

Infrastructure as Code Implementation to deploy a Web Application

Submitted in fulfillment of the 'J' component for the Course

VIRTUALIZATION CSE4011 SLOT F2

By
Project Group **P20**

Saksham Saini	17BCB0094
Vinay Verma	17BCE0110
G Lokesh	17BCE0684
Rahul Nair	18BCE0750

Under the guidance of Prof. Kalyanaraman P

**School of Computer Science and Engineering
VIT, Vellore**



6th June, 2020

CONTENTS	Page No.
1. Abstract	3
2. Motivation	3
3. Objectives	4
4. Literature Survey	5
4.1 Abstract	5
4.2 In Brief	6
4.3 Drawbacks of existing methodologies	13
4.4 Bibliography	14
5. Modules and Diagrams	17
5.1 Keywords	22
6. Drawbacks and Conclusion	23
7. Methodology and Introduction	30
8. Conclusion	40
9. Appendix	42

1. Abstract

- Infrastructure as Code (IaC) is the process of managing and provisioning computer data centers through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools also practices that include version control, peer review, automated testing, release tagging, release promotion, and Continuous Delivery.
- The wide adoption of virtualization and self-service cloud infrastructure has shifted the bottleneck from allocating servers to configuring them. Where it used to take weeks or months to allocate a server, now it can be done in a minute or two.
- IaC approaches are promoted for cloud computing, which is sometimes marketed as infrastructure as a service (IaaS). IaC supports IaaS, but should not be confused with it.

2. Motivation

- The older methods of infrastructure management — manual processes and documentation, brittle single-purpose scripts, and graphical user interface based tools — each had their uses in the past. Today, though, with the perpetual need to scale infrastructure, the doption of ephemeral infrastructure, and greater application system complexity, new ways of keeping things under control are needed.
- Previous methods of IT management won't work if you're trying to address modern scale and agility requirements. IT automation, sometimes referred to as infrastructure automation, is the use of software to create repeatable instructions and processes to replace or reduce human interaction with IT systems. Automation software works within the confines of those instructions, tools, and frameworks to carry out the tasks with little to no human intervention.

3. Objectives

- To successfully build a Web application which can communicate with Docker and should be dynamic.
- To deploy the web app with proper Nginx configuration to secure it and to load balance the app as well as implement reverse proxy.
- To make proper Dockerfile and docker-compose.yml files and connect all of the services.
- Technologies to use and implement:
 1. Containers – Docker (IaC)
 2. Container Orchestration – docker-compose (IaC)
 3. Webserver as reverse proxy– Nginx
 4. Web server as Load Balancer – Nginx
 5. Visualize Database – Adminer (IaC)

For Web application: Front end – html, css, js

Back end – go lang

Database - MySQL

4. Literature Survey

4.1 Abstract

There's no doubt that cloud computing has had a major impact on how companies build, scale, and maintain technology products. The ability to click a few buttons to provision servers, databases, and other infrastructure has led to an increase in developer productivity we've never seen before. While it's easy to build some simple cloud architectures, mistakes can easily be made while provisioning complex ones. Human error will always be present, especially when you can launch cloud infrastructure by clicking buttons on a web app. The only way to avoid these kinds of errors is through automation, and Infrastructure as Code [1][2][3][4][5] is helping engineers automatically launch cloud environments quickly and without mistakes.

Virtualization is the key to cloud computing, since it is the enabling technology allowing the creation of an intelligent abstraction layer which hides the complexity of underlying hardware or software [7][8][23][25][27][28]. Virtualization [6][9][10] is the process of running a virtual instance of a computer system in a layer abstracted from the actual hardware. Cloud computing is the on-demand availability of computer system resources, especially data storage and computing power, without direct active management by the user [29]. We make use of Docker technology [13][15][19][20][21] for proper deployment of the WebApp. We also use a variety of open source program platforms such as Ansible [12][24], Jenkins [24], Hadoop Distributed File System [11][17][18][19], Kubernetes [14], Nginx [16], GitLab [22], etc. We also analyse the CRUD methodology [26] and MVC methodology [30].

4.2 In Brief

1) The model Cloud WorkBench (CWB) framework is grounded on IaC to foster simple definition, execution, and repetition of benchmarks over a wide array of cloud providers and configurations. One core feature of CWB is that benchmarks, including the cloud configuration they are evaluating, can be defined entirely in code and by using the CWB web interface, essentially following the ideas of DevOps and IaC. The client library present in all the cloud VMs, along with all other required code (e.g., Linux packages required by a benchmark, or the benchmark code itself), is provisioned in the cloud VMs based on IaC configurations retrieved from a provisioning service. The provisioning service knows how to prepare a given bare VM to execute a given benchmark. The definition of client VMs and provisioning configurations follows the established notions of standard IaC tooling, e.g., Vagrant and Opcode Chef.

2) As part of the DevOps menu, many practices entail re-using standard tools from software development (e.g., code-versioning, code revision management, etc.) to manage what is known as infrastructure-as-code (IaC). IaC promotes managing knowledge and experience of plethora of subsystems as a single commonly available source of truth instead of traditionally reserving it for system administrators. This briefing introduces all essential technologies involved in supporting infrastructure-as-code, starting from the industrial standard for IaC, OASIS TOSCA, which stands for “Topology and Orchestration Specification for Cloud Applications” and elaborating more on connected technologies, such as deployment blueprints or TOSCA-ready orchestration engines. The purpose behind this proposal is to re-use successful and common software development practices to speed up software operations: think of using infrastructure code versioning for the purpose of de-bugging and back-tracking/version control or using infrastructure design patterns to quickly put together common solutions to known problems or even exploiting model-driven engineering to pull IaC specifications immediately from architectural design and development models. In fact, IaC is a key enabler of several DevOps tenets that heavily depend on automation.

3) In DevOps, infrastructure as code (IaC) scripts are used by practitioners to create and manage an automated deployment pipeline that enables IT organizations to release their software changes rapidly at scale. Low quality IaC scripts can have serious consequences, potentially leading to wide-spread system outages and service discrepancies

4) In this paper, the existing research is further developed by first applying code the IaC smells to other technologies and it is observed that similar results are achieved. The paper describes how it applied the code smells in two case studies to open and closed source IaC code repositories, where the presented results indicate that IaC smells are present in other tools and technologies. Furthermore, the results

show that IaC smells are agnostic to the applied technology and can be defined on a technology agnostic level. Secondly, the paper also introduces new code smells from the field of software engineering, which were not covered yet, to the domain of IaC. The paper presents a catalogue of 17 code smells which were applied to Chef and whose implementation is available as Open Source.

5) Infrastructure as Code, IaC is a model aimed to describe an infrastructure in a declarative format using a high-level abstraction of infrastructure components. IaC methods allow automation of software deployment. This approach allows user to create versions and process them just like any other code.

6) This paper defines and identifies the various techniques of paging in virtual environment and its uses. It defines the best condition/purpose for each technique and gives the comprehensive description on the methodology and effective utilisation.

7) Virtualization simply means generating resources, which are virtual, for example, virtual server, virtual network switch or virtual storage device, using a single resource. Through virtualization, you can run multiple VMs with different OS and that too sharing a single system. Since virtualization will isolate every environment, crashing of any one system does not affect the other. Virtualization in cloud computing simply is to allocate smaller system components in a siloed approach. Through virtualization in cloud computing cloud vendors can maximize resource utilization and reduce their overheads. Virtualization forms the backbone of cloud technology. Through virtualization, vendors are empowered to induce business scalability, robust security and resource flexibility.

8) The paper defines a design of Virtual Machine aware flow switch in the Cloud based Network Function Virtualization System, which capable of providing a service at a stable speed even though the number of virtual machines connected to a hypervisor is increased by extracting flow information about each of a plurality of packets and providing network virtualization.

9) Recently, combining virtualization technology is used in current server consolidation widely, where physical machines are transferred to VMs through P2V technique. As Virtualization technology can improve utilization of physical resources and save cost, it becomes the norm. But, the long-term continuous operations of VMM and VMs will lead to software aging (phenomenon of performance degradation). So, the paper analyses software rejuvenation of VMM and VMs in virtualization environment, combining the characteristics of virtual machine system.

10) As all PC servers are not suitable to be virtualized, it is not true that in HPC all applications are not suitable to be virtualized. To some extent, virtualization is just the effective way to enhance the extra value of HPC, and even the best choice to solve the traditional difficulties HPC has been faced with. The possible benefits of such a system entails improvements in development efficiency, integration of heterogeneous resources, provisioning of customized appliance, improvements in reliability and fault tolerance of the system, improvements in the security of HPC systems, etc. However, for every model there exist certain challenges for its further developments. Some of these include performance overheads brought by virtualization, efficient coordination of many VMMs, the management of a large number of VM, programming model and the support of software environment.

11) It can be seen that Big Data projects need large datasets and working with those huge datasets will be tough using regular databases and visualization tools. To overcome these problems, the general concise is to use software like Hadoop running on large number of servers. In this paper they show a detailed analysis of tools and methods used for these purposes. The main tools used are Hadoop and Amazon Cloud Services. With the analysis it is clear that the NX parser is the best for applications needing no validation. Next best is the Any23 due to its lack of efficiency. Jena is the last and slowest because it is performing validation while parsing. But all these have their own advantages and disadvantages. If an application has large data set and we want quality as our first priority, then we should go with Jena. Else if the validation is already performed or we need speed as the priority we should go with NX parser or Any23. In the intermediate comes the amazon EMR. It is good with small files like 0.4 to 4.7 GB files.

12) With the increasing demands of users and worker there is a significant increase in the servers and end stations. Due to these increasing numbers there are some difficulties while maintaining these servers. To manage these servers, we use data-centre orchestration and configuration management tools. In this paper they had considered building a new layer for the utilized Ansible orchestration tool. The developed framework is tested in 10 laboratories at Brno University of Technology. They created a new framework which can provide security for remote management and configuration of the selected network or network parts.

13) As today's world is moving towards a completely virtualized world, we need to find the functionalities and some drawbacks of regular models and develop new models to increase the efficiency and performance. In this paper they proposed to develop an LTE soft base station (SBS) in Docker, utilize a range of hardware accelerators (HA) to speed-up signal processing and design an efficient HA framework for multiple SBSs sharing the HA resources simultaneously. In their model the real computer has multiple FPGA accelerators with different functions, which are accelerated by unified Linux driver. For each SBS there is a virtual acceleration memory which are running in a

Docker container. This model is an exploration for current 5G technology. SBS running in Docker has flexible features to fulfil complex wireless requirements and adapt to fast technical iteration.

14) This paper deals with the MANO specification aspects with Kubernetes fundamental containers orchestration mechanism. This work helps in finding out that in which degree Containerized Network Functions (CNFs) can be managed with the use of Kubernetes platform. They practically checked that the Kubernetes can fulfil performance constraints and fundamental requirements imposed by MANO standard in the area of performance and fault management. The results show that the Kubernetes can provide powerful mechanism for autoscaling and self-healing capabilities. While autoscaling or failures, traffic is automatically distributed to new or healthy parts of the system respectively.

15) We know that new a days distributed technology is widely used. It provides a new stack of computing placed on virtualization of assets. Due to this method of growing the stack on the servers, the resources are not used effectively. In this paper they try to balance the work on every node. It helps in preventing overloading of hubs. They also show how services have been accessed by the node in a cluster with the help of docker swarm and Kubernetes. As a result, this paper proved that Kubernetes is superior than the docker swarm even though Kubernetes is difficult in installing process but is effective in major ways.

16) Nginx is famous for its high concurrent processing capability, while its performance and scalability are excellent. Nginx is a high-performance HTTP and reverse proxy server, in the case of highly concurrent connections, memory, CPU and other system resources have a very low consumption, and a steady operation. Therefore, the Nginx based Web load balance and cache optimization design under the high-concurrency environment is a very large practical significance.

17) Hadoop is an open source software, distributed processing of very large data sets on clusters. Built in commodity hardware, all the modules are designed in Hadoop environment with its success based on the assumption that hardware failures should be automatically handled by the framework. The Hadoop file system runs on standard or low-end hardware as distributed file system. HDFS provides native support of large data sets, high fault tolerance, and data throughput access in Mapreduce algorithm. HDFS is developed in Apache Hadoop. HDFS greatly improves the performance of the system with multinode clusters. It significantly improves the performance of the system during multiple user access to the system. It is also proved that the multinode cluster reduces time of execution and increases throughput.

18) The paper reviews the features of all DFS, chooses highly reliable, resilient DFS for big data. The paper also reviews and makes analysis of log mandate, format and proposes a common log format. The paper also identifies the security issues and safety of proposed DFS. The proposed model should ensure confidentiality of stored log, integrity and availability of proposed DFS. Implementation of proposed DFS with 3 nodes and optimization of DFS using open source software resources. Perform log analysis and simulation to obtain desired results like best performance of read, write latency and security features. Optimization of DFS using open source software resources.

19) The paper analyses the large Docker registry workload and depicts the potential for deployment time improvements for Docker image sharing. It also surveys and identifies the type of distributed file system that is suitable for fog computing environments. The paper defines a model consisting of a Docker background for image sharing as a framework. The model is capable of image sharing among multiple co located nodes. Lastly, the paper also compares the effectiveness of the proposed model's methodology of sharing cache among fog nodes in terms of average cache hit rate and reduction in average container deployment.

