



VAASAN AMMATTIKORKEAKOULU  
UNIVERSITY OF APPLIED SCIENCES

Ho Hoang Phuc

AUTO SCALING INFRASTRUCTURE  
FOR FIT RESTAURANT  
WITH NGINX AND DOCKER

Technology and Communication

2019

## **ACKNOWLEDGEMENT**

Firstly, I am grateful to my parents. Their recommendations, endless love and continuous support to me are priceless, unique things in the world since I was a child till I start moving and taking care my own life.

Secondly, I deeply appreciate Bui Thi Yen Phuong for her mental and financial support, and Hien Nguyen for being a great team mate during the progress. Without the help of Phuong, I cannot be here today to continue my chosen study path. Without the collaboration of Hien, there would be no deep discussion, ideas, and thorough thoughts in different parts of the study.

Next, a special thanks to the owner of Fit restaurant for giving me this opportunity. This was a great chance for me to dive deeper into my career, gain experience and take responsibility as a part of my life adventure, especially my career.

Moreover, I would like to express my gratitude to Dr. Ghodrat Moghadampour for being a friendly teacher, a responsible supervisor and sympathy supporter during my good old days at Vaasa University of Applied Sciences (VAMK).

And lastly, I also would want to convey my appreciation to Dr. Seppo Mäkinen and Dr. Chao Gao for being great teachers, all the office assistants as well as staff at VAMK that I had ever met for all of their services.

Ho Hoang Phuc  
Helsinki, Finland  
27.11.2019

## ABSTRACT

Author	Ho Hoang Phuc
Title	Auto Scaling Infrastructure for Fit Restaurant with Nginx and Docker
Year	2019
Language	English
Pages	55
Name of Supervisor	Dr. Ghodrat Moghadampour

---

The thesis covered the infrastructure and back-end development for Ravintola Fit restaurant located in Jyväskylä. The aim of the study was to develop a good system where different parts of structure were bundled as standalone containers working together to serve the end-users whilst maintaining their functionalities, portability as well as flexibility. The most important point, the infrastructure can be scaled vertically or horizontally based on the need due to the spike in traffic or business expansion since they all hosted on cloud computing service. This guarantees the demands are efficiently served, both the hardware cost and the experience of the user.

The groundwork contained four parts, the orchestration, load balancer, back-end, and database. Every piece was built from scratch without the help from any content management system and bundles. Docker Swarm and Docker command line interface were used to construct and manage the containers within the orchestration. Traffics travel back and forth the system were instructed by Nginx thanks to the lightweight, stability and high performance as a web server. The back-end server inquired Postgres database through GraphQL application programming interface, gathered data to feed the front-end server via the REST application programming interface, and held the integration for third-party services at the same time.

The work was successfully delivered to the restaurant and the web application has been in production for some times since the requirements were met. It even went beyond the expectation of the owner in the perspective of performance, cost and design.

# CONTENTS

ACKNOWLEDGEMENT .....	2
ABSTRACT.....	3
1 INTRODUCTION .....	1
1.1 Background .....	1
1.2 Purpose .....	1
1.3 Thesis Overall Structure.....	2
2 REQUIREMENTS AND ANALYSIS .....	3
2.1 Requirements Collection .....	3
2.2 Requirements Analysis.....	4
3 SOLUTION TECHNOLOGIES AND ARCHITECTURE.....	6
3.1 Adopted Technologies.....	6
3.1.1 Cloud Computing .....	6
3.1.2 Platform as a Service.....	7
3.1.3 HTTP/2 over SSL/TLS with Let's Encrypt .....	8
3.1.4 REST API .....	12
3.1.5 PostgreSQL and PostGraphile .....	14
3.1.6 Express server .....	16
3.1.7 Nginx web server .....	18
3.2 Docker and Container Orchestration .....	19
3.2.1 Docker Image and Container .....	19
3.2.2 Docker Engine.....	21
3.2.3 Dockerfile.....	21
3.2.4 Docker Registry .....	22
3.2.5 Docker Swarm.....	23
3.2.6 Docker Stack .....	24
3.3 Solution Architecture .....	27
4 IMPLEMENTATION.....	29
4.1 Orchestration .....	29
4.2 Load Balancer.....	32
4.3 Back-end.....	35
4.4 Database .....	37

4.5	Potential Services .....	37
5	INTERGRATIONS, ADJUSTMENT AND PERFORMANCE .....	39
6	SECURITY .....	40
7	SUMMARY .....	41

## **LIST OF ABBREVIATIONS**

<b>ACME</b>	Automatic Certificate Management Environment
<b>API</b>	Application Programming Interface
<b>AUFS</b>	Advanced multi-layered Unification FileSystem
<b>AWS</b>	Amazon Web Services
<b>CA</b>	Certificate Authority
<b>CLI</b>	Command Line Interface
<b>CMS</b>	Content Management System
<b>DevOps</b>	Development and Operations
<b>CORS</b>	Cross-Origin Resource Sharing
<b>CPU</b>	Central Processing Unit
<b>CRIME</b>	Compression Ratio Info-leak Made Easy
<b>CSRF</b>	Cross Site Request Forgery
<b>DES</b>	Data Encryption Standard
<b>DNS</b>	Domain Name System
<b>DV</b>	Domain Validation
<b>ECMAScript</b>	European Computer Manufacturers Association Script
<b>EFF</b>	Electronic Frontier Foundation
<b>EV</b>	Extended Validation
<b>GA</b>	Google Analytics

<b>GCM</b>	Galois/Counter Mode
<b>GZIP</b>	Compressed File Archive Zip
<b>HA</b>	High Availability
<b>HTTP</b>	Hypertext Transfer Protocol
<b>HTTPS</b>	Hypertext Transfer Protocol Secure
<b>IaaS</b>	Infrastructure as a Service
<b>ID</b>	Identification
<b>ISRG</b>	Internet Security Research Group
<b>IT</b>	Information Technology
<b>JSON</b>	JavaScript Object Notation
<b>MVCC</b>	Multi-version concurrency control
<b>ORDBMS</b>	Object-Relational Database Management System
<b>OS</b>	Operating System
<b>OV</b>	Organization Validation
<b>PaaS</b>	Platform as a Service
<b>PWA</b>	Progressive Web App
<b>RAM</b>	Random Access Memory
<b>REST</b>	Representational Stateless Transfer
<b>SaaS</b>	Software as a Service
<b>SEO</b>	Search Engine Optimization
<b>SSL</b>	Secure Socket Layer

<b>SSR</b>	Server Side Rendering
<b>TLS</b>	Transport Layer Security
<b>UFW</b>	Uncomplicated Firewall
<b>XML</b>	Extensible Markup Language
<b>YAML</b>	Yet Another Markup Language



## LIST OF FIGURES

Figure 1. Self-hosted premise versus cloud computing premise icebergs /2/.....	6
Figure 2. Types of Cloud computing /4/.....	7
Figure 3. Multiplexing helps lessening a number of requests needed for the first page load /6/.....	9
Figure 4. Communication between Browser and Server with and without Server Push.....	10
Figure 5. Overview of the SSL/TLS handshake /10/.....	11
Figure 6. Transaction isolation with Multi-version concurrency control in PostgreSQL /16/.....	14
Figure 7. GraphQL endpoint's query and result for restaurant dishes.....	16
Figure 8. Request to response workflow via Express server /24/.....	17
Figure 9. Nginx as a load balancer /27/.....	18
Figure 10. A Dockerfile content /31/.....	19
Figure 11. Structure of layers in Docker Container /31/.....	20
Figure 12. Docker containers share the same Kernel /33/.....	20
Figure 13. Docker architecture /34/.....	21
Figure 14. Docker registry and internal workflow /35/.....	22
Figure 15. Docker Swarm overview /37/.....	24
Figure 16. An example of database and api services in a Docker Stack YAML file.....	26
Figure 17. Solution Architecture.....	27
Figure 18. Environment variables reference to secret keys in Docker Stack.....	30
Figure 19. Volume and network setup with the Docker Stack.....	31
Figure 20. Numbers of replicas, policy and constraints of a service.....	32
Figure 21. Forcing users to use secure connection.....	33
Figure 22. Difference between an EV certificate (top) and DV certificate (bottom).....	34
Figure 23. Comparison between uncompressed and compressed responses.....	35
Figure 24. Back-end handles requests back-and-forth.....	36
Figure 25. Data journey for collecting, managing and supervising /41/.....	38

## **LIST OF TABLES**

Table 1. An example of a REST API /14/ .....	13
--	----

# 1 INTRODUCTION

## 1.1 Background

With the involvement of technologies in everyday life, more and more modern web applications have been produced to meet the demands of people, has led to the development in different spheres. The restaurant, where people usually come for meals because of good food and design, cannot hold for the same number of customers as they used to be if they do not leverage the development of technology to increase the flexibility and comfort. The use of technology can also be seen as a sparkle point to attract the attention and interest of customers.

Tuong Vi Nguyen, who operates a local restaurant in Jyväskylä, would like to have a web application for her restaurant to increase the number of customers come to restaurant, get feedback, and to do the integration with third party companies in the future.

During the progress, several discussions have been conducted and the requirements has been collected and updated all the time based on the owner's interests as well as the developers' recommendation.

## 1.2 Purpose

The thesis reports the development of Fit restaurant back-end system whose main intent is to serve the front-end requests, such as information about dishes, contact, Instagram and opening hours scheme. It could also be used as an endpoint to send data back and forth to the third-party APIs for later integration.

The system was divided into separated services, such as database, back-end, front-end. Each should be hosted on the same hosting and under the same domain but separated containers, hence the load balancer has been introduced as a service to increase the flexibility of the system. The implementation progress of the whole system is proceeded without any help from any Content Management System (CMS) therefore a good database structure was constructed and kept updating every time a new requirement is introduced.

### **1.3 Thesis Overall Structure**

The first part shows the reason and context of the thesis, while the progress of requirement collection and analysis covered in the second part provides a deep understanding about the direction of the system. The third part talks about all related technologies used in different parts of the system together with Docker, the environment for hosting, communication and management of all the containers. More details of how those technologies are formed and the workflow are introduced in the fourth part. The fifth part provides the information about the integration, adjustment, and performance of the system. The thesis ends with the discussion about the security layers and the summary regarding the status of the system and the satisfaction of the customer.

## 2 REQUIREMENTS AND ANALYSIS

### 2.1 Requirements Collection

All the requirements of the restaurant owner were written down, discussed, filtered and prioritized before the implementation took place. Several questions were made to clarify some points of view of the requested features and the potential problems that may be encountered during the development process.

This stage takes time to gain a good understanding about the features and the system itself. Acknowledging the matching between the illustration of the restaurant owner and technical solution will help saving significant amount of time for meetings, restructuring or re-implementing features.

There is no specific requirement for the back-end as the restaurant owner does not have any clue about back-end, development and operation (DevOps) but it was consulted, discussed and built so that the system can serve any designed requests from the web application and the future centric application.

The progress contains two iterations of development, and the requirements collection happens at the very beginning of each cycle. The reality shows that requirements may come suddenly at any time during the development progress, so below are all the requirements collected from the restaurant owner.

- Restaurant owner's requirements:
  - Banner
  - Restaurant introduction
  - Menu
  - Opening hours
  - Map
  - Feedback
  - Multilingual support for English and Finnish
  - Modern technologies development
  - Low setup cost and monthly basis fee
  - Simple, clean and neat design

- Developers' improvement suggestions:
  - Restaurant introduction with image
  - Dynamic opening hours
  - Separate menu page
  - Separate contact page
  - Commitments from restaurant
  - Interactive map
  - Instagram feed
- System roles and permissions: There are only two roles in the system and the roles are only used in the future centric application.
  - Owner role: The restaurant owner wants to update everything in the application without any assistance so one owner role is enough. The owner will have the maximum abilities to create, update and delete the contents, such as bilingual translation content, basic restaurant information, dishes, opening hours, Instagram ID and token.
  - Admin role: It is mostly used to debug the centric application in case an error happens. It has the same functionalities as the owner does and it can do the internal configuration to which the owner cannot see or make any change.
- Overlook features:
  - Online ordering system: Because of the scale and operation duration of the restaurant, the owner decided not to go for ordering system attached to the web application.
  - Finance and Profit report: As the restaurant uses separate services from companies to serve different purposes, it is hard to gather all necessary information into one place to produce report. The feature also goes beyond the scope of this thesis.

## **2.2 Requirements Analysis**

From above requirements, one database is needed to store data, and one back-end server is required to handle requests. The technologies will be determined based on

the current knowledge of developers, trend and benefits that contribute to the project outcome.

