`, March 2014. (Accessed on 08/06/2016).

[16] S. Martin, "Announcing reduced pricing on storage — blog — microsoft azure." `https://azure.microsoft.com/en-us/blog/storage-price-match/`, January 2014. (Accessed on 08/06/2016).

[17] F. Lardinois, "Google announces massive price drops for its cloud computing services and storage, introduces sustained-use discounts." https://techcrunch.com/2014/03/25/google-drops-prices-for-compute-and-app-engine-by-over-30-cloud-storage-by March 2014. (Accessed on 08/06/2016).

[18] J. Cohen, D. Moxey, C. Cantwell, P. Burovskiy, J. Darlington, and S. J. Sherwin, "Nekkloud: A software environment for high-order finite element analysis on clusters and clouds," in *2013 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 1–5, IEEE, 2013.

[19] P. Mehrotra, J. Djomehri, S. Heistand, R. Hood, H. Jin, A. Lazanoff, S. Saini, and R. Biswas, "Performance evaluation of amazon ec2 for nasa hpc applications," in *Proceedings of the 3rd workshop on Scientific Cloud Computing Date*, pp. 41–50, ACM, 2012.

[20] Q. He, S. Zhou, B. Kobler, D. Duffy, and T. McGlynn, "Case study for running hpc applications in public clouds," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pp. 395–401, ACM, 2010.

[21] J. Goecks, A. Nekrutenko, and J. Taylor, "Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences," *Genome biology*, vol. 11, no. 8, p. 1, 2010.

[22] APS, "What is science?." http://www.aps.org/policy/statements/99_6.cfm, November 1999.

[23] O. S. Collaboration *et al.*, "Estimating the reproducibility of psychological science," *Science*, vol. 349, no. 6251, p. aac4716, 2015.

[24] C. Drummond, "Replicability is not reproducibility: nor is it good science," 2009.

[25] D. L. Donoho, "An invitation to reproducible computational research," *Biostatistics*, vol. 11, no. 3, pp. 385–388, 2010.

[26] G. K. Sandve, A. Nekrutenko, J. Taylor, and E. Hovig, "Ten simple rules for reproducible computational research," *PLoS Comput Biol*, vol. 9, no. 10, p. e1003285, 2013.

[27] F. Moreews, O. Sallou, Y. Bras, *et al.*, “A curated domain centric shared docker registry linked to the galaxy toolshed,” in *Galaxy Community Conference 2015*, 2015.

[28] J. R. Lewis, “Ibm computer usability satisfaction questionnaires: psychometric evaluation and instructions for use,” *International Journal of Human-Computer Interaction*, vol. 7, no. 1, pp. 57–78, 1995.