SRH University Heidelberg

Master's Thesis

High Performance Cluster Simulation

Author:	Raman, Suruthi
Matriculation Number:	11012454
Address:	305,Willy Andreas Allee 5,
	Karlsruhe,
	Germany.
Email Address:	suruthiram19@gmail.com
Supervisor:	Prof. Dr. Achim Gottscheber (at SRH)
	Prof. Benedict, Dominic Savio (at SAP)
Begin:	01. October 2020
End:	25. March 2021

Declaration

I, Suruthi Raman, hereby declare that,

- The work done in this research project is my original work and does not contain any other people's work without being stated.
- Neither the whole work nor any part of this work is being submitted for another degree.
- The definitions and contents of the work of others have been clearly cited and indicated in reference.

R. Suruthi. ,25.03.2021

Signature of the student, Date

Abstract

In this project, two node applications are created using two different ports on the docker image. These two docker applications are configured in the third application using HA proxy method so that the two applications on the different ports will be running on the same port. HA proxy balances the workload between multiple servers to improve performance and reliability of servers. Applications are built in the docker image and executed on the docker container. This is how the high performance cluster simulation process is carried out.

Kurzfassung

In diesem Projekt werden zwei Knotenanwendungen erstellt, die zwei verschiedene Ports auf dem Docker-Image verwenden. Diese beiden Docker-Anwendungen werden in der dritten Anwendung mit der HA-Proxy-Methode konfiguriert, sodass die beiden Anwendungen auf den verschiedenen Ports auf demselben Port ausgeführt werden. HA-Proxy gleicht die Arbeitslast zwischen mehreren Servern aus, um die Leistung und Zuverlässigkeit der Server zu verbessern. Die Anwendungen werden im Docker-Image erstellt und auf dem Docker-Container ausgeführt. Auf diese Weise wird der Prozess der Hochleistungscluster-Simulation durchgeführt.

Notations and Abbreviations

OS	Operating System
VM	Virtual Machine
HV	Hypervisor
DE	Docker Engine
DC	Docker Container
I/O	Input-Output
CPU	Central Processing Unit
Cgroup	Memory limit
PaaS	Platform as a service
CI	Continuous Integration
CD	Continuous Delivery
UNIX	Uniplexed Information Computing System
API	Application Programming Interface
CLI	Common Line Interface
Json	JavaScript Object Notation
TLS	Transport Layer Security
DF	DockerFile
NPM	Node Package Manager
Btrfs	Filesystem for Linux(CoW)
Zfs	Zettabyte filesystem
Aufs	Multilayer filesystem
SHA	Secure Hash Algorithm
ARGS	Arguments
TCP	Transmission Control Protocol
HTTP	HyperText Transfer Protocol

ACLs	Access Control Lists
IP	Internet Protocol
SSL	Secure Sockets Layer
Maxconn	Maximum Connection
BE	Backend
FE	Frontend
Cfg	Configuration
DNS	Domain Name server
ECMA	European Computer Manufacture Association
LB	Load Balancer
RR	Round Robin

Contents

List of Figures

Chapter 1

1.1 Introduction:

The aim of modern software development is to run all of the programs on the same port or cluster to avoid interfering with the operations of other programs. Many computer systems linked to a group and operating as a single entity are simulated using cluster simulation. The field technologies Docker are included in this thesis. This type are used for deploying purpose . Docker is a containerization framework that uses Kubernetes to generate a image of a utility and the dependencies needed to run it.

Containers have an extremely environmentally friendly and granular system for combining software into the form of utility and keeping one's software program additives up to date and maintained, operating system store packages . It standardizes the distribution of utility applications by allowing apps to run as a field of the host. When a field is used, it has the most capacity to quickly example any field, allowing for rapid scale-up with the use of instancing a brief time span assignment within the form of a field. In this factor of view, the instantiating a field photo, need to be dealt with withinside an equal manner as a technique(like a provider or net app), despite the fact that while going for walks more than one times of the equal photo throughout more than one host servers, every field to run in a one-of-a-kind host server in different fault domains, for reliability.

Docker is accustomed to building a Docker image of a utility with the aid of a DockerFile. A Dockerfile contains all of the steps required to create the final image for deployment and distribution. The field image that are taken are reusable indefinitely. The images are used by Kubernetes to deploy the cluster. The use of Docker, it include the reusability of old developed resources and the quick setup of the environment, it's for testing or manufacturing process. The usage of Docker include the reusability of produced sources and the setup of the environment, it's for testing purpose. It is carried out using field technology, which is made possible by Docker and Kubernetes. The container generation is a relatively recent one that has been gaining popularity. (2018, Docker Company)

With a sector, all possible mismatches between unique software program variations and operating systems are eliminated. It enables to use a programming language and software device , can run it without problems within the confines . It is coupled with the use of implementation strategy, making the whole process agile, massively scalable, and most importantly-pipeline, which allows for an easily implemented production model of a utility to evaluate locally.

1.2 Lightweight virtualization

Based on a few different types of box ideas, such as having less assets and taking longer. It's a more versatile gear for packaging, submitting, and orchestrating each software program and application. Containers focused on existing virtualization developments allow for greater portability while also allowing for the use of running systems virtualization techniques. It offers an uncommon location for unique offerings to be run within the cloud platform. Virtualization systems claim to be able to deliver high overall efficiency.

In microservice-based totally applications, typical place workloads are represented. Using HTTP servers and a key-cost shop, it gains access to utility overall results. It tests similar characteristics such as startup time, image size, community latency, and memory throughout the deployment. Containers eliminate the ability to run a particular running device, as well as the strong distinction between digital systems. These procedures do seem to have their own dedicated unit, but it is unquestionably walking in a particularly remote environment.

The key advantages of containerization over full virtualization are the considerably lower device overhead during run-time and the specified time to install new times on a strolling host. The latter is especially important for automatically connect of large internet offerings.

1.3 Work flow of Containerization

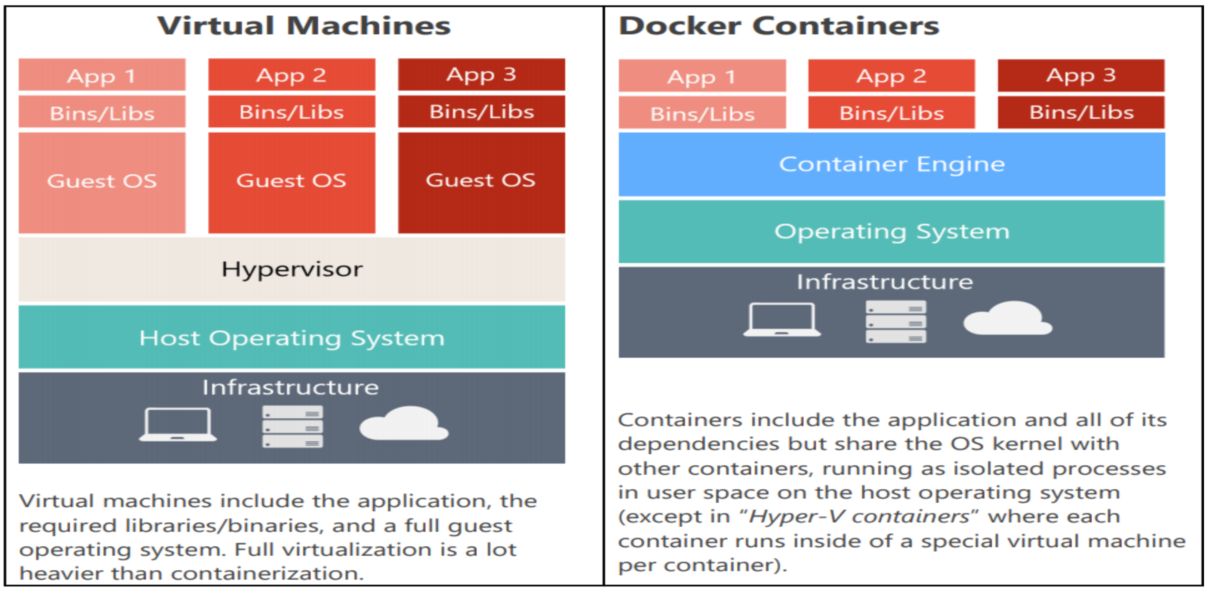
When doing something box-related, such as pushing or creating containers, there is a three-step process. It normally starts with a few different types of something that defines the package. This would be something like a Dockerfile in the Docker universe, and it could be a take place YAML in Cloud Foundry. Develop the actual image after that. As a result, the images that run on Docker may be a Docker Image.

The final element is a physical box that contains all of the runtimes and libraries needed to walk a utility that runs on virtual machines. A host gadget that runs, an after-run time engine (Docker Engine), and a few properties. After that, start distributing it as containers. It's a lot more light weight than deploying a couple of containers. In addition to the utility, it includes libraries. It allows continuous integration and distribution.

The Docker framework is a software framework that transfers resource control from the operating system to the hardware. It enables the container application's utility to be realized. Containers abstract the OS kernel from the entire hardware server. This method of virtualization will be active.

In most cases, a CONTAINERS will run until all of the operations specified in the IMAGE have completed. Containers can also run interactively on the console with a shell.

Fig 1 : VM vs Docker Container


Virtual Machines
App 1
App 2
App 3
Bins/Libs
Bins/Libs
Bins/Libs
Guest OS
Guest OS
Guest OS
Hypervisor
Host Operating System
Infrastructure
Virtual machines include the application, the required libraries/binaries, and a full guest operating system. Full virtualization is a lot heavier than containerization.
Docker Containers
App 1
App 2
App 3
Bins/Libs
Bins/Libs
Bins/Libs
Container Engine
Operating System
Infrastructure
Containers include the application and all of its dependencies but share the OS kernel with other containers, running as isolated processes in user space on the host operating system (except in "Hyper-V containers" where each container runs inside of a special virtual machine per container).

Traditional virtualization architecture

- The physical server that hosts virtual machines as the server.
- The host operating system that runs on a PC,(Linux or Windows)
- Virtual machines are hosted by hypervisor, both VMWare or Windows Hyper V.
- Then, as a Guest, install several operating systems as virtual machines on top of the current hypervisor, and host your applications on each Guest OS.

Dockers has enabled a new wave of virtualization.

- The physical server on which virtual machines are hosted as the server. As a result, this layer is unchanged.
- The base computer, such as Linux or Windows, is known as the host OS. As a result, this layer is unchanged.
- The Docker engine(DE), which is the next generation, has now arrived,it is used to run the OS, as well as run Docker containers in virtual machines.
- Docker containers(DC) are also used for many of the apps.

Each Docker container is typically a single process in terms of application architecture, which may be an entire app, a single service, or a microservice. When your application or service process runs within a Docker container, it contains all of its dependencies, meaning that it can be deployed in any Docker-compatible environment.

Docker containers have a range of benefits because they are sandboxes that run on the same shared OS kernel. They're simple to set up and use, and since they share the same kernel, they provide less isolation than VMs while using a fraction of the resources.

Both the host and guest operating systems are managed by the hypervisor. It's a virtual feature that enables you execute multiple systems on same server. It connects the operating system and the CPU. It is divided into two segments by virtualization: the first is Para-Virtualization, and the second is Complete Virtualization.

The docker tool is a form of operating system level virtualization that manages Linux containers, it shows that there are several isolated Linux containers on a single control host. The Linux kernel manages resources such as network, memory, CPU, and block I/O, as well as cgroups, without the need to start a virtualization machine.

Chapter 2

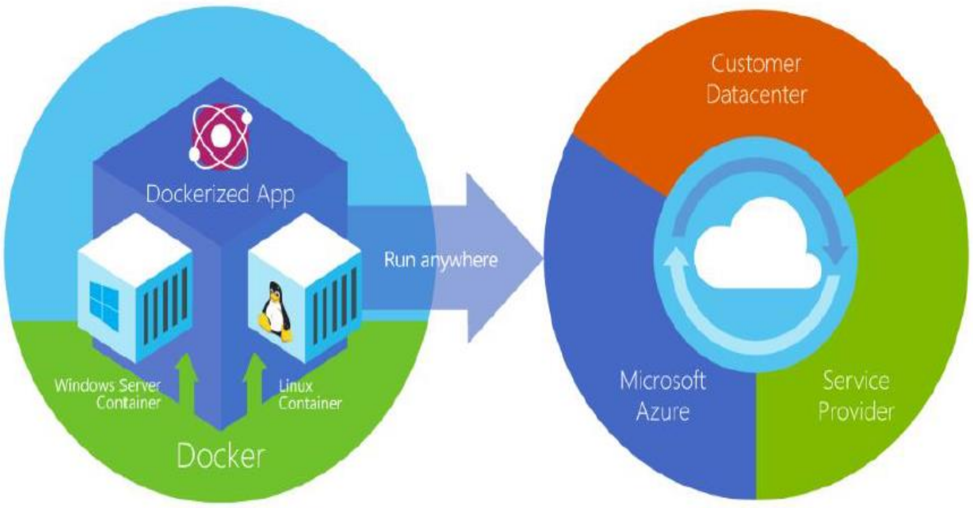
Docker

Docker is a container-based tool that simplifies the development, deployment, and execution of software. It allows a package and deploying of an application's specifications, including binaries and other sources. It enables you to decouple applications from infrastructure, allowing you to create apps more quickly. It's a group of system products that deliver apps in containers using OS-level virtual machines. The system that runs the container is known as Docker Engine. Because of the computer's customized settings for composing and checking code, the programmer should be assured that perhaps the program will run on every other windows computer.

Docker streamlines and speeds up our workflow while also encouraging builders to play with new equipment, utility stacks, and deployment environments for each project. On a single host, security allows multiple packaging containers to operate at the same time. It provides tooling and a forum for manipulating the packaging container's lifecycle.

- The use of packaging containers in the production of a utility and its assisting additives.
- It transforms into a system for dispensing and testing programs.
- If all is ready, place the utility as a container in the environment. If the surroundings are a neighborhood knowledge center or a cloud service, it works equally well.

Fig 2 : Docker deploys containers throughout the cloud's layers.



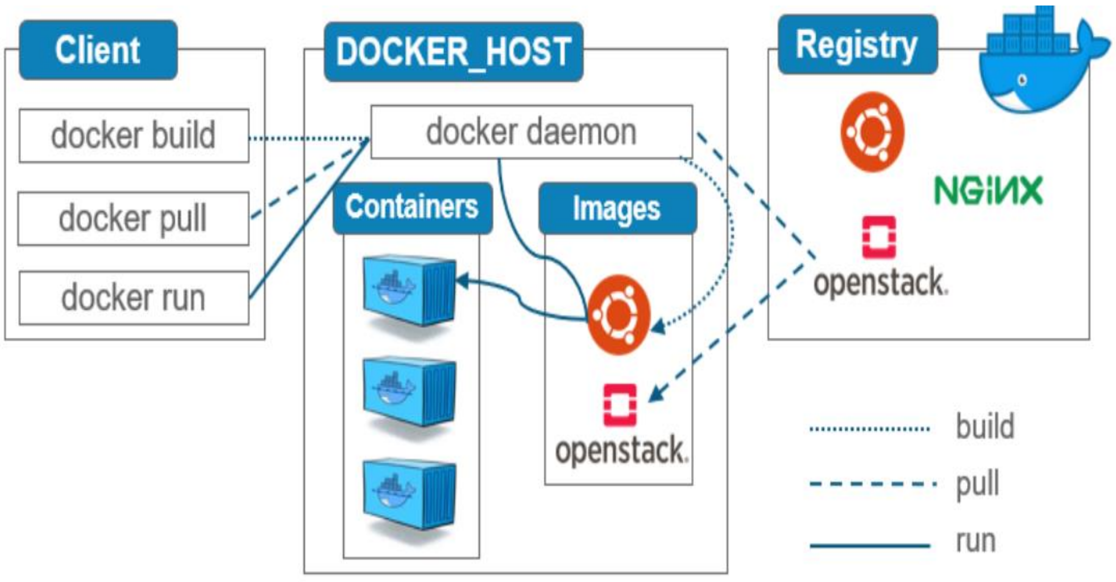
Programs are delivered quickly and on a daily basis. Containers are mainly for continuous integration and shipping (CI/CD) processes. It makes use of Docker to force its programs to look around and run automated and guided tests. Bugs discovered in the development platform can be rebuilt and re-deployed to the testing process for evaluation. Returning to the customer after the testing has completed is as easy as uploading the most recent image to the manufacturing environment.

It's flexible when it comes to implementation and scaling. Workloads are extremely transportable with container systems. It can run on a local laptop, in the cloud, or in a particular environment. It's simple to different workloads, scaling, and deliver new services . It walks more workloads on the same hardware, making it a more cost-effective alternative to a hypervisor-based totally digital computer.

2.1 Docker architecture

It's built around the consumer-server model. The User agent connects with the Docker daemon, which is in control of importing and issuing Docker containers. A docker user has the option of connecting to a remote docker daemon or running both the client and the daemon on the same machine. It describes how to use REST APIs in conjunction with UNIX sockets or group interfaces. It comes with both hard and soft containers, as well as features.

Fig 3 : Docker architecture



It's a type of software program that includes all of the software's dependencies and libraries. It is simple to install programs inside a running system's packing containers. It provides a full environment of equipment to create a field image, pulls certain image from registries, installs packing containers, and controls or orchestrates field walks through a cluster of machines.

It has made field adoption more common. It ensures a consistent runtime at all stages of a product's life cycle, including development, testing, and deployment. if docker is able to solve this complexity by providing a constant environment for the program.

In contrast to a digital computer, Docker packing containers are shorter and start up faster. By default, Docker Engine has built-in security.It is fueled by additives, including the fielded. The most powerful CLI and API for managing shipping containers. It enhances the Docker Engine's capabilities.

2.1.1 Docker Engine

It is the foundation of the Docker device and containerization age for designing and containerizing applications. Inside the docker engine, there are three additives in particular.

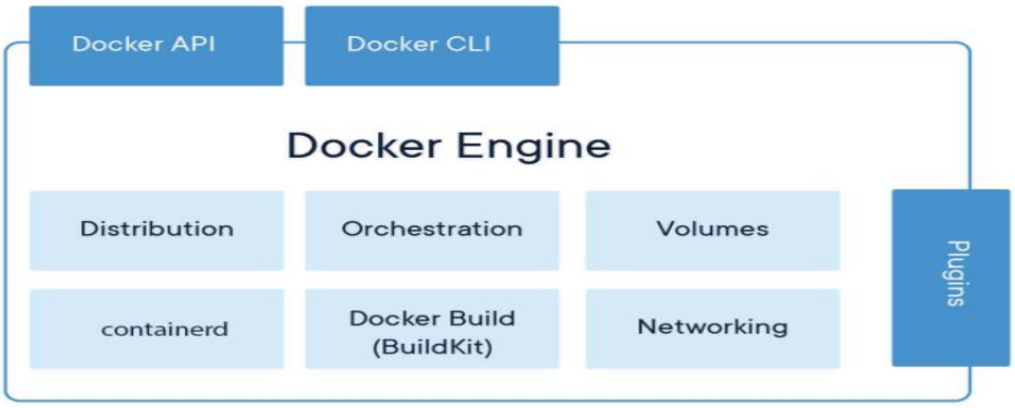
- To begin, there is a server docker daemon named docked. It allows and manages docker images, bins, and networks.
- Second, the Rest API was used to instruct the Docker daemon about what to do.
- Finally, there's the Command Line Interface (CLI) for entering Docker commands. It facilitates the tasks and workflows involved in developing, delivering, and running box-based applications.

Image, bins, networks, and garage volumes are all hosted by the Docker engine, which is a server-aspect daemon framework. It changes settings and conditions automatically to ensure that the actual country and the desired country are compatible at all times. Instead of deployment, each supervisor and employee node can be instantiated at runtime from a single disk image. It uses a declarative model to operate. It may make use of a number of plug-ins, such as images stored in the personal registry, a public supply on GitHub, or on DockerHub.

With the docker engine, it can manage the entire lifecycle of plug-ins, from installation to deletion. It simultaneously creates records volumes with box images and includes records copied from a figure image. Between containers, the volumes can be exchanged and reused.

It could enter docker bins going for walks on extraordinary machines using Docker Networking. Docker-compose is useful if our program covers many boxes. It has the access to start, stop, or other software's features. Docker swarm helps us to monitor a cluster of Docker engines if we orchestrate box through multiple hosts machines.

Fig 4 : Docker Engine



Docker provides simple tooling and packaging methods for bundling all software dependencies within a package that can then be run on Docker Engine. Containerized applications can now operate on any infrastructure.

Acceleration Transformation: The cornerstone of the Docker platform, enabling developers and operators to easily and securely reveal their ideas.

Other people's freedom: It works with every form of software, including cloud software and CRI-certified paintings.

Security: Since it is designed with safety in mind, it can run programs in remarkably controlled environments.

Customers may use Docker Engine's default group drivers to build custom bridge networks that manage box communication. Customers can draw up their own personal bridge networks without affecting box service, according to the company. Without an external key-fee store, Docker Engine networking and swarm mode will work together to build overlay networks on supervisor nodes. This overlay group is most efficient available to people nodes who need it for offerings, and it expands automatically to any new nodes entering the service.

Interfaces with a common line (CLIs)

In Docker, there are several CLIs:

Docker CLI : Docker It is the most commonly used CLI by Docker users. Docker kill, for example, is part of this CLI and adds a kill to the command. Docker API calls to Docker Daemon are converted into instructions.