A centric application, which is out of the scope of this thesis, will be used by the restaurant owner to manage data and settings. While the web application will only be used to show the necessary data to the customer and no authentication is needed since the customer does not need to login to use the web.

In the view that the multi-language support is required, both front-end server and database server will be in charged for this function. This means more complexity will be added on top of the system and should be taken into account from the early state of the progress.

In addition, a web application may encounter downtime or sudden increase in traffic someday in the future. With the modern development requirement in mind, the containerizing technology where any services can be launched, removed, scaled up and down easily and automatically based on the pre-setup will be used to achieve the goal.

Subsequently, a load balancer for traffic coordination and caching mechanism is needed when containerizing technology was introduced. All network requests will be handled properly and may also be cached for a better performance.

There are many providers for renting the domain and hosting. Finnish regulation might be applied on domain renting but the price. High price has its own benefits in the ecosystem and services, such as Amazon Web Services (AWS), while low price brings the basis services onto the table but may be poor in quality or connection between services or there will be more manual work to do. The monthly price will be the first priority before thinking about the ease of setup, management, deployment and integration.

In the end, after conducting several investigations and considerations, Digital Ocean was chosen for its services which fits the requirements the most.

### 3 SOLUTION TECHNOLOGIES AND ARCHITECTURE

#### 3.1 Adopted Technologies

##### 3.1.1 Cloud Computing

Cloud computing is the way of delivery computing as services for servers, database, storage, networking, software, analytics, and such over the Internet. It can be classified into some spacious categories including Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS).

Each of these has its own purpose and can be used separately or built on top of another. With different types of categories, the consumer needs to handle a certain amount of work to achieve the end result. /1/

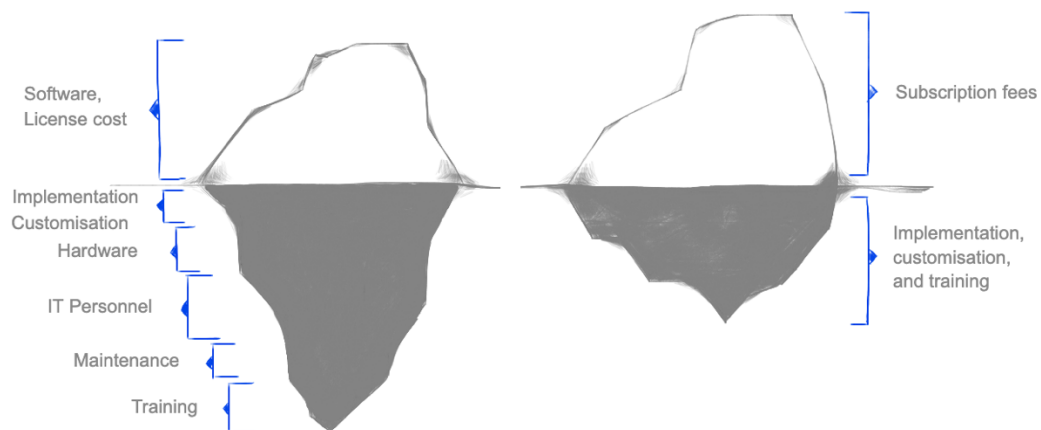


Figure 1. Self-hosted premise versus cloud computing premise icebergs /2/

A start-up or a small company trying to find its way to the cloud or taking advantages of technologies in the market should focus on its mainstream business to conquer more markets, pursue better targets and earn a better profit rather than establishing and managing all things from hardware to software, planning to maintenance. In other words, self-hosted premise requires gigantic amount of work which will be scattered into hardware facility, software, security and cost management, development, maintenance and professional personnel. The online premise with self-hosted approach is not the wrong way to go but it may be



considered not the right moment to invest time, focus and money from the beginning if technology is not the field to which the company belongs.

Cloud computing helps reduce the cost of investing, managing, and maintaining professional IT systems. In addition, consumers could benefit from all best practices and the flexibility in the scalability of IT structure later.

### 3.1.2 Platform as a Service

A platform as a service is the way vendors or service providers offer their customers the cloud environment to build, develop and maintain applications with for example, supported infrastructure, operating system (OS), network, storage but the consumer has to decide on which kind of environment and application will be deployed. /3/

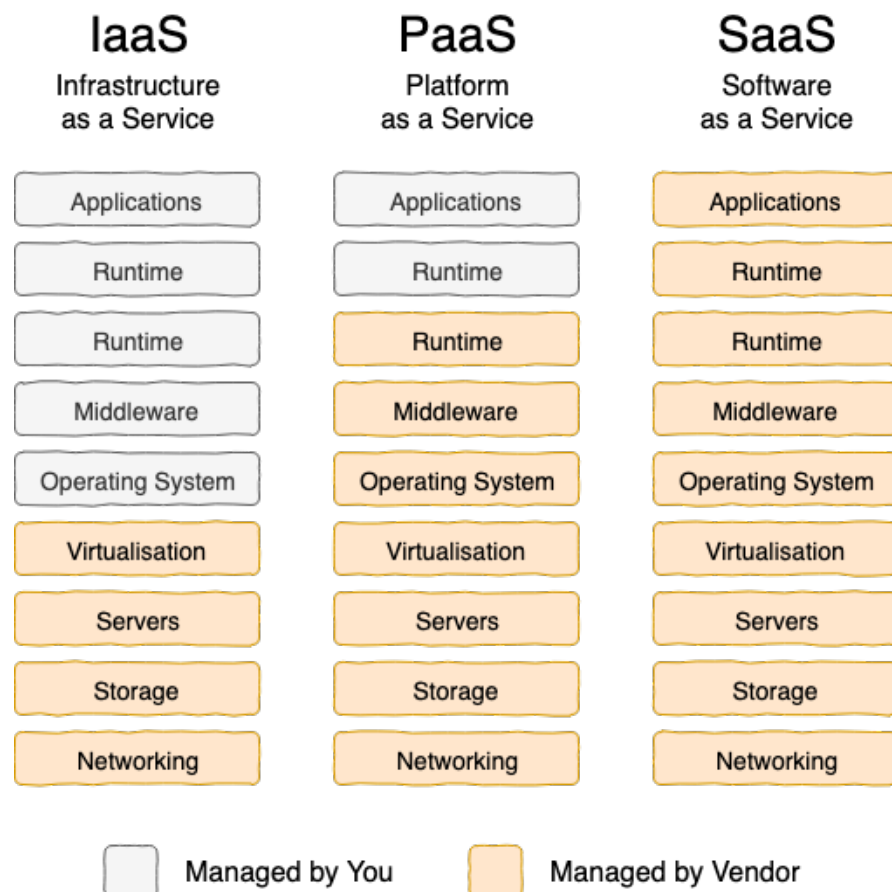


Figure 2. Types of Cloud computing /4/

In place of doing tremendous number of setups in different parts from hardware, location of the server, installation and licensing, PaaS enables small businesses to have their own camp on the Internet with affordable price on monthly basis, brings the ease of use and piece in mind for founders or developers who continuously work on their projects.

This kind of service suits those who already have their own pieces of software or ability to build their own. It may be varied between different providers on what services are provided, the collaboration between services, and most importantly, the services pricing.

There are some well-known PaaS providers, for instance, AWS, Oracle Cloud Platform, Google App Engine, OpenShift, Digital Ocean, etc.

### **3.1.3 HTTP/2 over SSL/TLS with Let's Encrypt**

Hypertext Transfer Protocol (HTTP) is an application protocol that helps web based applications on all over the world communicate with each other and exchange data. In other words, it is one of the foundation protocols to help carrying information around the world from one point to another. Hence, if a user navigates to a web application, the web browser makes a HTTP request to the server, then the server will send a HTTP response after a while back to the web browser. /5/

With HTTP/2, the transformation of transferring plain-text message in HTTP/1.1 to transferring encoded binary in HTTP/2 which enables multiple requests to be sent in parallel within a single connection, the so-called multiplexing. This means the loading time to the site can be reduced significantly in the first page load. /6/

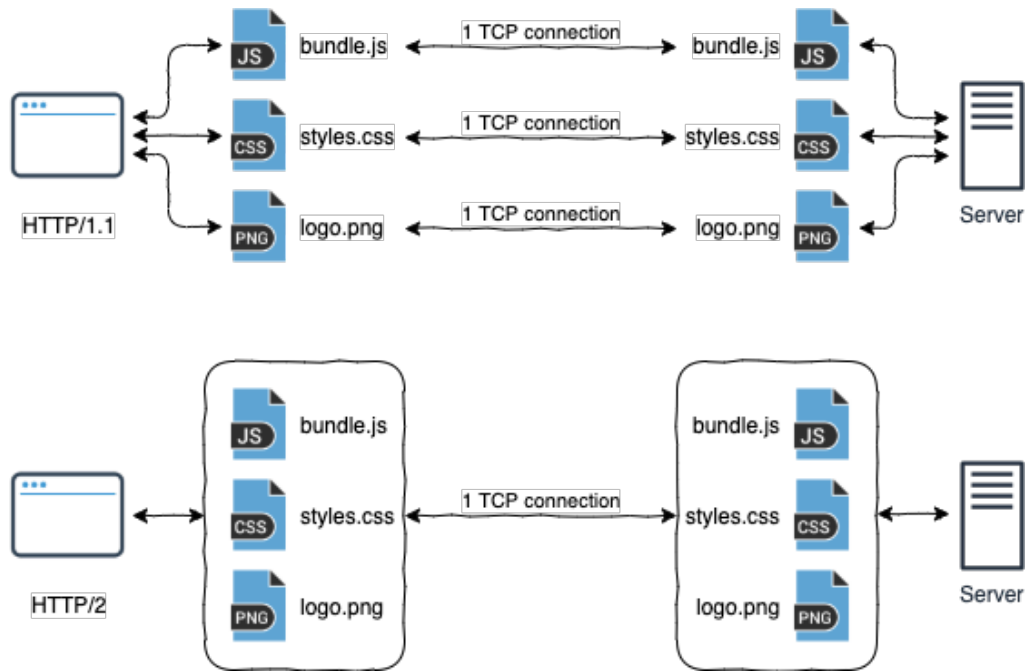


Figure 3. Multiplexing helps lessening a number of requests needed for the first page load  
/6/

In addition, various redundant fields in the request header can be compressed and kept track with the help of HPACK to only send changed indexes to reconstruct the shared fields and encode new fields. Hence, the speed and experience will be enhanced due to the small bandwidth needed for each request. /7/

Moreover, the necessary resources for the first page load do not need to wait until requested. With server push, the server can perform the transfer of resources ahead of time which significantly reduces the serving time. /8/