20) The primary objective of the paper is to evaluate the name resolution application of the NovaGenesis (NG) architecture in the Docker environment. In other words, the aim of the paper is to evaluate whether Docker is a reliable tool for experimentation and its compatibility with NG architecture, since this encompasses several network concepts and services that may not be as simple to experiment with Docker. It was seen through the results that the Docker provides necessary tools and easy implementations of the NG architecture. Packaging applications in containers facilitate horizontal scalability of processes, access and closure of multiple instances. Moreover, it is a platform that consumes few computational resources and is widely used in solutions that rely on rapid prototyping or fast system.

21) In this paper, the specialized features of the Docker technology are used to deploy a user-centred social network application that helps users to have benefit and enjoy activities in real life. The Docker is used as a platform for the application as it provides high performance as well as gives auto-scalability. It is also seen that Docker provides good reliability.

22) In this paper, we summarise how Learning Analytics (LA) methods have previously been used in Software Engineering (SE) education, with a focus on how student data generated through interaction with GitLab can be handled to gather significant information not only about the system-user interactions but also about the mistakes and lessons learned along the way. The paper also presents a description of a proposed pipeline for LA that incorporates the steps needed to prepare the data for LA,

from data extraction to data analysis, and taking into account the need for confidentiality, for which an exploration of alternatives for data anonymisation was performed.

23) This paper defines an approach of continuous integration and continuous deployment which is used to automate the deployment every time there is a new update in the codebase. There are multiple CI/CD tools available in the market to solve the above requirement. CI is a segment of practices involved in software programming principles. CI states that the entire code for an application should be kept in a common repository so that whenever the developer checks in the code into the repository, a script is triggered which picks the latest code from the repository, integrate it with the existing code and run the test cases designed according to the application. CD (Continuous Delivery) is defined as an ability to deploy new features, bug fixes into the live server as and when required. CD part is executed after a successful CI where all the updated code is lying in the master branch. A script is run which picks the code from the master branch, prepares the build and deploy it to a test environment/production environment.

24) The paper defines a model which consists of integrated delivery system by the development team, the version control repository, continuous integration server, build server and continuous integration feedback mechanism which constitute the collaboration system, and the repository change monitoring, automated build to automatic testing and continuous integration to promote the state feedback mechanism. The continuous integrated delivery control system is based on Jenkins continuous integration server to implement automatic operation and maintenance.

25) In this paper, an out-VM monitoring approach based on introspection, called vProVal, is proposed. The vProVal is designed to detect the hidden processes and rootkits that disable the security tool, running in the monitored VM in Kernel VM (KVM)-based cloud environment. It performs the malware detection from outside the VM at the KVM-layer and hence more robust to attacks. The introspection technique used is to extract the low-level details of a running VM from hypervisor by viewing its memory, trapping on hardware events, and accessing the vCPU registers.

26) In this paper, the CRUD operations (Create, Read, Update, Delete) were used to determine colour preferences. Essentially the paper tries to identify whether the CRUD operations cause certain colours to preferred over others and where each colour was given equal preference.

27) In this paper, the model derives the conclusion that the integration and simplification of the existing operator IT support system architecture with the new and constantly updating virtualization technology can effectively improve the IT support system's response capability, reduce IT input costs, and

accelerate the pace of innovation, incubation and deployment of new services. It also allows the reduction in the cost of new business investment, operation & maintenance, so as to effectively improve the competitiveness of enterprises.

28) The paper consists of research done on the utilisation of virtualisation for data centre computing, as well explains the various key factors of virtualisation. It defines the methodology of cloud computing and analyses the various benefits of a cloud computing server over a normal server in a data centre both theoretically and in real time use.

29) This paper describes the possible application of lightweight virtualization to Fog Nodes in the form of case studies. It defines the methodology of containers and depicts the difference in performance metrics like performance overhead, booting time, CPU overhead, etc between the utilisation of containers and virtual machines. It proceeds to prove the theoretical assumptions and hence proves the benefits of containers over virtual machines.

30) The model defined by this paper utilises the concepts of Model View Controller (MVC). The essence of the MVC design patterns is to demarcate the presentation (View) and the business logic (Model) through an interface integration layer (Controller). However, in the traditional MVC design patterns, there is a provision for an association between View and Model. Drawbacks of such association does not define the privacy of Data Model as well as the complete decoupling between View and Model. Therefore, in order to achieve an ultimate decoupling and privacy of Data Model, the model imitates an Inversion of Control in the Controller by applying the DI pattern which ensures an abstraction wrapper over Model.

4.3 Drawbacks of existing methodologies

- 1) In traditional systems unlike infrastructure as code we can't reuse successful and common software development practices to speed up software operations.
- 2) There is no standard language for infrastructure design in a software lifecycle phase. Infrastructure design include automation and configuration code, models, required dependencies and operational configuration parameters declaration.
- 3) In traditional process there are no methods of automatically building, provisioning, configuring and managing a software system.
- 4) The deployment of complex infrastructure with lots of hardware and software requirements becomes a problem and causes errors if using traditional deployment tools.
- 5) We can't describe an infrastructure in a declarative format using a high-level abstraction of infrastructure components.
- 6) Software deployment requires more time in traditional methods while compared to using infrastructure as code.

4.4 Bibliography

- 1) Cloud WorkBench – Infrastructure-as-Code Based Cloud Benchmarking-Joel Scheuner, Philipp Leitner, Jurgen Cito, Harald Gall
- 2) DevOps: Introducing Infrastructure-as-Code: Matej Artac, Tadej Borovsak, Elisabetta Di Nitto, Michele Guerriero, Damian Andrew Tamburri
- 3) Anti-Patterns in Infrastructure as Code: Akond Rahman
- 4) Code Smells in Infrastructure as Code: Julian Schwarz, Julian Schwarz, Horst Lichter
- 5) Domain-specific language for infrastructure as code: Valeriya Shvetcova, Oleg Borisenko, Maxim Polischuk
- 6) Paging techniques in Virtual Environment: Ekta, Manisha Agarwal
- 7) Overview of virtualization in cloud computing: Nancy Jain, Sakshi Choudhary
- 8) A Design of Virtual Machine aware Flow Switch in the Cloud based Network Function Virtualization System: Kang Il Choi, Bhum Cheol Lee
- 9) Software Rejuvenation in Virtualization Environment: Kehua Su, Hongbo Fu, Jie Li, Dengyi Zhang
- 10) Research on the Application of Virtualization Technology in High Performance Computing: Van lunhao, Lv Aili

- 11) Garcia, Ted, and Taehyung Wang. "Analysis of big data technologies and method-query large web public RDF datasets on amazon cloud using Hadoop and open source parsers." 2013 IEEE Seventh International Conference on Semantic Computing. IEEE, 2013.

- 12) Masek, Pavel, et al. "Unleashing full potential of ansible framework: University labs administration." 2018 22nd conference of open innovations association (FRUCT). IEEE, 2018.

- 13) Wu, Kunheng, et al. "The Design of Soft Base Station Based on Docker." 2018 IEEE 4th International Conference on Computer and Communications (ICCC). IEEE, 2018.

- 14) Gawel, Maciej, and Krzysztof Zielinski. "Analysis and Evaluation of Kubernetes Based NFV Management and Orchestration." 2019 IEEE 12th International Conference on Cloud Computing (CLOUD). IEEE, 2019.

- 15) Marathe, Nikhil, Ankita Gandhi, and Jaimeel M. Shah. "Docker Swarm and Kubernetes in Cloud Computing Environment." 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI). IEEE, 2019.

- 16) Web Load Balance and Cache Optimization Design based Nginx Under High concurrency Environment: Xiaoni Chi, Bichuan Liu, Qi Niu, Qiuxuan Wu

- 17) Formation of Single and Multinode Clusters in Hadoop Distributed File System: A. Aasha Begum, Dr. K. Chitra

- 18) Develop a Model to Secure and Optimize Distributed File Systems for ISP Log Management: Mohammad Shazzad Hossain, Pratik Kumar Bhowmik, Md. Abidur Rahman, Mahbub Uddin, Mohammad Abu Yousuf, Kazi Abu Taher

- 19) Docker Image Sharing in Distributed Fog Infrastructures: Arif Ahmed, Guillaume Pierre

- 20) A Docker-Based Platform for Future Internet Experimentation: Testing NovaGenesis Name Resolution: Elcio C. do Rosario, Victor Hugo D. D'Avila, Thiago Bueno da Silva, and Antonio Marcos Alberti

- 21) Building a Prototype Netcloudbook for User Activities-centred Social Network based on Booked Open-source and Docker Technology: Thu Thuy Trieu, Dana Petcu
- 22) A Methodology for Using GitLab for Software Engineering Learning Analytics: Julio Cesar Cortes Rios, Kamilla Kopec-Harding, Sukru Eraslan, Christopher Page, Robert Haines, Caroline Jay and Suzanne M. Embury
- 23) Comparison of Different CI/CD Tools Integrated with Cloud Platform: Charanjot Singh, Nikita Seth Gaba, Manjot Kaur, Bhavleen Kaur
- 24) Design and Implementation of Continuous Integration scheme Based on Jenkins and Ansible: Wang yiran PetroChina, Zhang tongyang PetroChina, Guo yidong PetroChina
- 25) vProVal: Introspection based Process Validation for Detecting Malware in KVM-based Cloud Environment: Preeti Mishra, Ishita Verma, Saurabh Gupta, Varun S. Rana and Kavitha Kadarla
- 26) Measuring users' color preferences in CRUD operations across the globe: a new Software Ergonomics Testing Platform: Petra Tomanová, Jiří Hradil, Vilém Sklenák
- 27) Cloud Computing Architecture Design of Database Resource Pool Based on Cloud Computing: Sui Yi, Li Yuhe, Wang Yu
- 28) Cloud Computing: Virtualization and Resiliency for Data Center Computing: Valentina Salapura
- 29) A Review on Container-Based Lightweight Virtualization for Fog Computing: M. Sri Raghavendra, Priyanka Chawla
- 30) Enterprise Service Bus Dependency Injection on MVC Design Patterns: Sidhant Rajam, Ruth Cortez, Alexander Vazhenin, Subhash Bhalla