Docker Daemon CLI: The Docker Daemon has its self command line interface, it uses the docker

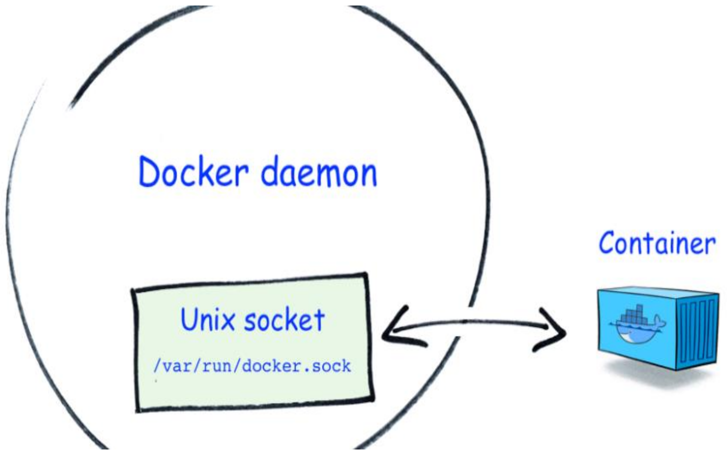
CLI for Docker Machine: The docker-machine command is part of this command-line.

This command docker-compose is part of the **Docker Compose CLI**.

2.1.2 Docker Daemon

It manages the Docker element and listens to Docker API calls. It can also interact with daemons to manipulate the Docker services environment. It gives access to a tool that is close to the typical subject-based completely swarm, which has successfully acted as a proxy for a cluster of Docker hosts. It also provides a location for the shared state, as well as the ability to restart bins in accordance with policy. It's also in charge of networks and volumes that are shared within a few bins. It is an appropriate location for one's equipment to be run as their own daemons.

Fig 5: Docker Daemon



The Docker daemon is started by way of a walking machine utility, not by way of a user. Automatically starting Docker at the same time as the tool reboots is much less complicated. Docker can be programmed to start automatically or manually using a docked button, used to start the daemon. It makes use of a JSON configuration file, holds all configuration in one location, and uses flags at the same time as it starts docked. We can use any of these choices together as long as we don't mention equal opportunity in the JSON file or as a flag.

TLS is used by the daemon in debug mode, and for ports that support this notion. It can also manually start the daemon and customize it with flags. It comes in handy when troubleshooting issues. All facts are saved in a single list by the Docker daemon. For each daemon, it is saved on this committed list. To take advantage of the exquisite list in the facts-root configuration choice. It enables debugging and troubleshooting on the daemon runtime hobby. If the daemon is not responding, it will upload strain to the entire stack trace of all threads.

2.1.3. Docker client

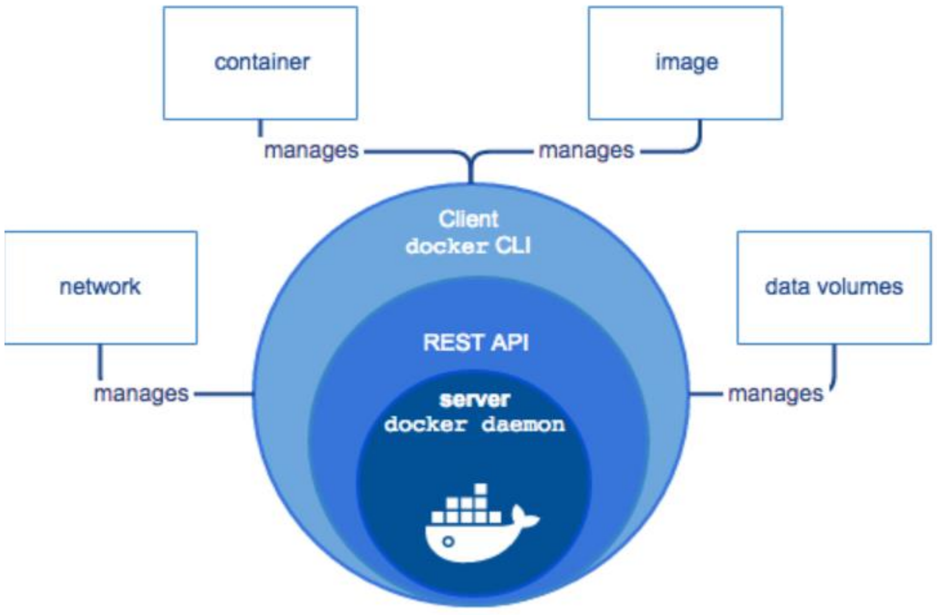
It is the most common way for dockers customers to communicate with docker. It makes use of the Docker API and uses instructions in combination with docker run. The patron sends certain instructions to docked. A couple of daemons are available to speak with the patron, interact with the Docker daemon. Using a method of naming natural surroundings is the best way to do this. It can configured by using the DockerClient .

The same environment variables are used by the docker command-line client. Using the URL to the Docker server, verify the host's CA certificate. There's also a course on how to list TLS certificates to use when connecting to a Docker host. A customer who wishes to connect with a Docker server.

Docker patron is used to communicate with the docker daemon in order to complete a constructive task such as jogging a container or creating a few image. Now it's your turn. In addition to the CLI, the Docker daemon exposes a REST API that can be used to gain access to providers exposed by the daemon. The docker client, which runs on a docker server, is designed to provide direct image retrieval from a registry. Clients using commands

- Docker build
- Docker pull
- Docker run

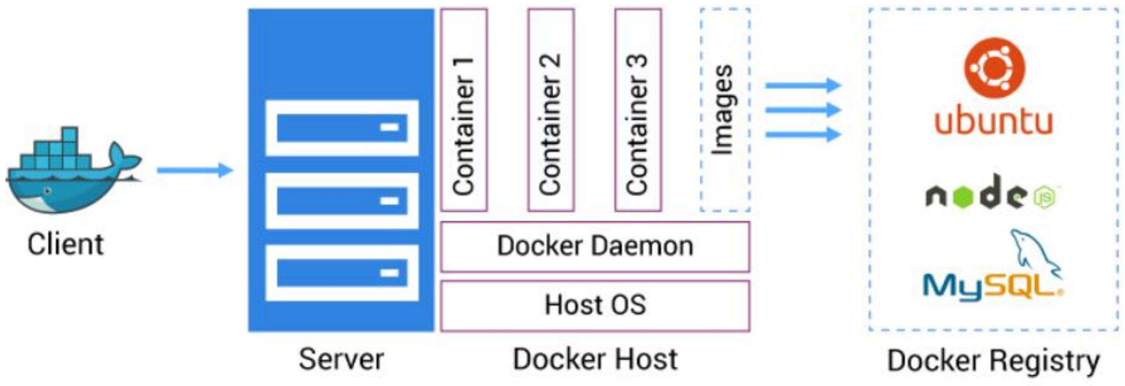
Fig 6 : Docker Client



Docker Server

Docker, as we all know, is a container management framework that does everything. It has a field runtime, managed field storage, networks, photo builds, and a number of other features. The bulk of these responsibilities necessitate the use of computing power. A Docker server is nothing more than a whole instance (VMs, computers, and so on) that can go for a stroll. Docker daemon method that inflip handles all docker-based services. It is responsible for walking the product and carrying out the responsibilities demanded by the client.

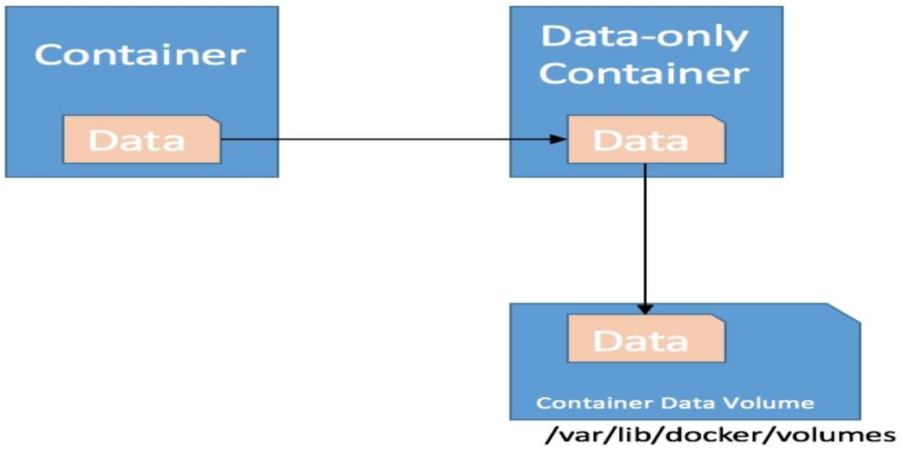
Fig 7 : Docker Server



Docker Host

It offers a full environment for executing and running applications. Docker daemon, Images, Containers, Network, and Storage are all included. The daemon is in charge of all container-related behaviour and receives information through CLI and API. It can also interact with daemons to control its operation. It fetches and maintains container images at the user's request, then helps in creating framework for a container command in a create file. A daemon that schedules other elements for the container's run may also be provided.

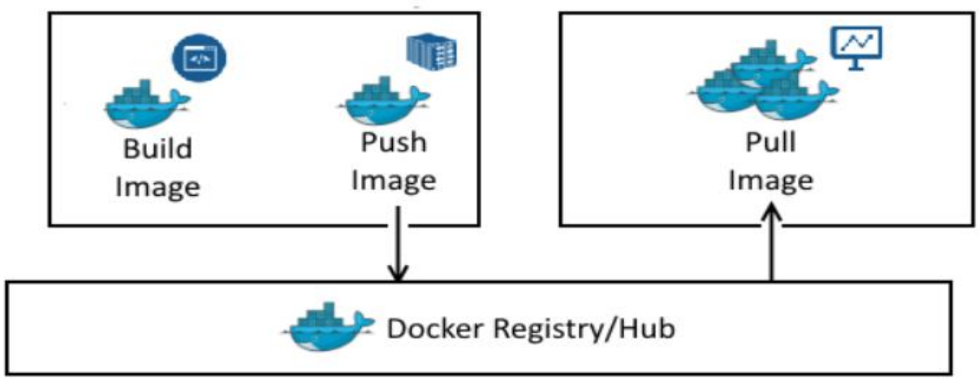
Fig 8: Host docker system



The Docker Hub is a publicly available resource for docker and its components. It's a Docker-based carrier for finding and exchanging field images. It is a repository of field images with a variety of content material properties, including field network developers, open supply tasks building and dispensing their codes in containers, and field network developers.

It provides non-public repositories with free public repositories for storing and exchanging images. It allows users to push and pull field photos as well as manage access to non-public field image repositories. Also, use high-quality field image provided by way of docker to robotically create field from GitHub.

Fig 9: Docker Hub



2.1.4. Docker Registries

Docker files are stored in a Docker registry. It's a stateless, highly scalable server-side framework that allows you to share Docker files. It's free and open source, with a permissive Apache license. It's possible to use a formal or informal docker database. Docker is a registry for anyone to use, and Docker is sets up to search it by image. It may be used to manage people.

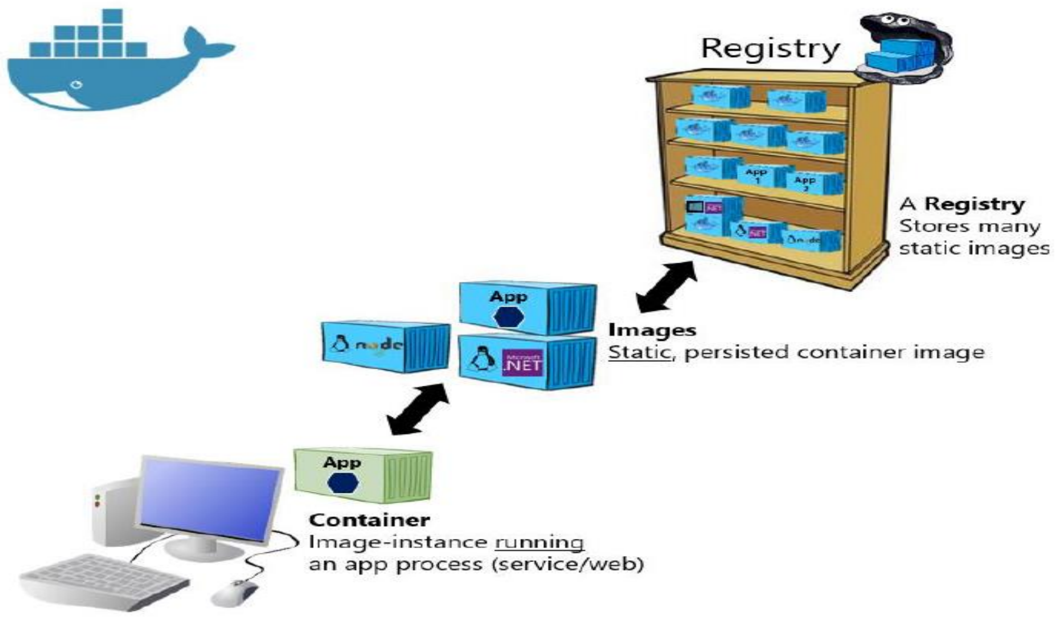
To use the Registry,

- you must: have complete control over where images are stored; and
- Own the entire image delivery pipeline. It closely incorporates image storage and delivery into the production process.
- To begin, start the registry, then pull or create some images from the hub, tag them so that they point to the registry, and push and pull them back.
- Finally, shut down the registry and delete all of the records.

2.2. Docker Objects

The data to docker objects mechanism includes images, containers, local daemons, sizes, networks, and swarm nodes and resources. It labels photos, keeps track of licensing details, and annotates connections between containers, amounts, and networks. It's a key-value pair with a string as the key. It assigns multiple labels to an object; each key value pair within an object must be unique.

Fig 10 : Basic Taxonomy of Docker

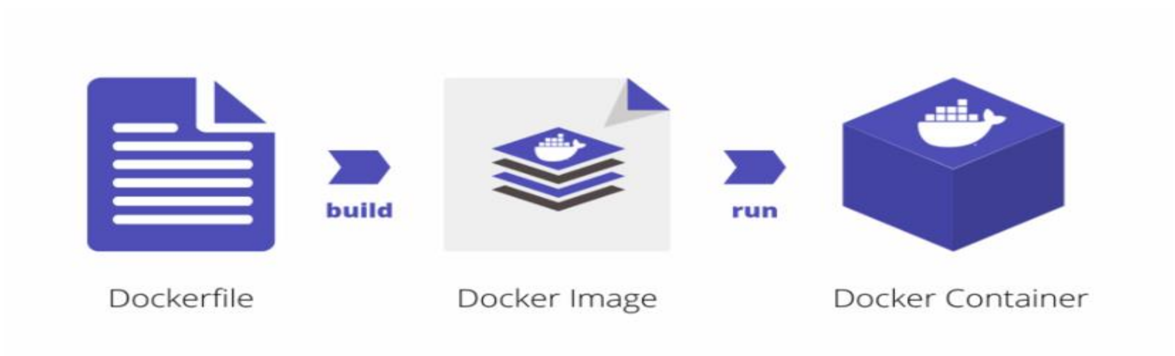


2.2.1 Docker Images

Docker Images are read-only models that provide instructions for constructing Docker containers. It provides a straightforward method for bundling applications and pre-configured server environments for private use or public sharing with other Docker users. Users can share and save images in both public and private repositories.

If you're new to docker, images are also a good place to start. It helps you to make your own unique photos. To create images for deploying code and assembling container-based services. Docker images can be downloaded from a Docker hub and used to start a new, modified docker file. You can make your own docker using a dockerfile.

Fig 11: Docker Images



2.2.2 Docker containers

Once a docker image is running, a docker container is created. This container contains all of the software and their environments. It can start, stop, and remove a Docker container using the Docker API or the CLI. It can attach storage to a container, connect it to networks, and create images based on the state.

To a great extent, a container is isolated from other containers and its host computer. It may monitor the degree to which a container network, storage, or other underlying subsystem is isolated from other containers or local machines. Its photos, as well as any configuration options, describe ,It enables us to build or begin it.

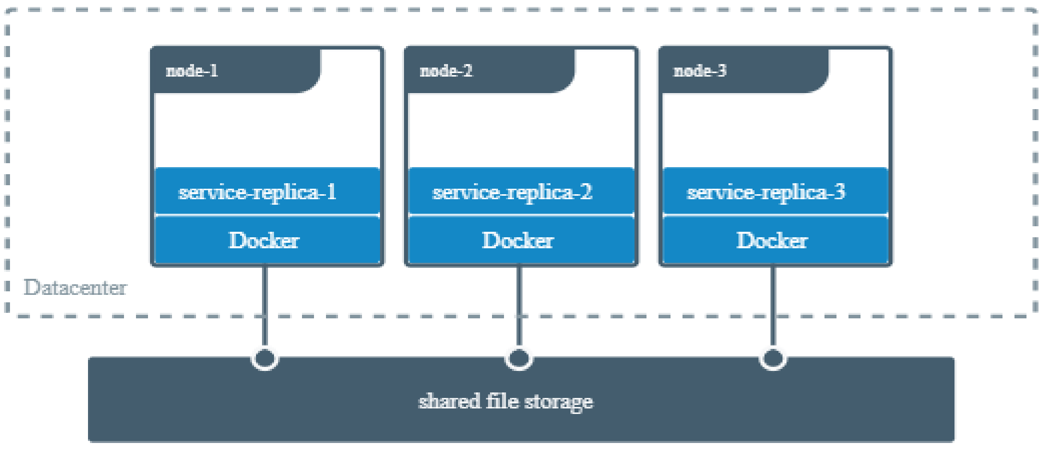
```
$ docker build -t app1 .
```

When you run this order, Container can pull the from the configured registry, or you can automatically execute a docker pull. Docker creates a new container that can be used to run docker container build, and it gives the container a read-write filesystem. It allows a container to build and change files and directories in the local filesystem while it is running. It provides network interfaces for the container to link to the default network. It also involves assigning an IP address to the container and, if using the exit button, stopping but not removing the container.

2.2.3. Docker Volumes

It's the most common method for storing data supported by Docker containers and used by them. Mounts are dependent on the host computer's directory structure and operating system, while volumes are managed entirely by Docker. To control volumes, it can use the Docker CLI commands from the Docker API. The volume directive in the Dockerfile and the docker run command on the command line are both used to install volumes. It makes a new volume available for containers to consume and store data. Docker assigns you a random name. Outside of the default union file structure, volumes are folders that reside on the host filesystem as regular directories and data. The content of a new volume may be pre-populated by a previous volume.

Fig 12 : Storage file system



Binds mounts from local hosts do even worse than Docker Desktop volumes. Containers share this resource. By storing volumes on remote hosts or cloud providers, it can encrypt their content and add other features.

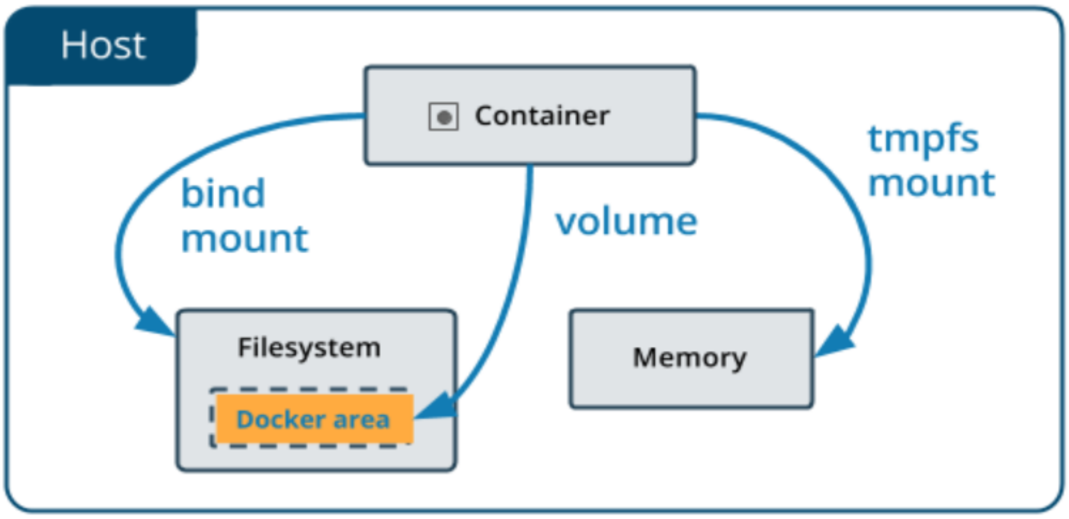
It provides the ability to create storage with the ability to create volumes, as well as a list of all volumes. The data volume is saved on the file system of the host computer.

Container for data volume: Another solution is to make a container host a volume and mount that volume to another container. This approach is independent of the application container and can be shared with another container.

Folder Mounts: Another method is to use a container to install a host's local directory. When it comes to directory mounts on the host machine used for source, the volume inside the docker volume folder is used. amplitude

Plugins for external storage: it allows you to link to external storage platforms. storage from the host to the data array All of this information can be found on the Docker plugin tab.

Fig 13: volumes



A container's capacity is not increased by using volume. It can use -v or -mount flag to start a container with a volume.

2.2.4. Docker Networks

Docker networking is a way for all of the isolated containers to connect with one another. Docker has a small number of network drivers.