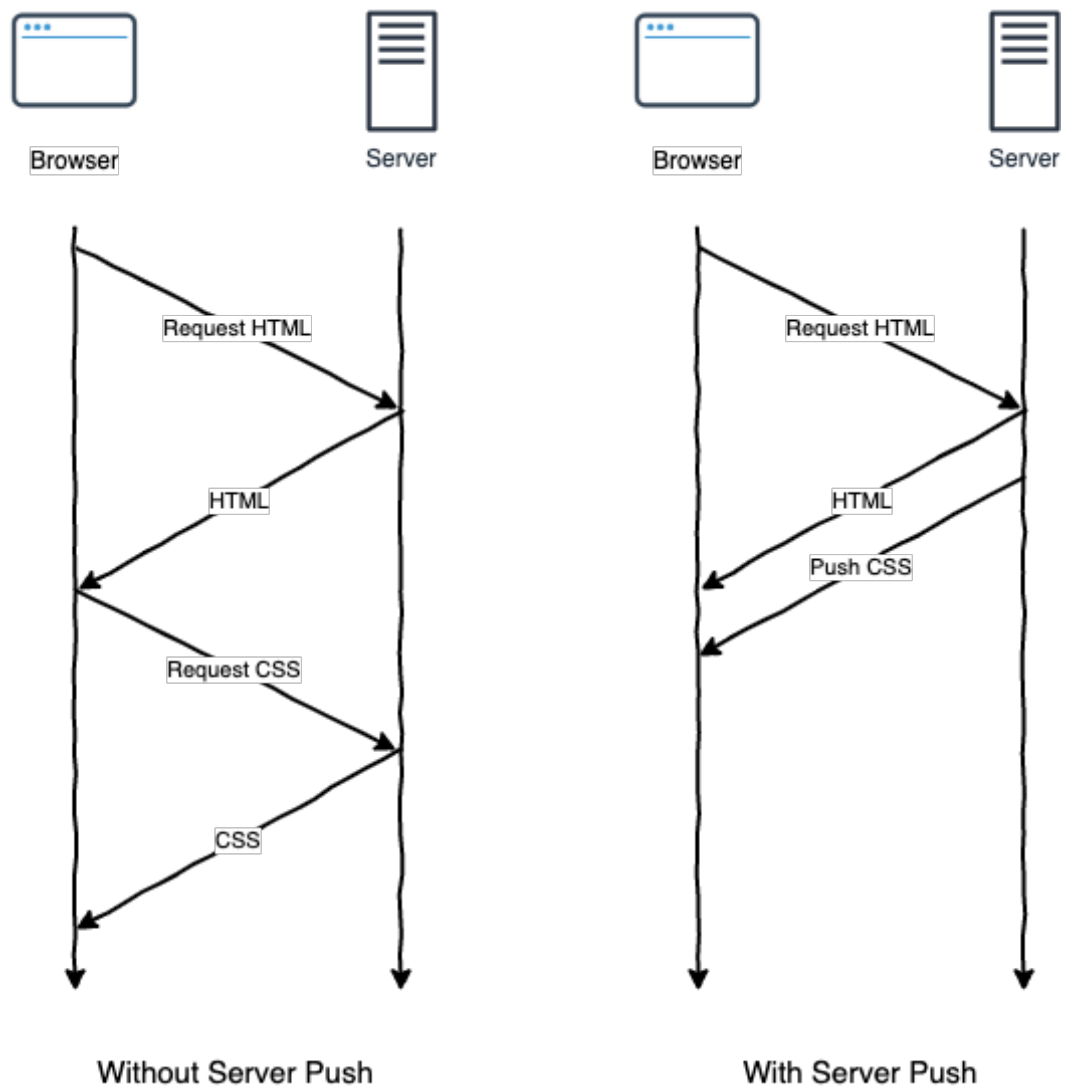


Figure 4. Communication between Browser and Server with and without Server Push

Secure Socket Layer (SSL) is the predecessor to Transport Layer Security (TLS) but both are cryptographic protocols providing data encryption and authentication for servers, applications over the Internet. Their aim is to prevent data integrity and privacy being stalked or spoiled by any third party. /9/

By writing HTTPS, it simply means that data exchange over HTTP is only made after the SSL/TLS connection has been established. To have that secure connection, a handshake process was introduced which includes several steps until the connection is assessed to be secured and ready for data exchange as demonstrated in the Figure 5 below.

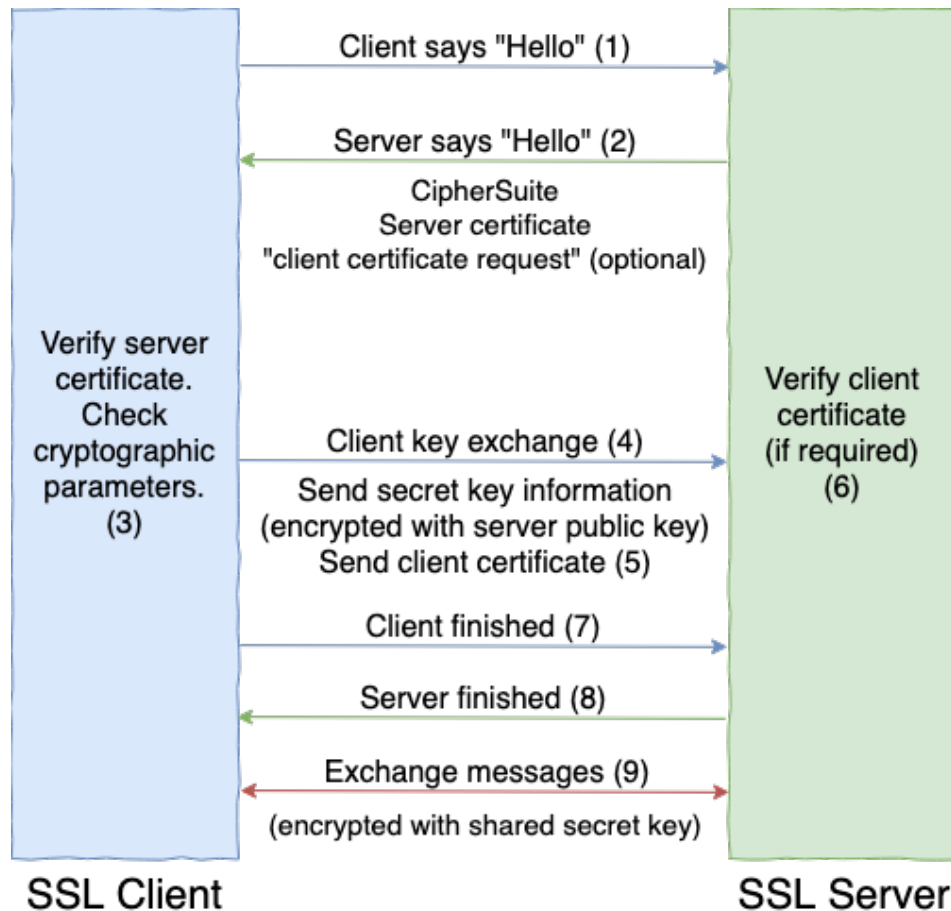


Figure 5. Overview of the SSL/TLS handshake /10/

On February 8, 2018, Google published a blog post announcement regarding unsecure web access. This means Chrome will show a first-stop warning page every time a user visits a webpage with no SSL/TLS implementation over the domain. /11/ Since then, every web hosting providers have started providing free or low-cost certificates to its consumers.

Let's Encrypt, the first non-profit Certificate Authority (CA) run by Internet Security Research Group (ISRG), distributes certificate for free to make the world a better place since April 12, 2016 /12/. The automation of issuance and renewal process can also be done via Certbot ACME client if consumers have the shell access. At the time, the expiration for a certificate is 90 days, which was considered based on the advantage of issuing automation and damage limitation over the occurrence of miss-issuance and key compromise.

### 3.1.4 REST API

Representational Stateless Transfer (REST) is a software architecture style that uses many other standards, such as XML, JSON, URI and HTTP to create a standard way for system communication over the Internet. RESTful design provides a good performance, simplicity, portability, and reliability by using a stateless protocol and standard operations. /13/

There are several key-points a RESTful design should follow:

- Client-server architectural style: Helps to increase the flexibility and scalability as the concerns of the interface and storage are separated thus any new evolvement or any change in either two will not affect each other functionalities. /13/
- Statelessness: The server will not store any client context as well as should not make use of any feature supporting stored context. In contrast, the client request should carry all the necessary information that the server can understand. This also helps to increase the reliability and scalability when recovering back from failure and releasing of resources. /13/
- Cacheability: It is all about the efficiency in data transfer and transparency. Cache enables data to be reused for different purposes and any response has to be clarified as cachable or non-cachable. /13/
- Uniform Interface: This is a significant feature in a RESTful design. The ability to simplify the system architecture and decoupling the implementation from the services are conducted by four interface constraints: resource identification in requests, resource manipulation over representations, self-descriptive message, and hypermedia as the turbine of application state. /13/
- Layered system: Some components within the system will be categorized into different layers. The higher layer will get support in functions and services from lower layers, and its understanding about lower or higher layers cannot go beyond what is provided by the interacting layer. This

breaks down the complexity, increase the flexibility and encapsulate legacy in a system. /13/

- Code on demand: Giving the server the ability to temporarily extend the functionalities by running applets or scripts, which also reduces the visibility and becomes a voluntary option in RESTful design. /13/

Application Programming Interface (API) is a set of definitions, tools and protocols to build software. It can be used within web application, operating system, hardware, library, etc. /14/

While talking about RESTful APIs, it can be understood as Web APIs or web service APIs. The designed architectural approach to provide sets of interfaces that different types of consumers can understand. It contains one or more non-identical endpoints exposed to the Internet, and serves Hypertext Markup Language (HTML), Javascript Object Notation (JSON), Extensible Markup Language (XML) or some other format as the payload via Hypertext Transfer Protocol (HTTP).

There are several typical HTTP methods such as:

Table 1. An example of a REST API /14/

<b>HTTP methods</b>	<b>URL</b>	<b>Instruction in request body</b>	<b>Description</b>
GET	/collection	Not required	Retrieve all the member resources
GET	/collection/:id	Not required	Retrieve only one member resource based on the given id
POST	/collection	Required	Create a new member resource
PUT	/collection/:id	Required	Update (if exist) or create (if not exist) a member resource based on the given id

DELETE	/collection/:id	Not required	Delete a member resource based on the given id
--------	-----------------	--------------	--

### 3.1.5 PostgreSQL and PostGraphile

PostgreSQL is the one of the most advanced open-source object-relational database system (ORDBMS) which offers different kinds of connectors, tools, and libraries thanks to its active communities. /15/

Some benefits that make PostgreSQL stand out, such as default support for Unicode and concurrency, working on different platforms (Linux, Mac, Window, etc.), performs really well in many tech stacks via extensions support and many paid features in MySQL such as partitioning, compression, concurrent index creation. /15/

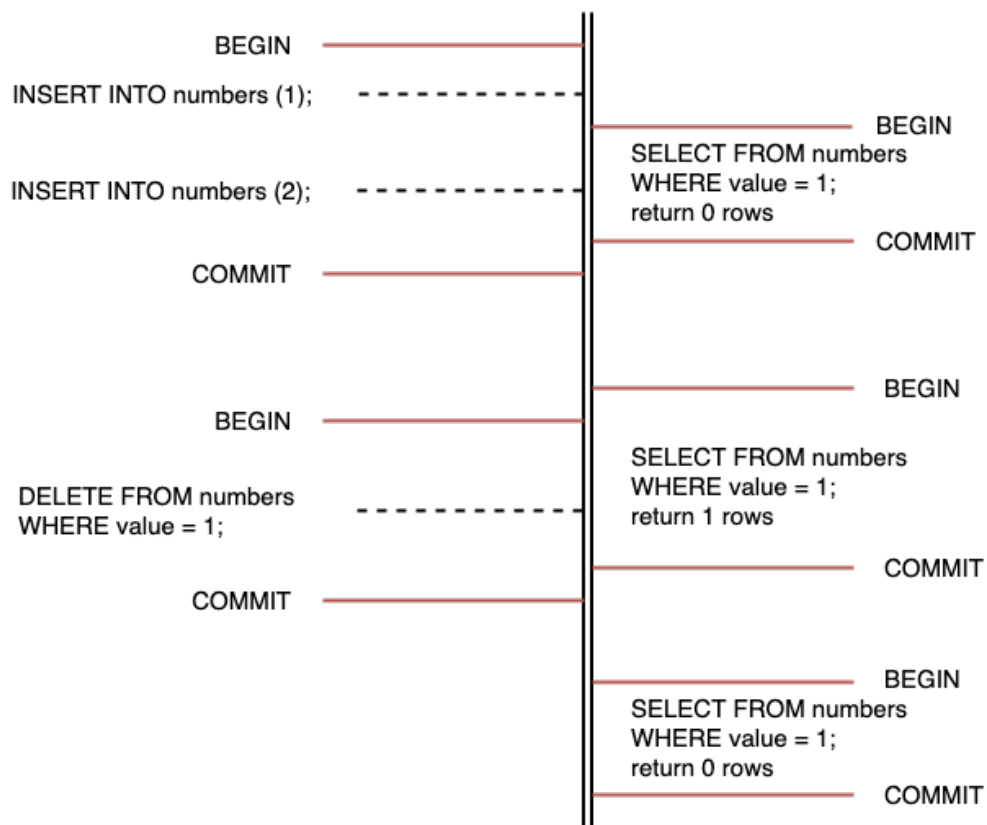


Figure 6. Transaction isolation with Multi-version concurrency control in PostgreSQL /16/



The above image illustrates one of PostgreSQL features, transaction isolation. Each transaction runs on an isolation level. For example, if there is SELECT queries run in between concurrent operations in the left, such as INSERT and DELETE, only the old data from which those left operations began will be returned.