5. Modules and Diagrams

Architecture of our web app

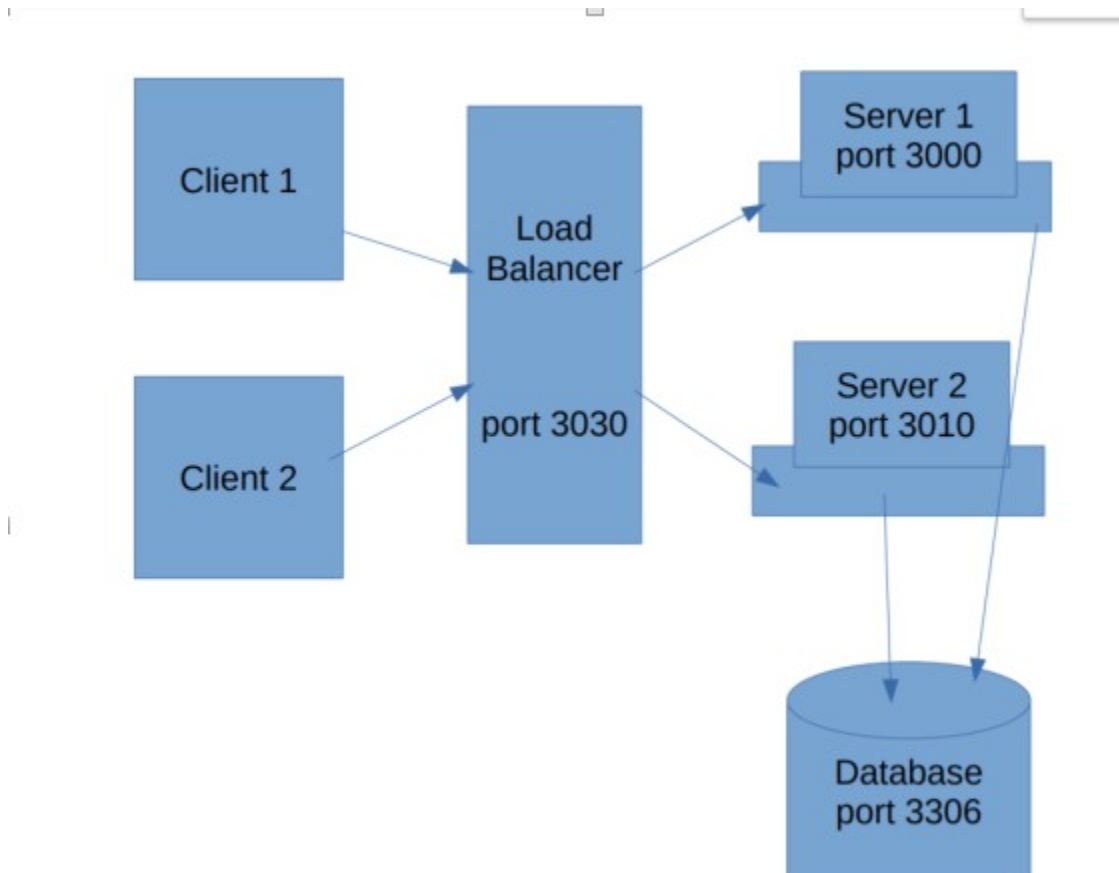


Fig1: Web application architecture

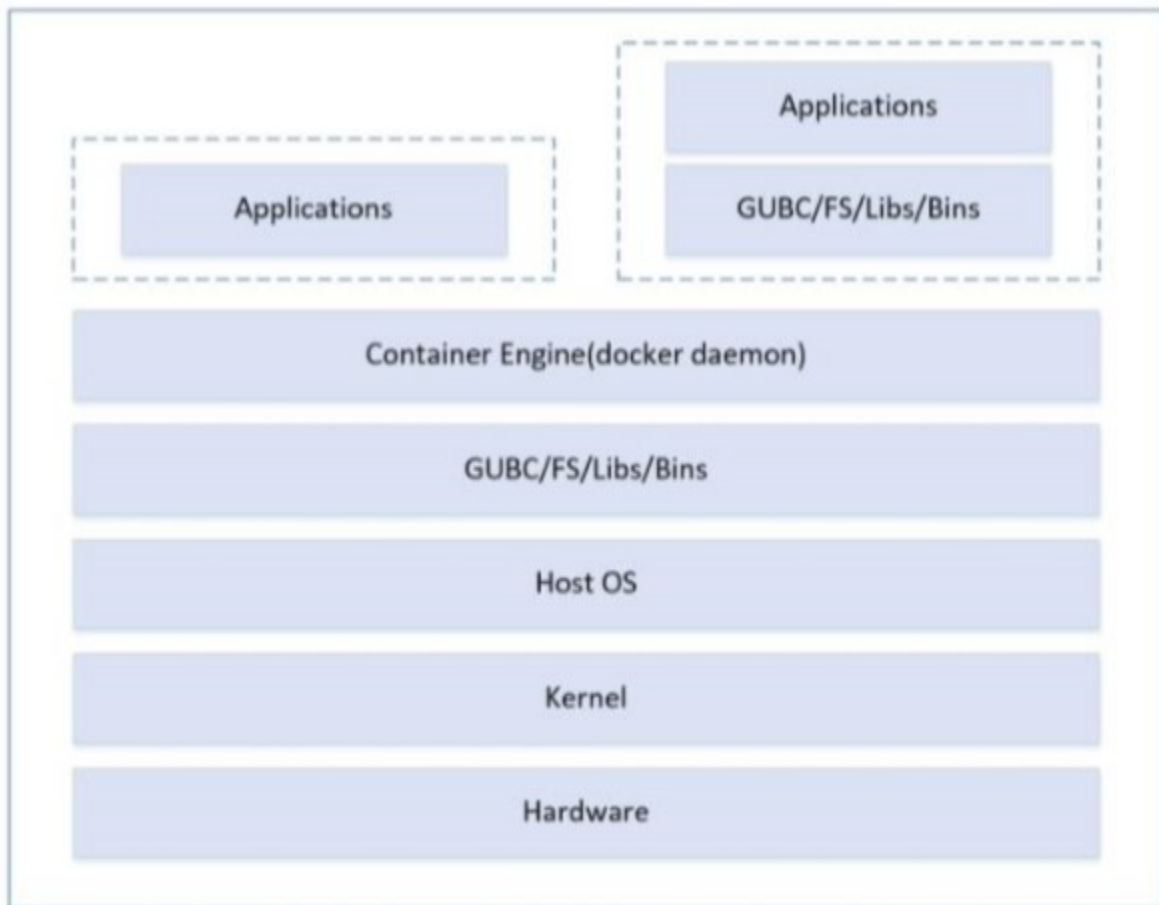


Figure 2. The Docker structure.

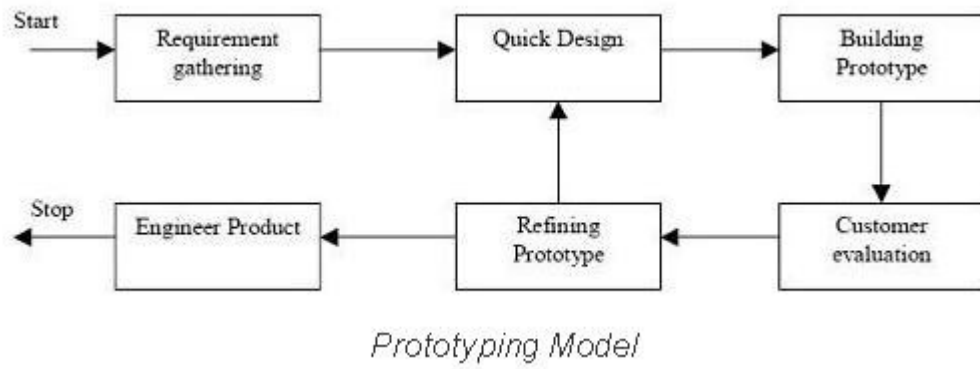


Fig3: Software Model: Prototyping model

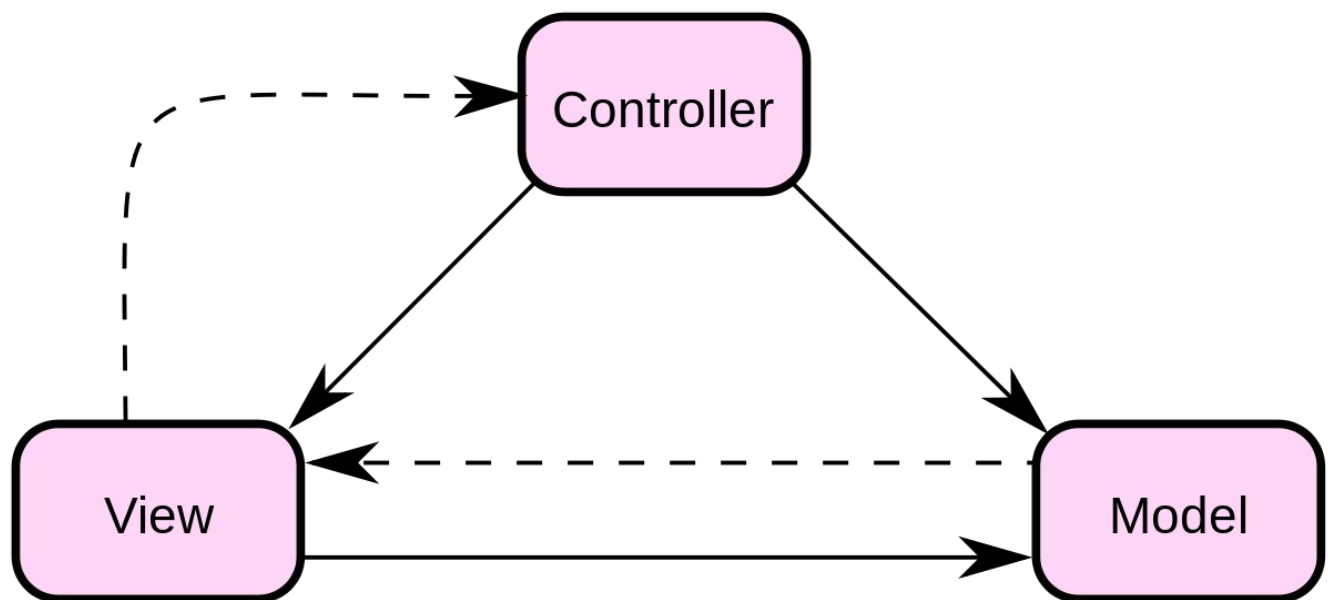


Fig4 : Software Architecture: MVC

Configure all servers to run app version 2.1

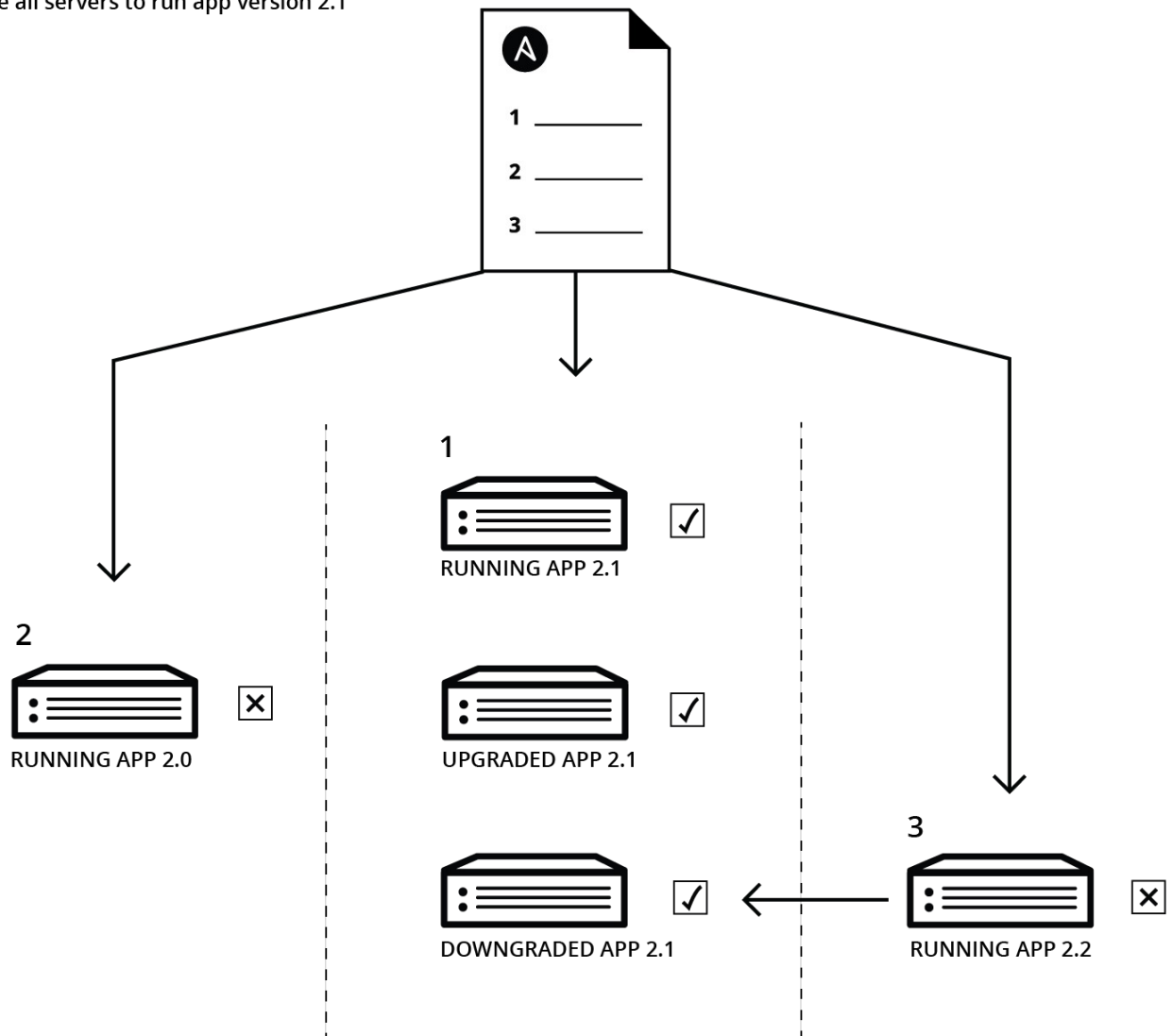


Fig5: Ansible demonstration

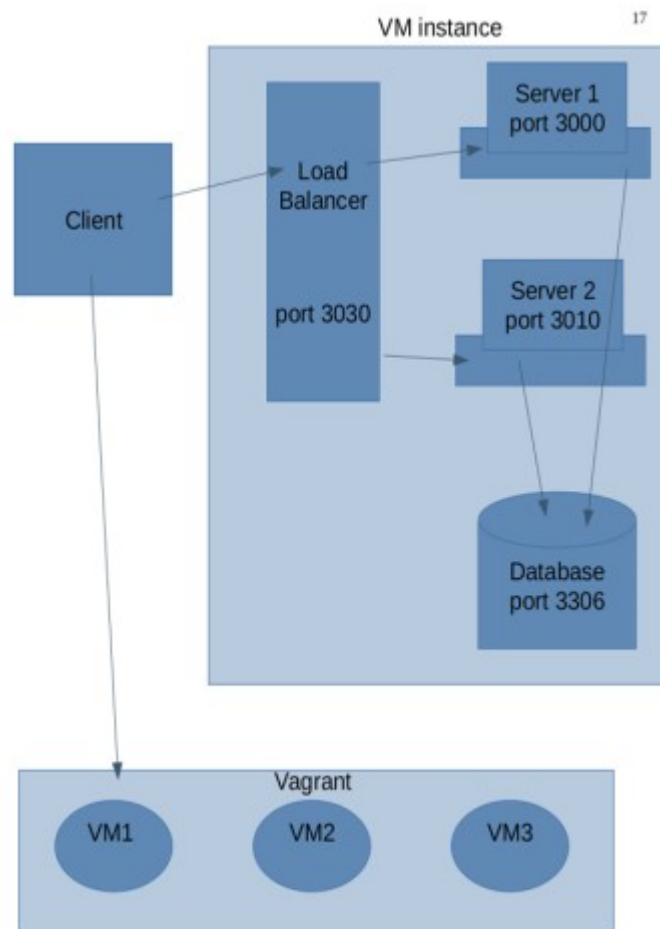


Fig6: Vagrant demonstration

5.1 Keywords

Infrastructure as Code (IaC),

DevOps,

Pagination,

Cloud Computing,

Amazon Web Services (AWS),

Hadoop, Distributed File System (DFS),

Virtualization, Big Data, Ansible,

Docker, Vagrant, Kubernetes,

Orchestration, Docker Swarm,

Load Balancer, Cache Optimizer,

Nginx,

Distributed Fog Infrastructure (DFI),

GitLab, Continuous integration and continuous delivery (CI/CD),

Jenkins,

K Virtual Machine (KVM),

Create-Read-Update-Destroy (CRUD),

Fog Computing,

Model-View-Controller (MVC).

