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 are increasingly complex. Our everyday software are packed with features that makes its usage difficult. To people without familiarity with the product, it will be a barrier of entry to use it even when it helps them tremendously.

This also ties in to the complexity of a research that use these softwares. 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 or even computation are replicable easily.

1.1 Motivation

To study how blood flow in a given vessel, Mazzeo and Coveney [2008] developed a fluid dynamic simulation software named HemeLB. Currently, it is actively developed and used by researchers to help their study. For example, Itani et al. [2015] used HemeLB for automated ensemble simulation of blood flow for a range of exercise intensities, Bernabeu et al. [2015] used it for detecting difference of retinal hemodynamics with regards to diabetic retinopathy, and recently Franco et al. [2015, 2016] used it to understand branching pattern of blood vessel networks.

As I have written in the proposal for this dissertation [Steven, 2016], HemeLB works by calculating fluid flow in parallel by using lattice-Boltzmann method [Mazzeo and Coveney, 2008]. 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 simulation, coupled with complexity of configuration of the software created a high barrier of entry for scientists and doctors to use HemeLB. Furthermore, as observed previously [Steven, 2016], 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 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 improvement 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 [Green, 2014].

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 [Steven, 2016], There are steps that are in place to make sure HemeLB and its simulation result are reproducible. First, the entire code base are publicly available on Github. Second, in running 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 [Bernabeu et al., 2015; Itani et al., 2015; Franco et al., 2015, 2016] that include all these information. These 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 integral part of medical decision in the future.

1.2 Objectives

Based upon 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 much efforts. On top of that, these infrastructures are available to everyone with a costs, 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 steps 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**. I will provide background information that are 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**. 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
- **Chapter 5**. Analysis
- **Chapter 6**. Future work

Chapter 2

Background

2.1 Current HemeLB workflow

Running a blood flow simulation using HemeLB currently consists of multiple steps. To understand how the proposed project can improve the current conditions, I will elaborate on how HemeLB workflow currently work.

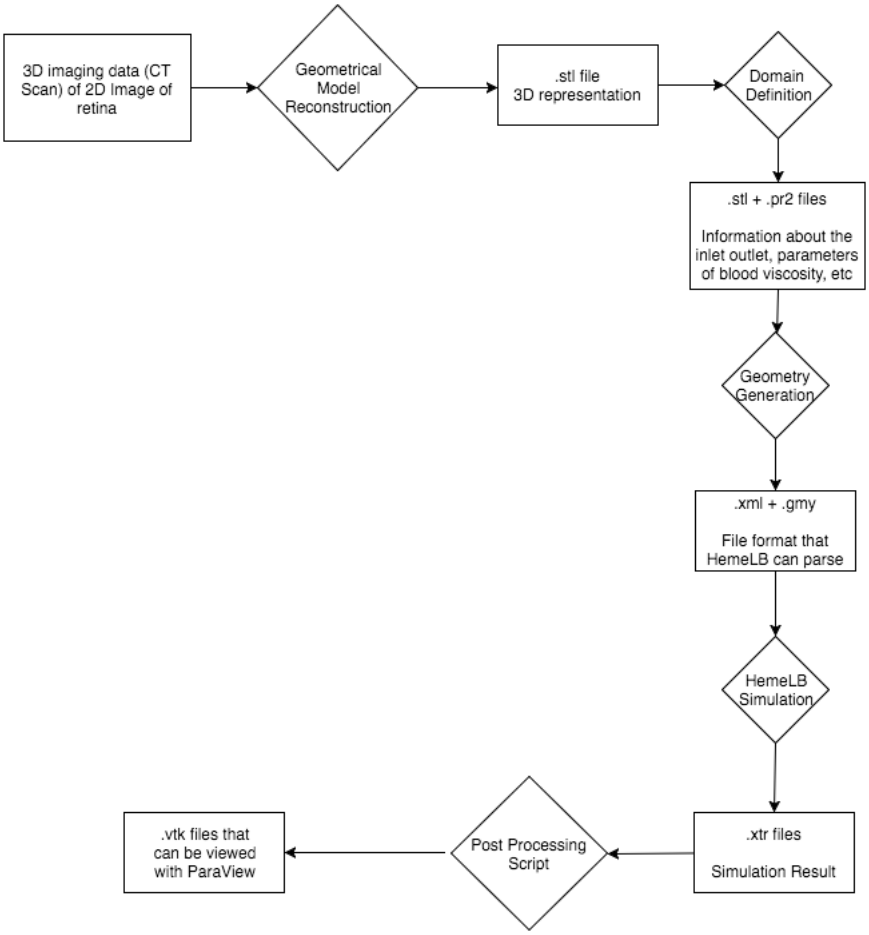

Figure 2.1: Current HemeLB workflow taken from Steven [2016]

