

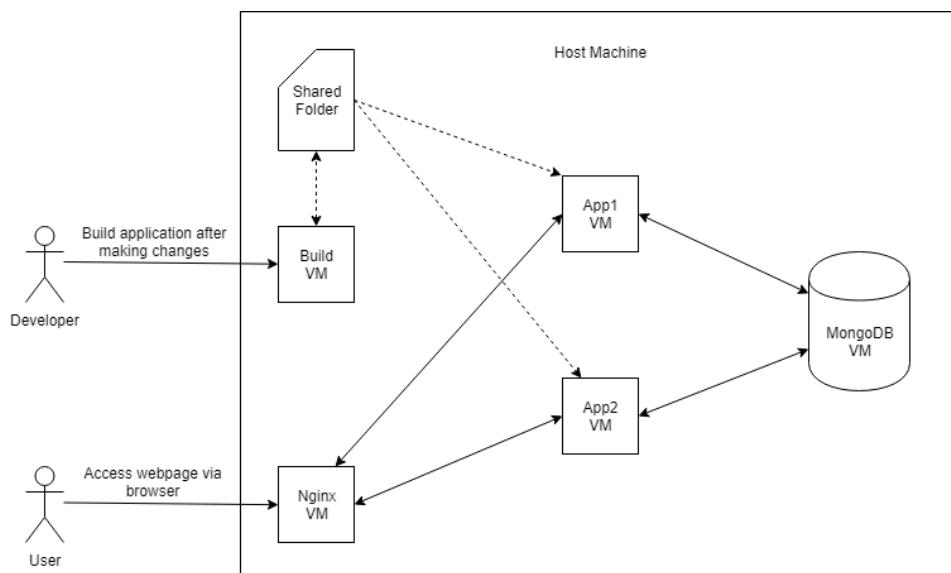
# COSC349 Assignment 1 Report

## Application Architecture

This application is a simple web-based recipe book system which allows users to store and view their favourite recipes. The application makes use of five virtual machines (all running the same version of Ubuntu Xenial to minimise additional downloads and complexity):

- **Build server:** The Go programming language development tools are installed on this virtual machine during provisioning so that the user does not need to worry about setting them up themselves on the host system. While it's easy to cross-compile Go code for any supported platform, the build server automates the entire process and removes any manual interaction which would otherwise be required. Source is accessed from, and executables are saved to the Vagrant shared folder which can be accessed from the host and any of the virtual machines.
- **Nginx load balancer:** This server runs an Nginx reverse proxy which accepts all incoming requests for the web application. It acts as a load balancer, using round-robin selection to route requests between two different application servers (see below). This allows us to distribute load between multiple application servers.
- **2 Application servers:** These two virtual machine instances are running an identical web service written in Go which is built using the standard http library. This service handles incoming requests, serving the front-end files (which are generated using the standard library template engine), and makes requests to store and retrieve data to and from the MongoDB database (see below), using the official MongoDB driver. The Bootstrap library is also used for styling the front-end pages. These virtual machines retrieve the application executable from the shared folder.
- **MongoDB database:** This virtual machine is running the MongoDB server in which all the recipes in the application are stored. It allows for flexible, persistent data storage of recipes, which can be very nicely mapped to its JSON-like document format. During virtual machine provisioning, some example recipes are automatically loaded into the database (sourced from <https://www.foodista.com> and licensed under CC BY 3.0).

The following diagram illustrates how the virtual machines interact:



## Building & Deployment

### First Build & Deployment

Vagrant is used to provision and manage the virtual machines (<https://www.vagrantup.com/>). Once Vagrant has been installed, running the command `vagrant up` in the root project directory will create and provision all five virtual machines, and build, deploy, and run the application.

This generally takes around 10 minutes to complete, although may take a shorter or longer time due to varying computer and download speeds.

During this process, various resources will be downloaded for the virtual machines. These resources, with their approximate download sizes are as follows:

- Ubuntu Xenial Vagrant Box: 271MB
- Go Language Tools: 122MB
- Nginx: 16MB (apt-get update) + 3MB (Nginx & dependencies)
- MongoDB: 16MB (apt-get update) + 81MB (MongoDB & dependencies)

**Total: 509MB**

At this point, the virtual machines and the applications running on them are all configured. The Go web application is automatically built, and the example data is loaded into the database.

Any virtual machines started by this process can be shut down by using the command `vagrant halt` or completely deleted using the command `vagrant destroy`.

No downloads are required for any subsequent builds.

### Subsequent Builds & Runs

If any of the five virtual machines are no longer running, the `vagrant up` command should be used once again to start the virtual machines. This will run much more quickly than the first time it was run (usually taking around 4 minutes).

Once this has been completed, the application should be ready to use again. If changes were made to the web application source code, it only needs to be rebuilt if the Go source files have been changed. Any changes to the static content and template files will be immediately applied without requiring any action.