6. Drawbacks and Conclusion

1. Client-server Software Architecture Style

Drawbacks:

1) Overloaded Servers:

When there are frequent simultaneous client requests, server severely get overloaded, forming traffic congestion.

2) Impact of Centralized Architecture:

Since it is centralized, if a critical server failed, client requests are not accomplished. Therefore, client-server lacks the robustness of a good network.

Conclusion:

According to our Web Application needs we are building an app which has a sign up, login and home screen only made for the use of clients. Hence, this software architecture style is in our best interest.

2. MVC Web Application architecture

Drawbacks:

- It is hard to understand the MVC architecture.
- Must have strict rules on methods.

Conclusion:

We are building such a web application that consists of some frontend, backend and database. Hence, describing basic conditions to use and implement a MVC software architecture.

3. Prototyping Model Software Model

Drawbacks:

- Leads to implementing and then repairing way of building systems.
- Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.
- Incomplete application may cause application not to be used as the full system was designed
- Incomplete or inadequate problem analysis.

Conclusion:

This web application is a basis or can be a initial/demo app for a big project. Hence, it categorizes under Prototyping model also due to the fact that we made it this way.

4. Golang - Backend

Drawbacks:

1) A young language, so it is still developing

Being a very young language, developers might find it difficult to make maximum use of the libraries. They might have to write the libraries themselves and there aren't many books or online courses to help, while in doubt.

2) Absence of manual memory management

Manual memory management is important, and the lack of it could lead to overhead garbage collection, issues like pauses, which in turn could lead to system programming.

3) Too simple

The simplicity of the language could be an issue, as it makes it superficial. And in an effort to make it simple, the language wasted several years of programming language progress.

4) Error handling isn't perfect

Though error handling is not perfect in Go, the imperfectness of it could get you. Solutions are still being searched for, and proposals have come up for error handling.

5) Runtime safety is not that good

Go is safe, but it doesn't deliver the level of the safety that Rust provides. The safety level is compile-time only and to a certain extent runtime. Go focuses on the speed of production, and Rust concentrates on the safety aspect.

Conclusion:

Golang is relatively new as when compared to other languages but it is very fast even when not using any frameworks. It also potential to replace C++ in the future if it gets developed enough. Hence, for learning purposes and to know about this modern language we choose it.

5. MySQL - Database

Drawbacks:

1) It's Got A Few Stability Issues

According to Digital Ocean, MySQL tends to be somewhat less reliable than its peers. These stability issues are related to the manner in which it handles certain functions (such as references, transactions, and auditing). While the database is certainly still usable in light of these problems, they do tend to make MySQL a poor choice for certain use cases.

2) It Suffers From Relatively Poor Performance Scaling

Although MySQL is equipped to handle a virtually limitless volume of data, it has a troubling tendency to come grinding to a halt if it's forced to deal with too many operations at a given time. This relatively poor performance scaling means that anyone with high concurrency levels should probably look into an alternative.

"In my experience," writes software engineer Koushik Ramachandra, "I have found that MySQL works better when you have a low write/read ratio, and offers low scalability as the read/write ratio grows."

3) Development Is Not Community Driven – and Hence Has Lagged

Since Oracle has taken the helm of MySQL's development, progress appears to have ground to a halt, with only one major release in the past several years. The company doesn't accept community-developed patches, nor has it bothered to offer users any sort of roadmap for MySQL development. There's really no way for developers to discuss the database management system with Oracle – and that's a problem.

4) Its Functionality Tends To Be Heavily Dependant On Addons

Although MySQL is relatively easy to set up, it tends to have less out-of-the-box functionality than many other database systems on the market. Certain features – such as text search and ACID compliance – are dependant not on the core engine but on applications and add-ons. While it's true that there exists a plethora of well-made applications for MySQL, tracking them down can sometimes be a pain, and might cause some developers to simply choose an alternative which – while not as easily installed – offers more immediate functionality.

5) Developers May Find Some Of Its Limitations To Be Frustrating

Not surprisingly, MySQL isn't designed to do everything (nor should it be). The database isn't fully SQL-compliant, and tends to be limited in areas including data warehousing, fault tolerance, and performance diagnostics (among others). Developers may find this relative dearth of functionality frustrating, particularly if they're used to a more full-featured alternative.

Conclusion:

As we know SQL queries are much faster than NoSQL queries, another reason being that our database is not big but small. Hence, we choose MySQL to work with, also since it's relatively easy to deal with

6. Haproxy - Load Balancer

Drawbacks:

- No native failure detection or fault tolerance and no dynamic load re-balancing.
- No capability other than round-robin.
- No way to ensure connection to the same server twice, if required.
- DNS cannot tell if a server has become unavailable.
- Cannot take into account the unknown percentage of users who have DNS data cached, with varying amounts of Time to Live (TTL) left. So, when TTL times out, visitors may still be directed to the 'wrong' server.
- Load may not be evenly shared as DNS cannot tell how much load is present on the servers.
- Each server requires a public IP address.

Conclusion:

One of our aim to do this project was to learn and implement how load balancing works and is done. So, we did load balancing on server side. Since we are using Haproxy for this and we don't expect much traffic on our network, so load balancing would be much easily handled with very less errors.

7. Docker – Containerization and Container Orchestration

Drawbacks:

1) Missing features

There are a ton of feature requests are under progress, like container self-registration, and self-inspects, copying files from the host to the container, and many more.

2) Data in the container

There are times when a container goes down, so after that, it needs a backup and recovery strategy, although we have several solutions for that they are not automated or not very scalable yet.

3) Run applications as fast as a bare-metal serve

In comparison with the virtual machines, Docker containers have less overhead but not zero overhead. If we run, an application directly on a bare-metal server we get true bare-metal speed even without using containers or virtual machines. However, Containers don't run at bare-metal speeds.

4) Provide cross-platform compatibility

The one major issue is if an application designed to run in a Docker container on Windows, then it can't run on Linux or vice versa. However, Virtual machines are not subject to this limitation. So, this limitation makes Docker less attractive in some highly heterogeneous environments which are composed of both Windows and Linux servers.

5) Run applications with graphical interfaces

In general, Docker is designed for hosting applications which run on the command line. Though we have a few ways (like X11 forwarding) by which we can make it possible to run a graphical interface inside a Docker container, however, this is clunky. Hence we can say, for applications that require rich interfaces, Docker is not a good solution.

6) Solve all your security problems

In simple words, we need to evaluate the Docker-specific security risks and make sure we can handle them before moving workloads to Docker. The reason behind it is that Docker creates new security challenges like the difficulty of monitoring multiple moving pieces within a large-scale, dynamic Docker environment.

Conclusion:

Main goal in our project is to implement the use of docker, since it is the only service (PaaS, OS virtualization) which is connecting all the different elements of our app like the database with server, load balancer with server, app with load balancer etc.

There are other services like these available but docker is one of the better open source and fairly popular service available.

8. Nginx - Web Server

Drawbacks:

- Traffic congestion has always been a problem in the paradigm of C / S. When a large number of simultaneous clients send requests to the same server might cause many problems for this (to more customers, more problems for the server). On the contrary, P2P networks each node in the network server also makes more nodes, the better bandwidth you have.
- The paradigm of C / S Classic does not have the robustness of a network P2P. When a server is down, customer requests cannot be met. In most part, P2P networks resources are usually distributed across multiple nodes of the network. Although some out or abandon download, others may still end up getting data download on rest of the nodes in the network.
- The software and hardware of a server are usually very decisive. A regular computer hardware staff may not be able to serve a certain number of customers. Usually you need specific

software and hardware, especially on the server side, to meet the work . Of course, this will increase the cost.

- The client does not have the resources that may exist on the server. For example, if the application is a Web, we cannot write the hard disk of the client or print directly on printers without taking before the print preview window of the browser.

Conclusion:

There are different types of web servers available for different use cases and hence we are using Nginx for our project.

Further Implementation

9. Ansible – Configuration management

Drawbacks:

- OS restrictions: Ansible is OS dependent, code working for one OS will not necessarily work for others, apart from that, you can't use a windows box as your Management Server.
- Once a playbook is running, you can't add or remove hosts from inventory file, this can be crucial if your use case is such that you have a real time code which generates IP addresses where the same script is supposed to run. This holds true for variables too, i.e. you can't add variables to global vars file at runtime.
- Ansible makes fresh connection to remote hosts for each and every module it executes, the disadvantage of this is that making so many connection attempts makes it prone to connection failures. And one connection failure can put the execution of the entire play in jeopardy.
- If you compute some variable at run-time, and want to reuse it across hosts, Ansible does not provide a direct/simple way to do it, you need to go through a lot of pain to be able to do it. You need to literally copy that value to all desired hosts.
- Running Ansible script manually is one thing, but I assume anyone picking up ansible is aiming for more than that, i.e. invocation level automation(a piece of code invoking ansible playbook for you): ansible does support this, but the last time I checked, that library was available only for python, also here's a thing: couple of things that work for manually invoked ansible just don't work in invoked version, like `delegate_to` feature, verbose logging etc.
- Error reporting isn't great to say the least. If your playbook fails because of some syntax issue, ansible points you to the line number and adds a generic syntax issue msg. You need to figure out where exactly the issue is, is it a semi colon or parenthesis or space. This can be frustrating.

Conclusion:

Main reason to integrate Ansible is for configuration management and to learn the same.

10. Vagrant – Configuration management

Drawbacks:

- A modern, complex application probably uses a number of external dependencies, like various database servers, message queue daemons and whatnot. Installing these on every developer's machine individually for a local development environment can be quite a hassle. Using a virtual machine image, you can distribute a pre-configured environment easily.
- This does not mean that you set this image up once and cross your fingers that it'll never corrupt. It also doesn't mean that developers are writing their code inside the virtual machine. To create the virtual machine, you should create scripts which can set up a base OS to the desired state. Look at Ansible, Chef, Puppet, simple shell scripts and similar tools for that. These scripts can be used to set up a virtual machine, your production system, test systems etc all from the same source. Creating and distributing a complete image of a virtual machine is just a time saver, since such scripts could take a while to run. It also allows you to easily reset your environment to a known good state if you have messed things up. The VM image is not your one and only source though; if it corrupts it doesn't matter, since it doesn't hold any data or code and since you can simply recreate it.

Conclusion:

Using Vagrant is not feasible, since it requires a lot of space and is generally used for complicated systems. Hence, it is more of a proposed addition to our project as a software performance amplifier.

7. Methodology and Introduction

In our project titled “Infrastructure as Code implementation to deploy a WebApp” main component is Docker, since only Docker is connecting all the different services to each other and running in such an environment that those services are only connected via a port and they exist in their own respective containers.

Docker

About:

Docker is a set of platform as a service products that uses OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels. All containers are run by a single operating-system kernel and are thus more lightweight than virtual machines.

OS Virtualization:

OS-level virtualization refers to an application paradigm in which the kernel allows conditions for multiple single user tasks. Such instances, called containers (Solaris, Docker), Zones (Solaris), private virtual servers (OpenVZ), subdivisions, virtual environments (VEs), virtual kernel (DragonFly BSD), or prisons (FreeBSD prison or a chroot prison), may look like virtual computers from the perspective of the systems in which they operate. A computer program running on a standard operating system can see all the resources (connected devices, files and folders, network shares, CPU power, equal hardware capacity) of that computer. However, applications operating within the container can only see the contents of the container and the devices provided in the container.

For programs that work like Unix, this feature can be seen as an improved implementation of the standard chroot method, which replaces the virtual root folder of the current operating system with its children. In addition to classification methods, the kernel often provides resource management to reduce the container workloads to other containers.

The term "container," while most popularly referring to OS-level virtualization systems, is sometimes ambiguously used to refer to fuller virtual machine environments operating in varying degrees of concert with the host OS, e.g. Microsoft's "Hyper-V Containers."

Working:

- Docker can package an application and its dependencies in a virtual container that can run on any Linux server. This helps provide flexibility and portability enabling the application to be run in various locations, whether on-premises, in a public cloud, or in a private cloud. Docker uses the resource isolation features of the Linux kernel (such as cgroups and kernel namespaces) and a union-capable file system (such as OverlayFS) to allow containers to run within a single Linux instance, avoiding the overhead of starting and maintaining virtual machines. Because Docker containers are lightweight, a single server or virtual machine can run several containers simultaneously.[33] A 2018 analysis found that a typical Docker use case involves running eight containers per host, but that a quarter of analyzed organizations run 18 or more per host.
- The Linux kernel's support for namespaces mostly isolates an application's view of the operating environment, including process trees, network, user IDs and mounted file systems, while the kernel's cgroups provide resource limiting for memory and CPU. Since version 0.9, Docker includes its own component (called "libcontainer") to directly use virtualization facilities provided by the Linux kernel, in addition to using abstracted virtualization interfaces via libvirt, LXC and systemd-nspawn.
- Docker implements a high-level API to provide lightweight containers that run processes in isolation.