A container's default network driver is

Bridge :When an application is running on containers, i.e. several containers communicating with the same Docker host, it uses this network.

Host: This driver breaks the network barrier between Docker containers and Docker hosts. There is no need for network separation between the host and the container when it is used.

Overlay: This component allows swarm services to communicate with one another. It's used while several applications are running containers on separate Docker hosts.

None: This drive switches off all network connections.

2.3.Docker Installation

Docker needs features found in recent Linux kernels to work properly, so on Mac OSX and Windows hosts, a virtual machine running Linux is needed. Currently, the most common way of installing and configuring this virtual machine is via Docker Toolbox, which uses VirtualBox internally, although there are plans to incorporate this functionality into Docker itself, using the operating system's native virtualization features. Docker is a native Linux program that runs on the host.

Docker can be used on a variety of platforms, including the following:

Server: Windows Server 2016 and various Linux distributions.

There are cloud providers such as AWS, Google Compute Platform, Azure, IBM Cloud, and others.

Download and run the installer from Docker for Windows.

Continue with the default options in the installer and enter your account credentials when prompted a submission.

Requirements :

Hardware Virtualization Technology is available on a 64-bit version of Windows 7 or higher on a system that supports it.Although the docker binary will run natively on Windows, you'll need a Linux virtual machine to build and host containers.1.00.20

Docker can now use Windows' native Hyper-V features to start up a small Linux machine to serve as a backend, so you don't need to install a separate VM after version 1.12.

2.3 Docker Desktop :

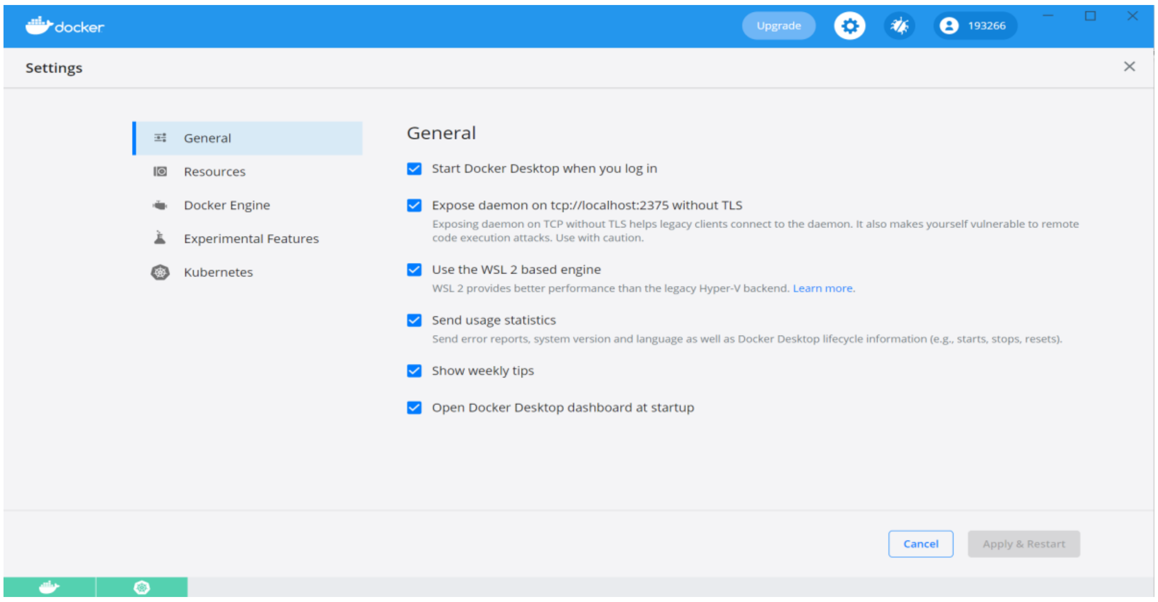
Docker has out-of-the-box support for Linux, but to run Docker on Windows, you'll need the following setup. Docker Desktop for Pc is a Windows 10-optimized edition of Docker. It's a native Windows framework that offers a simple development environment for shipping and running dockerized software. The easiest and most secure way to build Docker apps on Windows is to use windows-native Hyper-V virtualization and networking. It can run docker containers on both Linux and Windows. It allows you to create and share containerized apps and microservices. Docker Engine, CLI client, Docker compose, and Kubernetes are all included. Install and run the docker desktop first.

Docker ToolBox :

For Old versions of Windows 8.1, Docker ToolBox was introduced. and Windows 7, that require configuration in order to install Docker. Virtualization must be turned on.

- If Docker isn't already working, start it from the Start menu.
  After that, every terminal should be tightened up (either cmd or PowerShell)
- Once the terminal is up and running, sort
  $ docker run hello-world
- If all goes well, this should print a welcome note stating that the installation went smoothly.

Fig 14: Docker Desktop



General

- [✓] Start Docker Desktop when you log in
- [✓] Expose daemon on tcp://localhost:2375 without TLS
  Exposing daemon on TCP without TLS helps legacy clients connect to the daemon. It also makes yourself vulnerable to remote code execution attacks. Use with caution.
- [✓] Use the WSL 2 based engine
  WSL 2 provides better performance than the legacy Hyper-V backend. Learn more.
- [✓] Send usage statistics
  Send error reports, system version and language as well as Docker Desktop lifecycle information (e.g., starts, stops, resets).
- [✓] Show weekly tips
- [✓] Open Docker Desktop dashboard at startup

Chapter 3

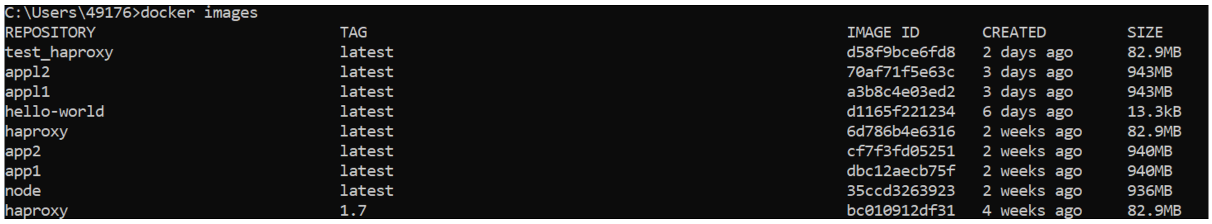
Docker Image steps

3.1.Docker Image

It's a read-only prototype for creating Docker containers. The Docker create components. It allows for the distribution of apps along with their runtime environment, all dependencies, and configuration required for them to run, removing the need to install packages and troubleshoot. Docker must be available on the target machine at all times. Docker can read instructions from a Dockerfile and generate images for you automatically. It's a text file with a straightforward and well-defined syntax. Images can be extracted and stored in either a formal or informal registry. The default tag for images is latest.

$ docker images

Fig : 15 : Docker Images



```
C:\Users\49176>docker images
REPOSITORY                   TAG                                                        IMAGE ID       CREATED        SIZE
test_haproxy                 latest                                                     d58f9bce6fd8   2 days ago     82.9MB
appl2                        latest                                                     70af71f5e63c   3 days ago     943MB
appl1                        latest                                                     a3b8c4e03ed2   3 days ago     943MB
hello-world                  latest                                                     d1165f221234   6 days ago     13.3kB
haproxy                      latest                                                     6d786b4e6316   2 weeks ago    82.9MB
app2                         latest                                                     cf7f3fd05251   2 weeks ago    940MB
app1                         latest                                                     dbc12aecb75f   2 weeks ago    940MB
node                         latest                                                     35ccd3263923   2 weeks ago    936MB
haproxy                      1.7                                                        bc010912df31   4 weeks ago    82.9MB
```

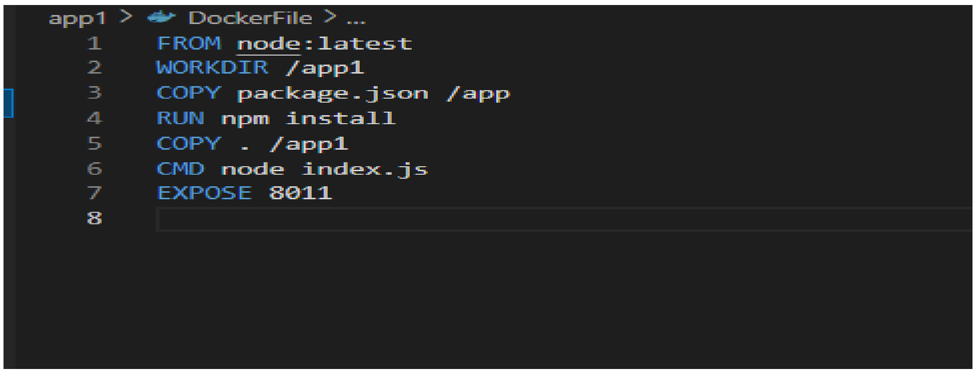
3.1.1Docker Image: Dockerfile

A Dockerfile and a background can be used to make an image.

- Dockerfile : instructions for putting the image together • Background : a set of files (i.e., applications , libraries)
- An image is often focused on another image.
- The syntax of Dockerfiles is similar to that of comment and instruction arguments.

The commands in a Dockerfile are executed in the file.

Fig 16: Docker File



```
FROM node:latest
WORKDIR /app1
COPY package.json /app
RUN npm install
COPY . /app1
CMD node index.js
EXPOSE 8011
```

Following any guidelines

COMING FROM: To select a parent image(mandatory)

RUNNING : Tell the docker daemon to run a command when creating images. Any instruction that is executed in a second pattern on top of the existing image will be accepted.

To change configuration files, use the ENV instruction.

EXPOSE : At runtime, the container listens on listed network ports.

When a container running,

CMD sets the default parameters. To include defaults for the container that executes.

WORKDIR : This specifies which directory on which the command must be executed.

EXPOSE: This command makes the container port clear.

3.1.2.Docker Image : build

A Dockerfile and a background are used by the docker construct command to create Docker images. The collection of files in the designated PATH or URL is referred to as a construct context. A COPY command may be used to refer to a file in the conext. Among many other things, these parameter may apply to Git repositories, before the tarball contexts, and text files.

Use this command to generate an image from a DockerFile: docker image construct [OPTIONS] URL Route -

To generate an image for a Nodejs application

'docker create app1' is a command to build an application using Docker.

BUILD state :

The FROM command tells Docker where to get the build image from. Based on the location, The operating system is extracted from Dockerhub and used as the framework for the final image.

The author's name and e-mail address were clearly stated in the MAINTAINER command. In this scenario, it's best to start over with Dockerfile.

The COPY command is used to copy objects from the source device to the container's target destination. It's used to specify a number of different outlets. It allows the image to copy the contents of the downloads directory to the image's root path.

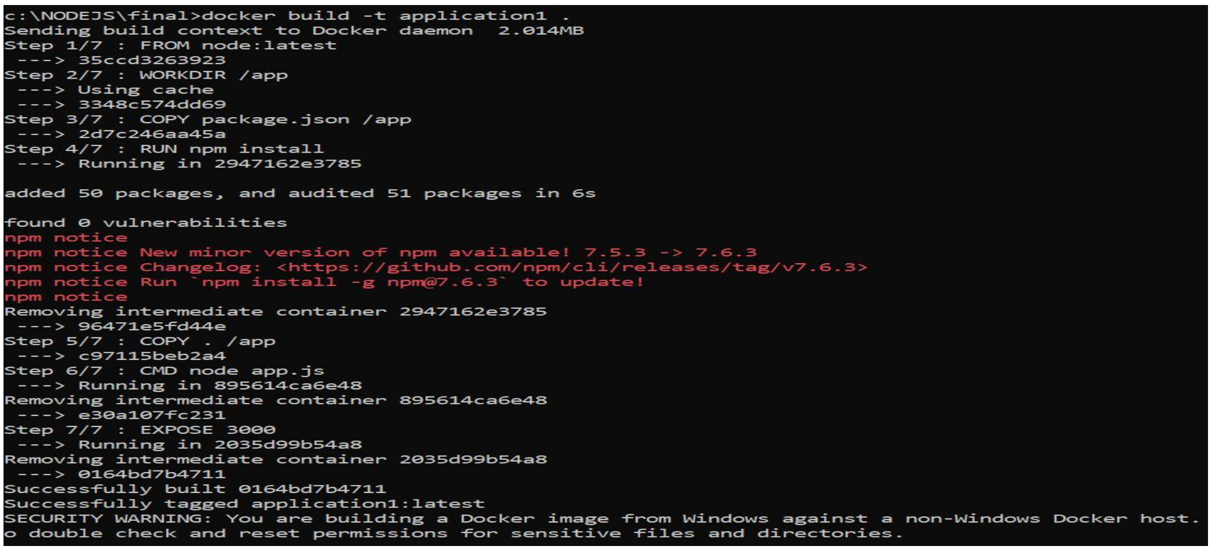
The ADD command is similar to COPY, but it helps you to remove a file from the source to the destination. When extracting a file into a particular Docker image directory.

RUN commands are executed inside a layer on the current image. The results are saved and used in the Dockerfile's subsequent measures. When it comes to downloading vital dependencies or updates. It must be modified, which necessitates the execution of an order. upgrade & upgrade.

The ONBUILD command inserts a instruction into the image, after executed. When making an image that will later be used as a foundation for other image. It stores instructions that will be executed in the next building stage.

DockerFile is used to generate the image of a container that will run a Nodejs program.

Fig 17 : Docker Image -Build



```
c:\NODEJS\final>docker build -t application1 .
Sending build context to Docker daemon  2.014MB
Step 1/7 : FROM node:latest
 ---> 35ccd3263923
Step 2/7 : WORKDIR /app
 ---> Using cache
 ---> 3348c574dd69
Step 3/7 : COPY package.json /app
 ---> 2d7c246aa45a
Step 4/7 : RUN npm install
 ---> Running in 2947162e3785

added 50 packages, and audited 51 packages in 6s

found 0 vulnerabilities
npm notice
npm notice New minor version of npm available! 7.5.3 -> 7.6.3
npm notice Changelog: <https://github.com/npm/cli/releases/tag/v7.6.3>
npm notice Run `npm install -g npm@7.6.3` to update!
npm notice
Removing intermediate container 2947162e3785
 ---> 96471e5fd44e
Step 5/7 : COPY . /app
 ---> c97115beb2a4
Step 6/7 : CMD node app.js
 ---> Running in 895614ca6e48
Removing intermediate container 895614ca6e48
 ---> e30a107fc231
Step 7/7 : EXPOSE 3000
 ---> Running in 2035d99b54a8
Removing intermediate container 2035d99b54a8
 ---> 0164bd7b4711
Successfully built 0164bd7b4711
Successfully tagged application1:latest
SECURITY WARNING: You are building a Docker image from Windows against a non-Windows Docker host.
o double check and reset permissions for sensitive files and directories.
```

3.1.3 Docker Image : Run

When starting the modified image with the predefined manner, Docker run generates an editable container layer it over.

Docker start, which is similar to the APT construct then start container, can be used to restore a stopped container with all of its past adjustments. It can be used in conjunction with docker commit to modify a container's order. In a new container, run a script.

RUN state :

The intention of the CMD instruction is to define a command that will be executed when the container is built from the final image. There can only be one CMD command, and if there are several, the most current is used. Within the jar, run a command shell. The ENV instruction is used to specify the application's environment when it is running within the container. When deciding users and passwords for a database case, these are heavily application specific. In this project, the best defined within a docker-compse.yml fine is used.

When a container is running, Docker uses the Expose instruction to inform it which ports it should listen to. The port to be listened using TCP or UDP. This command does not actually publish the port; it simply specifies which port will be made public.

The Volume instruction identifies a mount point and specifies that it will hold externally mounted volumes from a host or other container. Volumes is used to generate storage space for user data stores in Docker applications.

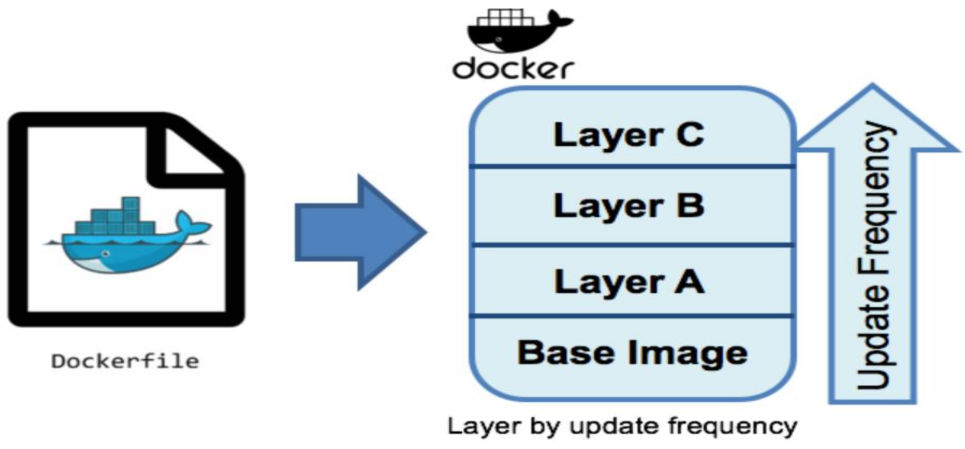
The ENTRYPOINT instruction enables the client to a container which will execute as a protocol. Only the last entry point in a list of multiple entry points will have an impact on the Docker file. It is used to prevent the end user from overriding the required executable.

3.1.4 Docker Image : Layers

Docker containers are application building blocks. Each containers are made up of a publish layer on top of an image with an accessible layer. These layers, also known as intermediate images, are created when commands in the Dockerfile are executed during the Docker image construction. It's a form of transparency on which imaging effects or images are applied and positioned underneath an imageEvery image consists of several layers.

Docker combines these layers into a single unified view using union file systems. The computer's operating architecture of a container is composed of layers layered on top of each other. It is based on the collection concept. Each layer in the optimal control to a Dockerfile instruction. The very last one would almost surely be publish.

Fig 18: Docker image Layer



$ docker inspect imageid to inspect an image, including image layers

3.1.5 Docker Image : storage

It's necessary to understand how Docker creates and stores images, which are then used by containers, in order to use storage drivers effectively. The most efficient way to save data from applications while minimizing performance issues. Containers should have no state. Only a small amount of data is written to the container's writable layer. Data can be stored on Docker volumes, however some workloads require writing data to the container's executable layer.Docker images are directory files that are placed in a directory. The storage drivers regulate how Docker host stores and handles images and containers.

If the kernel supports multiple storage. Docker has a list of storage drivers that are prioritized for use. For supported Linux , Overlay is the storage driver. Some storage drivers need a particular backing filesystem format. If you need to use a particular backup file system because of external specifications. The storage driver selected can have an effect on containerized application performance.

When it comes to storage drivers, there are usually two levels: file and block. The filesystem drivers union, overlay, and overlay work at the level They make better use of memory, but in write-intensive workloads, the container's writable layer may become unnecessarily big. Since the project involves no write-intensive workloads, the default overlay2- drivers were chosen. The device mapper, btrfs, and zfs block-level storage drivers function well with write-intensive workloads.