The reasons for PostgreSQL being chosen including:

- A longstanding history with regular updates.
- Open-source with an emphasis on extensibility and Atomicity, Consistency, Isolation, Durability (ACID) compliance. /17/
- An eco-system including community and vendors who are supportive, devoted with built-in support for PostgreSQL or zero-cost for using PostgreSQL as the database which Heroku could be taken as a luminary example. /18/
- The first residence for Multi-version concurrency control (MVCC) feature, which overcomes the database dead-lock, enables users to interact with the database concurrently thanks to the use of the snapshot of data at a single point of time underneath. /19/

PostGraphile (PostGraphQL) library, is a powerful combination of GraphQL and PostgreSQL, can be used on top of PostgreSQL schemas as a data loader from different sources. It automatically discovers all the tables, relationships, types, functions, etc. to make executing and querying data in a way as flexible as possible. /20/

PostGraphile has some built-in security features, auto generating documentation and auto supporting Create Read Update Delete operations (CRUD), saving network roundtrips, playing especially well with Server Side Rendering (SSR) and many more. /21/

```

1 {
2   allMenuDishes{
3     nodes{
4       id,
5       nameEn,
6       nameExtraEn,
7       descEn,
8       allergy,
9     }
10  }
11 }

```

```

{
  "data": {
    "allMenuDishes": {
      "nodes": [
        {
          "id": 1,
          "nameEn": "Chả giò (3pcs)",
          "nameExtraEn": "Spring rolls",
          "descEn": "Chicken, rice vermicelli noodle with vegetables, fish sauce",
          "allergy": "G, L"
        },
        {
          "id": 2,
          "nameEn": "Gỏi cuốn (3pcs)",
          "nameExtraEn": "Summer rolls",
          "descEn": "Prawn, chicken, rice noodle, vegetables, peanut butter sauce",
          "allergy": "G, L"
        },
        {
          "id": 3,
          "nameEn": "Tôm nướng Fit (3pcs)",
          "nameExtraEn": "Fit's grilled prawns",
          "descEn": "Prawns, Fit's marinade",
          "allergy": "G, L"
        },
        {
          "id": 4,
          "nameEn": "Mực nướng sa tế",
          "nameExtraEn": "Grilled squid with chili sauce, served with seaweed salad",
          "descEn": "Squid, chili sauce, seaweed",
          "allergy": "G, L"
        }
      ]
    }
  }
}

```

Figure 7. GraphQL endpoint's query and result for restaurant dishes

The screen demonstrates the query structure on the left, and its result on the right. GraphQL endpoint returns the same data structure to the query, which brings the ease for the data processing progress later.

### 3.1.6 Express server

Express (Express.js) is a fast, lightweight and high-performance Node.js based web framework. As a server side scripting framework, Express can be used to build a Javascript-based back-end rapidly and easily within minutes. /22/

Even though Node.js is a single threaded event loop model, its I/O APIs are designed to be non-blocking in order to serve even loop efficiently. In addition, Node.js also has a cluster module which was born to spawn more separated worker processes. This results in to the outstanding performance of Express when it can take all the advantages of Node.js. /23/

Express comes with a robust routing mechanism where APIs definition becomes easier to set up and maintain. Besides the ability to add a chain of functions in between the request of user and response of server, the so-called middlewares in the way the server can react and process properly with different types of requests,

especially for handling cross-origin resource sharing (CORS), cross site request forgery (CSRF), authentication and proxy requests. /22/

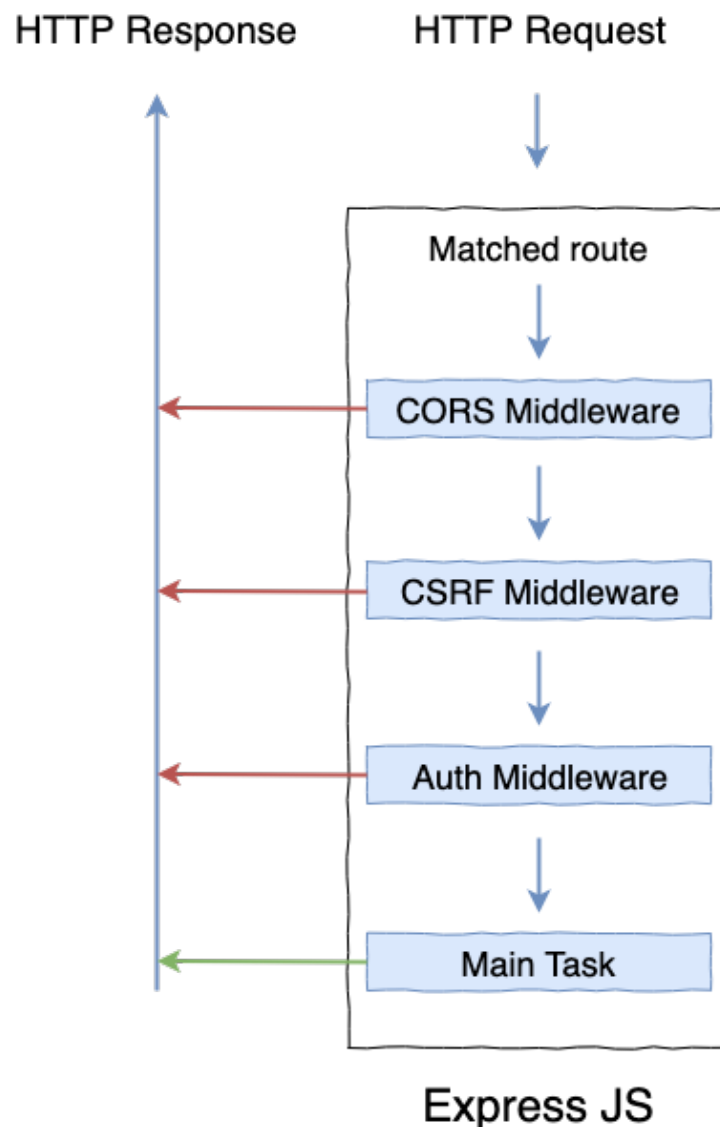


Figure 8. Request to response workflow via Express server /24/

Figure 8 describes the workflow of a request going through an ExpressJS server. The request will be examined against pre-defined routes on the server in order, the first match route will forward the request to layers of middlewares. If the request does not pass the conditions of any middleware, the middleware can deny the request, and prevent it to be proceeded further by sending back an appropriate response to the client.

### 3.1.7 Nginx web server

Nginx is an open source web server written by Igor Sysoev in C procedural language. Even though it was designed to be asynchronous, event driven and single threaded, Nginx can operate as a high-performance load balancer, reverse proxy, caching or streaming server. /25/

Having the ability to handle a huge number of concurrent requests with a low memory usage makes Nginx a true gem in its type. Moreover, FastCGI was introduced to effectively enhance the concurrency in Nginx, which was used by many different well-known servers such as Apache, Jetty, Microsoft IIS, and Zeus /26/

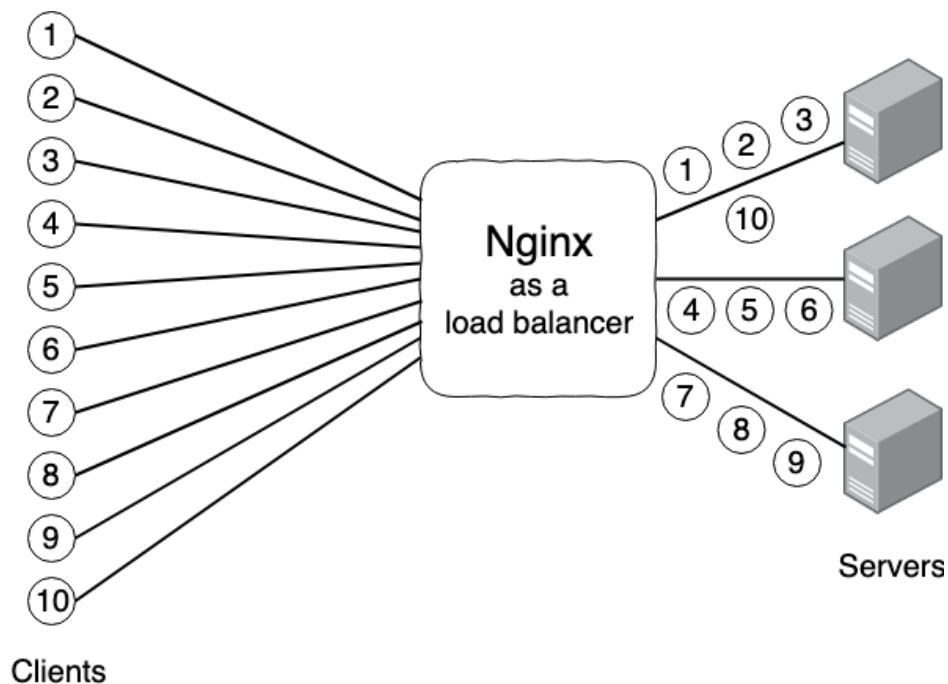


Figure 9. Nginx as a load balancer /27/


Besides the support for HTTP/2, IPv6, TLS/SSL, OpenBSD, WebSockets, GZIP compression, Nginx gained the popularity in community and adopted by a large number of prominent companies, including IBM, Microsoft, Salesforce, Gitlab, Facebook, Apple and Intel. /28/ Nginx also allows the use of third party modules so the customization could gain its way to utilize Nginx to work more efficient and take advantage of different parts in many ways.

## 3.2 Docker and Container Orchestration

Docker, a software platform, first released in 2013, that allow users to build and run applications on environment-isolated containers, is developed by Docker Inc. /29/