Components:

The Docker software as a service offering consists of three components:

- **Software:** The Docker daemon, called `dockerd`, is a persistent process that manages Docker containers and handles container objects. The daemon listens for requests sent via the Docker Engine API. The Docker client program, called `docker`, provides a command-line interface that allows users to interact with Docker daemons.
- **Objects:** Docker objects are various entities used to assemble an application in Docker. The main classes of Docker objects are images, containers, and services.

A Docker container is a standardized, encapsulated environment that runs applications. A container is managed using the Docker API or CLI.

A Docker image is a read-only template used to build containers. Images are used to store and ship applications.

A Docker service allows containers to be scaled across multiple Docker daemons. The result is known as a swarm, a set of cooperating daemons that communicate through the Docker API.

- **Registries:** A Docker registry is a repository for Docker images. Docker clients connect to registries to download ("pull") images for use or upload ("push") images that they have built.

Registries can be public or private. Two main public registries are Docker Hub and Docker Cloud. Docker Hub is the default registry where Docker looks for images. Docker registries also allow the creation of notifications based on events.

Tools:

Docker Compose is a tool for defining and running multi-container Docker applications. It uses YAML files to configure the application's services and performs the creation and start-up process of all the containers with a single command. The docker-compose CLI utility allows users to run commands on multiple containers at once, for example, building images, scaling containers, running containers that were stopped, and more. Commands related to image manipulation, or user-interactive options, are not relevant in Docker Compose because they address one container. The docker-compose.yml file is used to define an application's services and includes various configuration options. For example, the build option defines configuration options such as the Dockerfile path, the command option allows one to override default Docker commands, and more.

Implementation of Docker and Docker-compose in our project:

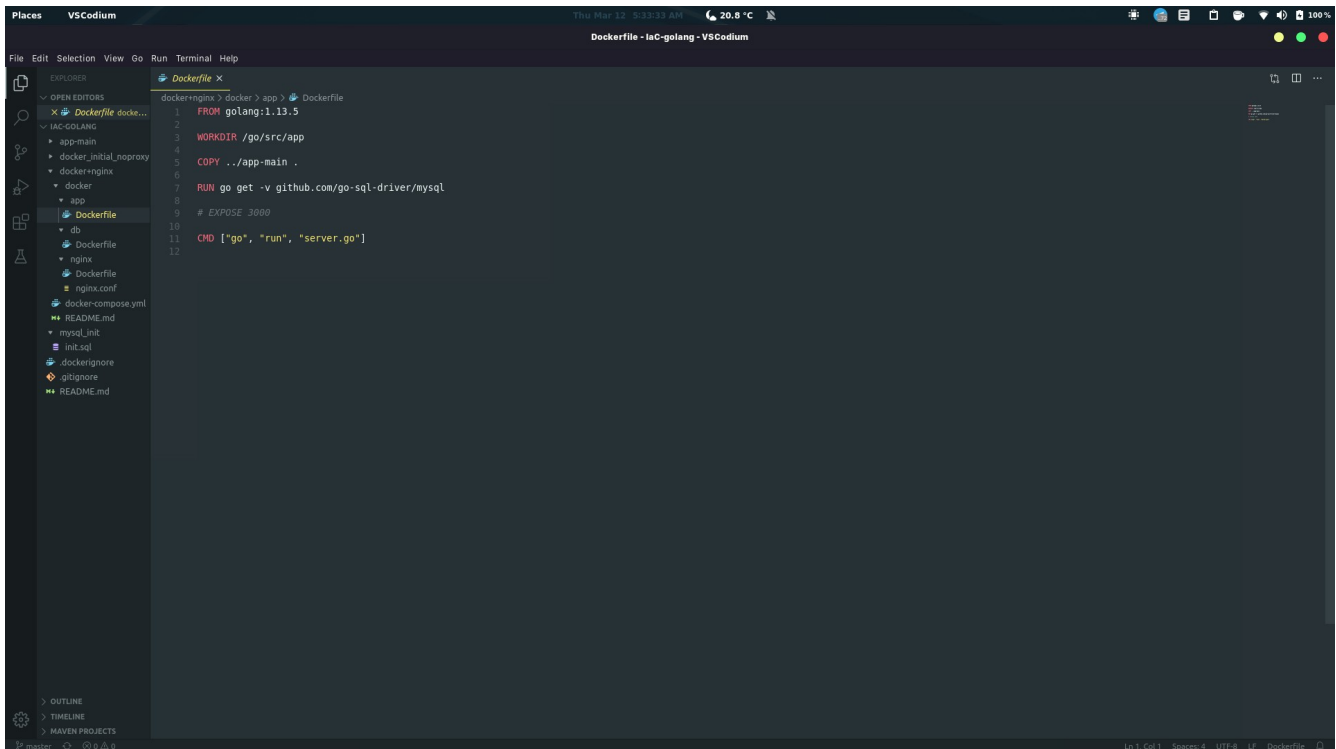


Fig7: Dockerfile for main app

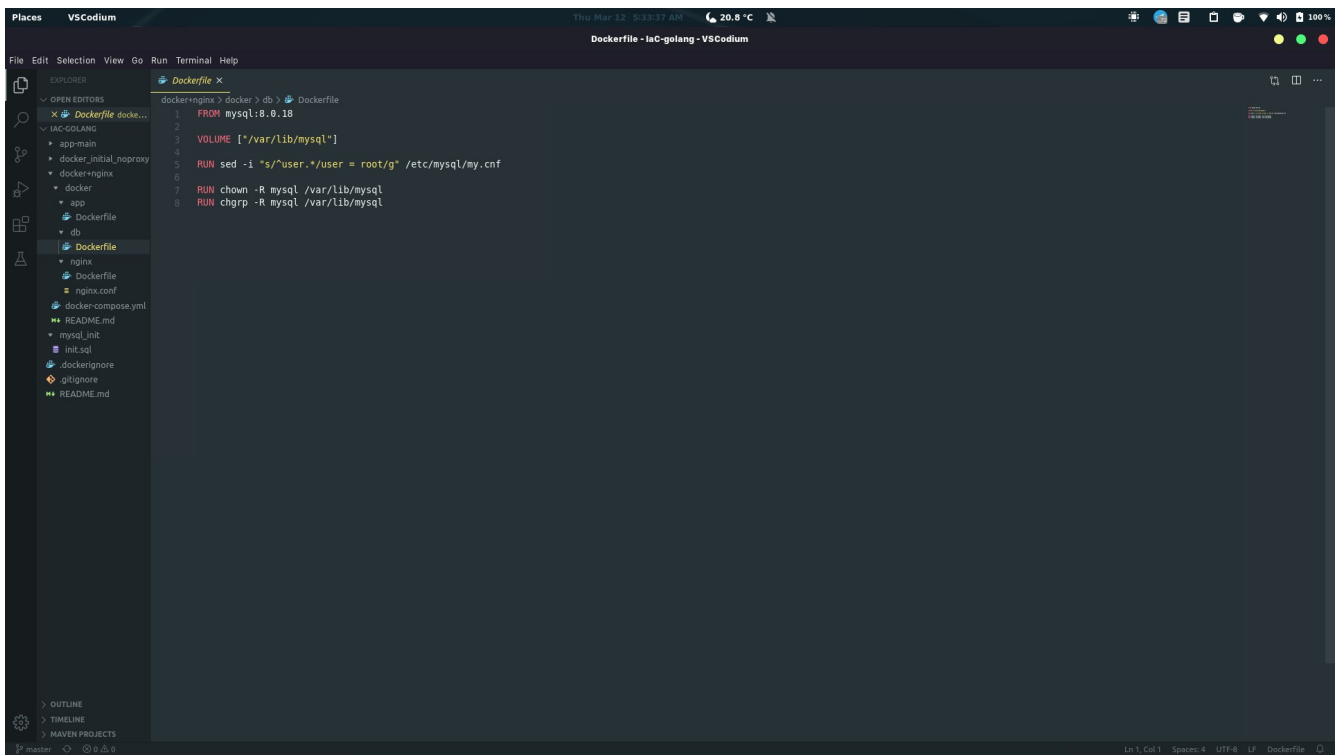


Fig8: Dockerfile for database

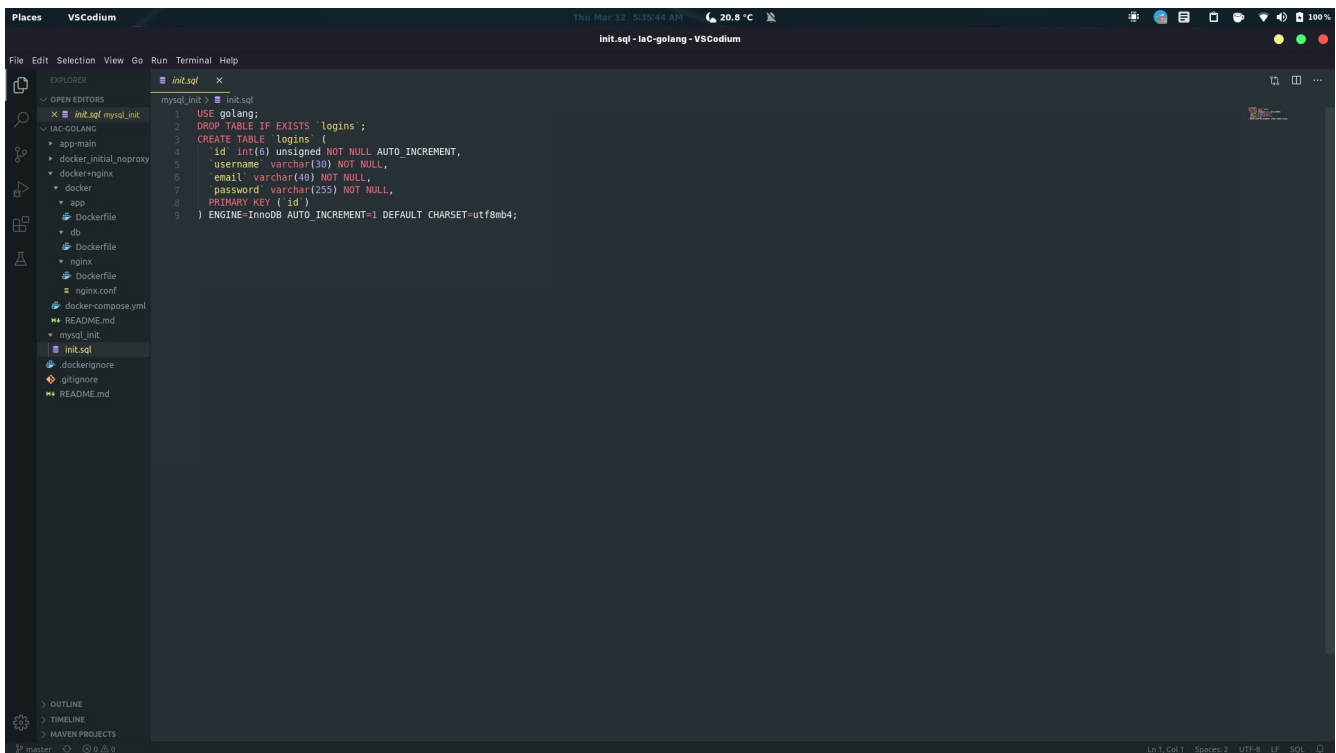
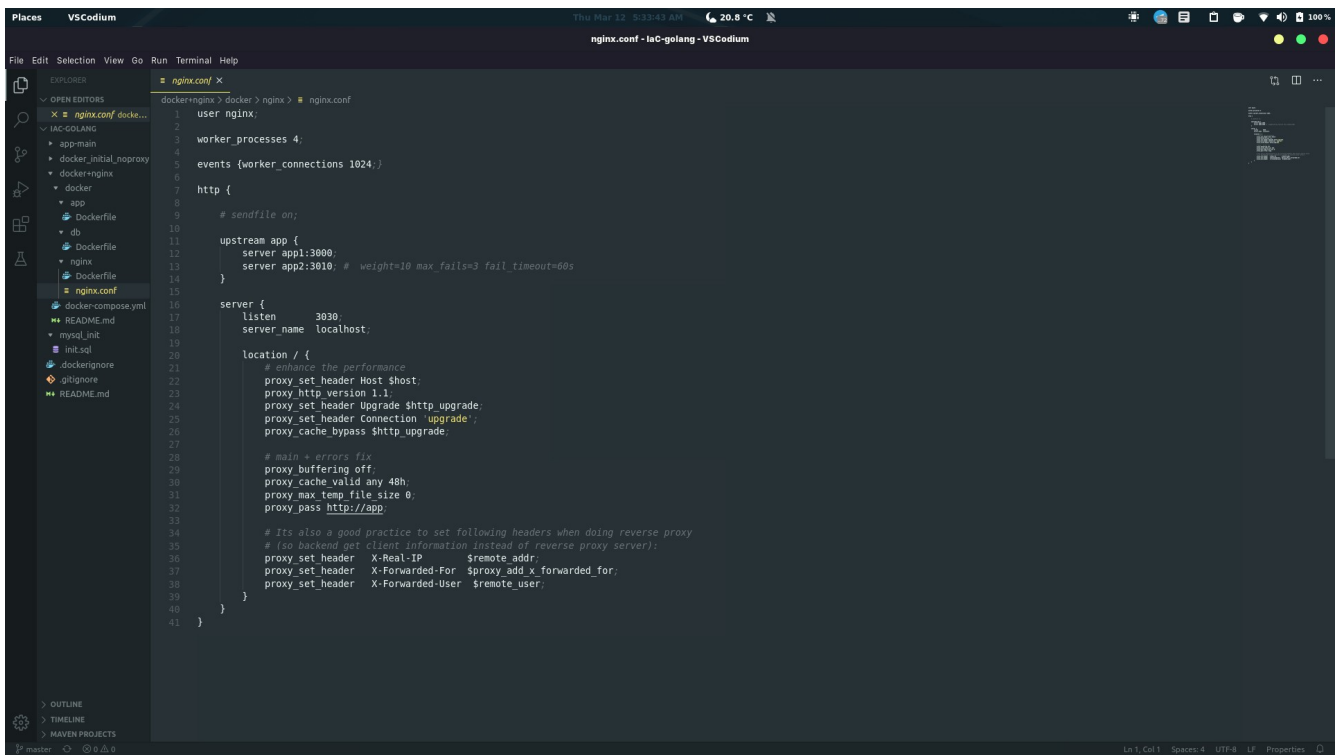