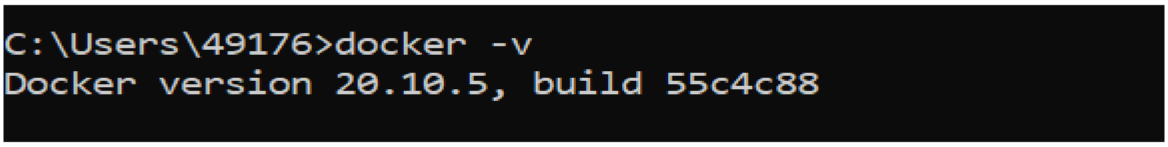
3.2.Commands : Image Handling

1. Docker info :

System- wide info on Docker installation

$ docker version

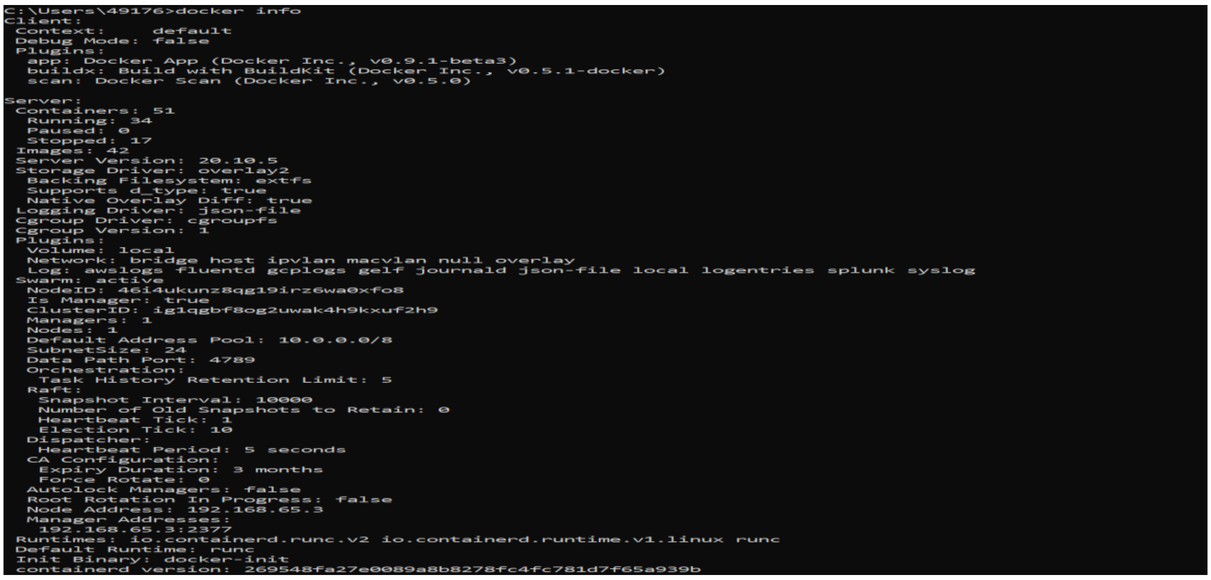
Fig 23 : Docker commands



```
C:\Users\49176>docker -v
Docker version 20.10.5, build 55c4c88
```

$ docker info

Fig 23 : Docker commands



```
C:\Users\49176>docker info
Client:
 Context:    default
 Debug Mode: false
 Plugins:
  app: Docker App (Docker Inc., v0.9.1-beta3)
  buildx: Build with BuildKit (Docker Inc., v0.5.1-docker)
  scan: Docker Scan (Docker Inc., v0.5.0)

Server:
 Containers: 51
  Running: 34
  Paused: 0
  Stopped: 17
 Images: 42
 Server Version: 20.10.5
 Storage Driver: overlay2
  Backing Filesystem: extfs
  Supports d_type: true
  Native Overlay Diff: true
 Logging Driver: json-file
 Cgroup Driver: cgroupfs
 Cgroup Version: 1
 Plugins:
  Volume: local
  Network: bridge host ipvlan macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog
 Swarm: active
  NodeID: 46i4ukunz8qg19irz6wa0xfo8
  Is Manager: true
  ClusterID: ig1qgbf8og2uwak4h9kxuf2h9
  Managers: 1
  Nodes: 1
  Default Address Pool: 10.0.0.0/8
  SubnetSize: 24
  Data Path Port: 4789
  Orchestration:
   Task History Retention Limit: 5
  Raft:
   Snapshot Interval: 10000
   Number of Old Snapshots to Retain: 0
   Heartbeat Tick: 1
   Election Tick: 10
  Dispatcher:
   Heartbeat Period: 5 seconds
  CA Configuration:
   Expiry Duration: 3 months
   Force Rotate: 0
  Autolock Managers: false
  Root Rotation In Progress: false
  Node Address: 192.168.65.3
  Manager Addresses:
   192.168.65.3:2377
 Runtimes: io.containerd.runc.v2 io.containerd.runtime.v1.linux runc
 Default Runtime: runc
 Init Binary: docker-init
 containerd version: 269548fa27e0089a8b8278fc4fc781d7f65a939b
```

This command displays information such as how many images, containers, and their status, as well as the storage driver.

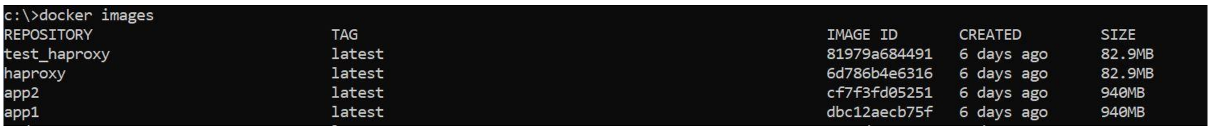
- Operating device, architecture, and total memory are all things to consider.
- Docker Registry is a registry for Docker containers.
- Status of the Docker Swarm

2. Docker Image Display :

The easiest place to list Docker images would be to use “docker files” without any arguments. You'll see a list of any and all Docker new images on your device when you execute the ordering.

$ docker images

Fig 23 : Docker commands



```
c:\>docker images
REPOSITORY                    TAG                                       IMAGE ID       CREATED         SIZE
test_haproxy                  latest                                    81979a684491   6 days ago      82.9MB
haproxy                       latest                                    6d786b4e6316   6 days ago      82.9MB
app2                          latest                                    cf7f3fd05251   6 days ago      940MB
app1                          latest                                    dbc12aecb75f   6 days ago      940MB
```

List all of the images, including their layers.

$ ls docker images

Option to sort images by name and tag, to sort images by image digests (SHA256), to filter images, and to format the performance

$ docker images –reference-filter

3. Docker Image Download :

To download an image, open docker pull.If no tag is specified, Docker Engine defaults to the :latest tag.

$ docker pull

$ docker pull –all-tags

4. Remove Docker Image :

one or more photos should be deleted

docker image rm [options] is the command to use. [IMAGE...] IMAGE [IMAGE...] IMAGE [IMAGE...] IM

$ --f , -f – Force removal of images

The command removes all stopped containers, dangling images and unused networks.

$ docker rmi imageid

5. Save Docker Image :

Create a tar archive with one or more images.

Usage : docker save [OPTIONS] IMAGE [IMAGE..]

6. Push docker image :

A registry may be used to push an image or a repository.

Usage : docker image push

7. Import Docker Image

To build a file system image, import the contents of a tarball.

Usage : docker image import

8. History of Docker Image :

Show how a image has evolved over time.

Usage : docker image history

9. Run Command

```
$ docker run [OPTIONS] IMAGE [ COMMAND] [ARGS]
```

The most popular choices are:

—name	give the container a name -d set the container to detached mode
-I	am an active participant
-t	do not assign a pseudo-tty
—expose	makes a number of ports within the container visible.
-p	To the host, publish the port of a container or a set of ports.
-v	Bind and mount a volume with the -v option.
-e	Variables in the environment are set
--link	Attach a connection to other containers with —link.

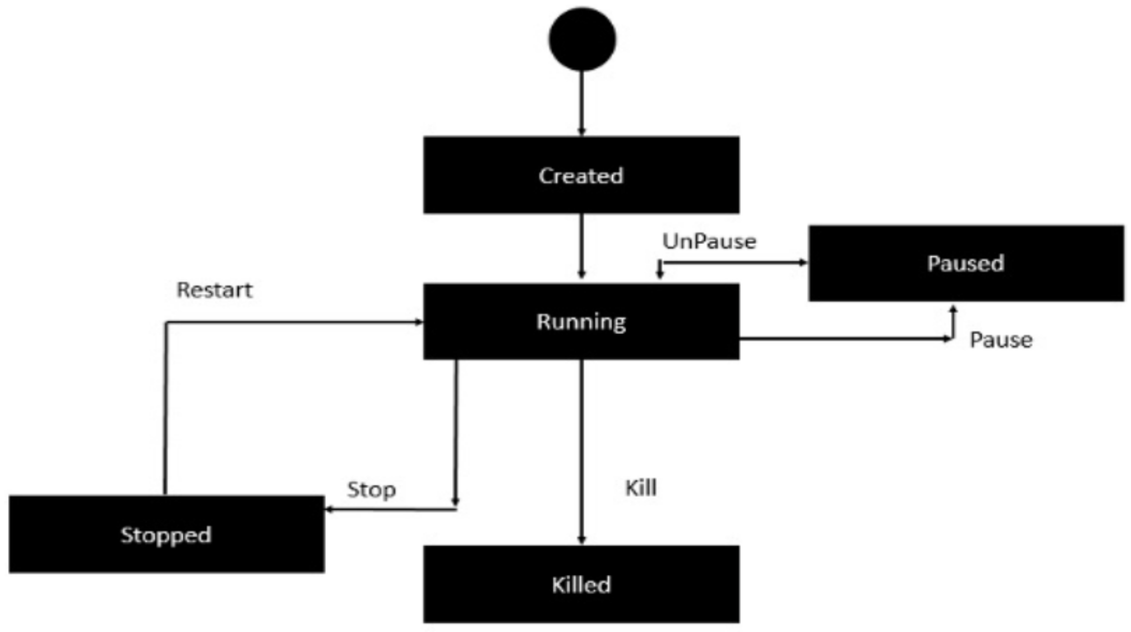
Chapter 4

Docker- Containers - Lifecycle

A Docker container image is a small, standalone software package that contains everything required to run an application, including code, runtime, system resources, system libraries, and settings. It is a basic piece of software that bundles and transports programs from one programming environment to another easily and reliably. It transforms into containers at runtime, and Docker containers-images transform into containers when run on Docker Engine. The same system can be used by containerized applications, which is designed for Window frames systems. Containerized applications are simpler, and Docker has the industry's best default isolation capabilities.

The Docker container technology was first released as an open source Docker Engine in 2013. Existing computing is based on containers. It's distinct in that it focuses on device operators' need to distinguish program dependencies from infrastructure. The Docker container technology is used by several of these providers for their -native laas system.

Fig 20 : Lifecycle



- The Docker container will be in the generated state at first.
- The Docker container begins when you press the docker run button.
- To delete an existing Docker, use the kill command.
- To take a Docker to retain, should use Docker halt instruction.
- The stop command is used to place a container on hold.
- The run command is used to return a container to a running state after it has been stopped.

4.1. Create Container

Docker Image is similar to a design that is used to create Docker containers. The run command is used to create containers using the Read Only templates. Build a container with the appropriate image to run it later.

```
 $ docker create –name <container-name> <image-name>
```

4.2. Start Container

Docker start is the instruction used to start or reopen a multiple simultaneous Docker containers that have been stopped:

```
  docker start   container id
```

4.3 Running a Container

Docker pulls the file, checks for the existence of images on the host, and if none are found, Docker downloads it from Docker Hub. If an image already exists, Docker can use it to construct a new container. Furthermore, Once Docker has the file, it uses it to construct a container. It provides a network interface that connects the docker container to the local host and allocates a network. A file system is allocated and a read-write layer is installed. The container is generated in the file system, and the image is given a read-write layer. Searches for and attaches an accessible Ip from a database to generate an IP address. It runs a program which is based on processes.

It links and logs standard I/O and errors to see how the application is working and collects and provides application performance. Start the docker container with the specified image and command method. The -d flag instructs the container to run in the background.

```
$ docker run -it -d –name< container-name>< image-name> bash
```

4.4 Docker kill

One or more containers are killed by the docker destroy subcommand. The SIGKILL signal is sent to the container's key process (default). Use the container ID, ID-prefix, or name to disable a container. The container was properly stopped after using the kill button, running containers must be killed.

```
  $ docker kill  CONTAINER NAME OR ID
```

4.5 Docker pause

The pause command suspends all processes in the containers you choose. When the Cgroup method is unable to record, it is put on hold. Hyper-V containers may be paused at that time.

```
    $ docker pause my_ container
```

4.6 Containers Catalogue

The basic format for using docker is

$ docker command

- Type the follows into terminal to see a list of all Docker containers that are currently running.
  $ docker ps
- Identify everything containers, both active and inactive,
  $ docker ps -a
- Use -aq to get a list of containers by ID.
  $ docker ps -aq
- -s to catalogue the container sizes
- $ docker ps -l
  The ps command produces a number of columns of data:
  The ps command displays classified in a variety of columns:
  Image – the container's base operating system image Container ID – a letters and numbers identifier for each container
  The container was designed and made available.
  Any ports that will be forwarded to the container for networking purposes are defined here.
  The Docker connected steps you a title.

4.7 Stop Container

To stop a container, use the docker stop command:

$ docker stop container_id

To get rid of a container

container_ id $ docker kill

To bring all running containers to a halt,

$ docker stop (docker ps -a -q)

One of the most important benefits of containers is their speed, as demonstrated by Docker. Containers can be used for development, testing, and deployment. When they've created an atmosphere, it can be moved for testing.

The applications that are designed within a Docker container are extremely portable. The output of these portable applications can easily be reduced to a single feature**.**

Docker's scalability enables it to be mounted on several servers, data servers, and cloud platforms. It can also be setup on PC. Containers can be conveniently transferred from the cloud to a local host. Adjustments are simple to create.

Rapid Delivery: A docker container's format is standardized, putting one job ahead of another. The duty is to deploy and manage the containerized server, which offers a consistent, stable, and enhanced environment.

Density: It makes better use of available resources, allowing more containers to run on a single virtual machine host.

Chapter 5

Docker Compose

Compose is a Docker specification that creates and manages and run multi-container Docker systems. Several of the platform's features can be managed using a Folder. From setup, it creates and starts all services. Compose begins and runs the whole program when you run docker-compose up.

Docker Compose that allows to build & run multi-container Docker applications. You describe the collection of docker containers that an application needs in a configuration file with Compose. Then you build and start all of the services in a single host with a single order.

Docker Compose is particularly beneficial in the following scenarios:

- Use your own computer, set up a development environment with all of the necessary services running. To put it another way, stop manually planning your development environment on your own or on various devices.
- Run integration tests in automated test environments that have the same characteristics as the production environments.

Compose works in a variety of settings, including manufacture, planning, design, validation, and continuous integration and delivery. To run docker-compose successfully, you must have access to the yml file directory.Using Compose is three-step process.

1. Build a Dockerfile to describe the application's environment.

2. The functions that create up the application are defined in docker-compose.yml so that they should operate simultaneously.

3. Running docker-compose up to start and run combine in the app. If you'd like to contribute or simply work on a project, Docker Compose is still in developing stage.

Reporting bugs and making outward form via Github's issue tracker. To discuss the matter in real time, access the Docker Communities slack medium docker-compose. To add code or data updates, create a request form on Github.

Compose has commands for handling an application's entire lifecycle:

- Activate, deactivate, and rebuild services
- View the status of currently operating programs.
- Stream the performance of operating services' log files.
- Activate a service with a one-time command.

On a single host, there are many isolated environments.

It uses a project name to keep environments apart from one another. It may use the project name in a variety of ways:

• On a development computer, make several copies of a single environment and run a stable copy for each project function.

• It identify the project title to a special build number on a CI servers to keep installs from interfering with each other.

• To prevent initiatives from interfering with each other, it maintains a certain command prompt.

The automatic target location is the basis name of the project folder. To define a configuration file, just use -p command prompt option or the COMPOSE New Project environments variable.

When creating containers, remember to save volume data.

All volume used by providers is preserved by Compose. If docker-compose up detects a container from a previous run, the volumes from the old container are copied to the new container. This procedure ensures that any data produced in large amounts is secured. It uses docker-compose on a Windows machine to examine environment variables and make appropriate adjustments for particular requirements.

This file defines two containers: site, which contains our Java code for launching a web server, and server, which contains our Java code for launching a server. The construct property appears in the first instance, while the image property appears in the second. It means that in order for a web container to function, it must first generate a Docker image that can be found on our local file system. The image for the haproxy server, on the other side, is taken from the DockerHub repository.

Because they are interconnected sometimes in way, Compose simulates two devices with different IP addresses in the same web server with this docker-compose.yml file. Only ports 8011 is accessed outside the our docker-machine, and Container creates a connection between this port and the container host 8080.

Only the containers that have modified should be recreated.

The configuration used to build a container is cached by Compose. Compose re-uses the same containers to restart a service that hasn't changed. Containers that are reused to rapidly alter the climate**.**

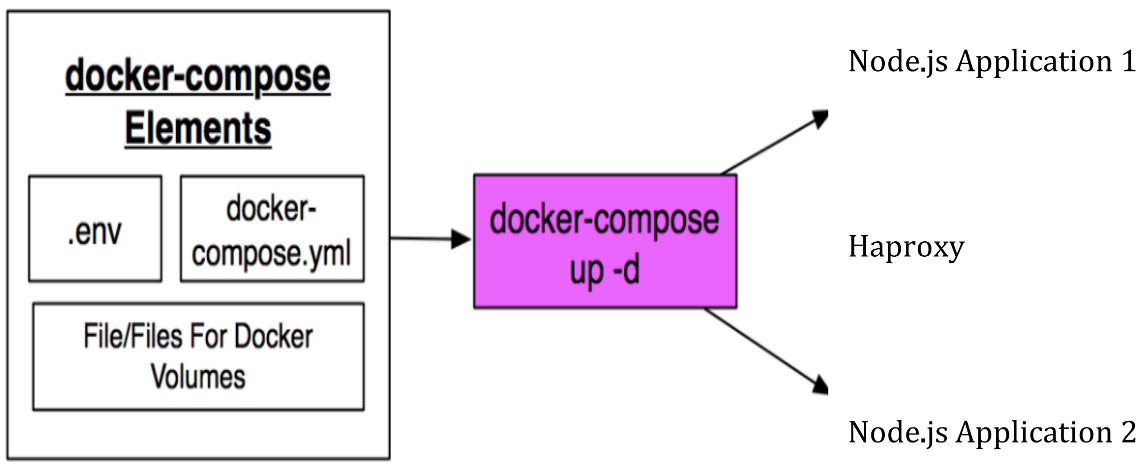
A composition that is variable and moves between environments

The configuration used to build a container is cached by Compose. Compose re-uses the same containers when you restart a service that hasn't changed. By reusing containers, you can make fast adjustments to the climate.

Development environments

It's essential to be able to run and connect with a program in a safe environment. Just use compose command prompt tool to create and communicate with an environment. The compose files will be used to record and modify most of a platform's services requirements The command - line interface tool can create and start two or more containers by each requirement with a simple step (docker-compose-up)

Fig 21 :Docker Compose



It can condense a multi-page "development team getting going guide" into a simple processor compose file and a few commands, making it simple to get initiated for designers. It helps you to configure and record the service dependencies of your application. The yaml files are configuration files that the Docker engine interprets, but they also serve as files that explain how multi-container applications are placed together.

Environments for automated testing

The automated test suite is part of the continuous deployment or continuous integration phase. End-to-end automated testing necessitates a test area. In a Compose file, Compose offers a simple way to build and kill isolated testing environments, i.e. destroy these environments

```
$ docker-compose up -d
$ ./run_tests
$  docker -compose down.
```

Deployments of only one host

Compose started with a focus on development and testing workflows, but as time went on, it introduced more production-oriented features. The simplest way to deploy an application is to run it on a single server, which also serves as the development environment. If the application grows in size, it can be scaled up and run on a cluster.

It's the fastest way to manage a software platform and deploy an executable on a single site. It can use env variables to deploy an application to a remote docker host.

Characteristics

- Compose's features that make it powerful are as follows:
- On a single host, you may have multiple isolate environments.
- When creating containers, remember to save volume data.
- Just make new containers.
- a composition that is variable and moves between environment.

Chapter 6

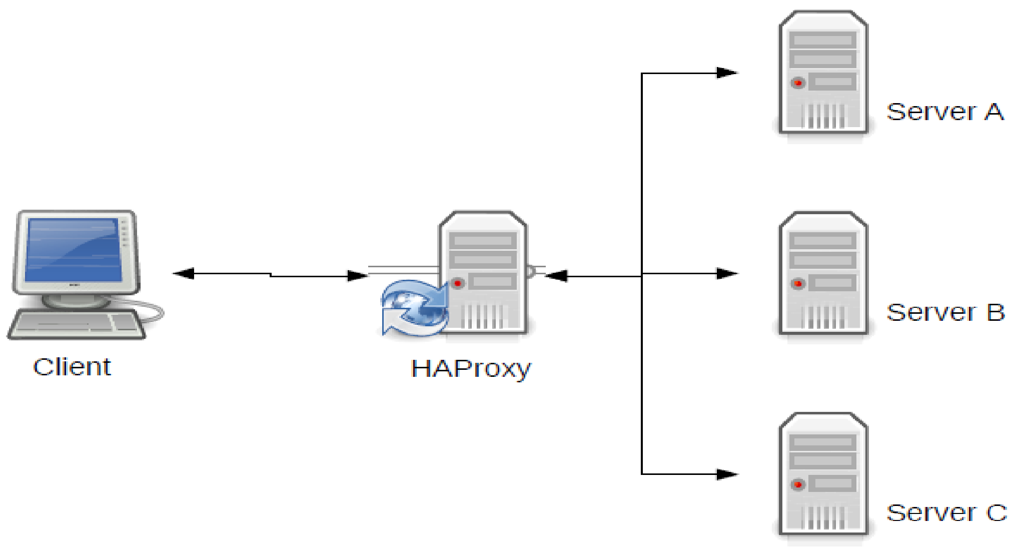
HAProxy-High Availability Proxy

It's a TCP/HTTP load balancer that's free to use. A load balancer's job is to spread incoming requests across an array of upstream servers so that no single upstream server is overworked or overloaded. The majority of websites have a large number of customers to represent. Several servers must be supported behind a load balancer to ensure a tolerable service period, according to the testing of a cluster-based network load delivery for both static and dynamic HTTP requests. It's a dependable system that can accommodate more connections per second.

HAProxy has never crashed in a production setting, and it has been stated to be highly productive and fault tolerant. It can be configured for various load balancing methods such as Round-Robin, Source, and so on. It also supports sessions, logs, and tracking statistics. ACLs, or Access Controls, improve system performance and reliability by distributing data over multiple servers

You can test various conditions in HAproxy and take measures depending on the outcomes. These criteria apply to any aspect of a request or response, including pattern detection, testing IP addresses, and evaluating recent request rates.

Fig 22: HAProxy



- ACLs are one of the capabilities.

  Make the decision depending on the info you've collected.

  Requests are blocked, the backend is selected, and the headers are rewritten.

- TCP mode(Layer 4)

  SSL passthrough, and basic TCP services

  ACLs are now available.

- Mode HTTP (Layer 7)

  ACLs for inspecting HTTP headers

  Persistence when it comes to cookie insertion

- Stick Tables
  Record data in tables, such as source addresses Search table for new connections, and choose a backend
- SSL termination, compression, and peers are all new features in HAProxy 1.5.