2.2 Cloud computing

2.3 Containerization Technology

2.4 Other High Performance Computing

Chapter 3

Design and Specification

I will discuss on the design and specification of HemeWeb. What are the resources needed to implement it and the design of the architecture.

3.1 HemeWeb System Specification

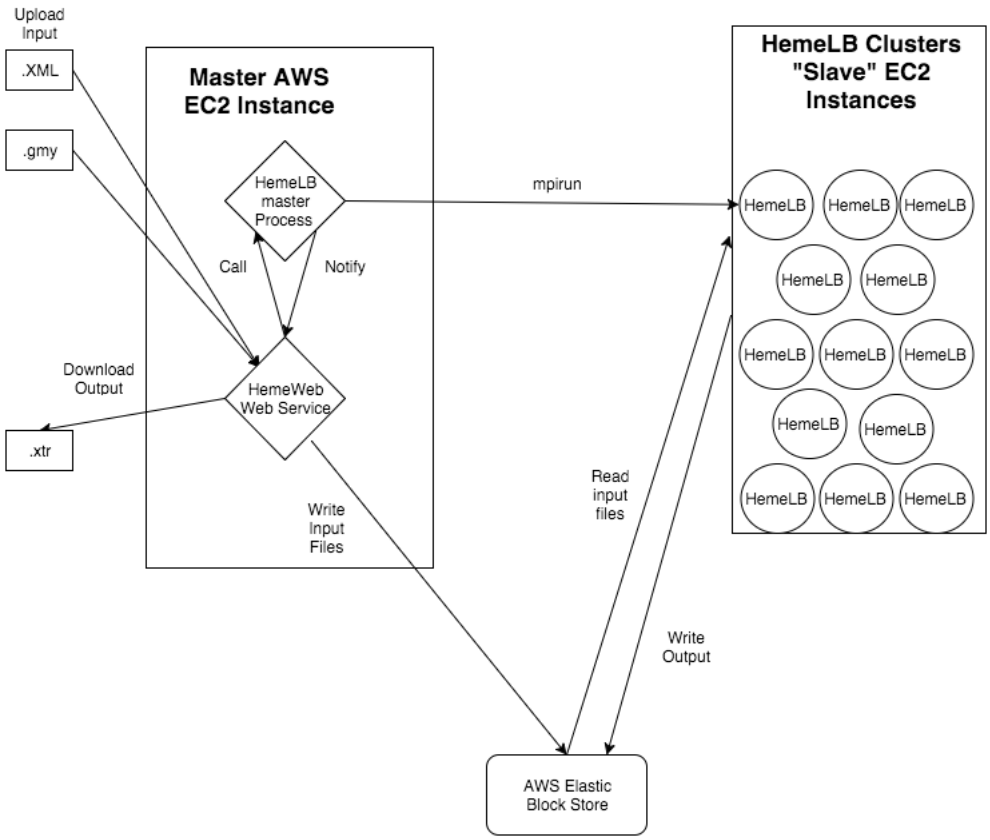

Figure 3.1: Planned HemeWeb architecture phase 1 from Steven [2016]

Figure 3.1 above was the original plan on how the system architecture will look like. The system will consist mainly of one master instance which will take care of starting up HemeLB cluster to compute the simulation. This master instance will run HemeWeb web application in which the user will interact with. Users will provide the web application with the inputs it needs to run. Which are the geometry file (.gmy) and the job configuration file (.xml).. With this input, the master instance will start up the appropriate HemeLB core cluster and run mpirun command to initiate the job execution. When execution is done, the output will be available for download to the users. In addition to that, the job files will be uploaded to amazon S3 so that it will be available for other instances of HemeWeb if it is shared, or for newly created instance with the same amazon AWS account.