### 3.2.1 Docker Image and Container

Docker Image contains a set of read-only advanced multi-layered unification filesystem (aufs) layers as the result of the Docker build process /30/. Each layer corresponds respectively to an instruction in the Dockerfile, which is called image layer. The layer of recent instruction will stay on top of the last instruction.



```
FROM ubuntu:15.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

carbon  
carbon.now.sh

Figure 10. A Dockerfile content /31/

When a Docker image is run and turned into a container, an additional layer called read-write layer is added on top of the image layers to handle all manipulations, such as adding, modifying and deleting files in the below layers. This maximizes the efficiency for sharing and copying files by minimizing Input and Output (I/O) as well as the size of the following layers. /31/

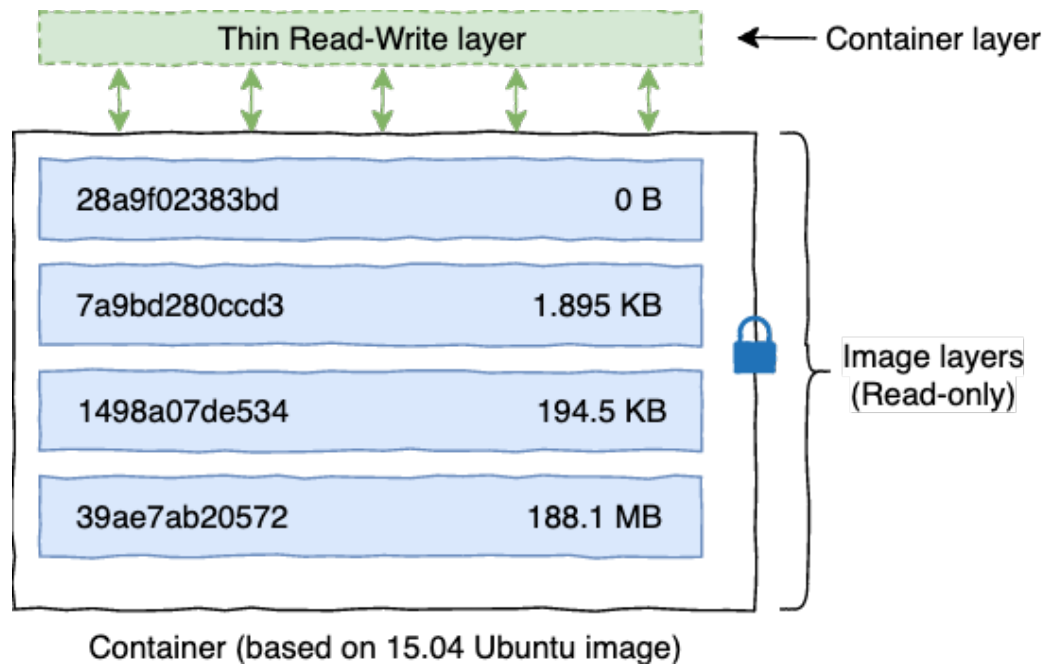


Figure 11. Structure of layers in Docker Container /31/

A Docker container is an instance of Docker container image which can be simply understood as a bundled piece of software with the source code, tools, runtime, libraries, settings and dependencies needed.

Docker containers are hosted within a single host operating system (OS) and can share the host kernel together and still maintain their own isolation environments thanks to the help of Docker Engine. /32/

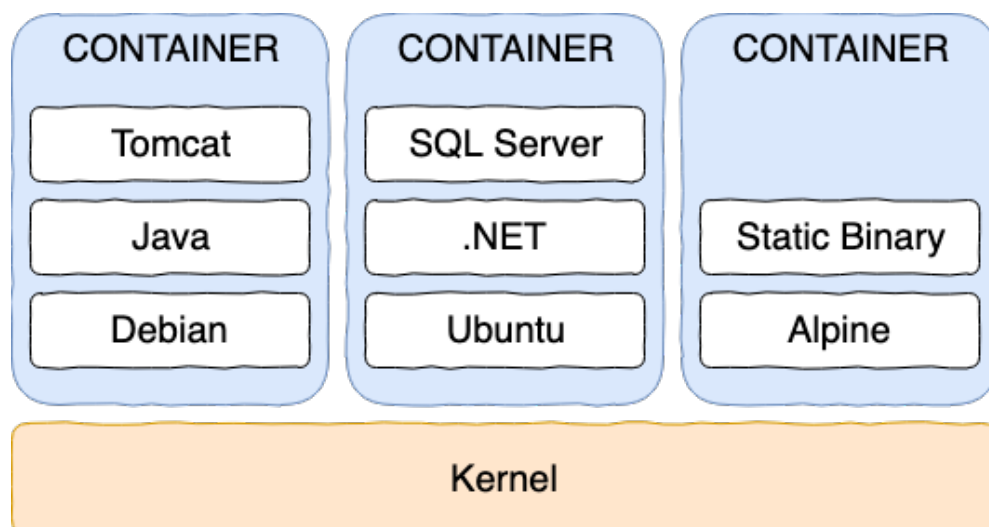


Figure 12. Docker containers share the same Kernel /33/

### 3.2.2 Docker Engine

Docker engine is a client-server application which first creates server side Docker daemon process to host images, containers, storage volumes and networks, and then allows users to use the Docker client – Command Line Interface (CLI) tool to interact with the Docker daemon. /34/

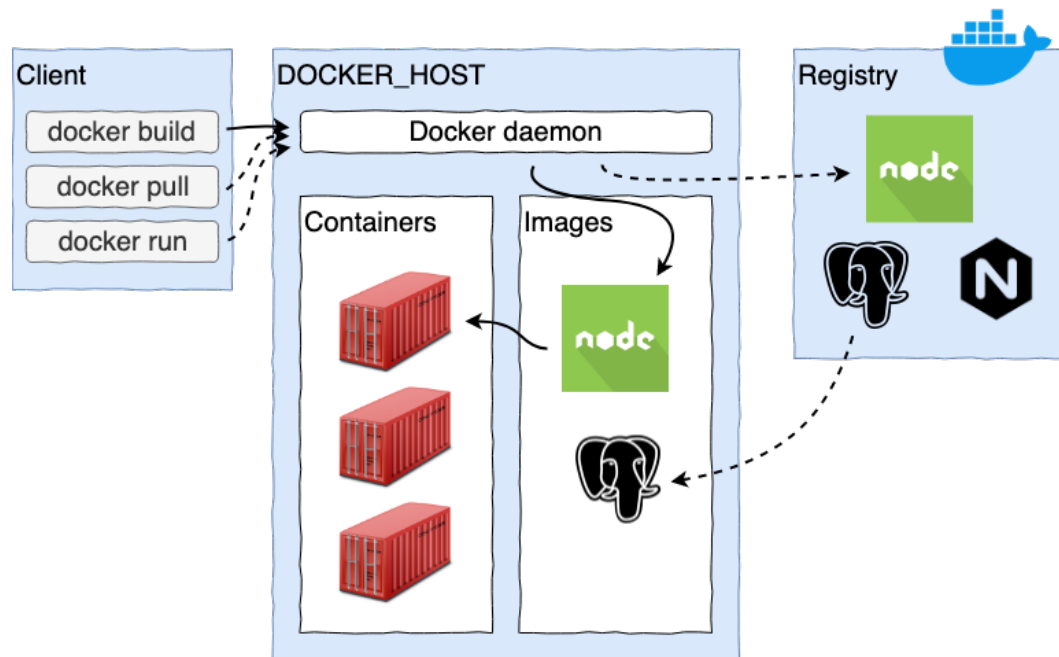


Figure 13. Docker architecture /34/

As Figure 13 above demonstrates, Docker daemon itself will listen to Docker API requests from the client and manipulate Docker components including containers, images and the registry, as well as communicate with other daemons to delivery its services. The advantage of Docker is that the Docker client and Docker daemon do not have to be host on the same system. A Docker client can communicate with many Docker daemons on different systems.

### 3.2.3 Dockerfile

A Dockerfile is required for any source code that would like to compile into a Docker container image. Dockerfile contains a list of instructions which tells Docker what needs to be done to produce a desired image. /34/

### 3.2.4 Docker Registry

Another important component is Docker registry, which is a Github-alike with commands such as pull, push, commit and tag within the Docker ecosystem. It is a central place to store all available Docker container images. A public version of this registry over the Internet provided by Docker Inc. is known as Docker Hub where users can register and store their images without the need to host their own registry. /34/

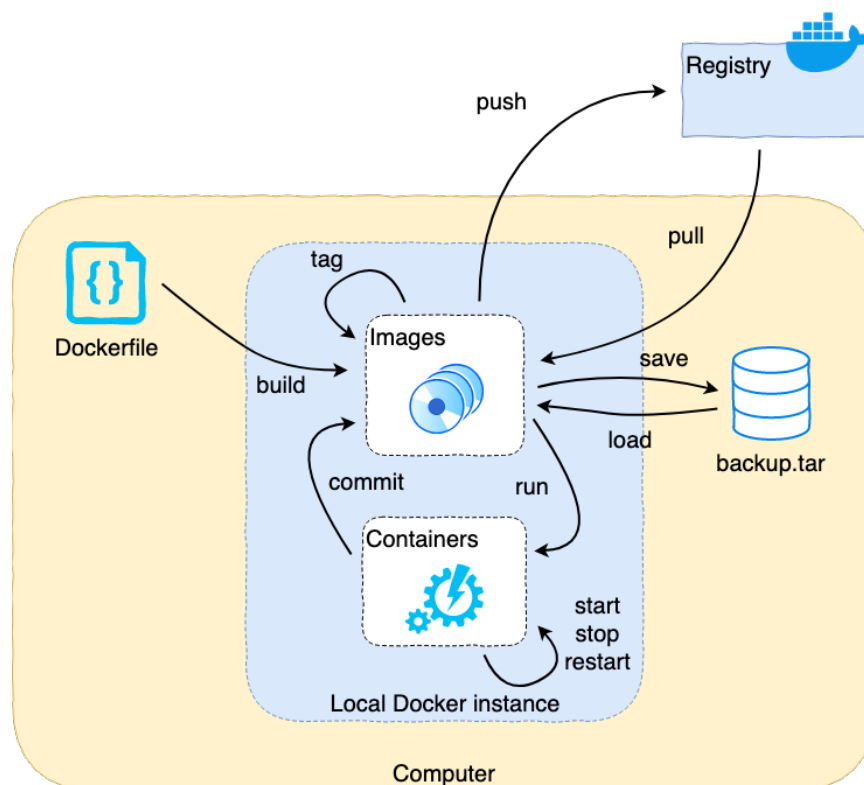


Figure 14. Docker registry and internal workflow /35/

Figure 14 the most doable internal workflow between components of Docker. A package of the source code and a Dockerfile can be built to form an image which comes with a tag. It can then be pushed to the registry for storage, and pulled for later use. The transformation between the image and container can be done via run and commit commands. The container itself has a set of basic commands including start, stop and restart. While the image can be compressed into a tar file via save command, and decompressed to an image with load command.



### 3.2.5 Docker Swarm

Docker Swarm or Swarm mode is a tool used to manage multiple running Docker Engines as a cluster. This also brings some more benefits such as: /36/

- Use CLI to interact with the Swarm and internal application services.
- Help decentralizing in a better way with the assistance of Docker Engine to distinguish node, manager or worker in the runtime instead of doing it manually in the deployment time.
- Make service declaration more understandable by letting the Docker Engine know what are the desired states of the services in the Swarm.
- Bring the power of the overlay network to the services by assigning automatically the proper addresses to containers, subsequently create a networking between the hosts.
- Domain Name System (DNS) will be added as another benefit so the Swarm manager can do the load balancing and the administrator can query against the containers.
- Any update on the nodes can be controlled via Swarm manager by adding the deployment delay or rolling back to the old version of the service.

The CLI will now be proxied via the Swarm and run on the cluster, including authorizing a machine to join the Swarm as a worker or master/manager node. In addition, Docker Agent is introduced by registering itself once in each node and acting as a messenger so the swarm knows the status of all the nodes as illustrated in the following Figure 15. /37/

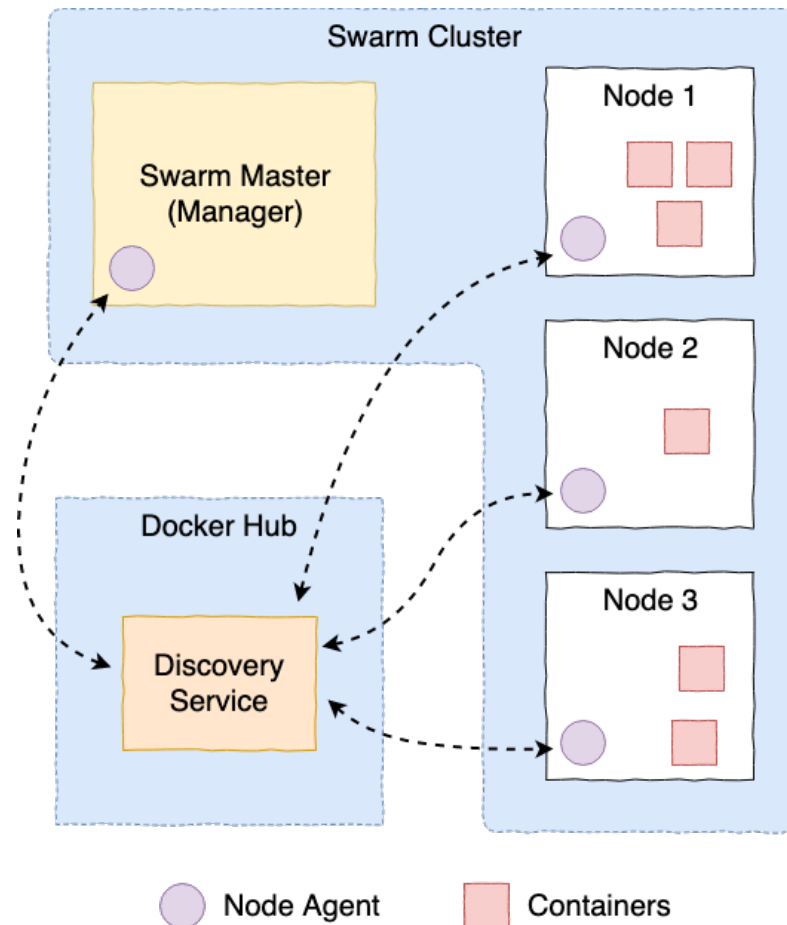


Figure 15. Docker Swarm overview /37/

Swarm also offers some benefits such as high-availability (HA) feature which guarantees no downtime to services by keeping containers available at all time, scheduling strategies to determine where and when a new container should be placed or respawn, and enforcing TLS mutual authentication and encryption to each node by default in the Swarm. /36/

Since there are multiple Docker Engines, a Register mentioned in the earlier part is needed mainly for pulling and pushing images. However, Ravintola Fit has its own Register on [Docker Hub](#) so a local Register is not necessary anymore.

### 3.2.6 Docker Stack

Docker Stack is the way to make all the connected services which may share dependencies scaled and orchestrated in conjunction. In other words, Stack is just

the way multiple services are put into a single file which will bring the ease for searching, understanding and managing. In order to use Stack within Swarm, an additional file in “Yet Another Markup Language” (YAML) format is required. All the necessary services should be defined within that file where each service declaration starts from the 1st level below the services directory. /38/

Figure 16 shows that there are two services inside the stack including database and api. Each service is constructed from a custom image where all the necessary setup was done beforehand. In addition, the port mapping for the service is also required so Docker will know to which link other services or forward requests. While the volumes option is only required when there is a need for data storage and will not be erased when the service respawns or stops by any reason. The volume will only be removed when the administrator of the host explicitly deletes that reserved portion of storage. Other parts such as environment and deploy will be covered later in the orchestration part.