How to Setup HAProxy

To begin, obtain the source tarball from the official website, create and install it, then copy the standalone proxy executable to the directory and run haproxy as a command.

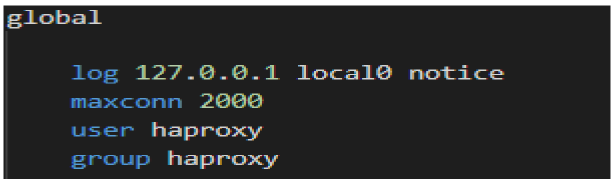
6.1 HAProxy configuration

There are several configuration options, and they are as simple to use as the installation.

Global Options:

It can be used on both frontend and backend servers and has a wide range of applications. It has debugging features such as setup logging and force level. If rysyslog is needed for logs (global). It will run as user and group haproxy, with a maximum number of connections permitted (maxconn) and stats unix socket enabled

- Global
- Maxconn
- User and group haproxy
  Fig23: Haproxy Settings



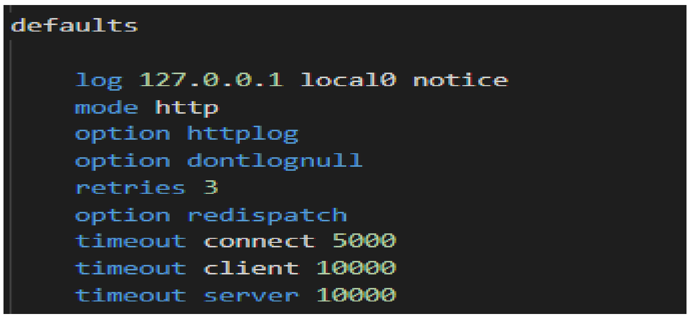
```
global

    log 127.0.0.1 local0 notice
    maxconn 2000
    user haproxy
    group haproxy
```

Default Settings :

These options are for the backend servers' default configurations. If no block settings are specified, the popular default, which includes all the listen and backend parts, will be used.

- Logging: It makes use of the logging options set in the Global settings.
- Mode: This controls how requests are parsed, and it employs http handling.
- As just a choice, httplog

Fig23: Haproxy Settings



```
defaults

    log 127.0.0.1 local0 notice
    mode http
    option httplog
    option dontlognull
    retries 3
    option redispatch
    timeout connect 5000
    timeout client 10000
    timeout server 10000
```

Setting up the backend:

A back - end is a collection of servers that collect and send forwarded applications. It is defined in the HAProxy configuration's backend portion. It can handle requests sent from the frontend as well as any custom settings.

• Stability

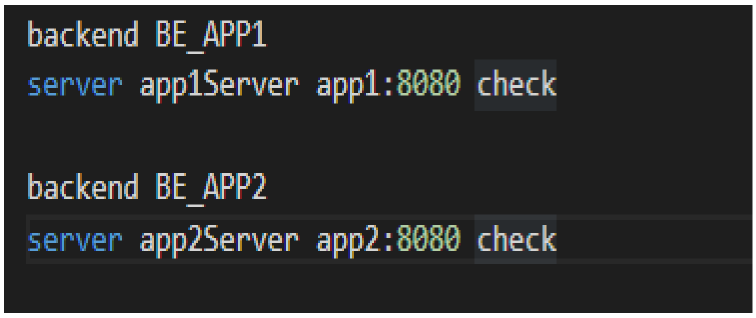
The algorithm for load balancing. Round robin is the norm, and it searches all available servers for the next available connection.

• A list of servers, ports, and hosts

• Cookie: Sets a cookie that the load balancer can use to direct future requests to the same server.

It may have one or more servers; typically, raising the number of servers in the backend would increase load capacity by spreading the load over many servers. Increased reliability can also be accomplished by changing one's demeanor.

Fig23: Haproxy Settings

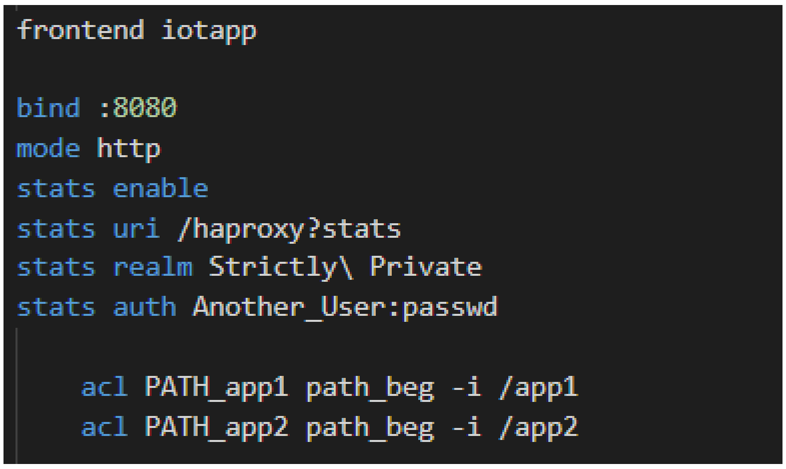

```
backend BE_APP1
server app1Server app1:8080 check

backend BE_APP2
server app2Server app2:8080 check
```

The balancing roundrobin for such the based algorithm described in the Load Balancing Optimization techniques section. As mentioned in the characteristics of load balancing sections, layer proxying would be used in mode http. The verify option, which appears at the end of the server instructions, indicates that certain backend servers should really be examined.

Front-end :

The way requests are forwarded to backends is described by a frontend. It is defined in the HAproxy configuration's frontend portion. It has networking rules that determine which backend server is capable of handling the client packet.

Fig23: Haproxy Settings


```
frontend iotapp

bind :8080
mode http
stats enable
stats uri /haproxy?stats
stats realm Strictly\ Private
stats auth Another_User:passwd

    acl PATH_app1 path_beg -i /app1
    acl PATH_app2 path_beg -i /app2
```

- Bind
  It specifies the port on which incoming requests will be sent
- ACLs
  Make a variable depending on the condition you're testing.
- Use_backend rules, it defines backends to use depending on which ACL condition are matched or a default_backend rule that handles every case.

- Default
  If the backend you're using doesn't fit the other rules, you'll get an error.
- Capture
- The valid User browser is included in the HAProxy request logs.
- Use_backend
  Routs to a particular group of backend settings depending on the true or false value of variables.

A frontend may be set up to handle different protocols congestion.

Security

HAProxy is considered safe by its users because it includes features such as chroot isolation, falling to a user/group upon startup, and preventing disk access after initialization that restrict the attack surface in the event of a security problem. It can inspect traffic and track a client's actions over a series of requests, blocking them if they appear to be malicious. It has the ability to set up access control lists, which specify policies for granting access based on data. Rate limiting and IP lists are also available.

The haproxy-wi web interface has the following features:

• Enable/disable servers via the stats page without restarting HAProxy with this HAProxy web interface.

• Using a web browser, view and analyze HAProxy logs.

• Updates to HAProxy servers can be easily pushed out.

• Edit or add frontend or backend servers.

• Manage user functions and server classes.

• Go back to a previous setup.

Load Balancing Algorithms

When load balancing, the load algorithm decides which server in the backend will be selected. HAProxy has a number of algorithm choices. A weight parameter may be allocated to a server in the load balancing algorithm to control how often it is selected in comparison to other servers. It can be used to spread TCP connections across multiple servers. The algorithm can also be used to disperse requests across servers when an HTTP profile is added to the VS and link multiplexing is allowed.

When persistence is allowed, the load balancer handles the first link from a client, and all subsequent connections or requests are sent to the same server. Request-based HTTP selects a new service for each HTTP request, regardless of TCP connections. The link is terminated as soon as the Https database server accepts the demand, and new TCP connections are established using Receiver methods apart from Web services. Until either supplier or the user closes the linkage, it stays available.

It uses roundrobin to include a variety of algorithms.

Round Robin

One of most extensively used and fundamental clustering algorithms. Most similar servers are used to get the same functions. They even have the same email address, but their Domain names are different. All IP addresses and their domain names are tracked by a DNS server. When a request for the IP address of a domain name is filed. Client requests are transmitted to applications servers in a circular manner. If there are previous two servers, for instance, the first customer order would be sent to the first server on the list, the second client requests to both the second server, and the third spoken to the first computer.

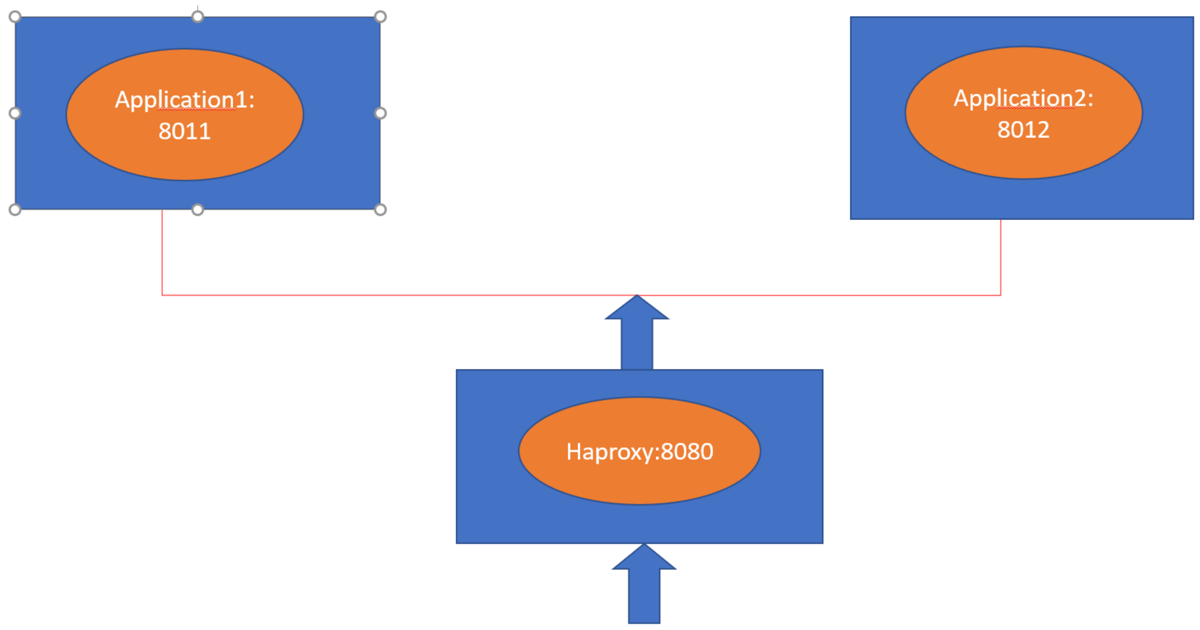
All applications are likely to be profitable. In terms of accessibility, computation, and system requiring, all large number of applications are believed to be similar.

Chapter 7

Working Concept

The aim of this project is to create two Nodejs applications that will run on a local host. And HAproxy is used to link these two applications to a third application. (workload distributed through many servers) These applications were generated in a Docker image and run on Docker Compose.

Fig 24: Working concept



Our Node.js app (application 1 &2). It generates the service from a call application image. We expose ports 8011 and 8012, as well as an environment variable called SERVICE PORTS that contains the ports we exposed, as well as some update and restart settings for them. What's important to note is that we bring all of those containers into a network called web application.

The third service is HAProxy, which is based on the haproxy Docker uses in their cloud (it doesn't construct it or generate a Dockerfile for it; instead, we just tell it where to get the image from). It is dependent on the application service, so it will not start until the application service is complete (all containers are up and running). The HAProxy container opens this folder to learn more about applications through its network .We place this container in the web network and expose port 8080. We simply tell it to always place this container on the manager node in the deploy environment (this setting is similar to Docker compose, when we have a few nodes).

Step 1 : creating two Nodejs application

To start with, Using Node.js, build two Nodejs applications in a folder. The elements of a Node.js framework are as follows.

Required modules were imported: It necessitates the use of a directive to load Node.js modules.

To use the http module and save the returned HTTP instance in a http variable

Var http = require (“http”);

Create Server : A server that, like an HTTP server, listens for client requests.

It built a http exception and a website exception using the http.createServer() technique. It then used the listen method to connect it to ports 8011 and 8012, passing it a function with request and response parameters.

Read the request and respond: The server established in the design phase will read and listen to Web applications from of the user, which could be a website or a console.

Start our HTTP server by putting everything together in a file called nodex.js.

Step 2 : Installing Packages

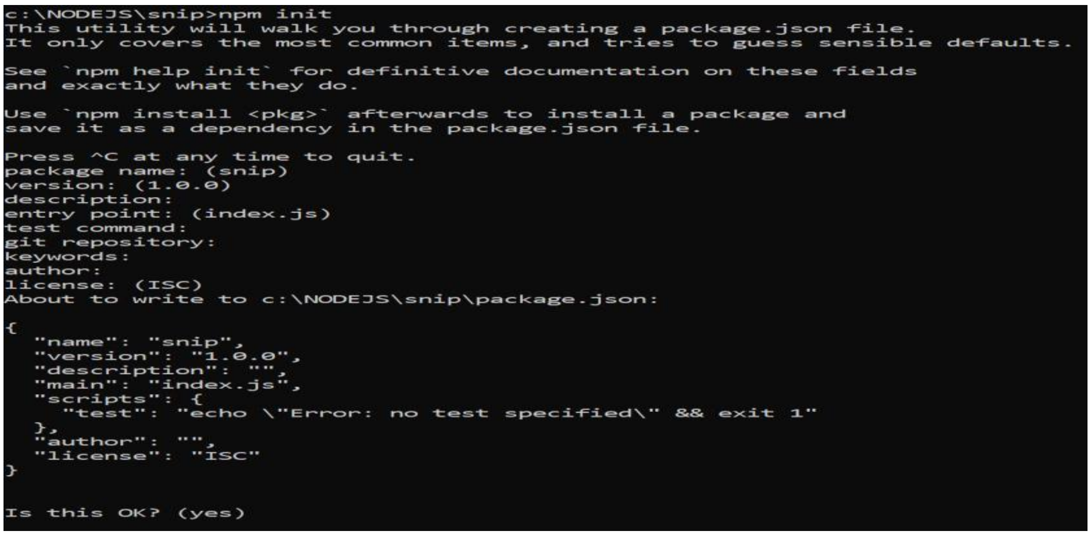
The Node Package Manager (NPM) has two features.

- 1.On nodejs.org, there are online repositories for node.js packages.
- 2. Install node.js packages using a command line utility.

Firstly, Installing npm

Using command **npm init**

Fig 25 : Installing packages



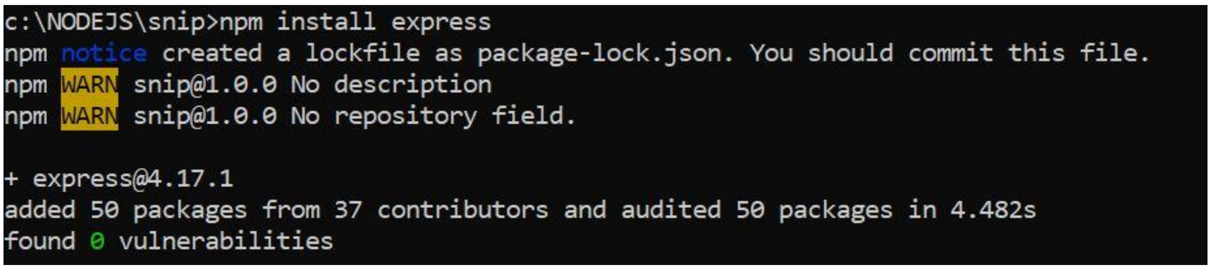
```
c:\NODEJS\snip>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (snip)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to c:\NODEJS\snip\package.json:

{
  "name": "snip",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}

Is this OK? (yes)
```

- When you run npm init, it will prompt you for some simple configuration information, such as the project name (app1), version, and starting point (index.js). When the server begins, it will search for index.js to execute.
- Then, using npm, install Express (Node Package manager). To building a web server, the express function is used.
- And then installing express

  Using command **npm install –save express**

The command above adds the Express dependency to the project. This dependency is saved in the package using the –save suffix. json is a type of data.

Fig 25 : Installing packages



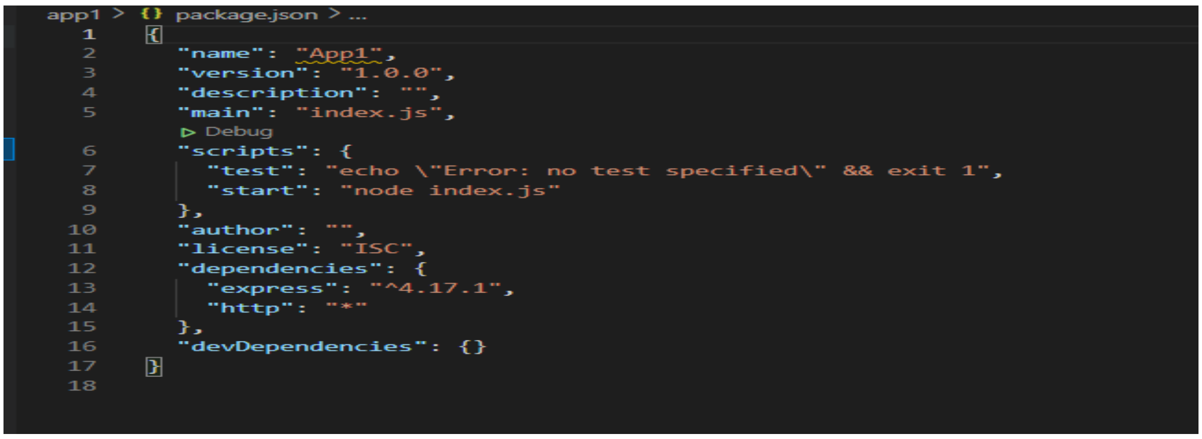
```
c:\NODEJS\snip>npm install express
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN snip@1.0.0 No description
npm WARN snip@1.0.0 No repository field.

+ express@4.17.1
added 50 packages from 37 contributors and audited 50 packages in 4.482s
found 0 vulnerabilities
```

Step 3 : Getting the application

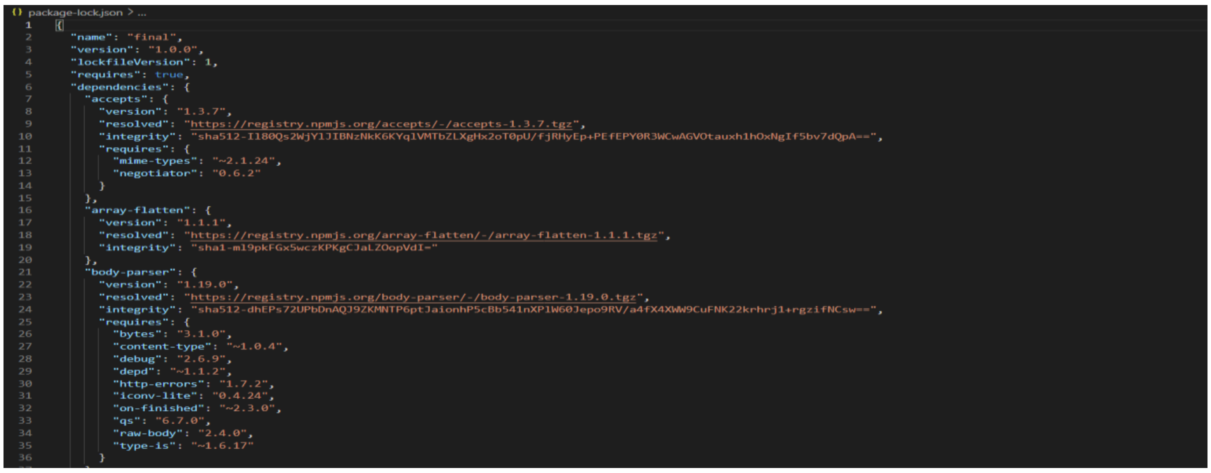
Get the program and its source code into the computer before running it. The repo will almost always be cloned. The application is contained in a file that I have made.

Automatically download the app's contents using the comments in the installation box. It can either download the entire project or uninstall the program folder after downloading it as a file. Once the projects has been built, open this in the Code Editor. package.json and subdirectories were included in the package.json format (package.lock).