3.2 HemeLB core container

Part of the designed system above is to have HemeLB cluster available for the HemeWeb master instance. In order to spin up the cluster easily, I decided to make a HemeLB core container only available in the docker hub (https://hub.docker.com). This container is based on an earlier work to ship HemeLB and all of its setup tools in a docker container so peers can run HemeLB without configuring their system a lot. However, for HemeWeb purpose, the whole suit of the setup tools are not essential for the simulation. What I planned to do is to strip it down to its bare minimum, so the container only contains the binary necessary for the simulation to run.

Chapter 4

Implementation

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

4.1 HemeLB core docker container

In this section I will explain how I developed the HemeLB core docker container. This part is the most straightforward of the project. There has been effort to make HemeLB portable by packaging it with the setup tools and a portable desktop environment using docker. It is done with the rationale to be shared with peers without having them to configure their work station too much.

In this phase, I modified the previous available image to handle few problems I found when trying to run HemeLB with it. 1) The size is big because it consists of the setup tools, not only the HemeLB binary, and 2) It cannot be run as a daemon. To solve this issue, I used a different base image than the previous. I used the ubuntu base provided by phusion. This image handle services correctly and allow the container to be ran as a daemon with SSH service running. This allows HemeLB to run with multiple containers. Also, I purged all package that are not needed for the container, resulting in a minimal sized container image.

4.2 Deployment

In addition to have a working prototype available. I also wanted to make sure that the process of deploying the web application is as painless as possible. Therefore,

I developed a deployment script for automatically configuring the architecture with minimal manual intervention. The script is made with the help of a configuration management and orchestration tools called ansible (https://www.ansible.com).

The goal of the script are as follow: 1. Provision the required master instance 2. Configure the master instances with the correct security and network settings 3. Configure and Install the HemeWeb correctly 4. Configure and install all the services needed by HemeWeb 5. Provision required compute instance 6. Configure the compute instances with the correct security and network settings 7. Configure the compute instances to run HemeLB core docker container

In the development of the script, I faced few problems in regard of orchestrating the infrastructure. Mainly with regards to cloud vendors having non-similar application programming interface to access or control the resources. This led to some part of the deployment needing manual intervention. One of this part is creating an image from the running instance so the master instance can request a new pre-configured instance instead of configuring from the start.

I developed the script as modular as I can so that it can be refactored easily. I separated the cloud vendor specific instruction into its own folder for each vendor. This part mainly deal with the provisioning, security, and network configurations. The more common part, like installing softwares and services needed, are abstracted into the common scripts that are called by the vendor specific script after provisioning instances are done. With this in mind, I had been able to run the deployment script on three different cloud vendors. They are Google Cloud Platform, Amazon Web Service, and Digital Ocean. After the deployment script is done, the architecture will be ready to run HemeLB simulation on the cloud.

This script is online as part of the codebase that is published on github. It is available on https://github.com/SeiryuZ/HemeWeb/tree/master/deployment .

4.3 HemeWeb

This section will discuss the bulk of the work that is developed for the project. The main interface for users to interface with HemeWeb, the web application. I developed this web application using django web framework for my familiarity with the framework, so I can focus my time on developing the web application.