```
version: "3.7"

services:
  database:
    image: fit_custom/db_postgres
    ports:
      - "5432"
    volumes:
      - database_location:/var/lib/mysql/data
    environment:
      - DB_USERNAME:/run/secrets/db_username
      - DB_PASSWORD:/run/secrets/db_password
      - DB_CONNECTION_STRING:/run/secrets/db_connection_string
    deploy:
      replicas: 1
      restart_policy:
        condition: on-failure
      placement:
        constraints:
          - node.role == manager
          - node.hostname == master_node2
    networks:
      - back_end

  api:
    image: fit_custom/api_node
    ports:
      - "6000"
    deploy:
      replicas: 1
      restart_policy:
        condition: on-failure
      placement:
        constraints:
          - node.hostname == master_node1
    networks:
      - front_end
      - back_end
```

Figure 16. An example of database and api services in a Docker Stack YAML file

### 3.3 Solution Architecture

Foremost, the system is designed so that all application images will be hosted as service containers across multiple nodes with the help of Docker Swarm and Docker Stack. All incoming requests will be handled by an Nginx load balancer as a frontier container. The load balancer will determine which front-end container to forward the request if there are more than one container of the same service.

Next, the front-end container will serve the request and send a response back to the client. Due to the fact that the front-end has its own server since it uses Server Side Rendering (SSR) technology, it is eligible to be wrapped within a container as a standalone processing unit serving a specific need in the system.

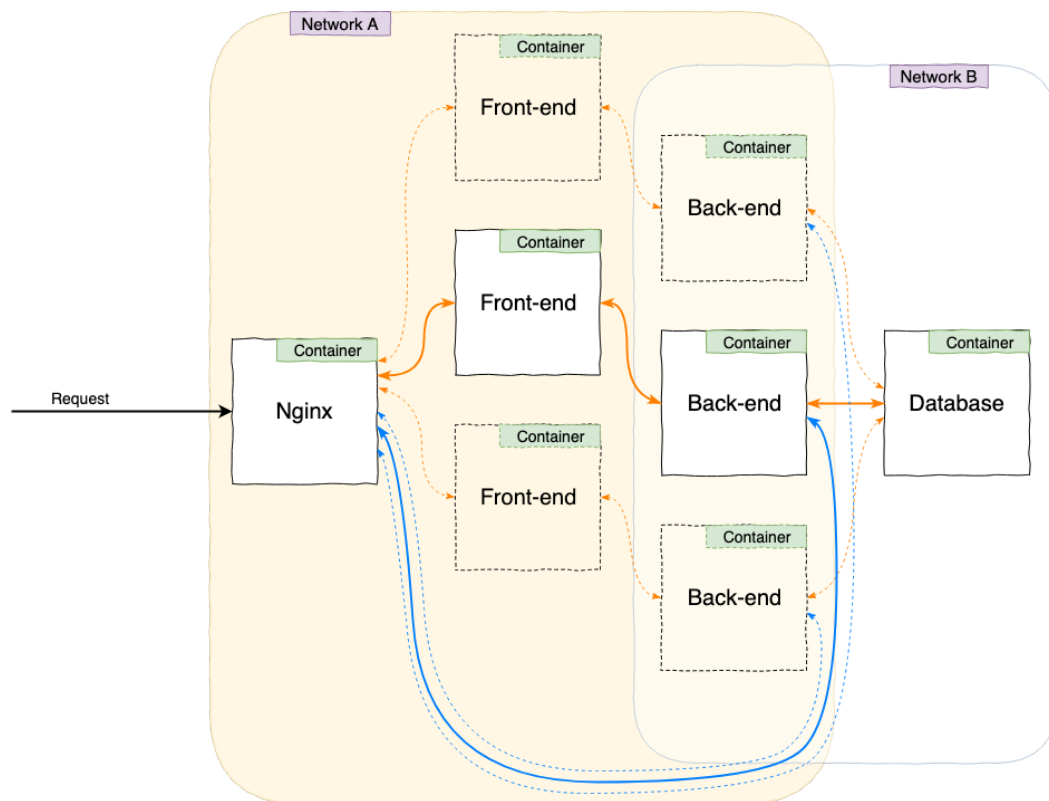


Figure 17. Solution Architecture

Besides providing data to serve front-end container, the back-end container also exposes to the outside world through Nginx for later integration with third party APIs and interacts with the database container for any data inquiry or manipulation.

Lastly, database container stores all the data for the whole system. It also contains functions and views which was designed to fit the need of the back-end container.

## 4 IMPLEMENTATION

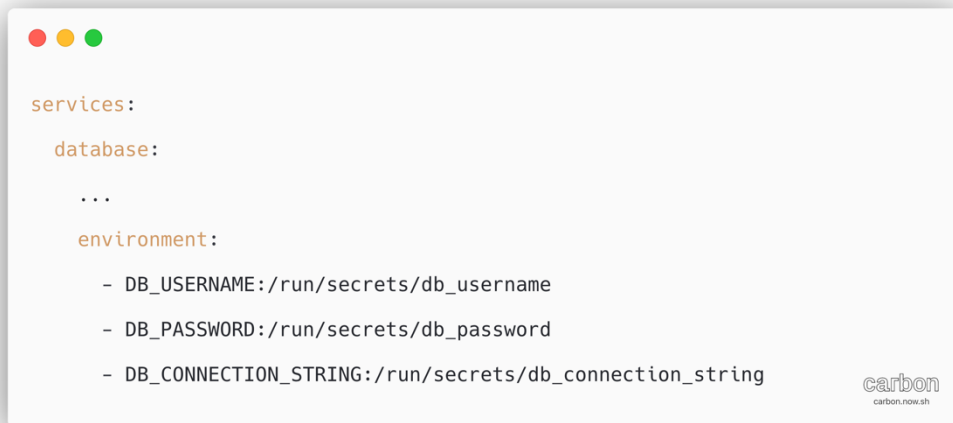
### 4.1 Orchestration

Instead of buying the hardware, hiring personnel for management or maintenance, the business owner should concentrate on the mainstream business and use the cloud computing in conjunction with PaaS as a good alternative selection to host the whole application without spending a big trunk of money on monthly budget. Basically, everything is rented except the application and the runtime environment which are the main focus of this thesis.

Since Docker has been widely used in many companies around the globe in both development and production environments, owing to the fact that it brings the environment transition from one to another as smooth as possible. Any necessary service can be built, tested and deployed as a standalone container within the Docker environment to serve a specific purpose without thinking whether it is supported by the providers or not. This would be a good base for the whole orchestration to get started with the help of Docker Swarm and Stack. Because Swarm can group all machines into one cluster, then connect, monitor and manage all of them as nodes while Stack can group all related services together and reveal all of them under certain aspects such as network, volume, environment variables, and secret keys.

Some sensitive information is stored as Docker pre-defined secret keys. The secret key itself is just a key-value pair data but a good thing is that the key can only be revealed when querying on the behalf of the attached container. This means only the container for which the key is created and served can reference to get its value in plain text. Otherwise, only error message “No such file or directory” is showed even though the secret key actually exists.

The attached container can access the secret keys either using the direct approach by querying straightforward to “/run/secrets/” where the secrets are located or using the indirect approach by creating an environment variable to make a reference back to the secret key as shown in the figure below.

A screenshot of a code editor window with a white background and a grey border. At the top left, there are three colored circles (red, yellow, green) representing window control buttons. The code is written in a monospaced font. It shows a 'services:' section with a 'database:' sub-section. Under 'database:', there are three dots indicating more services. Below that is an 'environment:' section with three lines of environment variables, each preceded by a hyphen. The variables are: DB\_USERNAME:/run/secrets/db\_username, DB\_PASSWORD:/run/secrets/db\_password, and DB\_CONNECTION\_STRING:/run/secrets/db\_connection\_string. In the bottom right corner of the code editor, there is a logo for 'carbon' with 'carbon.now.sh' written below it.

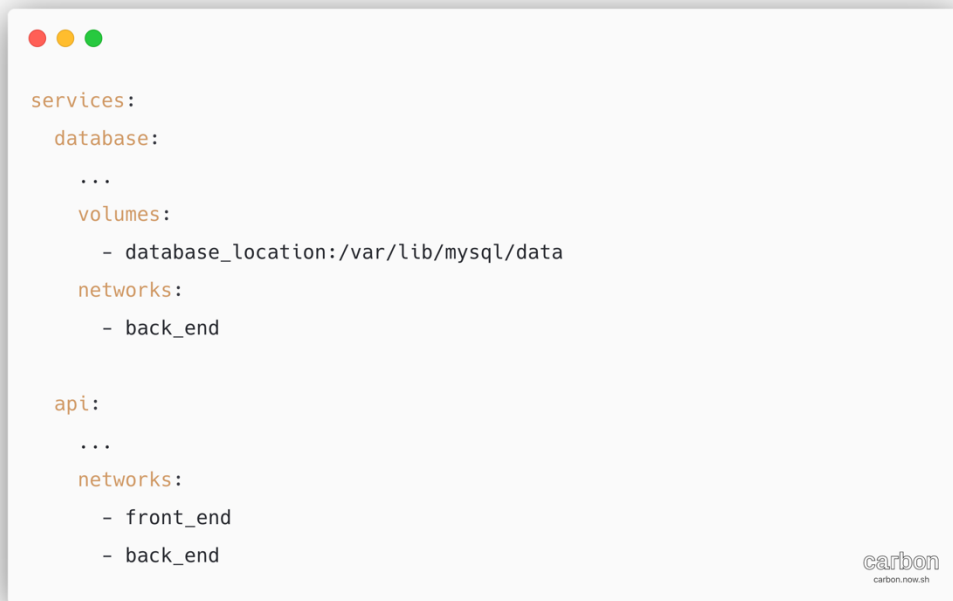
```
services:
  database:
    ...
  environment:
    - DB_USERNAME:/run/secrets/db_username
    - DB_PASSWORD:/run/secrets/db_password
    - DB_CONNECTION_STRING:/run/secrets/db_connection_string
```

Figure 18. Environment variables reference to secret keys in Docker Stack

Networks within the system are separated based on the use and purpose of different parts. The load balancer will forward the request to the front-end and the back-end so a network will be shared between those three. The front-end only needs to know the existing of the back-end which will serve its data inquiries, and it does not need to know the existence of the database from which the data come. Therefore, the back-end and the database can be grouped into another network. This yields another security benefit which will be discussed later in this paper.

Only one volume is needed for the database to store data. The volume provided by Docker environment is different from the storage within the container itself. All the data within the container-based storage will be removed if the service instance is deleted. While the volume storage will keep the data until it is deleted explicitly from the CLI by the administrator.





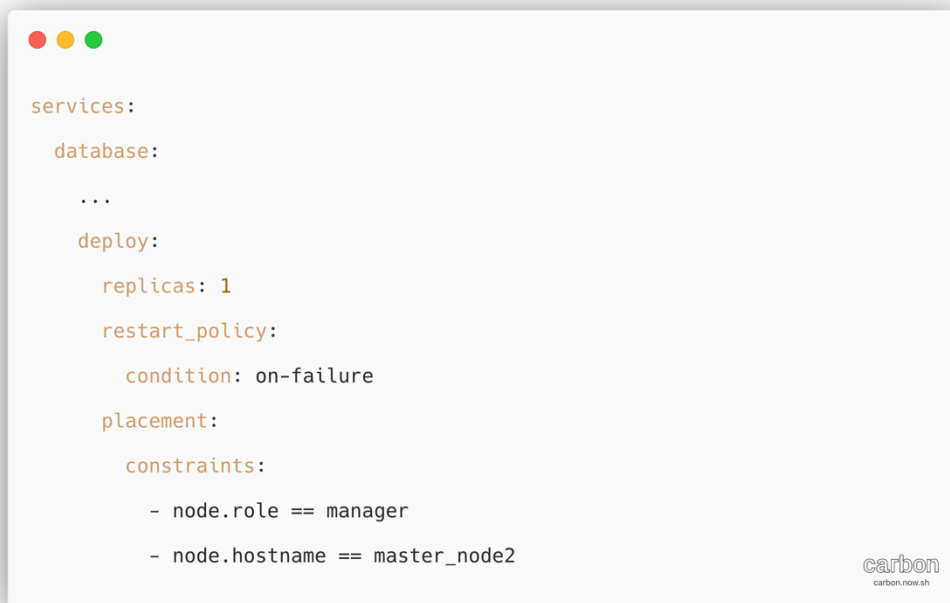
```
services:
  database:
    ...
  volumes:
    - database_location:/var/lib/mysql/data
  networks:
    - back_end

api:
  ...
  networks:
    - front_end
    - back_end
```

carbon  
carbon.now.sh

Figure 19. Volume and network setup with the Docker Stack

Since Docker Stack acts as a source of truth where all related services are listed, the numbers of replicas, restart policy, as well as one or more constraints can be put on top of each service to identify how many instances should be created, what to do when an instance failed to start, and which node a service can run when it gets initialized. Nevertheless, the replicas can be changed dynamically from the Docker CLI later based on the need, while the constraints and policy cannot. The whole Stack needs to be destroyed, updated, and re-deployed in order to apply new constraints or policy.