To build the Go source files, simply run the shell script `hostbuild.sh` on the host machine (note that `build.sh` is used by the build server and should NOT be run by the host). This script does two things:

- Runs the `build.sh` script on the build server to compile the Go source code.
- Restarts the app services on both application virtual machines if the build succeeded.

The commands to perform these operations can be run independently, and are as follows:

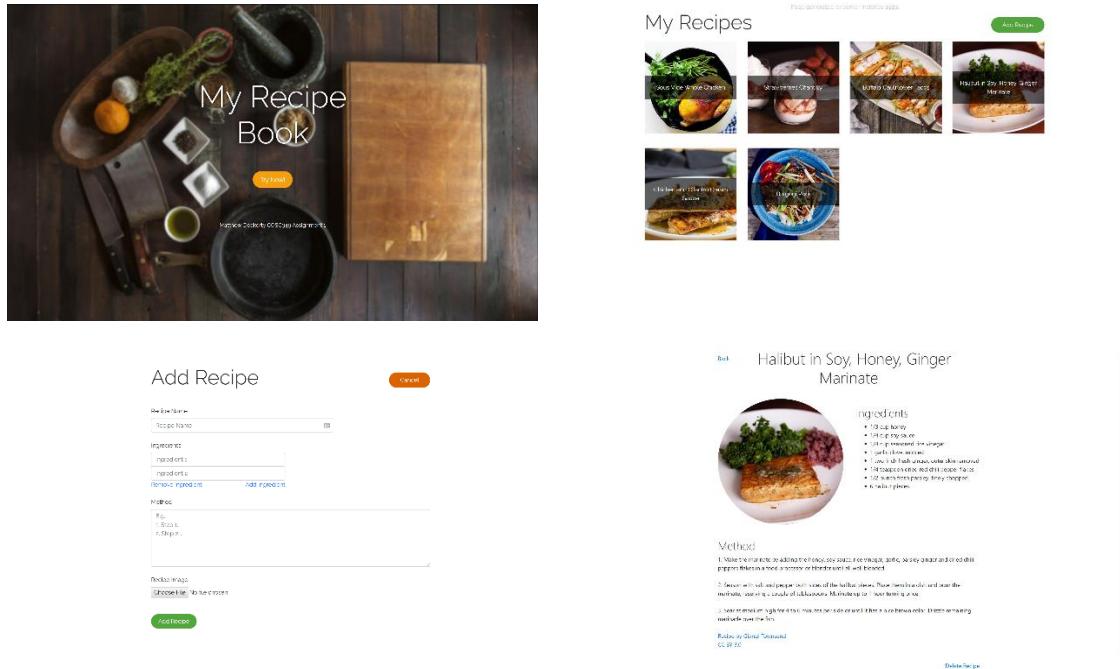
```
vagrant ssh build -c "exec /vagrant/build.sh"
vagrant ssh app1 -c "sudo systemctl restart app"
vagrant ssh app2 -c "sudo systemctl restart app"
```

## Application Usage

The application is very simple and provides 4 main features: viewing all recipes, viewing details for a specific recipe, adding new recipes, and deleting recipes.

Localhost port 8080 has been forwarded to the Nginx virtual machine, and so the web application can be accessed via <http://localhost:8080>.

Some screenshots of the application are shown below, and a short video clip demonstrating using the application can be found at <https://youtu.be/bSMsAYRreik>.



## Potential Modifications

In its current state, the application is very minimal and there are many potential improvements which could be made. These include the following:

- Basic recipe data such as ingredients and method are stored by the application, however, most recipes also contain other information such as preparation & cooking time, and the number of servings the recipe makes. These attributes could be added to the recipe model, the add recipe form, and the view recipe page, to provide more detailed information to anyone following a recipe.
- Recipes can currently be added and deleted but editing functionality has not been implemented. This means that if a user wishes to modify a recipe, they must delete the existing recipe, write up the entire recipe again, and re-add it. It would be beneficial to allow users to modify existing recipes. This would require creating (and auto-filling) a front-end form and implementing the back-end functionality to update recipes stored in the database.
- In the existing implementation, all the recipes in the database are fetched and displayed on the recipes list page. This isn't an issue with a small number of recipes but could cause future performance and scalability issues when many recipes have been stored. A common approach used to avoid this problem is to implement some form of pagination – where

recipes would be split over multiple pages that the user could navigate between. In addition to pagination, a simple search feature could also be implemented to allow users to easily find recipes without having to scroll through the entire list.

- It's often difficult to follow a recipe that uses different measurement units to those which we are familiar with. This would be a more extensive modification, but it would be helpful to provide automatic conversions between units of measurement so that people following the recipes do not have to stop and convert units while cooking.

If a developer were to make any changes to the system, they would need to follow the instructions outlined in the *Building & Deployment* section of this document (see above) to build and re-run the application.