Fig9: Database – MySQL

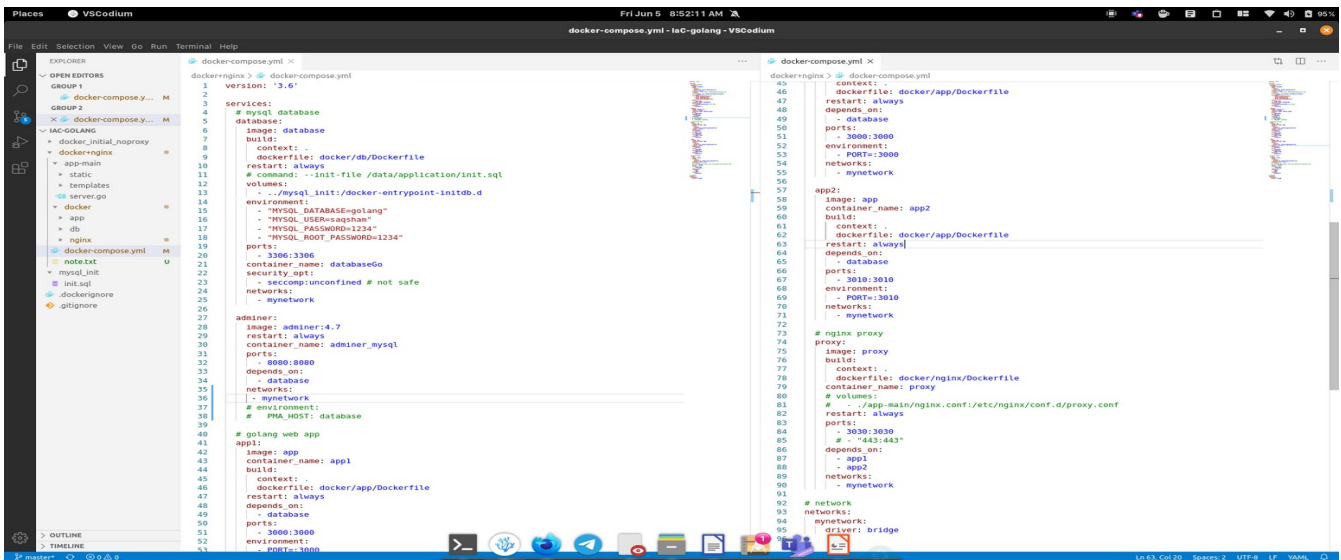


```

1 user nginx;
2
3 worker_processes 4;
4
5 events {worker_connections 1024;}
6
7 http {
8     # sendfile on;
9
10    upstream app {
11        server app1:3000;
12        server app2:3010; # weight=10 max_fails=3 fail_timeout=60s
13    }
14
15    server {
16        listen 3030;
17        server_name localhost;
18
19        location / {
20            # enhance the performance
21            proxy_set_header Host $host;
22            proxy_http_version 1.1;
23            proxy_set_header Upgrade $http_upgrade;
24            proxy_set_header Connection 'upgrade';
25            proxy_cache_bypass $http_upgrade;
26
27            # main + errors fix
28            proxy_buffering off;
29            proxy_cache valid any 48h;
30            proxy_max_temp_file_size 0;
31            proxy_pass http://app;
32
33            # Its also a good practice to set following headers when doing reverse proxy
34            # (so backend get Client Information instead of reverse proxy server);
35            proxy_set_header X-Real-IP $remote_addr;
36            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
37            proxy_set_header X-Forwarded-User $remote_user;
38        }
39    }
40
41 }

```

Fig10: Nginx Webserver config file



```

1 version: '3.6'
2
3 services:
4   # mysql database
5   database:
6     image: database
7     build:
8       context: .
9       dockerfile: docker/db/Dockerfile
10    restart: always
11    # command: --init-file /data/application/init.sql
12    volumes:
13      - ./mysql_init/docker-entrypoint-initdb.d
14    environment:
15      - MYSQL_DATABASE=golang
16      - MYSQL_USER=admin
17      - MYSQL_PASSWORD=1234
18      - MYSQL_ROOT_PASSWORD=1234
19    ports:
20      - 3306:3306
21    container_name: databaseGo
22    security_opt:
23      - seccomp:unconfined # not safe
24    networks:
25      - mynetwork
26
27   adminer:
28     image: adminer:4.7
29     restart: always
30     container_name: adminer_mysql
31     ports:
32       - 8080:8080
33     depends_on:
34       - database
35     networks:
36       - mynetwork
37     # environment:
38     #   PMA_HOST: database
39
40   # golang web app
41   app1:
42     image: app
43     container_name: app1
44     build:
45       context: .
46       dockerfile: docker/app/Dockerfile
47     restart: always
48     depends_on:
49       - database
50     ports:
51       - 3000:3000
52     environment:
53
54   app2:
55     image: app
56     container_name: app2
57     build:
58       context: .
59       dockerfile: docker/app/Dockerfile
60     restart: always
61     depends_on:
62       - database
63     ports:
64       - 3010:3010
65     environment:
66       - PORT=3010
67     networks:
68       - mynetwork
69
70   # nginx proxy
71   proxy:
72     image: proxy
73     build:
74       context: .
75       dockerfile: docker/nginx/Dockerfile
76     container_name: proxy
77     # volumes:
78     #   - ./app-main/nginx.conf:/etc/nginx/conf.d/proxy.conf
79     restart: always
80     ports:
81       - 3030:3030
82       - 443:443
83     depends_on:
84       - app1
85       - app2
86     networks:
87       - mynetwork
88
89   # network
90   networks:
91     mynetwork:
92       driver: bridge

```

Fig 11: Docker-compose.yml file for the whole application to load balance and scaling on two ports

```

server.go
105 email := r.FormValue("email")
106 password := r.FormValue("password")
107 insForm, err := db.Prepare("INSERT INTO logins (username, email, password)")
108 if err != nil {
109     panic(err.Error())
110 }
111 // defer insForm
112 insForm.Exec(username, email, password)
113 // log.Println("INSERT: username: " + username + " | email: " + email)
114 // fmt.Println("1 row inserted")
115 http.Redirect(w, r, "/login", 301)
116 }
117 }
118
119 func login(w http.ResponseWriter, r *http.Request) {
120     // fmt.Println("method:", r.Method) // get request method
121     if r.Method == "GET" {
122         t, _ := template.ParseFiles("templates/login.html")
123         t.Execute(w, nil)
124     } else {
125         // db work
126         r.ParseForm()
127         username := r.FormValue("username")
128         password := r.FormValue("password")
129         db := dbConn()
130         defer db.Close()
131         selDB, err := db.Query("SELECT password FROM logins WHERE username=?", username)
132         if err != nil {
133             panic(err.Error())
134         }
135         // log.Fatal(err)
136     }
137     logingo := logins{}
138     for selDB.Next() {
139         var password1 string
140         err = selDB.Scan(&password1)
141         if (err != nil) {
142             panic(err.Error())
143         }
144         logingo.password = password1
145         if (logingo.password != password) {
146             http.Redirect(w, r, "/login", 301)
147             break
148         } else {
149             http.Redirect(w, r, "/home", 301)
150             break
151         }
152     }
153 }
154
155 func main() {
156     port := os.Getenv("PORT")
157
158     fs := http.FileServer(http.Dir("static"))
159     http.Handle("/static/", http.StripPrefix("/static/", fs))
160     http.HandleFunc("/", sayhelloName)
161     http.HandleFunc("/signup", signup)
162     http.HandleFunc("/login", login)
163     // http.HandleFunc("/.js/css", assets) // static files
164     listener, err := net.Listen("tcp", port)
165     if err != nil {
166         panic(err)
167     }
168
169     fmt.Println("Using port:", listener.Addr().(*net.TCPAddr).Port)
170     panic(http.Serve(listener, nil))
171
172     // err := http.ListenAndServe(":3000", nil) // port setup
173     // if err != nil {
174     //     log.Fatal("ListenAndServe: ", err)
175     // } else {
176     //     // log.Println("Server Listening on Port: %v", port)
177     // }
178 }

```

Fig 12: Backend source code

```

Building app2
Step 1/5 : FROM golang:1.13.5
--> ed081345a3da
Step 2/5 : WORKDIR /go/src/app
--> Using cache
--> 5e565adf9f8b
Step 3/5 : COPY /app-main .
--> Using cache
--> 808233aadd11
Step 4/5 : RUN go get -v github.com/go-sql-driver/mysql
--> Using cache
--> 948bab316836
Step 5/5 : CMD ["go", "run", "server.go"]
--> Using cache
--> 80b9098b16c6

Successfully built 80b9098b16c6
Successfully tagged app:latest
Building proxy
Step 1/4 : FROM nginx:1.16
--> dfcfd8e9a5d3
Step 2/4 : MAINTAINER Saksham Saini
--> Using cache
--> 9f26d64db359
Step 3/4 : RUN rm /etc/nginx/conf.d/default.conf
--> Using cache
--> a902a5543d49
Step 4/4 : COPY docker/nginx/nginx.conf /etc/nginx/nginx.conf
--> Using cache
--> f140d2e81748

Successfully built f140d2e81748
Successfully tagged proxy:latest
Starting databaseGo ... done
Starting app2 ... done
Starting appl ... done
Starting adminer_mysql ... done
Starting proxy ... done
Attaching to databaseGo, app2, adminer_mysql, appl, proxy
app2 | Using port: 3010
adminer_mysql | [Fri Jun  5 03:33:40 2020] PHP 7.4.6 Development Server (http://[::]:8080) started
databaseGo | 2020-06-05 03:33:37+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.18-1debian9 started.
databaseGo | 2020-06-05 03:33:37+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
appl | Using port: 3000
databaseGo | 2020-06-05 03:33:37+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.18-1debian9 started.
databaseGo | 2020-06-05T03:33:38.007959Z 0 [Warning] [MY-011070] [Server] 'Disabling symbolic links using --skip-symbolic-l
inks (or equivalent) is the default. Consider not using this option as it' is deprecated and will be removed in a future rel
ease.
databaseGo | 2020-06-05T03:33:38.008094Z 0 [System] [MY-010116] [Server] /usr/sbin/mysqld (mysqld 8.0.18) starting as proce
ss 1
databaseGo | 2020-06-05T03:33:42.645193Z 0 [Warning] [MY-010068] [Server] CA certificate ca.pem is self signed.
databaseGo | 2020-06-05T03:33:42.901158Z 0 [Warning] [MY-011810] [Server] Insecure configuration for --pid-file: Location '
/var/run/mysqld' in the path is accessible to all OS users. Consider choosing a different directory.
databaseGo | 2020-06-05T03:33:42.930560Z 0 [System] [MY-010931] [Server] /usr/sbin/mysqld: ready for connections. Version:
'8.0.18' socket: '/var/run/mysqld/mysqld.sock' port: 3306 MySQL Community Server - GPL.
databaseGo | 2020-06-05T03:33:43.272225Z 0 [System] [MY-011323] [Server] X Plugin ready for connections. Socket: '/var/run/
mysqld/mysqlx.sock' bind-address: '::' port: 33060
[]