```
services:
  database:
    ...
  deploy:
    replicas: 1
    restart_policy:
      condition: on-failure
    placement:
      constraints:
        - node.role == manager
        - node.hostname == master_node2
```

Figure 20. Numbers of replicas, policy and constraints of a service

With this setup, the orchestration located remotely on the cloud can be manipulated directly from the desk of administrator via the CLI and executed by Swarm manager. Most importantly, the infrastructure can be scaled both vertically and horizontally after all. PaaS enables more computing power can be added to the existing machines by increasing the number of Central Processing Unit (CPU) cores, Random Access Memory (RAM) or even storage capacity. While Swarm also allows the existing architecture to extend its strength either by having more machines to join the Swarm network or increasing service containers to share the workload.

## 4.2 Load Balancer

Even though the Docker Swarm itself already has a built-in load balancing on each node, it is not flexible enough to solely depend on the load balancing of the swarm. By introducing Nginx as a balancer, developers can choose which routing algorithm needs to redirect requests of any user to a proper server. Nginx offers a few options such as:

- Round Robin: all requests will be evenly distributed to all servers.
- Least connections: the latest request will be delivered to the server handling least connection with clients.
- IP hash: requests will be handled based on the hash of the IP address in the request.

The load balancer of the restaurant uses Round Robin for dealing with upcoming requests. In addition, the redirect mechanism was also applied into the load balancer. If there are any request tries to connect to the load balancer via the port 80 will be redirected to the port 443 and continue being handled as normal as demonstrated in Figure 21 below. Introducing SSL on port 443 helps reducing the risk of being hijacked and scaling down to a few number of techniques hackers used to attack in order to conduct a proper diagnose and solution for the threat.

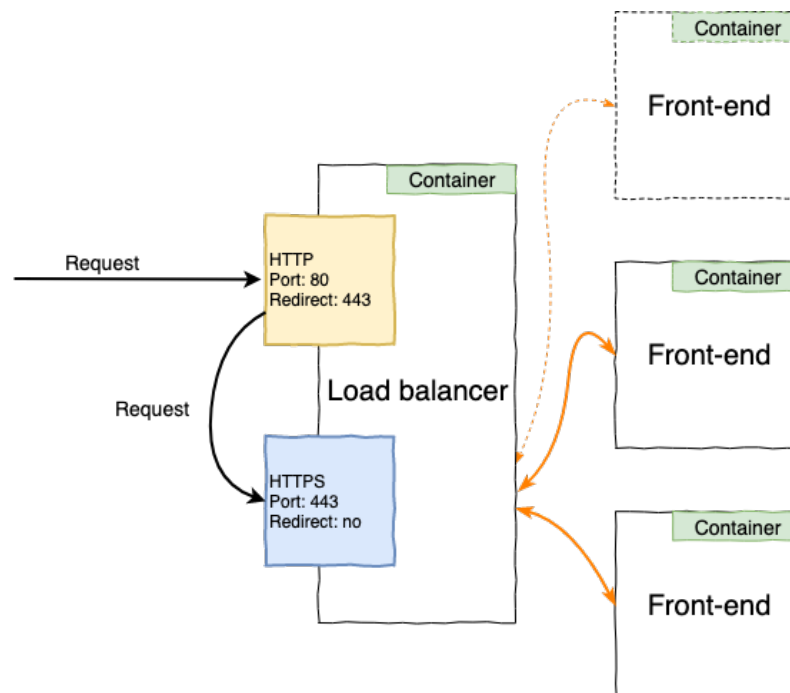


Figure 21. Forcing users to use secure connection

Port 80 is a well-known port for serving Web content back and forth a HTTP server with a high probability of being sniped by hackers. The same thing with port 443 but with a lower risk. There are still many attack techniques such as SSL Hijacking which leverages the vulnerability of the transition from HTTP to HTTPS to extract

for benefits /39/ or SSL/TLS downgrade attack which will tell the server to use a weaker version of SSL/TLS /40/. Even though the restaurant does not contain any sensitive information of customers, staff or payment transaction, a proper HTTPS configuration was implemented to first build the trust with any visitor lands on the site and maintain a good level of security for the system.

To prevent any possible hijack even further, enforcing server-side cipher suite was also applied, such as Diffie-Hellman, Galois/Counter Mode (GCM), and the use of weak cipher was avoided such as DES/3DES, RC4. The latest SSL/TLS is also preferable on several latest releases of modern browsers for example. Chrome, Firefox, Safari, and Edge.

However, using SSL/TLS requires certificate distributed by a CA. This is made possible thanks to an open CA belonging to the non-profit ISRG whose goal is to make secure Internet available everywhere by offering free certificates as well as delivering a certificate under an automation process with Certbot. Certbot, a project of Electronic Frontier Foundation (EFF), is designed to protect the freedom and privacy of people on the Internet. Once noticeable thing when using a certificate from Let's Encrypt is that there is no organization name displayed next to the padlock symbol. That is the visible difference between an Extended Validation (EV) and Domain Validation (DV) certificates. Let's Encrypt does not offer EV or Organization Validation (OV) certificates as the issuance process cannot be automated.

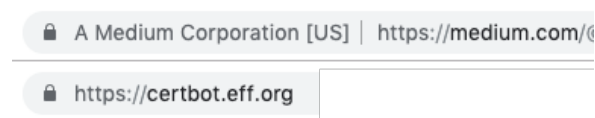


Figure 22. Difference between an EV certificate (top) and DV certificate (bottom)

Furthermore, HTTP/2 is also more preferable than HTTP/1.1 in any cases. By enabling HTTP/2, the site is able to deliver the content faster by using multiplexing, redundant header fields compression, and proactive server pushing for resources. Almost all current releases of Chrome, Firefox, Safari and Edge support HTTP/2.

HTTP/2 does not use GZIP for header compression due to the security exploitation in data compression with Compression Ratio Info-leak Made Easy (CRIME). Instead, a new header-specific compression scheme was created for compression mechanism within HTTP/2. The load balancer of the restaurant was configured to use GZIP over SSL/TLS connection but only for some certain types of file which does not contain any security value to extract.

Figure 23 shows the weight of the first package returned after making the two separate requests to the web application of the restaurant using Linux curl command. The only difference of the first request over the other is it did not use GZIP.



```
URL='https://ravintolafit.fi'
PLAIN="$(curl $URL --silent --write-out "%{size_download}\n" --output /dev/null)"
GZIPPED="$(curl $URL -H "Accept-Encoding: gzip, deflate" --silent --write-out "%{size_download}\n" --output /dev/null)"