Fig 25 : package file



```
app1 > {} package.json > ...
{
  "name": "App1",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  ▷ Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node index.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1",
    "http": "*"
  },
  "devDependencies": {}
}
```



```
package-lock.json > ...
{
  "name": "final",
  "version": "1.0.0",
  "lockfileVersion": 1,
  "requires": true,
  "dependencies": {
    "accepts": {
      "version": "1.3.7",
      "resolved": "https://registry.npmjs.org/accepts/-/accepts-1.3.7.tgz",
      "integrity": "sha512-Il80Qs2WjYlJIBNzNkK6KYqlVMTbZLXgHx2oT0pU/fjRHyEp+PEfEPY0R3WCwAGVOtauxh1hOxNgIf5bv7dQpA==",
      "requires": {
        "mime-types": "~2.1.24",
        "negotiator": "0.6.2"
      }
    },
    "array-flatten": {
      "version": "1.1.1",
      "resolved": "https://registry.npmjs.org/array-flatten/-/array-flatten-1.1.1.tgz",
      "integrity": "sha1-ml9pkFGx5wczKPKgCJaLZOopVdI="
    },
    "body-parser": {
      "version": "1.19.0",
      "resolved": "https://registry.npmjs.org/body-parser/-/body-parser-1.19.0.tgz",
      "integrity": "sha512-dhEPs72UPbDnAQJ9ZKMNTP6ptJaionhP5cBb541nXPlW60Jepo9RV/a4fX4XWW9CuFNK22krhrj1+rgzifNCsw==",
      "requires": {
        "bytes": "3.1.0",
        "content-type": "~1.0.4",
        "debug": "2.6.9",
        "depd": "~1.1.2",
        "http-errors": "1.7.2",
        "iconv-lite": "0.4.24",
        "on-finished": "~2.3.0",
        "qs": "6.7.0",
        "raw-body": "2.4.0",
        "type-is": "~1.6.17"
      }
```

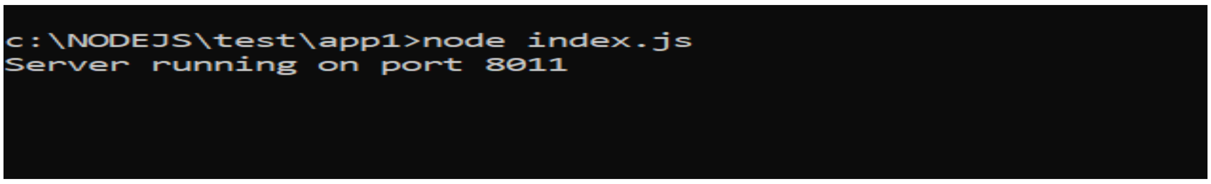
Step 4 : Make index.js File and write code

Make a document named index.js in the installation directory. I wrote the script using the POST/GET approach.

The Express package was imported using a required purpose with the first line; this method contains an object which can be used to modify the software. To begin listening on a specific host or port, a correctly implemented would be used. (For example, ports 8011 and 8012) We set up a route to check if the res.send() function returned the server response. This route doesn't do the actual database CRUD functions, but it is enforced to provide a route to check if the res.send() function returned the server response.After everything is done , Run the application using **npm start**

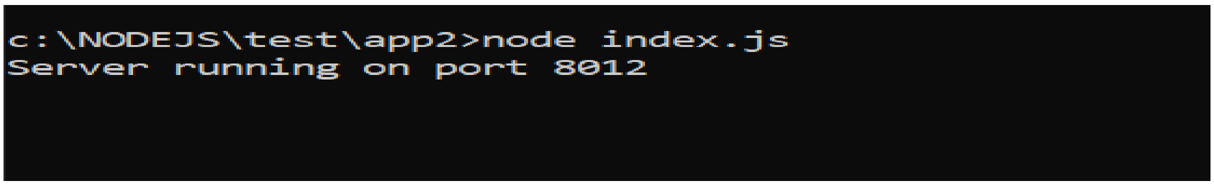
Fig 25 : Running application

Application 1 running on port 8011



```
c:\NODEJS\test\app1>node index.js
Server running on port 8011
```

Application 2 running on port 8012

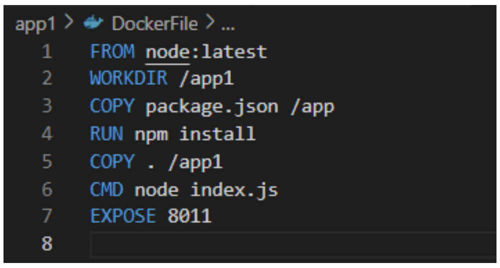


```
c:\NODEJS\test\app2>node index.js
Server running on port 8012
```

Step 5: Dockerizing the Node Server

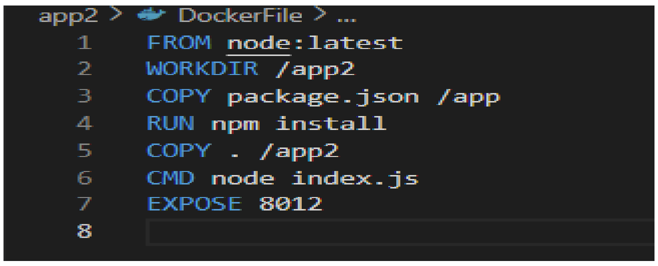
The server is ready to deploy, have to build the image and for that, write a Docker File.

Fig 25 : DockerFile



app1 > DockerFile > ...

```
FROM node:latest
WORKDIR /app1
COPY package.json /app
RUN npm install
COPY . /app1
CMD node index.js
EXPOSE 8011
```



app2 > DockerFile > ...

```
FROM node:latest
WORKDIR /app2
COPY package.json /app
RUN npm install
COPY . /app2
CMD node index.js
EXPOSE 8012
```

The images are made up of several layers, which are built up in each of the Docker-file phases.
1. Begin with the FROM keyword, which tells Docker which image to use as the base image, with node version 13 as the target.
2. WORKDIR denotes the images(/app) working directory in Docker. In this folder, CMD or RUN commands are executed.
3. The letters CP stand for copy, as in copying json file to application
4. RUN, The npm install command required the package's dependencies to be specified. json, which has been copied to the /app directory
5. Moving the documents from either the root folder to the /app filesystem using all inputs.
6. CMD stands for command node index.js, which is used to launch the nodejs server on the run file.
7. EXPOSE, it informationEXPOSE, it informs user container that it needs to open port

To Start building image.

```
Docker build app1 .
```

The Docker create function is being used to create the image from Docker-file orders. The -t flag is used to tag images with their node-server names, which determines the construct context for this image. file dockerignore

In the same directory as Dockerfile, create a.dockerignore file.

Modules for Node.js

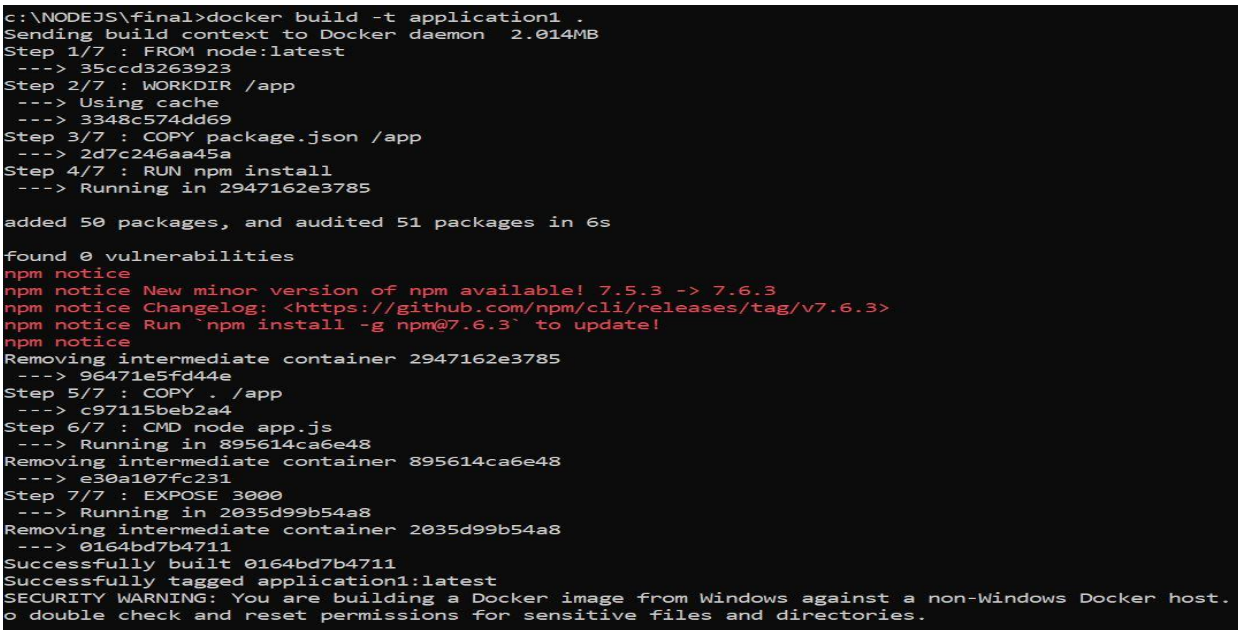
This prevents local modules from being copied into the docker file, potentially overwriting modules built with the image.

Step 6 : Build the image

Go over to the directory where the Dockerfile is maintained and run the command prompt to generate the Docker image .It's easier to use docker images command with the -t flag because it helps you to tag the images.

Build command

Fig 25 : Building stage



```
c:\NODEJS\final>docker build -t application1 .
Sending build context to Docker daemon  2.014MB
Step 1/7 : FROM node:latest
 ---> 35ccd3263923
Step 2/7 : WORKDIR /app
 ---> Using cache
 ---> 3348c574dd69
Step 3/7 : COPY package.json /app
 ---> 2d7c246aa45a
Step 4/7 : RUN npm install
 ---> Running in 2947162e3785

added 50 packages, and audited 51 packages in 6s

found 0 vulnerabilities
npm notice
npm notice New minor version of npm available! 7.5.3 -> 7.6.3
npm notice Changelog: <https://github.com/npm/cli/releases/tag/v7.6.3>
npm notice Run `npm install -g npm@7.6.3` to update!
npm notice
Removing intermediate container 2947162e3785
 ---> 96471e5fd44e
Step 5/7 : COPY . /app
 ---> c97115beb2a4
Step 6/7 : CMD node app.js
 ---> Running in 895614ca6e48
Removing intermediate container 895614ca6e48
 ---> e30a107fc231
Step 7/7 : EXPOSE 3000
 ---> Running in 2035d99b54a8
Removing intermediate container 2035d99b54a8
 ---> 0164bd7b4711
Successfully built 0164bd7b4711
Successfully tagged application1:latest
SECURITY WARNING: You are building a Docker image from Windows against a non-Windows Docker host.
o double check and reset permissions for sensitive files and directories.
```

$ docker images

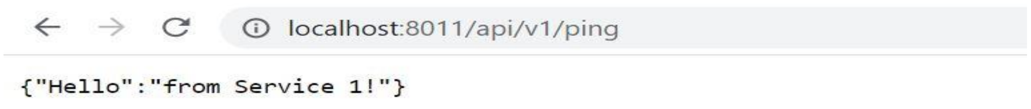
Step 7 : Test and Run the Container

Run the container and make sure it's functioning properly.

Execute the command

Use the Nodejs image to run a container. The -d flag in the run command means that the container will run in detach mode. –name is an optional parameter. The container -p option specifies the address on which their server is working; the container port is first, followed by the host port. It's important to determine what image, our app1 image, has been used to execute the container. Now use the command to search the server by going to localhost:8011/api/v1/ping.

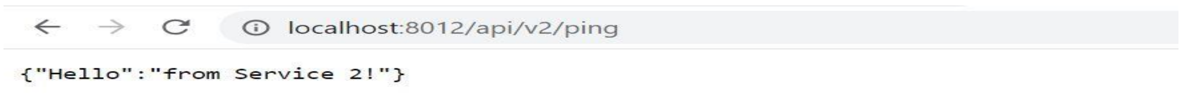
Fig 25 : Checking server



localhost:8011/api/v1/ping

```
{"Hello":"from Service 1!"}
```

Second Application :localhost:8012/api/v2/ping



localhost:8012/api/v2/ping

```
{"Hello":"from Service 2!"}
```

In the Docker Dashboard, you will see the application. You must be easier to see all the programs running once you've generated a file and launched the docker.

step 7: Configure with HAproxy.

This is a software program. When within that are performing at about the same moment and using HAproxy to interact with one another. Docker composer is used to execute a multi-container Docker program. To begin, create YAML that specifies two docker images, one for express application and the other for describing and expressing the construct.

Docker containers are stateless, so microservices benefit from them. Docker compose is a command that defines containers and their interactions. For example, the express part of our application relies on HAproxy, as specified in the yaml file. Create a docker-compose.yaml in directory.

The web component is expected to use the name of the HAproxy service to connect with it.

The configuration describes how to use docker to compose several web applications. It's mainly used for running multiple containers in a Docker program on websites.

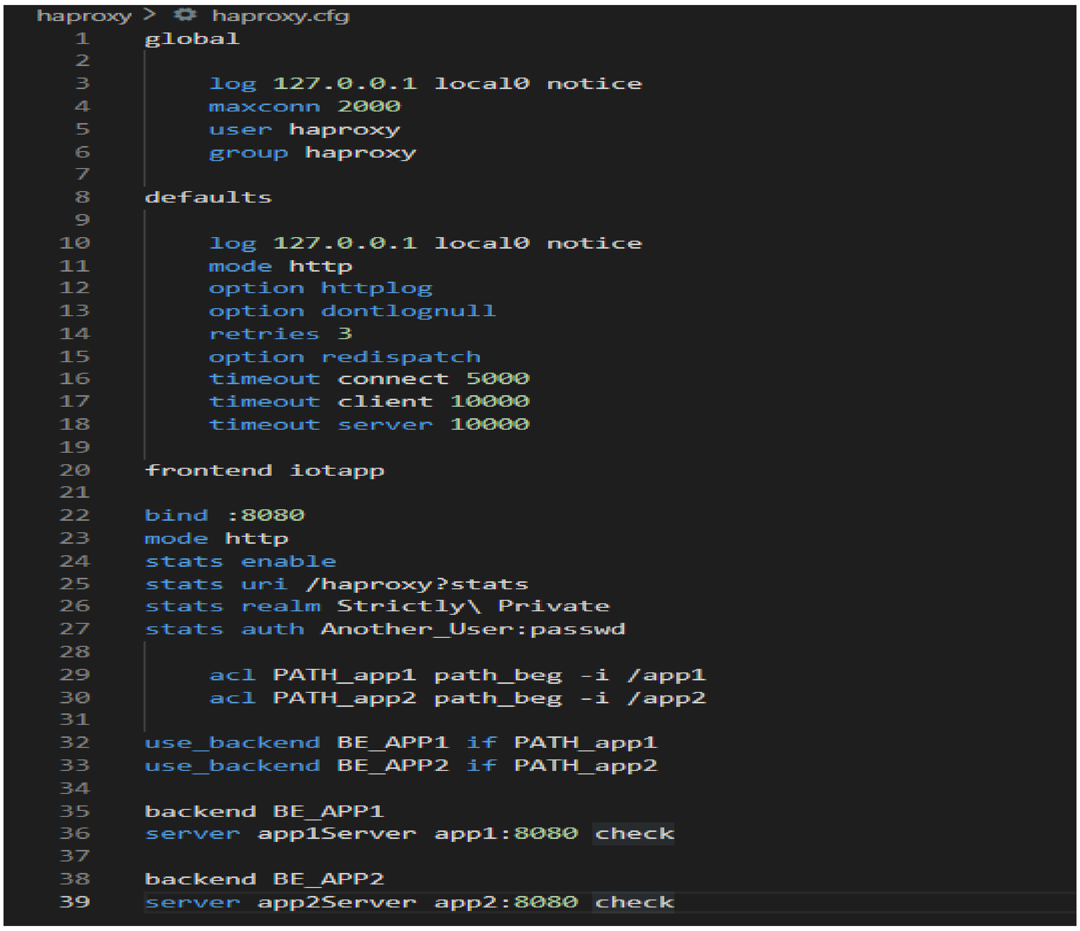
The log directive specifies a syslog server for logging messages; rsyslog is already installed and running at the specified address, and it modifies the configuration files.

The maxconn command states that the maximum number of simultaneous links on the frontend. The possible values is 2000. The HAProxy method is directed to the user or group specified by the user and group instructions. Mostly during TCP procedure, that both client and the server must accept or submit data. The client / server timeouts should be set to the very same number.

The retries directive defines the number of retries a VPS can make in the event of a communication failure. In the event of a malfunction, this option allows session redistribution. This file contains settings for both the frontend and the backend, and it tells HAproxy to listen on port 8080 for the app name, which is used to define an application.

The balancing sent by the task scheduling algorithm. The Round Robin method was employed. You name being mentioned in logs and alerts. This response includes a lot of information.

Fig 26: HAProxy Config



```
haproxy > haproxy.cfg
global

    log 127.0.0.1 local0 notice
    maxconn 2000
    user haproxy
    group haproxy

defaults

    log 127.0.0.1 local0 notice
    mode http
    option httplog
    option dontlognull
    retries 3
    option redispatch
    timeout connect 5000
    timeout client 10000
    timeout server 10000

frontend iotapp

bind :8080
mode http
stats enable
stats uri /haproxy?stats
stats realm Strictly\ Private
stats auth Another_User:passwd

    acl PATH_app1 path_beg -i /app1
    acl PATH_app2 path_beg -i /app2

use_backend BE_APP1 if PATH_app1
use_backend BE_APP2 if PATH_app2

backend BE_APP1
server app1Server app1:8080 check

backend BE_APP2
server app2Server app2:8080 check
```

We'll use HAProxy for our HTTP server, which means we'll need to build a HAProxy container that will listen on port 8080 and load balance requests to the various Containers for Node.js via port 8080. Let us just create our docker-compose.yml folder, which is then used to construct our containers (Node.js applications using HAProxy).

Chapter 8

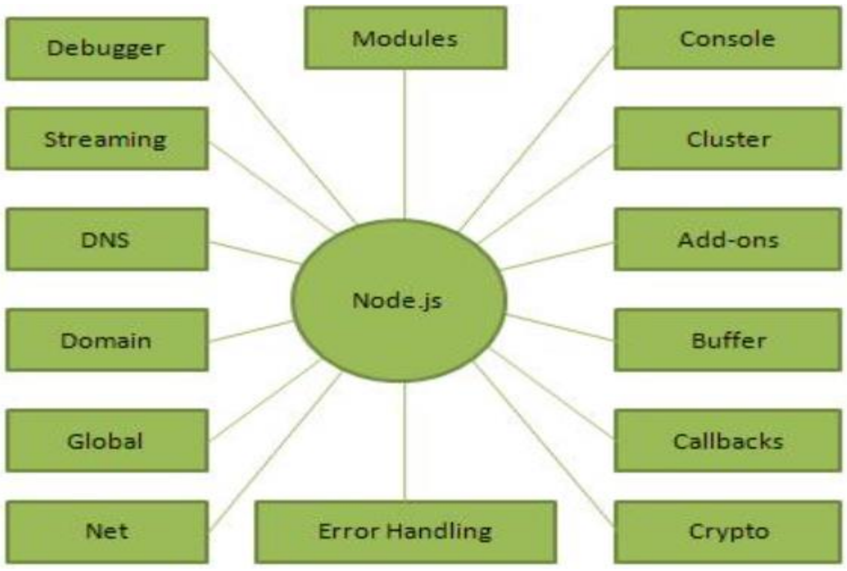
Software used

8.1 Node.js

Because it is an accessible and inter JavaScript developed for real, Node.js is a popular tool for the this type of development. It runs the JavaScript engine, which makes Node.js very quick. A Node.js application that runs in such a single procedure does not need to establish a new threaded for each order. Its simple design includes a set of async I/O iterators that keep Javascript from being blocked. It allows Node.js to conduct hundreds of simultaneous users without the overhead of thread virtualization maintenance on a single computer. Even if we don't have a full understanding of the language, we can write server code with client code written in a different language.

It has a huge advantage in that Node.js application, the latest ECMAScript standards, can be used without difficulty in choosing the script version to use by modifying the Node.js version and enabling specific features by using flags while running Node.js. It comes with a great standard library, as well as networking support. Node.js is capable of creating, opening, researching, publishing, deleting, and closing files mostly on website, and generating dynamic page content. It has the ability to gather information from a multitude of databases. It has the ability to add, remove, and change data in a database.

Fig 27 : Concept of Nodejs



Node.js Express Framework

It offers one of the most straightforward and successful methods for setting up a web server. Its popularity is due to its simplistic nature, which focuses on the most essential features of a server. It is a Node.js web application platform that offers a comprehensive collection of features for building web and mobile applications.

• Enables middlewares to respond to HTTP requests by allowing them to be configured.

• Defines a routing table for performing various acts based on HTTP process and URL.

• It enables the interactive rendering of HTML pages using template arguments.

For the backend, Express is interchangeable with other web application frameworks in the same way as frontend apps are.

Install the express platform with NPM Package and use it to create a web application. Creates a directory within node modules in the node modules directory.

body-parser is a node.js middleware that can handle JSON, Raw, Text, and URL encoded type data.

Cookie-parser parses cookie headers and fills req.cookies with an array keyed by cookie names.

Multer is a Node.js middleware for multipart/form data handling.

Node.js has a number of features.

There are a few main features that distinguish Node.js as asynchronous and event-driven:

The Node.js library's APIs are all asynchronous, or non-blocking. It was a server that doesn't allow with an API to deliver returns before proceeding. After calling an API, the server passes on to the next one, and Node.js' event notification mechanism aids the server in obtaining a response from the previous API request.

Because it is developed by google Chrom JavaScript Engine, the Node.js repository executes code very quickly.

It is highly scalable because it uses a single dimensional model with incident looping. It enables the server to react in a non-blocking manner and increases its scalability. It employs a single threaded program that can handle a much greater number of requests than conventional servers such as HTTP servers.

No Buffering: Data is never buffered in Node.js applications. The data is literally output in chunks by these applications**.**

Node.js is distributed under the MIT license.

• **I**/O-bound Applications Node.js is proving to be a perfect technology in these fields.