4.3.1 Architecture

4.3.1.1 Web application components

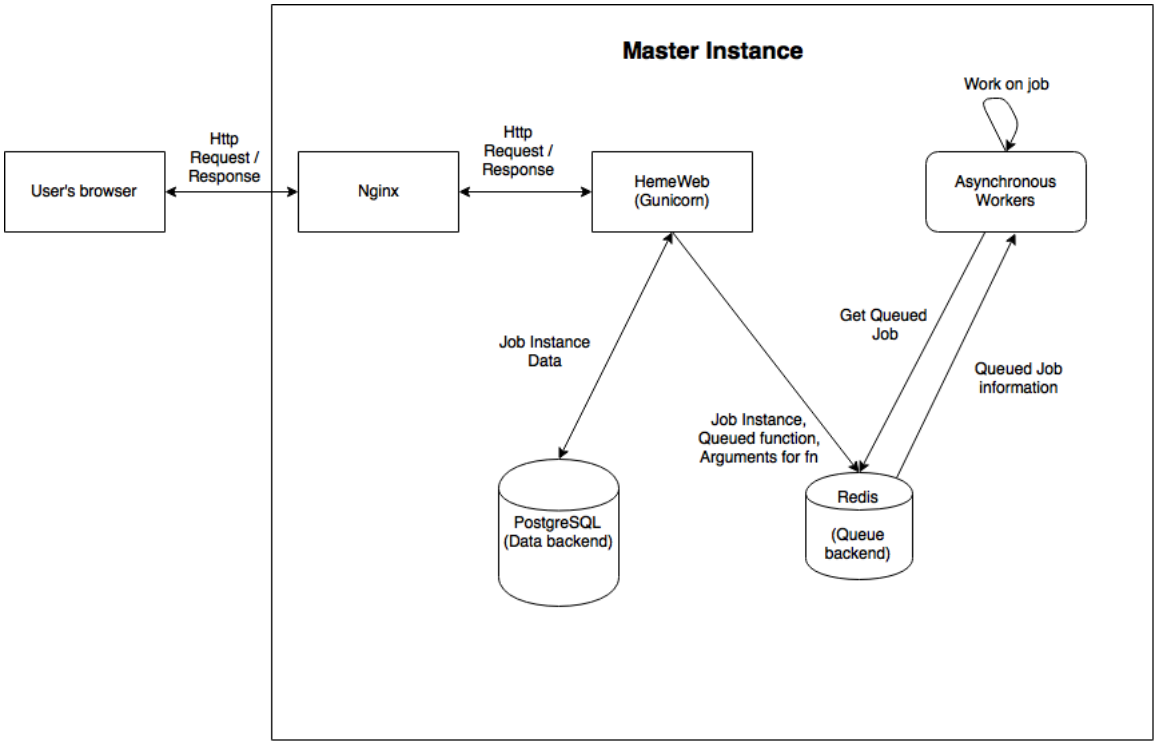

Figure 4.1: HemeWeb architecture

Figure 4.1 above illustrate how HemeWeb interacts with the other components on the master instances. HemeWeb, a django web application, lives on the master instance with various components that it interacts with.

User's browser will send a HTTP request to the master instance, in which nginx frontend will handle. It will act as a reverse proxy, and proxy those Http request towards the web application server process, which is handled by green unicorn python library. This library will run the HemeWeb python code to process the HTTP request by the user, it will server a static HTTP response, or handle job related logic that might interact with the PostgreSQL database service. The database will persist job information locally on the instance and will be wiped out when the master instance is terminated.

Another way the HemeWeb application can interact is to submit a job into the queue which use Redis datastore as the queue backend. HemeWeb use third party library called django-rq that handles asynchronous background tasks handling. It uses the pub

/ sub mechanism that Redis has to create a light weight background job workers. It will store the function to be executed, the job instance, and parameters to used by the function. A background worker that is called using cronjob at an interval will look at the queue and work on a job if there's any. The worker will execute the function and update the instance with relevant job execution result. The worker then will go back to idle until next invocation.