```

Fig 13: Building from Docker-compose using `docker-compose up --build`

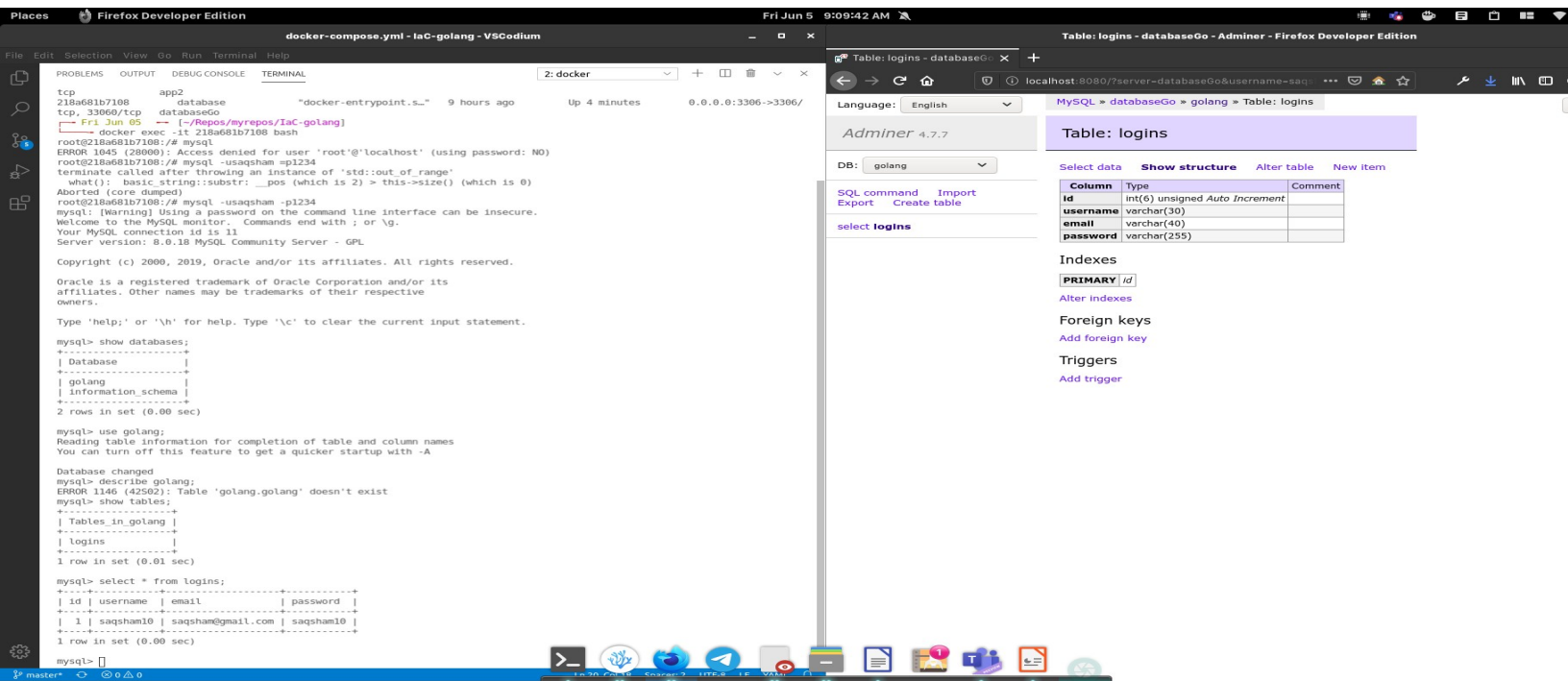


Fig 14: Database check

```

Fri Jun 05 10:00:00 AM [~/Repos/myrepos/IaC-golang]
docker exec -it beab7b027ce3 bash
root@beab7b027ce3:/# nginx -h
nginx version: nginx/1.16.1
Usage: nginx [-?hvVtTq -s signal] [-c filename] [-p prefix] [-g directives]

Options:
  -?, -h      : this help
  -v          : show version and exit
  -V          : show version and configure options then exit
  -t          : test configuration and exit
  -T          : test configuration, dump it and exit
  -q          : suppress non-error messages during configuration testing
  -s signal   : send signal to a master process: stop, quit, reopen, reload
  -p prefix   : set prefix path (default: /etc/nginx/)
  -c filename : set configuration file (default: /etc/nginx/nginx.conf)
  -g directives : set global directives out of configuration file

root@beab7b027ce3:/# nginx -v
nginx version: nginx/1.16.1
root@beab7b027ce3:/# nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
root@beab7b027ce3:/#

```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
beab7b027ce3	proxy	"nginx -g 'daemon of..."	About a minute ago	Up About a minute	80/tcp, 0.0.0.0:30
30->3030/tcp	proxy	"entrypoint.sh docke..."	About an hour ago	Up About a minute	0.0.0.0:8080->8080
294f505bfc94	adminer:4.7	"entrypoint.sh docke..."	About an hour ago	Up About a minute	0.0.0.0:8080->8080
/tcp	adminer_mysql	"go run server.go"	9 hours ago	Up About a minute	0.0.0.0:3000->3000
7e9de97edee2	app	"go run server.go"	9 hours ago	Up About a minute	0.0.0.0:3010->3010
/tcp	app1	"go run server.go"	9 hours ago	Up About a minute	0.0.0.0:3010->3010
582e192f8aeb	app	"go run server.go"	9 hours ago	Up About a minute	0.0.0.0:3010->3010
/tcp	app2	"go run server.go"	9 hours ago	Up About a minute	0.0.0.0:3010->3010
218a681b7108	database	"docker-entrypoint.s..."	9 hours ago	Up About a minute	0.0.0.0:3306->3306
/tcp, 33060/tcp	databaseGo	"docker-entrypoint.s..."	9 hours ago	Up About a minute	0.0.0.0:3306->3306

Fig 15: Nginx test check and docker containers

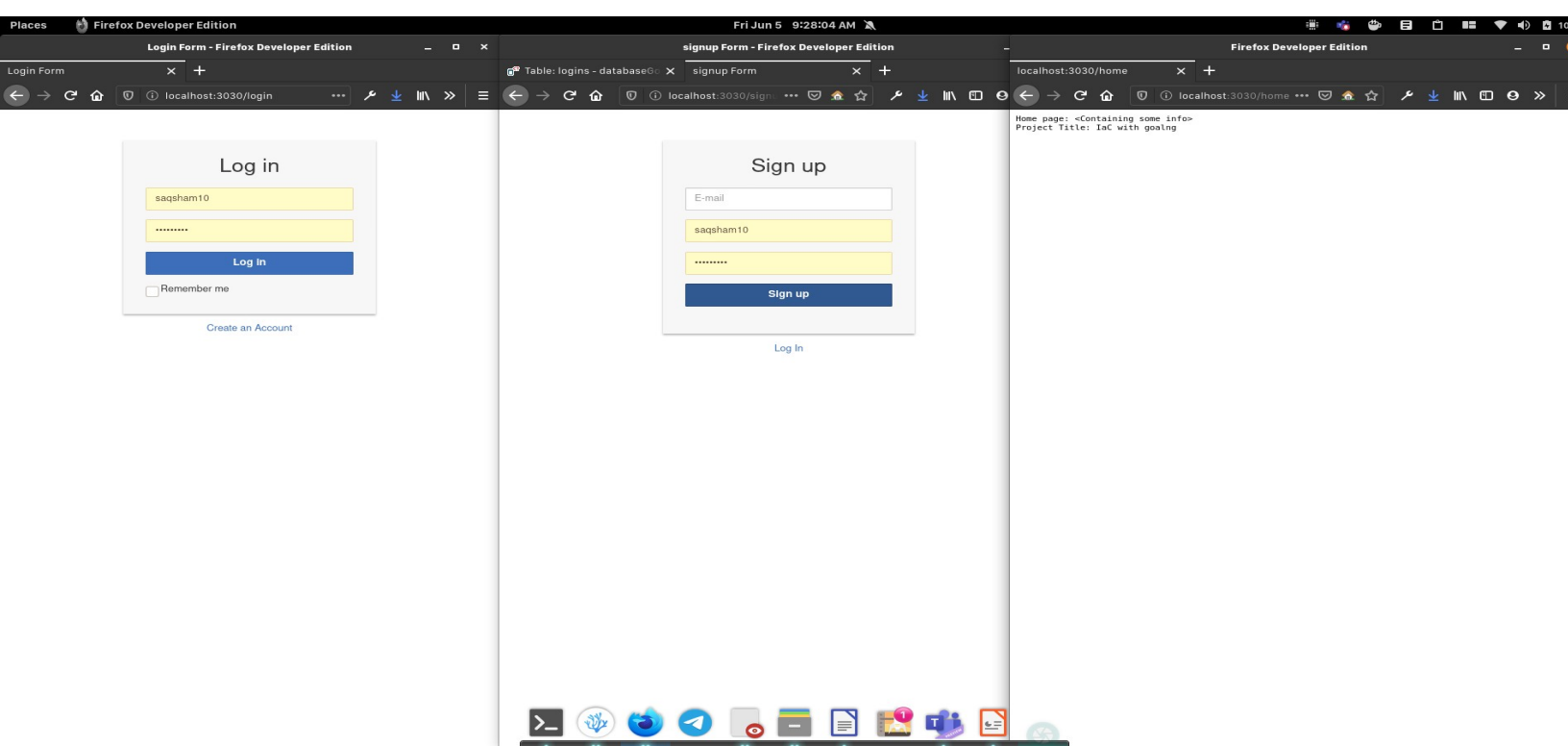


Fig 16: Our Application running on a reverse proxy localhost:3030

```
Fri Jun 05 [~/Repos/myrepos/IaC-golang]
docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
beab7b027ce3	proxy	"nginx -g 'daemon of..."	20 minutes ago	Up 20 minutes	80/tcp, 0.0.0.0:3030->3030/tcp	proxy
294f505bfc94	adminer:4.7	"entrypoint.sh docke..."	2 hours ago	Up 20 minutes	0.0.0.0:8080->8080/tcp	adminer_mysql
7e9de97edee2	app	"go run server.go"	9 hours ago	Up 20 minutes	0.0.0.0:3000->3000/tcp	app1
582e192f8aeb	app	"go run server.go"	9 hours ago	Up 20 minutes	0.0.0.0:3010->3010/tcp	app2
218a681b7108	database	"docker-entrypoint.s..."	9 hours ago	Up 20 minutes	0.0.0.0:3306->3306/tcp, 33060/tcp	databaseGo

Fig 17: Docker processes and other information