• Streaming Data Applications

• Real-time Data-Intensive Applications (DIRT)

• Applications based on JSON APIs

• Software with a single page

Using Nodejs with Docker :

- ➢ Streamlines the application deployment process; improves application portability;
- ➢ simplifies the diversion management process.
- ➢ Component Reuse at a Distance
- ➢ a very small footprint

7.2. Docker

Docker is a container-based technology that provides it easier to develop, distribute, and execute code. It allows the user to combine an application's requirements, containing binaries as well as other specifications, into a single package and deploy it. It enables you to separate programs from resources, comes to developing apps faster. It's a set of infrastructure as a service products that deliver services in containers using Desktop software solutions. Independent of any personalized settings used for writing and testing, the author can also be confident that the programme will run with any other Windows computer.

In some ways, Docker is similar to a virtual machine in that it allows programs to run on the same linux kernel system as the host computer and only needs applications to be shipped with items that aren't already running on the host computer. It improves performance while also reducing the size of applications.

They will also get a huge advantage by using one of the thousands of programming previously built to operate in a Docker container. It has a lot of flexibility and room for expansion.

decreases the number of systems needed Containers provide protection for applications running in a shared environment, but they are not a replacement for appropriate security measures. It includes a web-based simulator as well as a command-line simulator. It's basically a container system that uses linux kernel characteristics like namespaces and treatment conditions to build containers on top of a linux.

DockerFile: Docker creates images automatically by reading commands from Dockerfile text files, which contain all commands required to create a given image in the correct order. A dockerfile has a particular format and collection of instructions that must be followed. A dockerfile command is represented by all the other read-only levels in a docker image. The components are located on top of the other, and each one is a delta of the modifications done by the project level.

It creates an image container and places a new executable layered on top of the existing layers. All changes to the executing container, including changing existing files, modifying existing files, and deleting, are composed to a container level.

Docker-compose.yml: Compose is compatible with all environments, including manufacturing, creation, and testing, as well as continuous integration workflows. It configures application services using a YAML file. All of the services are developed and started from the configuration**.**

Docker Containerization's Usage:

1. **Stable and Segregated Environment:** It can build predictable environments that are isolated from other applications, allowing developers to spend less time testing and more time focused on launching features and functionality for users.

2. This can be related to contemporary technologies, making it both expense and fast to implement. As the test suite progresses, it becomes much more effective and easier to use.

3. **Mobility- Ability to Run Anywhere:** Docker images are free of environmental constraints, allowing any deployment to be consistent, movable, and scalable, with the added bonus of being able to run anywhere. Docker images are aimed at OS (Linux, Mac OS, VMs), which provides a few advantages for both development and deployment. In addition, there are powerful orchestration systems like Kubernetes and products like AWS.

4. **Reliability and Automation:** For new application containers, fast delivery and, once again, shorter deployment are essential. It has benefits in terms of upkeep. Containerization isolates an application from other programs operating on the same device. It's about writing code that uses repeatable technology and configuration.

5. Maintaining an appropriate atmosphere for continuous development and deployment (CI/CD) testing and implementation. Docker container is made to maintain all of their configurations and requirements. It is a straightforward and quick technique of reverted back.

6. Versatility: Another advantage of using Docker is its flexibility. It makes it possible to create, validate, and distribute images that can be distributed across various computers. It is capable of installing a patch, checking it, and releasing it to output. It helps you to quickly start and stop services or applications in the cloud**.**

7.Increase Productivity & Security Management

8. **Modularity and Scalability:** Containerization allows you to segment an application and refresh, clean up, and restore it without needing to reinstall it. It can construct an architecture for any application that consists of small processes interacting with APIs. This is highly cost-effective and time-saving since the production cycle is complete and all problems are resolved.

7.3.HAProxy.cfg

The action of the HAProxy load balancer is driven by the HAProxy configuration files. The global segment includes options that impact the HAProxy process as a whole. User, party, log directives, and stats are all included in this section. It works by assigning different weights to each server behind the load balancer. It's also the smoothest and most equitable algorithm since the server processing time is evenly distributed. Round Robin allows for the adjustment of server weights.

Haproxy is installed in front of the web server, in the haproxy.cfg format, which is located below the configuration files. a new section or several new sections Aside from global and default settings,

• **Frontend:** This section describes a reverse proxy that listens for incoming requests on a particular IP address and port.

• **Backend:** It specifies a set of servers to which the frontend can send requests.

• **Listen :** A shorthand notation that unifies frontend and backend functionality into a single command.

It can use any of the machine's IP addresses. It has the ability to add a large number of attach directives to the frontend. In web server logs, it catches the client source IP address. Before being sent to the web servers, it added a header called forward for to the incoming request.

Rewriting URLs is one of the features.

- Rate management
- Termination of SSL/TLS
- Help for the Proxy Protocol
- Recording of links and HTTP messages

Chapter 8

Analyzing Result

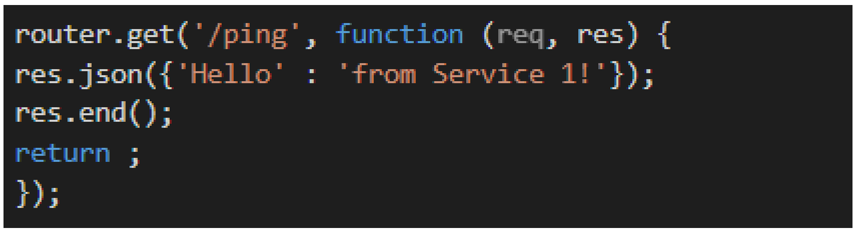
Case 1 : Application1

Both GET and POST are standard HTTP requests used to create APIs. GET needs have been used to send a bit of data because data is sent in the headers, while POST demands are used to deliver a big quantities of data because data is sent in body. Get or POST applications can be handled with Express.js.

It uses the Express application object to describe path, which is analogous to HTTP methods. The way an implementation listens to customer request is referred to as routing. The specification defines a function called when it sends a frame to the specified path and HTTP process (handler). Applications to the required paths and procedures are monitored by the user.

GET Process:

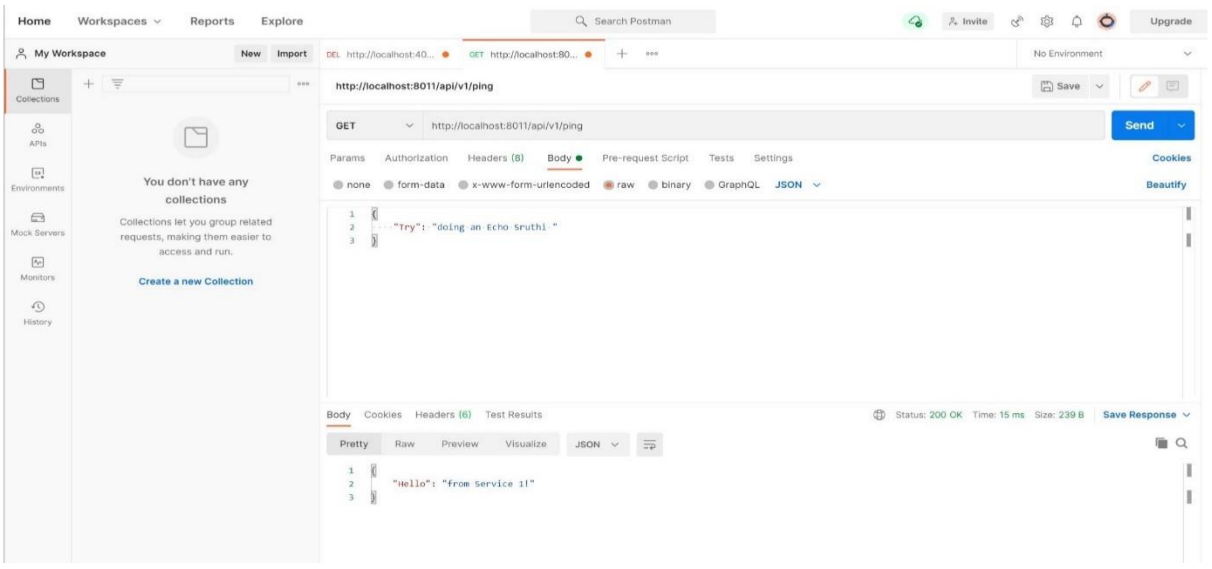
i.e. app. get() to handle GET requests



```
router.get('/ping', function (req, res) {
res.json({'Hello' : 'from Service 1!'});
res.end();
return ;
});
```

In my first application, using a POSTMAN , trying a GET request to 127.0.0.1:8011/api/v1/ping

Fig 28 : GET method



http://localhost:8011/api/v1/ping

GET http://localhost:8011/api/v1/ping

```
{
    "Try": "doing an Echo Sruthi "
}
```

Status: 200 OK Time: 15 ms Size: 239 B

```
{
    "Hello": "from Service 1!"
}
```

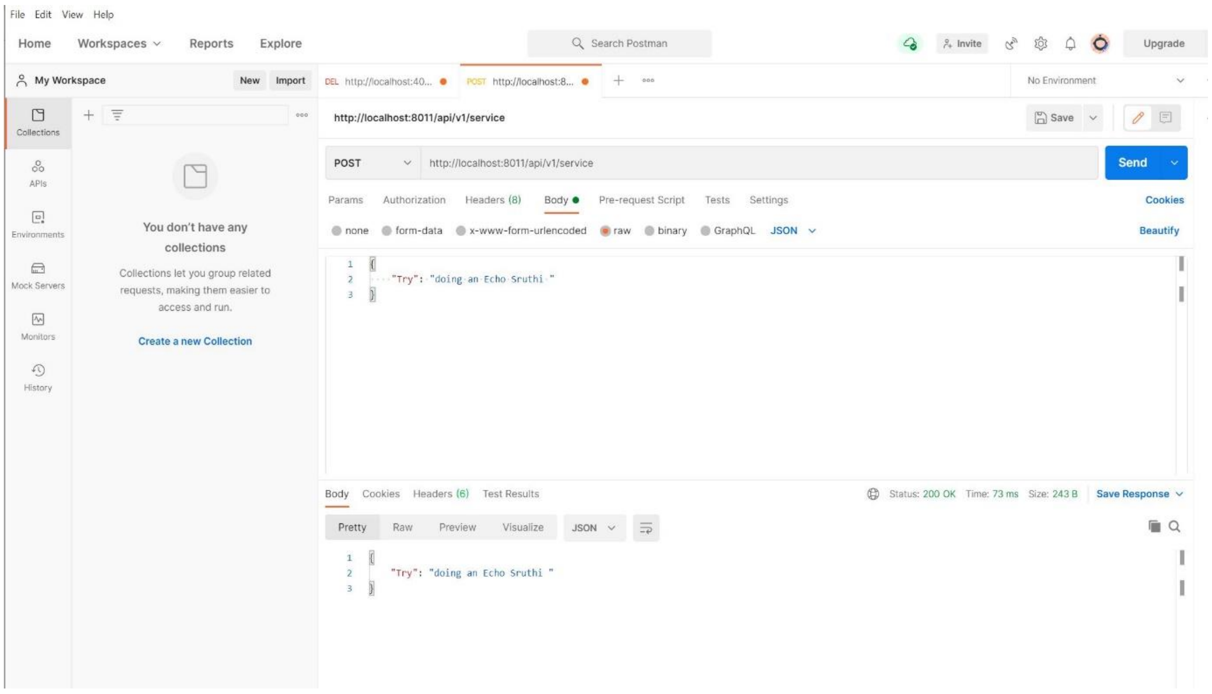
POST Method :

Although the POST method is safe since the data is not visible in the URL bar, it is not as widely used as the GET method. It is more sufficient than POST and is used more often.

```
router.post('/service', function (req, res) {
processPost(req, res);
});
```

and then trying a POST request to 127.0.0.1:8011/api/v1/service in POSTMAN

Fig 28 : POST method


File Edit View Help
Home Workspaces Reports Explore
Search Postman
Invite
Upgrade
My Workspace
New Import
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
You don't have any collections
Collections let you group related requests, making them easier to access and run.
Create a new Collection
Save
Send
Params Authorization Headers (8) Body Pre-request Script Tests Settings
Cookies
Beautify
Body Cookies Headers (6) Test Results
Save Response
Pretty Raw Preview Visualize

DEL http://localhost:40... POST http://localhost:8...

http://localhost:8011/api/v1/service

POST http://localhost:8011/api/v1/service

none form-data x-www-form-urlencoded ● raw binary GraphQL JSON

```
{
    "Try": "doing an Echo Sruthi "
}
```

Status: 200 OK Time: 73 ms Size: 243 B

JSON

```
{
    "Try": "doing an Echo Sruthi "
}
```

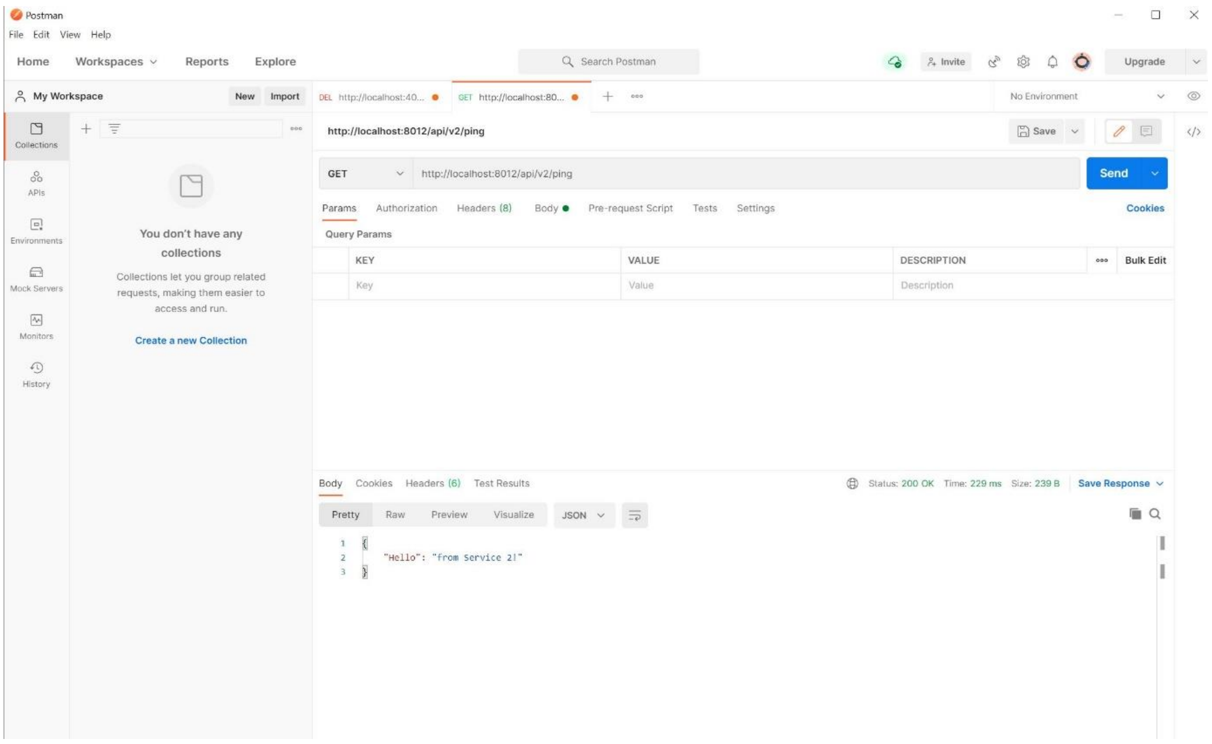
Case 2 : Application2

In my Second application ,i.e. app. get()

```
router.get('/ping', function (req, res) {
res.json({'Hello' : 'from Service 2!'});
res.end();
return ;
});
```

Using a POSTMAN , try a GET request to 127.0.0.1:8012/api/v2/ping

Fig 29 : GET method


Postman
File Edit View Help
Home Workspaces Reports Explore
Search Postman
Invite
Upgrade
My Workspace New Import
Collections
APIs
Environments
Mock Servers
Monitors
History
You don't have any collections
Collections let you group related requests, making them easier to access and run.
Create a new Collection
No Environment
Save
Send
Params Authorization Headers (8) Body Pre-request Script Tests Settings
Cookies
Query Params
Bulk Edit
Body Cookies Headers (6) Test Results
Save Response
Pretty Raw Preview Visualize JSON

DEL http://localhost:40... GET http://localhost:80...

http://localhost:8012/api/v2/ping

GET http://localhost:8012/api/v2/ping

KEY	VALUE	DESCRIPTION
Key	Value	Description

Status: 200 OK Time: 229 ms Size: 239 B

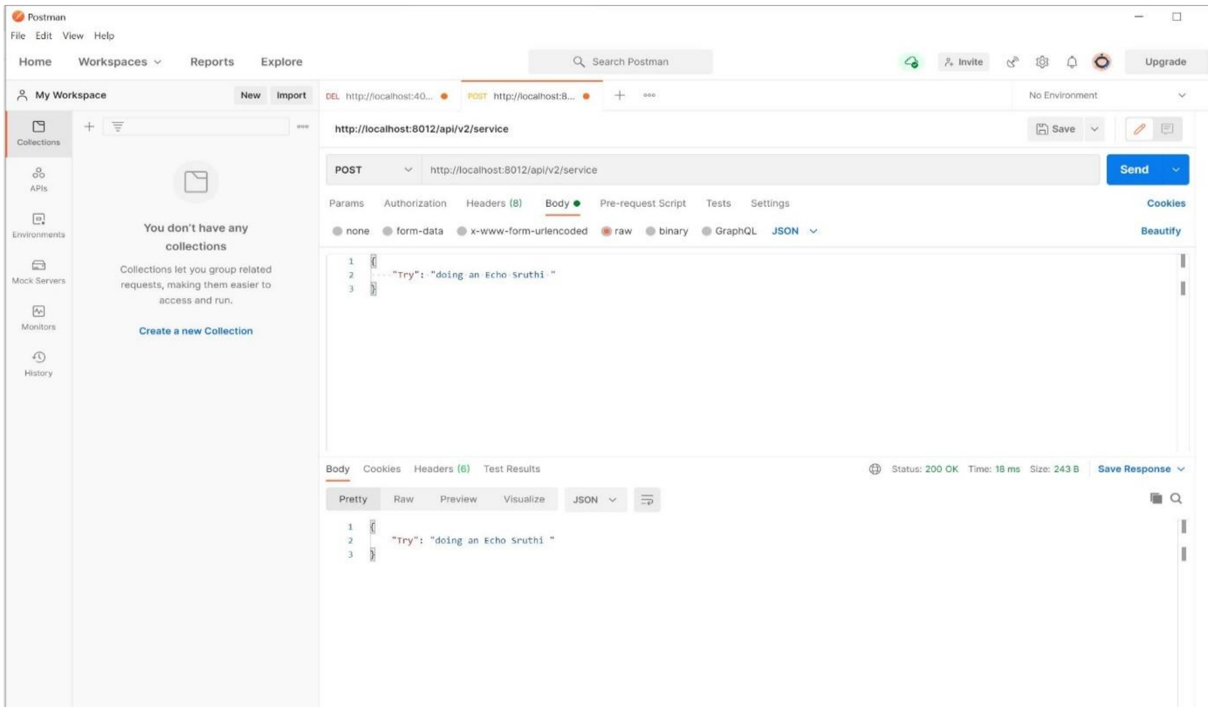
```
{
    "Hello": "from Service 2!"
}
```

Post Request :

```
router.post('/service', function (req, res) {
processPost(req, res);
});
```

then trying a POST request to 127.0.0.1:8012/api/v2/service in POSTMAN

Fig 29 : POST method


Postman
File Edit View Help
Home Workspaces Reports Explore
Search Postman
Invite
Upgrade
My Workspace
New Import
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
History
You don't have any collections
Collections let you group related requests, making them easier to access and run.
Create a new Collection
Save
Send
Params Authorization Headers (8) Body Pre-request Script Tests Settings
Cookies
Beautify
Body Cookies Headers (6) Test Results
Save Response
Pretty Raw Preview Visualize

DEL http://localhost:40...