4.3.1.2 Docker components

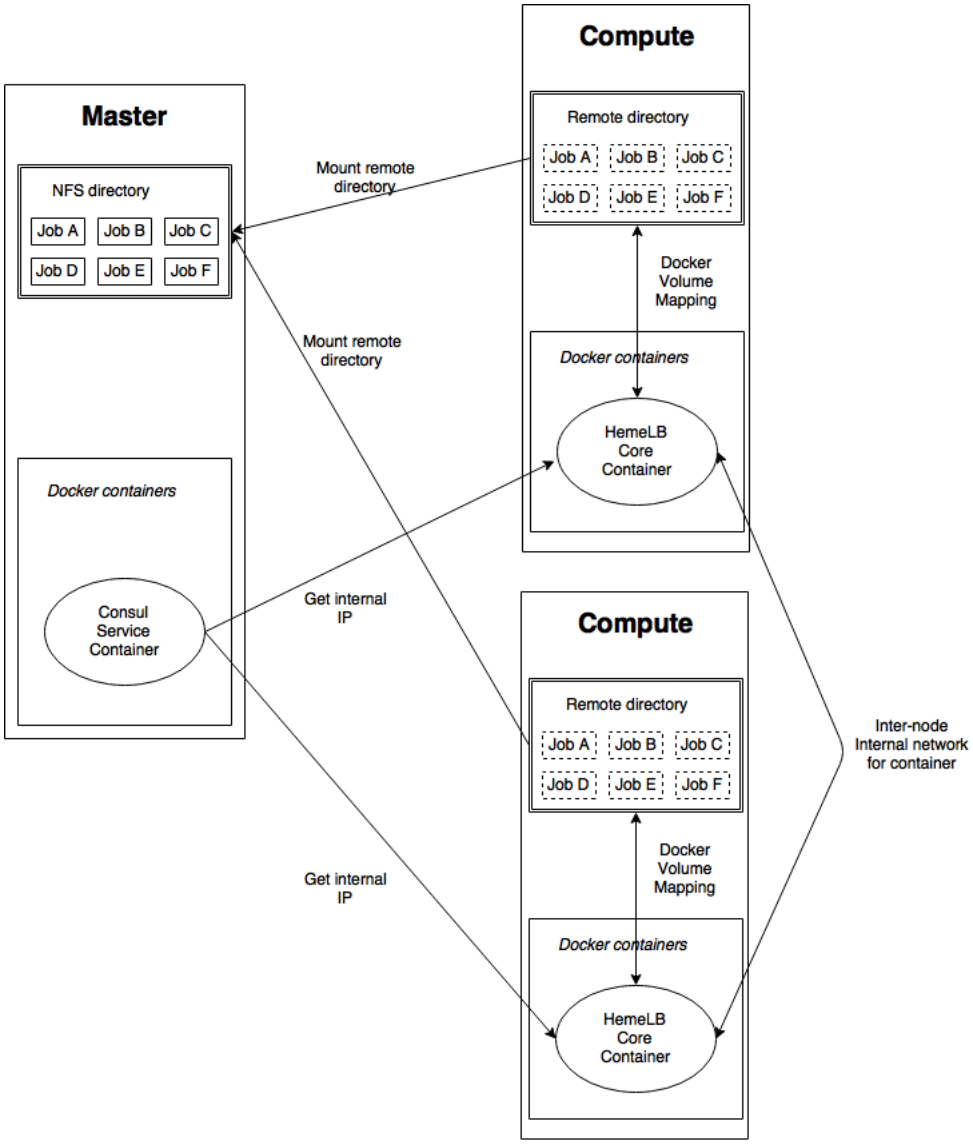

Figure 4.2: HemeWeb docker component

In this section, I will discuss how the docker components interact with its host and each other as illustrated on figure 4.2. In the master instances, job directories are required to be exported via the Networked File System service. These directories need to be exported for the purpose of sharing the folders and files with the compute node. The web application actually do not really interact with the Networked File System, it create or modify the folder, but configuration is handled during deployment process.

Also, in the master instance we have one docker containers running at all time. This container handle the consul service. This service is used by docker service to co-ordinate inter-host communication. With this service running, docker service running on other hosts will be able to coordinate with the master instance with regards to container network, give its container non-conflicting internal IP. In a nutshell, containers on different hosts can communicate as if they are on a local network. This feature is key in allowing HemeLB simulation to be run inside docker container on multiple host on the cloud platform.

On the other hand, the compute node will first mount the job directory from the master instance. This is done in order to allow HemeLB containers to have shared access on the input and output folders they are going to use. The container will gain access to the folder on the host by using the docker volume mapping features.

Docker containers on the compute node will only run HemeLB core containers. The compute node will pull the appropriate HemeLB core version from docker hub, but if specified version is locally cached on the instance, no network activity will be made. When running the container, the compute unit will coordinate with the consul service on master instance with regards to internal network configuration. Allowing multiple HemeLB core containers on different hosts to communicate with each others.

After this configuration is done, then the job execution will be done by one of the compute node. It will execute an mpi job for HemeLB with internal IP of all hosts that are started for this particular job.