```

proxy | 172.24.0.1 - - [05/Jun/2020:03:51:10 +0000] "GET /favicon.ico HTTP/1.1" 499 0 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:77.0) Gecko/20100101 Firefox/77.0"
proxy | 172.24.0.1 - - [05/Jun/2020:03:51:14 +0000] "GET / HTTP/1.1" 200 64 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:77.0) Gecko/20100101 Firefox/77.0"
proxy | 172.24.0.1 - - [05/Jun/2020:03:51:25 +0000] "GET /signup HTTP/1.1" 200 1836 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:77.0) Gecko/20100101 Firefox/77.0"
proxy | 172.24.0.1 - - [05/Jun/2020:03:51:25 +0000] "GET /static/js/bootstrap.min.js HTTP/1.1" 304 0 "http://localhost:3030/signup" "Mozilla/5.0 (X11; Linux x86_64; rv:77.0) Gecko/20100101 Firefox/77.0"
proxy | 172.24.0.1 - - [05/Jun/2020:03:51:25 +0000] "GET /static/js/jquery.min.js HTTP/1.1" 304 0 "http://localhost:3030/signup" "Mozilla/5.0 (X11; Linux x86_64; rv:77.0) Gecko/20100101 Firefox/77.0"
proxy | 172.24.0.1 - - [05/Jun/2020:03:51:25 +0000] "GET /static/css/bootstrap.min.css HTTP/1.1" 200 150063 "http://localhost:3030/signup" "Mozilla/5.0 (X11; Linux x86_64; rv:77.0) Gecko/20100101 Firefox/77.0"
proxy | 172.24.0.1 - - [05/Jun/2020:03:51:30 +0000] "POST /signup HTTP/1.1" 301 0 "http://localhost:3030/signup" "Mozilla/5.0 (X11; Linux x86_64; rv:77.0) Gecko/20100101 Firefox/77.0"
proxy | 172.24.0.1 - - [05/Jun/2020:03:51:30 +0000] "GET /login HTTP/1.1" 200 1853 "http://localhost:3030/signup" "Mozilla/5.0 (X11; Linux x86_64; rv:77.0) Gecko/20100101 Firefox/77.0"
proxy | 172.24.0.1 - - [05/Jun/2020:03:51:30 +0000] "GET /static/css/bootstrap.min.css HTTP/1.1" 304 0 "http://localhost:3030/login" "Mozilla/5.0 (X11; Linux x86_64; rv:77.0) Gecko/20100101 Firefox/77.0"
proxy | 172.24.0.1 - - [05/Jun/2020:03:51:33 +0000] "POST /login HTTP/1.1" 301 0 "http://localhost:3030/login" "Mozilla/5.0 (X11; Linux x86_64; rv:77.0) Gecko/20100101 Firefox/77.0"
proxy | 172.24.0.1 - - [05/Jun/2020:03:51:34 +0000] "GET /home HTTP/1.1" 200 64 "http://localhost:3030/login" "Mozilla/5.0 (X11; Linux x86_64; rv:77.0) Gecko/20100101 Firefox/77.0"
proxy | 172.24.0.1 - - [05/Jun/2020:03:56:38 +0000] "GET / HTTP/1.1" 200 64 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:77.0) Gecko/20100101 Firefox/77.0"
proxy | 172.24.0.1 - - [05/Jun/2020:03:56:48 +0000] "GET /signup HTTP/1.1" 200 1836 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:77.0) Gecko/20100101 Firefox/77.0"
proxy | 172.24.0.1 - - [05/Jun/2020:03:56:59 +0000] "GET /login HTTP/1.1" 200 1853 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:77.0) Gecko/20100101 Firefox/77.0"
proxy | 172.24.0.1 - - [05/Jun/2020:03:57:38 +0000] "GET /home HTTP/1.1" 200 64 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:77.0) Gecko/20100101 Firefox/77.0"

```

Fig 18: Showcasing that our reverse proxy is working fine

8. Conclusion

8.1 Derived from Literature Survey

→ What we inferred from the literature survey is that we have considered the utilization of Infrastructure-as-Code, which empower us to promote our comprehension of its ideology and use in the vast of Virtualization, to be a key attribute of enabling best practices in DevOps – Developers become more involved in defining configuration and Ops teams get involved earlier in the development process. . We likewise comprehended the possibility of Virtualization in terms of Paging Tables and the Aging Factor of a framework utilized for Virtualization itself.

→ We saw that Virtualization goes about as a key to Cloud Computing and also attempted to identify the upsides of Cloud Computing. We additionally examined the significant utilization of the applications like Hadoop, Haproxy, Nginx etc. and not to mention the technologies or services like Dockers, Kubernetes, Ansible and others (may/may not be previously mentioned in this report).

8.2 Derived after Completing this Project

→ The main goal of this project was to basically implement something related to the field of Virtualization, and Virtualization itself being so vast there was a long list of technologies and services available as open-source and pay-to-use as well. So, we decided do something which may open our eyes to this vast field, hence we choose to develop and deploy a simple Web Application using Docker and to further learn more we implemented other services such as a Web Server – Nginx and a Load Balancer – Haproxy since it is relatively easy to configure them with Docker or so we inferred after researching the same on google and reading different articles on medium etc..

→ The working, components etc. of Virtualization are briefly discussed in the Experimental Setup section, where we have explained all about Docker and OS level Virtualization.

→ Now about DevOps, by definition it is the union of people, process, and products to enable continuous delivery of value to the end users. The contraction of “Dev” and “Ops” refers to the replacing siloed Development and Operations to create multidisciplinary teams that now together with shared and efficient practices and tools. Essential DevOps practices include agile planning, continuous integration, continuous delivery, and monitoring of applications.

→ What we learned after researching about DevOps and doing this project was that it is a combination of different practices and there is no one or maybe the only right way to follow or do it. Although the developer should have a DevOps mindset while on a Real World project.

→ Further we learned that there are many job opportunities as well, since it is an interesting field from my perspective that was good to know.

→ Now about Infrastructure as Code (IaC), by definition it is the management of infrastructure (networks, virtual machines, load balancers, and connection topology) in a descriptive model using same versioning as DevOps team uses for source code, is a key DevOps practice and is used in conjunction with continuous delivery.

→ We learned after researching about IaC and doing this project was that most of it is comprised of configuration files and downloading the related images and specifying versions. For instance if we want to run a MySQL database for an application, then we only need to write “FROM mysql:latest” in the Dockerfile, when we’ll run the file it will search for latest MySQL version otherwise download it (pulls the required repo from the dockerhub). There is no need to install MySQL on the remote machine, also it saves space.

→ In the future, if we make or if someone will make an application which is larger in size and have a sufficient user-base then we can use Vagrant to scale this project, Firebase to store our useful data, and we can host the application using Netlify, Firebase and or Heroku. Basically deploy a complete cloud web application. Other than doing the above mentioned stuff we can also deploy the whole app using only Amazon Web Services (AWS).

For example:

- Using AWS Elastic Beanstalk to make the application, it supports languages and services like Java, .NET, Node.JS, Python, Ruby, Docker, and Go. We will first configure the EB application, then setup our EB environment where our application will be launched into.
- Amazon DynamoDB, usually used by mobile, web, gaming, ad tech, IoT, and other applications that need low-latency data access at any scale.
- Using Amazon Route 53 to get a domain for our website. we will then connect that domain name through the Domain Name System (DNS) to a currently running EC2 instance (such as a WebApp, or website running WordPress, Apache, NGINX, IIS, or other Website platform)
- Using Amazon CloudWatch to collect metrics and logs from the operating systems for our EC2 instances.

9. Appendix

Directory Structure:

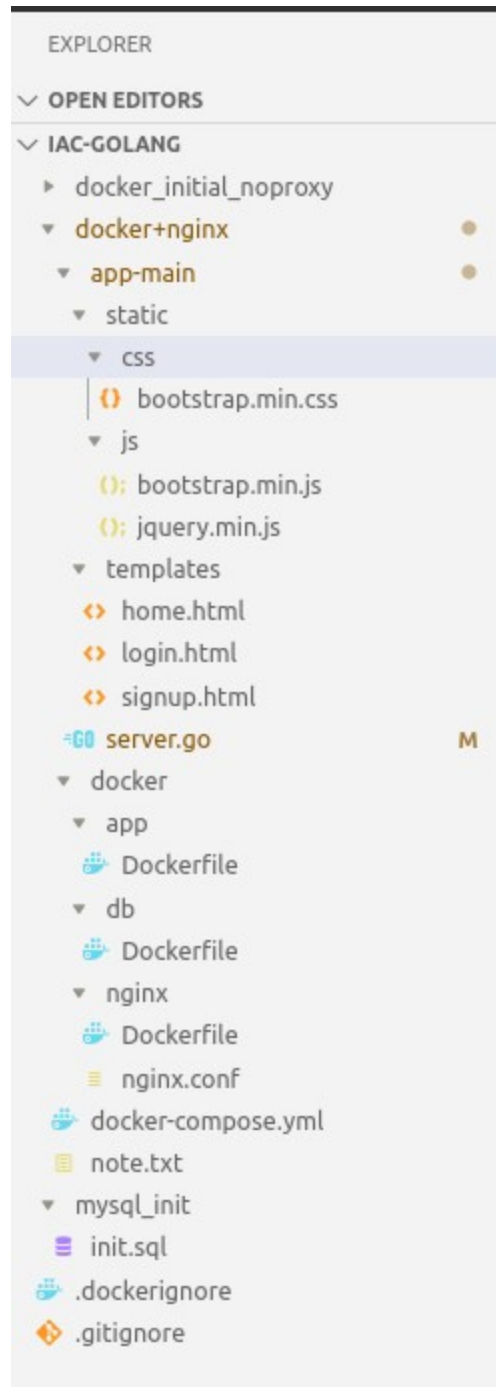


Fig 19: Directory Structure of the Project

Backend Code:

```

package main

import (
    "fmt"
    "html/template"
    "net/http"
    "os"
    "crypto/md5"
    "net"
    "time"
    "io"
    "strconv"
    "database/sql"
    _ "github.com/go-sql-driver/mysql"
)

type logins struct {
    id          int
    username    string
    email       string
    password    string
}

func dbConn() (db *sql.DB) {
    dbDriver := "mysql"
    dbUser := "saqsham"
    dbPass := "1234"
    dbName := "golang"
    db, err := sql.Open(dbDriver, dbUser+":"+dbPass+"@tcp(databaseGo)/"+dbName)
    if err != nil {
        panic(err.Error())
    }
    return db
}

func sayhelloname(w http.ResponseWriter, r *http.Request) {
    r.ParseForm() // Parse url params passed
    fmt.Fprintf(w, "Home page: <Containing some info>\nProject Title: IaC with goalng") // writing data to response
}

func signup(w http.ResponseWriter, r *http.Request) {
    if r.Method == "GET" {
        crutime := time.Now().Unix()
        h := md5.New()
        io.WriteString(h, strconv.FormatInt(crutime, 10))
        token := fmt.Sprintf("%x", h.Sum(nil))

        t, _ := template.ParseFiles("templates/signup.html")
        t.Execute(w, token)
    } else {
        // POST data
        r.ParseForm()
        token := r.Form.Get("token")
        if token != "" {

```

```

    }
    // validation
    username := r.Form["username"]
    if len(username) == 0 {
        t, _ := template.ParseFiles("templates/signup.html")
        t.Execute(w, nil)
    }
    // db work
    db := dbConn()
    defer db.Close()
    if r.Method == "POST" {
        username := r.FormValue("username")
        email := r.FormValue("email")
        password := r.FormValue("password")
        insForm, err := db.Prepare("INSERT INTO logins (username, email,
password) VALUES (?, ?, ?)")
        if err != nil {
            panic(err.Error())
        }
        insForm.Exec(username, email, password)
        http.Redirect(w, r, "/login", 301)
    }
}

func login(w http.ResponseWriter, r *http.Request) {
    if r.Method == "GET" {
        t, _ := template.ParseFiles("templates/login.html")
        t.Execute(w, nil)
    } else {
        // db work
        r.ParseForm()
        username := r.FormValue("username")
        password := r.FormValue("password")
        db := dbConn()
        defer db.Close()
        selDB, err := db.Query("SELECT password FROM logins WHERE username=?",
username)
        if err != nil {
            panic(err.Error())
        }
        logingo := logins{}
        for selDB.Next() {
            var password1 string
            err = selDB.Scan(&password1)
            if (err != nil) {
                panic(err.Error())
            }
            logingo.password = password1
            if (logingo.password != password) {
                http.Redirect(w, r, "/login", 301)
                break
            } else {
                http.Redirect(w, r, "/home", 301)
                break
            }
        }
    }
}

```

```

    }
}

func main() {
    port := os.Getenv("PORT")
    fs := http.FileServer(http.Dir("static"))
    http.Handle("/static/", http.StripPrefix("/static/", fs))
    http.HandleFunc("/", sayhelloname)
    http.HandleFunc("/signup", signup)
    http.HandleFunc("/login", login)
    listener, err := net.Listen("tcp", port)
    if err != nil {
        panic(err)
    }

    fmt.Println("Using port:", listener.Addr().(*net.TCPAddr).Port)

    panic(http.Serve(listener, nil))
}

```

Database initialization:

```

USE golang;
DROP TABLE IF EXISTS `logins`;
CREATE TABLE `logins` (
  `id` int(6) unsigned NOT NULL AUTO_INCREMENT,
  `username` varchar(30) NOT NULL,
  `email` varchar(40) NOT NULL,
  `password` varchar(255) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8mb4;

```

Nginx configuration file:

```

user nginx;

worker_processes 4;

events { worker_connections 1024; }

http {
    # sendfile on;
    upstream myapp {
        server app1:3000 weight=2;
        server app2:3010 weight=4; # weight=10 max_fails=3 fail_timeout=60s
    }

    server {
        listen 3030;
        server_name localhost;

        location / {
            # enhance the performance
            proxy_set_header Host $host;

```

```

    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection 'upgrade';
    proxy_cache_bypass $http_upgrade;

    # main + errors fix
    proxy_buffering off;
    proxy_cache_valid any 48h;
    proxy_max_temp_file_size 0;
    proxy_pass http://myapp;

    # Its also a good practice to set following headers when doing reverse
proxy
    # (so backend get client information instead of reverse proxy server):
    proxy_set_header    X-Real-IP        $remote_addr;
    proxy_set_header    X-Forwarded-For  $proxy_add_x_forwarded_for;
    proxy_set_header    X-Forwarded-User $remote_user;
}

location /nginx_status {
    # Enable Nginx stats
    stub_status on;
    # Only allow access from your IP e.g 1.1.1.1 or localhost #
    allow 127.0.0.1;
    # Other request should be denied
    deny all;
}
}
}

```

Dockerfiles

For web application:

```

FROM golang:1.13.5

WORKDIR /go/src/app

COPY /app-main .

RUN go get -v github.com/go-sql-driver/mysql

CMD ["go", "run", "server.go"]

```

For database

```

FROM mysql:8.0.18

VOLUME ["/var/lib/mysql"]

RUN sed -i "s/^user.*/user = root/g" /etc/mysql/my.cnf

```

```
RUN chown -R mysql /var/lib/mysql
RUN chgrp -R mysql /var/lib/mysql
```

For Nginx

```
FROM nginx:1.16
```

```
MAINTAINER Saksham Saini
```

```
RUN rm /etc/nginx/conf.d/default.conf
```

```
COPY docker/nginx/nginx.conf /etc/nginx/nginx.conf
```

Docker Compose file :

```
version: '3.6'
```

```
services:
  # mysql database
  database:
    image: database
    build:
      context: .
      dockerfile: docker/db/Dockerfile
    restart: always
    # command: --init-file /data/application/init.sql
    volumes:
      - ../mysql_init:/docker-entrypoint-initdb.d
    environment:
      - "MYSQL_DATABASE=golang"
      - "MYSQL_USER=saksham"
      - "MYSQL_PASSWORD=1234"
      - "MYSQL_ROOT_PASSWORD=1234"
    ports:
      - 3306:3306
    container_name: databaseGo
    security_opt:
      - seccomp:unconfined # not safe
    networks:
      - mynetwork

  adminer:
    image: adminer:4.7
    restart: always
    container_name: adminer_mysql
    ports:
      - 8080:8080
    depends_on:
      - database
    networks:
      - mynetwork
    # environment:
    #   PMA_HOST: database

# golang web app
app1:
  image: app
```

```

container_name: app1
build:
  context: .
  dockerfile: docker/app/Dockerfile
restart: always
depends_on:
  - database
ports:
  - 3000:3000
environment:
  - PORT=:3000
networks:
  - mynetwork

```

```

app2:
  image: app
  container_name: app2
  build:
    context: .
    dockerfile: docker/app/Dockerfile
  restart: always
  depends_on:
    - database
  ports:
    - 3010:3010
  environment:
    - PORT=:3010
  networks:
    - mynetwork

```

```

# nginx proxy
proxy:
  image: proxy
  build:
    context: .
    dockerfile: docker/nginx/Dockerfile
  container_name: proxy
  # volumes:
  #   - ./app-main/nginx.conf:/etc/nginx/conf.d/proxy.conf
  restart: always
  ports:
    - 3030:3030
    # - "443:443"
  depends_on:
    - app1
    - app2
  networks:
    - mynetwork

```

```

# network
networks:
  mynetwork:
    driver: bridge

```