echo $PLAIN vs $GZIPPED
> 54413 vs 11355
```

carbon  
carbon.now.sh

Figure 23. Comparison between uncompressed and compressed responses

GZIP helps the first data package to reduce almost 5 times in size from 54413 kilobytes (kB) down to 11355 kB. It was tested in the CLI by means of a bash script with the help of the curl command as shown in the figure.

### 4.3 Back-end

With current experience and knowledge in scripting language, the back-end was decided to be built in the form of a lightweight, yet high performance Express.js

server thanks to the enormous advantages of Node.js along with its ecosystem. However, there are many versions of Javascript, such as ECMAScript 2015, ECMAScript 2016, ECMAScript 2017, that make it hard to develop an application with new syntaxes of the language while keeping things backward compatible with older versions. Hence, it would be a nightmare without the support of Babel Javascript compiler which will convert any new syntax to the one any browser can understand. This would bring the freedom and comfortability to those, who work on it.

The back-end is designed to resolve requests not only from the front-end but the third-party requests going through the load balancer as well. Requests are served via REST API while any data inquiry sending to the database is made possible via Postgraphile as shown in Figure 24 below. Postgraphile can establish a GraphQL server itself while mapping all tables, indexes, relationships, functions, and views from a given PostgreSQL database.

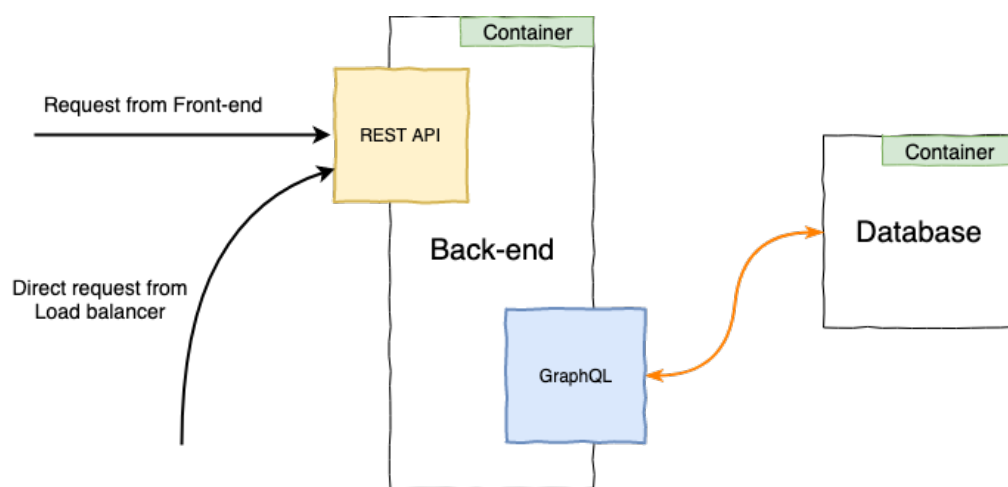


Figure 24. Back-end handles requests back-and-forth

Basically, Postgraphile acts like a dependency sits in back-end and directly serves any request with GraphQL API to the front-end. That is achievable, straightforward and brings least setup to the back-end but it also creates problems in some cases when data needs to be handled with some filtering functions or changes in structure due to the authentication or other security reasons. Alternatively, some flexibility can be added by only using Postgraphile to discover all the data living inside the

database, then taking any appropriate portion of data needed to serve the request, customizing and processing before sending the response back to the client via REST API.

#### **4.4 Database**

The database is the simplest piece of software in the structure as it is a PostgreSQL server persisting data in several different shapes. There are not too many functions and tables in the database for the current setup but a good candidate can be mentioned for a short introduction, the random identification (ID) generation function.

Its purpose is to generate long random ID for any user who has hands on the system to avoid using increasing subsequence of numbers. A PostgreSQL extension called pgcrypto, which can be utilized for hashing and encrypting, is used to help making random ID at a required length.

Theoretically, a random string can be generated in the back-end and forwarded to the database for persistence, however, the ID string can also be produced dynamically inside the database. In fact, the latter approach yields some benefits such as reducing security thread for persistent ID being transferred from one to another container, other tables can also use the function by way of a shared method to generate their own IDs in many different lengths if necessary. Furthermore, the overhead is excluded from the back-end.

The database was designed in the way that it can handle the bilingual information in Finnish and English, the nested categories of the menu dishes, and the hour scheme for both the opening hour display and the future booking integration. There are many ways to achieve those but they were designed and made with the present thought of best practices, as well as different sorts of normalization.

#### **4.5 Potential Services**

This system is built so that there should not be any barrier in the ability to extend or integrate services to the system in the near future. Based on the need of the

customer, any services can be part of the system in the form of containers without effecting the running ones. Some consideration services to serve specific needs such as Grafana and Elastic Stack.

Grafana is an open-source metric analytics and visualisation suite, it provides a beautiful dashboard for management users to have a look at the metrics of running services in the system, or create their own filters depends on their demand. Hence, the owner can have an appropriate react or adjustment to the event.

Elastic Stack is an open-source compound of ElasticSearch, Logstash and Kibana. Firstly, any persistence data will be sent to Logstash to ingest, parse, transform and transport data to ElasticSearch. ElasticSearch gives its users the ability to collect, search, filter, analyse gigantic amount of data in the matter of seconds. Kibana also provides a visualized dashboard, the owner can take the advantage of a powerful stack while generating more business value in the end.

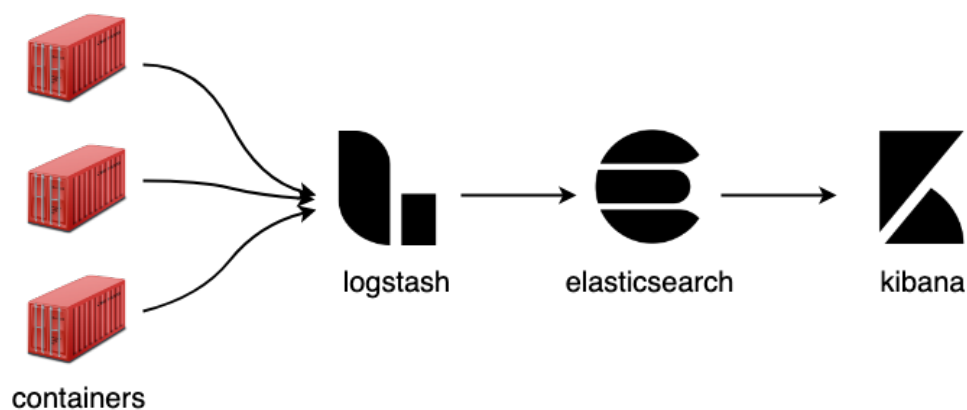


Figure 25. Data journey for collecting, managing and supervising /41/

This means any kinds of data or logs from different containers within the system can be dumped into a single place for managing and supervising. There might be some abnormal things which will unveil themselves in some perspectives when filtering, or doing the data combination for visualization.

Elastic stack benefits the owner the most if they are planned to grow big in the future. Otherwise, small scale operation will not gain any steep help while facing a large amount of work for setup and maintenance.



## **5 INTERGRATIONS, ADJUSTMENT AND PERFORMANCE**

Since marketing is an important part of a growing business, Google Analytics (GA) was installed in the latest requirement. GA is a service provided by Google to track and create reports based on traffic, geolocation, devices, and such. A Google Tag is required to be placed on the web application so the behaviours of the user will be tracked and sent to Google server for analysis.

From very beginning, the system was set up to see if there is any need to keep the entire application run on two nodes all the time. After one month collecting user footprints, it was discussed and considered to scale the application down to run solely on a node to reduce the monthly cost of renting hardware from PaaS provider.

The site performance of the restaurant was measured using Lighthouse inside Chrome browser and the calculation was done on the web application as a whole. Lighthouse is open-source and does the audits automatically on many criteria such as performance, accessibility, best practices, Search Engine Optimization (SEO), and Progressive Web App (PWA) against the web application.

The tool is located in terms of a separate tab in Chrome developer tool, reveals many different stats a developer needs to take care. However, in some cases, a few requirements from Lighthouse will be ignored intentionally due to the limitations in operation or unsupported feature on the web application. One of those could be images loaded from Instagram. While Lighthouse assesses these images it can save a part of their sizes to increase the load performance, and nothing can be done to decrease the size of images from a third-party source.

## 6 SECURITY

The machine on PaaS has its own firewall which is known as a cloud firewall and can be used to lock all the ports except the necessary ones. Furthermore, the Uncomplicated Firewall (UFW) can also be enabled from inside of the machine to only open the essential ports.

But one thing should be understood correctly, UFW is a host-based tool which is the default firewall configuration tool for Ubuntu while the PaaS cloud firewall is a network-based tool with the ability to apply the same set of rules for several groups of machines on PaaS. With this acknowledged, if UFW is considered, every single machine within PaaS should be configured to enable UFW one by one. That is unnecessarily tedious and the restaurant containers did not go with UFW but the cloud firewall instead as they might scale down at some point for relocation, upgrade or technical issue.

In addition, Docker Swarm has its own mechanism to strengthen its security over containers, internal communication via TLS. Docker Swarm demands its nodes to have TLS authentication to make the communication more secure. The user does not need to take any action to enable this feature since it is the default implementation. Other options are external 3<sup>rd</sup> party CA, internal corporate CA or self-signed CA where the last one is considered the least secure but the easiest to configure.

Besides, the secret key within the container cannot be accessed from the outside. It is encrypted during the transmission and only readable to the running services which were assigned.

Moreover, dividing Docker Swarm networks were divided into different zones to have one more security layer. This helps preventing the potential risk of unauthorized accessibility as mentioned in orchestration implementation.

These security performances are used to make different parts of the system more secure, hence, increase the overall security of the whole system.

## 7 SUMMARY

This thesis describes the development process of Ravintola Fit digital premise from the basic requirements to fully functional web application. This will definitely contribute to the business development of the restaurant by acting as an online base to promote what is being served, prices, opening hours, and more to the customers in advanced while tracking traffics and gathering feedbacks to input the next move.

Starting from some fundamental demands as the web application should show the information, such as detailed menu, map, and opening hours to using new technology, so it is stable enough, loads fast and has least barriers to do the integration or continue working in the future. Since the back-end, development and operations has no specific requirements on their own, an architecture was designed and proposed to the owner; thus a big picture of the whole system was agreed from both sides. Firstly, the orchestration was built with the help of Docker and its ecosystem.

The whole system was developed with the ability to scale, security and flexibility in mind so the application will not get locked into any cloud provider, while it still guarantees the functionalities are maintained and the system is secure at the same time. Yet, it might not be the best design in the world for such the application starting from small scale but rather fit to the specific needs of the owner and so much effort had been put into researching, learning, experimenting before the system took its first step.

In the conclusion, the web application meets the demands and the expectation of the owner since it loads fast, does not flicker, and is good-looking and low cost. From another perspective, the functionalities were added over time while the communication and delivery schedule were also maintained. There were some more functionalities in the roadmap and the system will continue being developed and delivered in the near future.

## LIST OF REFERENCES

- /1/ Microsoft. What is cloud computing? Accessed 01.11.2018.  
<https://azure.microsoft.com/en-in/overview/what-is-cloud-computing/>.
- /2/ Eastman Kodak Company. For Print Service Providers, the Sky's the Limit with Cloud Computing. Accessed 05.11.2018.  
[https://www.kodak.com/us/en/prinergy-workflow/Blog/Blog\\_Post/?ContentId=4295002215](https://www.kodak.com/us/en/prinergy-workflow/Blog/Blog_Post/?ContentId=4295002215).
- /3/ Wikipedia. Platform as a Service. Accessed 15.11.2018.  
[https://en.wikipedia.org/wiki/Platform\\_as\\_a\\_service](https://en.wikipedia.org/wiki/Platform_as_a_service).
- /4/ The getsix® Group. The Types of Cloud Computing. Accessed 19.11.2018.  
<https://getsix.eu/resources/glossary/the-types-of-cloud-computing/>.
- /5/ Wikipedia. Hypertext Transfer Protocol. Accessed 04.12.2018.  
[https://en.wikipedia.org/wiki/Hypertext\\_Transfer\\_Protocol](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol).
- /6/ Fores, M. 2016. A Beginner's Guide to HTTP/2 and its Importance. Accessed 14.12.2018.  
<https://www.advancedwebranking.com/blog/beginners-guide-to-http2>.
- /7/ Internet Engineering Task Force (IETF). Proposed standard - HPACK: Header Compression for HTTP/2. Accessed 15.12.2018.  
<https://tools.ietf.org/html/rfc7541>.
- /8/ Internet Engineering Task Force (IETF). Hypertext Transfer Protocol Version 2 (HTTP/2). Accessed 16.12.2018.  
<https://tools.ietf.org/html/rfc7540#page-60>.
- /9/ Wikipedia. Transport Layer Security. Accessed 15.12.2018.  
[https://en.wikipedia.org/wiki/Transport\\_Layer\\_Security](https://en.wikipedia.org/wiki/Transport_Layer_Security).
- /10/ IBM Knowledge Center. An overview of the SSL or TLS handshake. Accessed 07.12.2018.

- [https://www.ibm.com/support/knowledgecenter/en/SSFKSJ\\_7.1.0/com.ibm.mq.doc/sy10660\\_.htm](https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_7.1.0/com.ibm.mq.doc/sy10660_.htm).
- /11/ Google Security Blog & Emily Schechter, Chrome Security Product Manager. 2018. A secure web is here to stay. Accessed 19.12.2018.  
<https://security.googleblog.com/2018/02/a-secure-web-is-here-to-stay.html>.
- /12/ Wikipedia. Let's Encrypt. Accessed 23.12.2018.  
[https://en.wikipedia.org/wiki/Let%27s\\_Encrypt](https://en.wikipedia.org/wiki/Let%27s_Encrypt).
- /13/ Fielding, T. Roy. 2000. Architectural Styles and the Design of Network-based Software Architectures. Accessed 01.01.2019.  
[https://www.ics.uci.edu/~fielding/pubs/dissertation/rest\\_arch\\_style.htm](https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm).
- /14/ Wikipedia. Representational state transfer. Accessed 10.01.2019.  
[https://en.wikipedia.org/wiki/Representational\\_state\\_transfer](https://en.wikipedia.org/wiki/Representational_state_transfer).
- /15/ The PostgreSQL Global Development Group. PostgreSQL 11.2 Documentation. 1996-2019. Accessed 17.01.2019.  
<https://www.postgresql.org/docs/11/index.html>.
- /16/ Heroku Dev Center. PostgreSQL Concurrency with MVCC. Accessed 22.01.2019. <https://devcenter.heroku.com/articles/postgresql-concurrency>.
- /17/ Wikipedia. PostgreSQL. Accessed 10.11.2019.  
<https://en.wikipedia.org/wiki/PostgreSQL>.
- /18/ Salesforce. Heroku Addons. Accessed 10.11.2019.  
<https://elements.heroku.com/addons/heroku-postgresql>.
- /19/ The PostgreSQL Global Development Group. Chapter 9. Multi-Version Concurrency Control. Accessed 10.11.2019.  
<https://www.postgresql.org/docs/7.1/mvcc.html>.
- /20/ Gillam, Benjie. 2019. Accessed 18.01.2019.  
<https://www.graphile.org/postgraphile>.

- /21/ Open source Community. PostGraphile Github. Accessed 29.01.2019.  
<https://github.com/graphile/postgraphile>.
- /22/ StrongLoop, IBM, and other expressjs.com contributors. Express - Fast, unopinionated, minimalist web framework for Node.js. Accessed 03.02.2019. <https://expressjs.com>.
- /23/ Node.js Foundation. Don't Block the Event Loop. Accessed 04.02.2019.  
<https://nodejs.org/es/docs/guides/dont-block-the-event-loop/>.
- /24/ freeCodeCamp.org & Kevin Kononenko. 2017. Going out to eat and understanding the basics of Express.js. Accessed 11.02.2019.  
<https://medium.freecodecamp.org/going-out-to-eat-and-understanding-the-basics-of-express-js-f034a029fb66>.
- /25/ Wikipedia. Nginx. Accessed 18.02.2019.  
<https://en.wikipedia.org/wiki/Nginx>.
- /26/ mcarbonneaux. FastCGI.com Archives. Accessed 25.02.2019.  
<https://fastcgi-archives.github.io>.
- /27/ Nginx Inc. Choosing an NGINX Plus Load-Balancing Technique. Accessed 26.02.2019. <https://www.nginx.com/blog/choosing-nginx-plus-load-balancing-techniques/>.
- /28/ Kinsta Inc. What Is Nginx? A Basic Look at What It Is and How It Works. Accessed 27.02.2019. <https://kinsta.com/knowledgebase/what-is-nginx>.
- /29/ Wikipedia. Docker (software). Accessed 03.03.2019.  
[https://en.wikipedia.org/wiki/Docker\\_\(software\)](https://en.wikipedia.org/wiki/Docker_(software)).
- /30/ Wikipedia. aufs. Accessed 10.11.2019. <https://en.wikipedia.org/wiki/Aufs>.
- /31/ Docker Inc. Images and Layers. Accessed 04.11.2019.  
<https://docs.docker.com/storage/storagedriver/#images-and-layers>.

- /32/ Docker Inc. What is a Container? Accessed 15.03.2019.  
<https://www.docker.com/resources/what-container>.
- /33/ Docker Inc. DOCKER 101: INTRODUCTION TO DOCKER WEBINAR RECAP. Accessed 05.03.2019. <https://www.docker.com/blog/docker-101-introduction-docker-webinar-recap>.
- /34/ Docker Inc. Docker overview. Accessed 16.03.2019.  
<https://docs.docker.com/engine/docker-overview/>.
- /35/ Mazin, Arnaud. 2014. Docker registry first steps. Accessed 23.03.2019.  
<https://blog.octo.com/en/docker-registry-first-steps/>.
- /36/ Docker Inc. Swarm mode overview. Accessed 29.03.2019.  
<https://docs.docker.com/engine/swarm>.
- /37/ Gupta, Arun. Clustering Using Docker Swarm 0.2.0 (Tech Tip #85). Accessed 27.03.2019. <http://blog.arungupta.me/clustering-docker-swarm-techtip85/>.
- /38/ Docker Inc. Docker Guides. Accessed 28.03.2019.  
<https://docs.docker.com/v17.09/get-started/part5>.
- /39/ Sanders, Chris. 2010. Understanding Man-In-The-Middle Attacks - Part 4: SSL Hijacking. Accessed 01.04.2019. <http://techgenix.com/understanding-man-in-the-middle-attacks-arp-part4/>.
- /40/ Team Cinnamon. Downgrade Attacks. Accessed 09.04.2019.  
<https://tlseminar.github.io/downgrade-attacks/>.
- /41/ Berman, Daniel. 2017. Docker Logging with the ELK Stack – Part One. Logz.io. Accessed 11.04.2019. <https://logz.io/blog/docker-logging>.