4.3.1.3 Job instance structure

HemeWeb use django python library to handle the heavy weightlifting with regards to web functionalities. The way job instance is created is also dependent on django's model hierarchy. **Job** is a class that represent one instance of job simulation. It inherits from django.db.models.Model class that exist within django infrastructure. Django's model class handles the interaction with the database and with the django environment and HemeWeb's job class only need to extend this class to add its own functionality.

HemeWeb's Job class has the following attributes:

- id: This is a unique UUID field that represent 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 ID between these instances.
- input_file, stl_file, profile_file, output_file, configuration_file: These attribute keep track of the files that are used by the job. It is stored as path to the file in the local file system, but with django functionalities HemeWeb can work with the file as an object.
- container_image: This attribute determine which container of HemeLB core will be used in the simulation. Currently, this field is set manually on the code.
- 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 on the 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

4.3.1.4 Job instance directory structure

HemeWeb structure each job's files and configurations into its own folder. I will discuss how it is structure to provide clearer picture on how the application package and work with the job's files.

<UPLOAD_FOLDER_DIR>/<JOB_ID>
<UPLOAD_FOLDER_DIR>/<JOB_ID>/inputs/*
<UPLOAD_FOLDER_DIR>/<JOB_ID>/logs/*
<UPLOAD_FOLDER_DIR>/<JOB_ID>/outputs/*
<UPLOAD_FOLDER_DIR>/<JOB_ID>/metadata

HemeWeb installation can change the upload folder directory as the basis of all job's folder will be located. Job ID will be generated by the web application using UUID4 scheme, so ID's generated by different instances of HemeWeb should theoretically have really low chance of clashing. This is also why adding previous job to the HemeWeb instance is not problematic, because using auto increment ID will be a problem for multiple installation.

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 output files on 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.

4.3.2 Simulation workflow

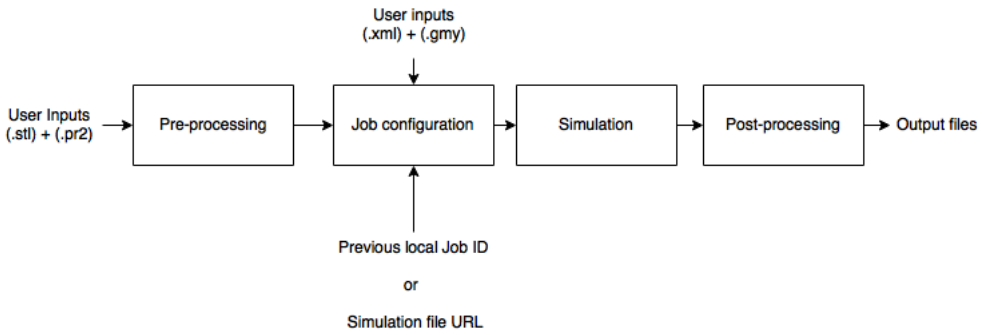

Figure 4.3: HemeWeb flow

Figure 4.3 illustrate how the HemeWeb web application works. it consists of mainly 4 core activity that will be discussed in details in the following section.

4.3.2.1 Pre-processing

HemeWeb handle pre-processing of inputs that are needed so that HemeLB simulation can parse the files. User provide a geometry file (.stl) and a profile file (.pr2) to the web application. HemeWeb will then create a job instance with these two files, save

them locally on master instances and queue the pre-processing job.

The asynchronous worker on master instance will then pick up 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 on 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.

4.3.2.2 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 4.3 , there are 4 possible entry points for this step. 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. This entails on configuring how many instances should be started, what are the type of the instance, and the hemelb-core container version. After configuring all these parameters, then the user can queue the job into the queue system.

4.3.2.3 HemeLB simulation

Once job instance are queued into the simulation queue, free asynchronous workers 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.

HemeLB core container 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 unit 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.

4.3.2.4 Post-processing

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

However, the post-processing steps are not done until converting the output. There are further steps that HemeWeb took to make sure that the simulation files, configura-

tions, and results are preserverd. 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 support amazon S3. The simulation file are uploaded to this storage and made available for public for other HemeWeb instance to use. Also, with the job files persisted on persistent storage. The next HemeWeb instance deployed can take advantage of these files that it can download them as previous job available.