POST http://localhost:8012/api/v2/service

```
{
    "Try": "doing an Echo Sruthi "
}
```

Status: 200 OK Time: 18 ms Size: 243 B

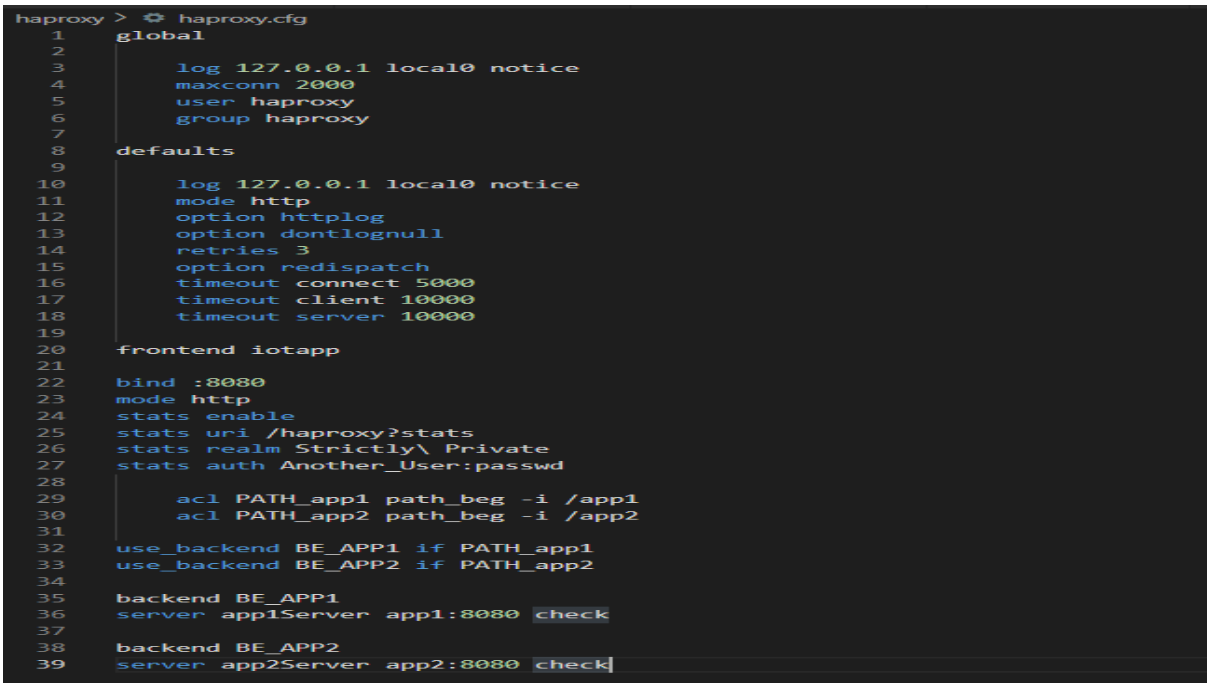
JSON

```
{
    "Try": "doing an Echo Sruthi "
}
```

Case 3 : Haproxy cfg.

In Haproxy configuration file , everything is mentioned about global , default and both frontend and backend settings

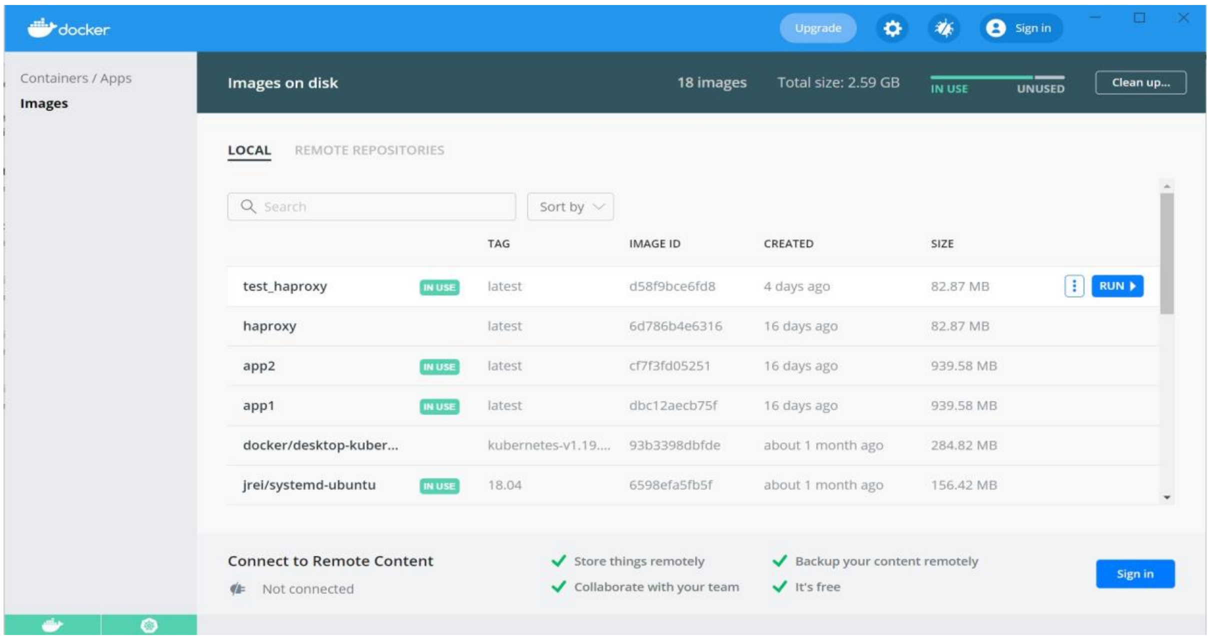
Fig 30 : Haproxy cfg



```
haproxy > ⚙ haproxy.cfg
global

        log 127.0.0.1 local0 notice
        maxconn 2000
        user haproxy
        group haproxy

defaults

        log 127.0.0.1 local0 notice
        mode http
        option httplog
        option dontlognull
        retries 3
        option redispatch
        timeout connect 5000
        timeout client 10000
        timeout server 10000

frontend iotapp

bind :8080
mode http
stats enable
stats uri /haproxy?stats
stats realm Strictly\ Private
stats auth Another_User:passwd

        acl PATH_app1 path_beg -i /app1
        acl PATH_app2 path_beg -i /app2

use_backend BE_APP1 if PATH_app1
use_backend BE_APP2 if PATH_app2

backend BE_APP1
server app1Server app1:8080 check

backend BE_APP2
server app2Server app2:8080 check
```

In all my applications are shown below in docker dashboard ,

Fig 30 : Haproxy configuration



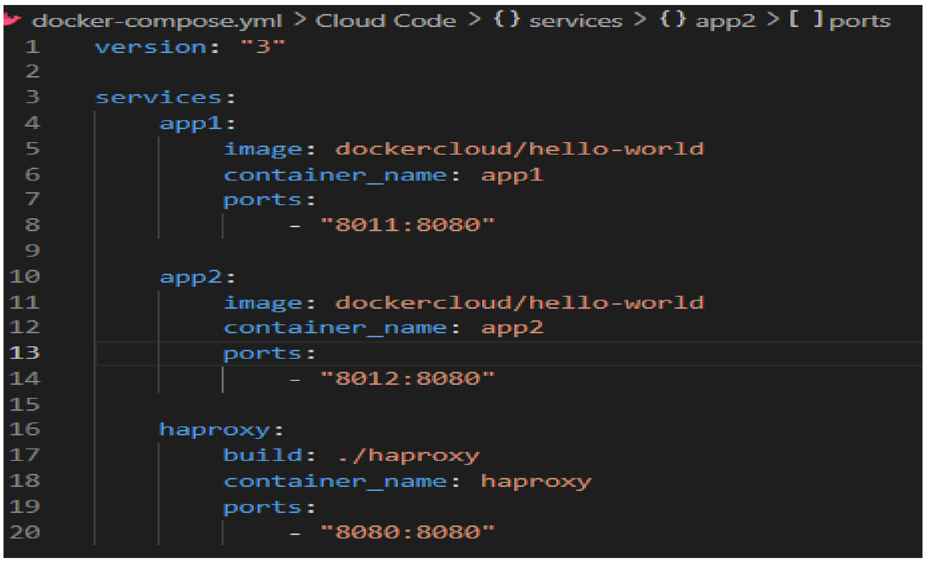
Images on disk 18 images Total size: 2.59 GB IN USE UNUSED

		TAG	IMAGE ID	CREATED	SIZE
test_haproxy	IN USE	latest	d58f9bce6fd8	4 days ago	82.87 MB
haproxy		latest	6d786b4e6316	16 days ago	82.87 MB
app2	IN USE	latest	cf7f3fd05251	16 days ago	939.58 MB
app1	IN USE	latest	dbc12aecb75f	16 days ago	939.58 MB
docker/desktop-kuber...		kubernetes-v1.19....	93b3398dbfde	about 1 month ago	284.82 MB
jrei/systemd-ubuntu	IN USE	18.04	6598efa5fb5f	about 1 month ago	156.42 MB

Case 4 : Docker Compose

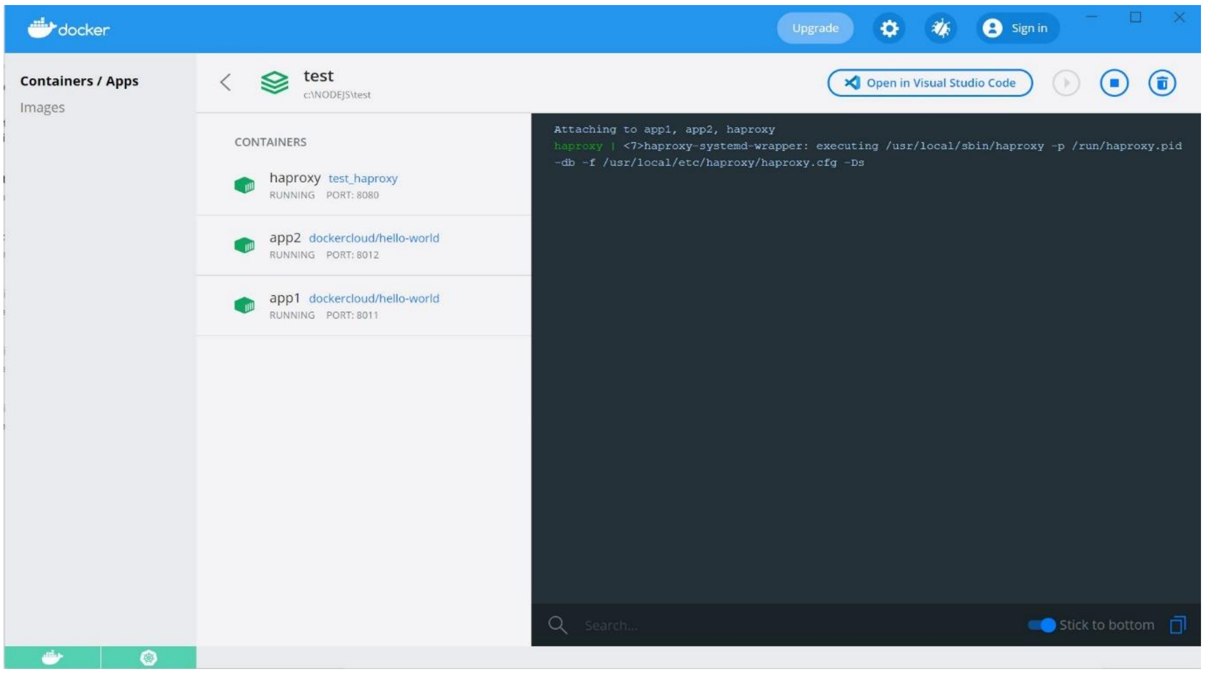
It's a platform for making and maintaining inter Docker software, and it modifies everything for of the applications' services and creates and starts all of them from configuration. It's composed of a single port hosting different environments. In order to prevent builds from competing with one another. I listed application 1 and application 1 and haproxy ports, as well as container id, in the compose yaml file.

Fig 31 : compose



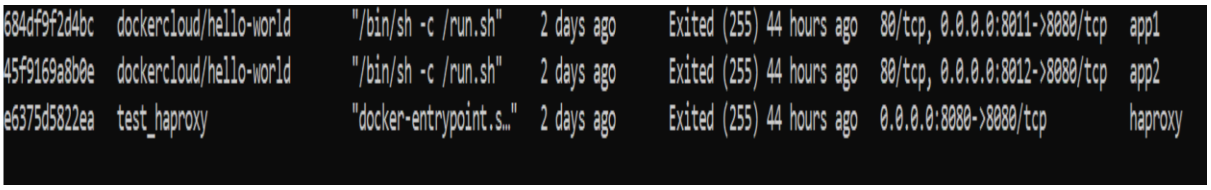
```
docker-compose.yml > Cloud Code > {} services > {} app2 > [ ] ports
version: "3"

services:
    app1:
        image: dockercloud/hello-world
        container_name: app1
        ports:
            - "8011:8080"

    app2:
        image: dockercloud/hello-world
        container_name: app2
        ports:
            - "8012:8080"

    haproxy:
        build: ./haproxy
        container_name: haproxy
        ports:
            - "8080:8080"
```

Once yaml file is running , it showing in docker desktop



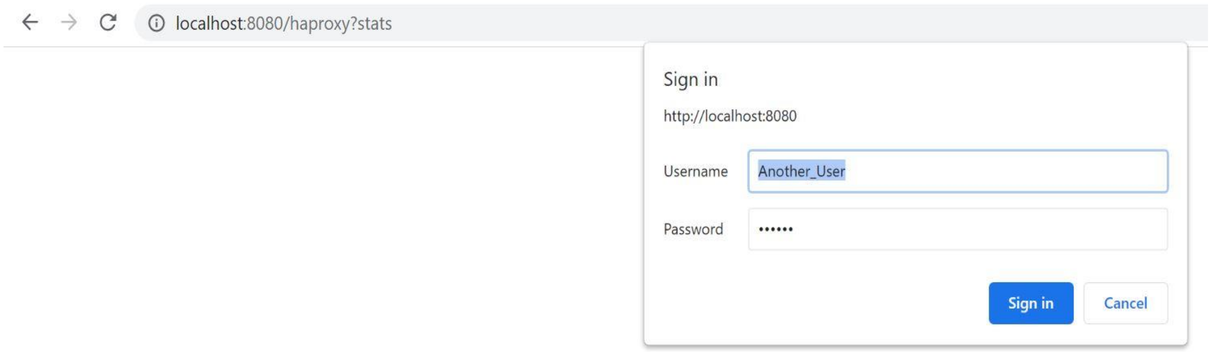
```
Attaching to app1, app2, haproxy
haproxy | <7>haproxy-systemd-wrapper: executing /usr/local/sbin/haproxy -p /run/haproxy.pid
-db -f /usr/local/etc/haproxy/haproxy.cfg -Ds
```

App1 and app2 will be running on port 8080



```
684df9f2d4bc   dockercloud/hello-world    "/bin/sh -c /run.sh"    2 days ago    Exited (255) 44 hours ago   80/tcp, 0.0.0.0:8011->8080/tcp   app1
45f9169a8b0e   dockercloud/hello-world    "/bin/sh -c /run.sh"    2 days ago    Exited (255) 44 hours ago   80/tcp, 0.0.0.0:8012->8080/tcp   app2
e6375d5822ea   test_haproxy               "docker-entrypoint.s…"  2 days ago    Exited (255) 44 hours ago   0.0.0.0:8080->8080/tcp           haproxy
```

After we will login haproxy configuration using http://localhost:8080/haproxy?stats



localhost:8080/haproxy?stats

Sign in

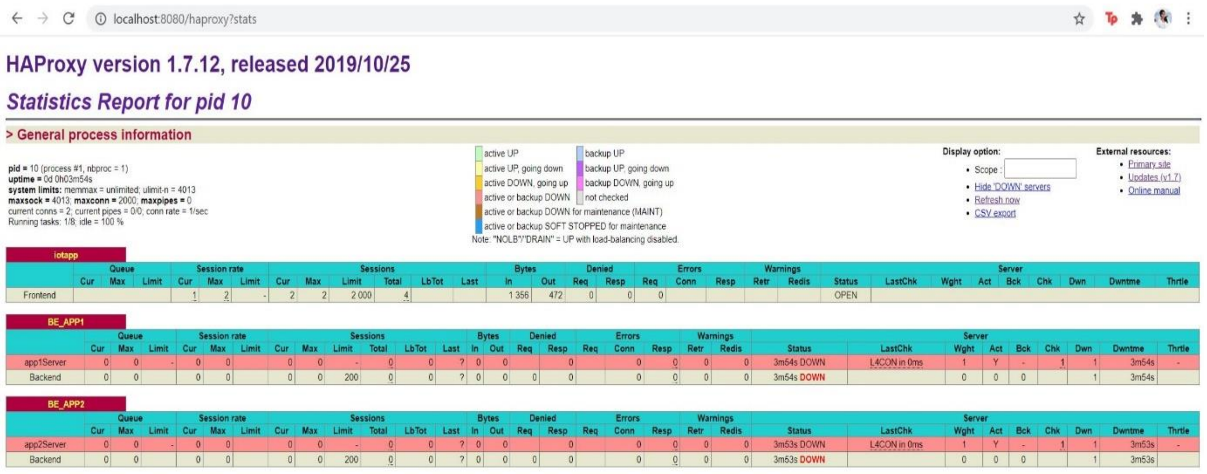
http://localhost:8080

Username: Another_User

Password: ••••••

In haproxy configuration my two application will showing and running

Fig 32 : Haproxy cfg



localhost:8080/haproxy?stats

HAProxy version 1.7.12, released 2019/10/25

Statistics Report for pid 10

> General process information

pid = 10 (process #1, nbproc = 1)
uptime = 0d 0h03m54s
system limits: memmax = unlimited; ulimit-n = 4013
maxsock = 4013; **maxconn =** 2000; **maxpipes =** 0
current conns = 2; current pipes = 0/0; conn rate = 1/sec
Running tasks: 1/8; idle = 100 %

active UP | backup UP
active UP, going down | backup UP, going down
active DOWN, going up | backup DOWN, going up
active or backup DOWN | not checked
active or backup DOWN for maintenance (MAINT)
active or backup SOFT STOPPED for maintenance
Note: "NOLB"/"DRAIN" = UP with load-balancing disabled.

Display option:
- Scope :
- Hide 'DOWN' servers
- Refresh now
- CSV export

External resources:
- Primary site
- Updates (v1.7)
- Online manual

iotapp

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings			Server						
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				1	2	-	2	2	2 000	4			1 356	472	0	0	0					OPEN								

BE_APP1

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings			Server						
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
app1Server	0	0	-	0	0		0	0	-	0	0	?	0	0		0		0	0	0	0	3m54s DOWN	L4CON in 0ms	1	Y	-	1	1	3m54s	-
Backend	0	0		0	0		0	0	200	0	0	?	0	0	0	0		0	0	0	0	3m54s DOWN		0	0	0		1	3m54s	

BE_APP2

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings			Server						
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
app2Server	0	0	-	0	0		0	0	-	0	0	?	0	0		0		0	0	0	0	3m53s DOWN	L4CON in 0ms	1	Y	-	1	1	3m53s	-
Backend	0	0		0	0		0	0	200	0	0	?	0	0	0	0		0	0	0	0	3m53s DOWN		0	0	0		1	3m53s	

Chapter 9

Conclusion

The thesis' aim was to build a Docker image of the Nodejs application and configure it with HAProxy. The target was met, and the application's test version was made available. The next move will be to automate and deploy to a real production environment using the existing file generated during the process.

The thesis project, on the other hand, was a lot more exciting. I had never heard of Docker or HAProxy before starting, so it took a lot of testing and trying out new stuff to get to the desired result.

A most challenging part of this thesis has been the introduction of an entirely new topics. It took a lot of reading through the manuals and trial and error because I had so little knowledge of Docker and HAproxy. As the thesis progressed, keeping track of sources became more difficult, and potential typing errors became more common, necessitating more attention.

The configuration of the Dockerfile and docker-compose.yml file, in my view, was done well. When moving on to the compose file, there is no problem with the application being deployed. Also, for a first-timer, the Haproxy setup was well-done.

What might have been done differently, for example, was to use a larger cluster, such as the minimal cluster, which is designed for local growth but not for deployment. However, since they all use the instructions for the installation process differs. Additionally, instead of using the docker-compose.yml file for deployment, will try in Kubernetes YAML files .

This project has broadened my expertise in the fields of containerization, virtualization, domains, and networking in general, and has put me on the path to becoming a DevOps professional in the future.

Chapter 10

Reference

https://searchitoperations.techtarget.com/definition/Docker-Engine

https://geekflare.com/docker-architecture/

https://devopedia.org/docker

https://docs.docker.com/get-started/overview/

https://hub.docker.com/editions/community/docker-ce-desktop-windows

https://docs.docker.com/config/daemon/

https://dockerlabs.collabnix.com/beginners/components/server_client.html#index

https://docs.docker.com/registry/

https://docs.docker.com/docker-hub/

https://docs.docker.com/config/labels-custom-metadata/

What is Docker? The spark for the container revolution | InfoWorld

https://docs.docker.com/get-started/overview/#docker-registries

https://docs.docker.com/storage/volumes/

https://www.practical-devsecops.com/lesson-2-docker-images-docker-layers-and-registry/

https://www.docker.com/resources/what-containera

https://medium.com/@BeNitinAgarwal/lifecycle-of-docker-container-d2da9f85959

https://closebrace.com/tutorials/2017-03-02/creating-a-simple-restful-web-app-with-nodejs-express-and-mongodb

https://support.ptc.com/help/thingworx/platform/r9/en/index.html#page/ThingWorx/Help/ThingWorxHighAvailability/HAProxyExample.html

https://searchnetworking.techtarget.com/definition/HAProxy

https://dzone.com/articles/8-lessons-learned-using-docker-compose https://docs.docker.com/compose/compose-file/

https://www.digitalocean.com/community/tutorials/how-to-use-haproxy-to-set-up-http-load-balancing-on-an-ubuntu-vps

https://towardsdatascience.com/deploy-a-nodejs-microservices-to-a-docker-swarm-cluster-docker-from-zero-to-hero-464fa1369ea0

https://www.digitalocean.com/community/tutorials/how-to-create-a-cluster-of-docker-containers-with-docker-swarm-and-digitalocean-on-ubuntu-16-04

https://callistaenterprise.se/blogg/teknik/2017/12/18/docker-in-swarm-mode-on-docker-in-docker/

https://www.metaltoad.com/blog/docker-containers-clusteringorchestration

https://www.ansys.com/products/platform/ansys-high-performance-computing