4.3.3 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.

4.3.3.1 Cloud vendors features and API difference

Cloud vendors each have their own API and features. This has led to some problems when trying to create a common API to do certain task. One notable problem is that the absence of image creation from running instance task from one of the cloud vendors.

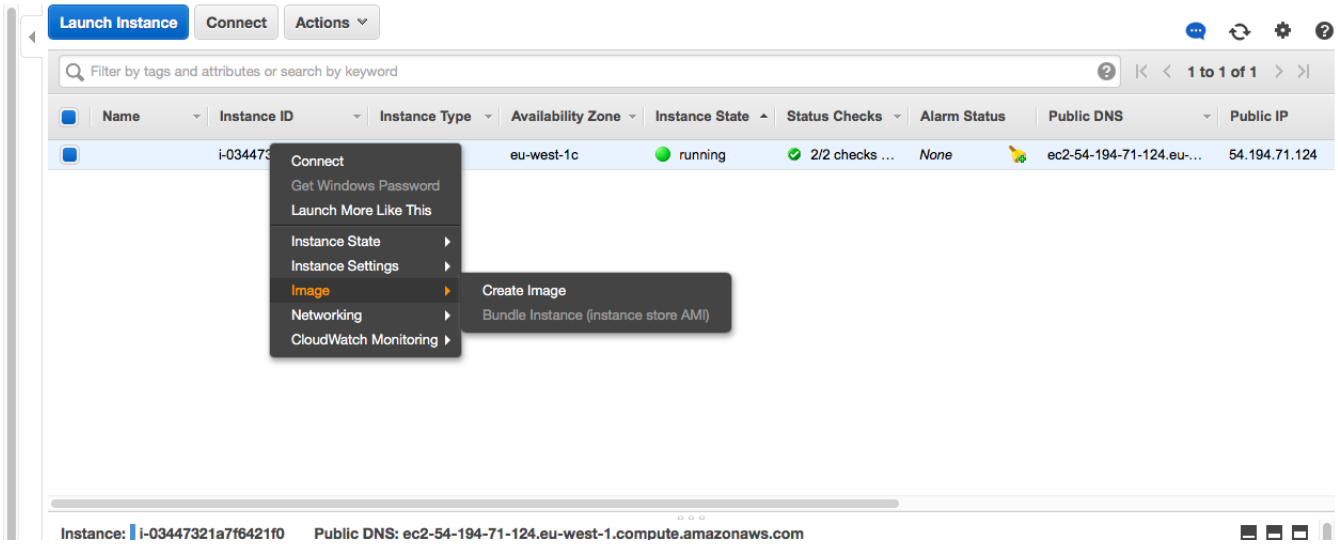

Figure 4.4: Manual image creation

Figure 4.4 shows how I decided to solve the problem. By instructing users to manually create image from running instance. Users have to go the web interface of cloud vendors, right-click on the running instance, and create image from it. This is a simple workaround which is less complicated than accommodating different features

and API from different cloud vendors.

Another problem is 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 this functionalities out to become more generic, however for the interest of time, I decided not to.

4.3.3.2 Security

Highlight inter job access

Private networking on cloud vendors

Bibliography

Bernabeu, M. O., Lu, Y., Lammer, J., Aiello, L. P., Coveney, P. V., and Sun, J. K. (2015). 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)*, pages 8070–8073. IEEE.

Franco, C. A., Jones, M. L., Bernabeu, M. O., Geudens, I., Mathivet, T., Rosa, A., Lopes, F. M., Lima, A. P., Ragab, A., Collins, R. T., et al. (2015). Dynamic endothelial cell rearrangements drive developmental vessel regression. *PLoS Biol*, 13(4):e1002125.

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

Green, C. (2014). Computer simulation could become 'integral' in the diagnosis of.

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

Mazzeo, M. D. and Coveney, P. V. (2008). Hemelb: A high performance parallel lattice-boltzmann code for large scale fluid flow in complex geometries. *Computer Physics Communications*, 178(12):894–914.

Steven, S. (2016). Hemeweb: Container based high performance computing scenario in cloud infrastructure for hemelb. Informatics research proposal, University of Edinburgh.