IT3036 Final Project Report

Jordan Christopher Wentland

April 27, 2022

**Summary**

When deciding on a final project, I wanted to explore a concept we touched very little on, but one that I was very interested in. Because of this curiosity, I settled on exploring more technological applications for Docker, specifically Docker Compose when it comes to an enterprise. The project I ended up settling on was using Docker to create multiple web application containers using the HTTPD image as well as using an NGINX container to act as a reverse proxy and a load-balancer for the web app(s) being used by an enterprise. Using containers for an enterprise is a very easy and scalable way for running web applications. If more instances are needed for smooth operation, we can simply spin up more containers! What this project does, is creates a simple way for accessing these web app containers while also ensuring scalability, reliability, and usability. The main focal points of this project are the use of docker-compose and NGINX to create an easily accessible web application environment for developers. By using docker-compose we can create all of our containers in one manageable and editable stack all on the same docker network for security purposes. Docker-compose makes creating multi-container projects extremely easy due to its YAML formatting. Through the use of NGINX, we can orchestrate our network traffic to direct to different containers/websites and even load balance these containers to ensure an even distribution of work across our servers.

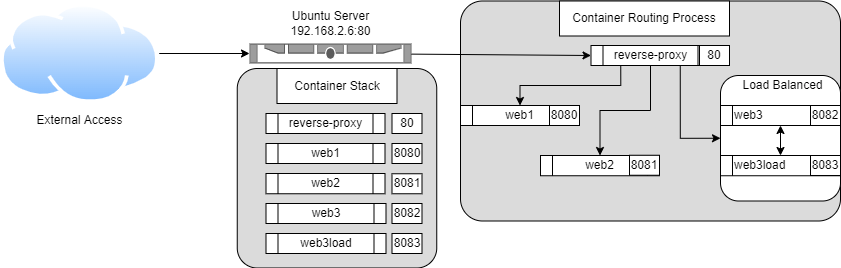
**Alternatives and Options – containers**

Before deciding on using docker for the containerization platform for this project, another option was also considered. This option was using Kubernetes for orchestration. Kubernetes is a container orchestration platform that allows you to distribute a load of processing under many clusters or instances. When comparing Docker to Kubernetes, it is not quite fair to compare the two. Docker runs containers while the main focus of Kubernetes is to orchestrate and scale said containers. Because of this aspect, I decided to more focus on this project on a single web app environment and opted to use docker and docker-compose for the creation of this project.

**Alternatives and Options – reverse proxy + load balancer**

When exploring alternatives, many options come to mind depending on where the content is hosted, what kind of content is being hosted, budgetary concerns, hardware requirements, etc. To help better focus our exploration into alternatives, we will look at the three major competitors in the containerized space: HAProxy, Traefik, and Nginx. It's important to understand that this analysis will be coming from the enterprise perspective, not from an open-source individual users' perspective. All three of these software can act as both a load balancer and as a reverse proxy. The first major thing to consider when selecting technology is pricing. HAProxy starts at roughly $2500 a year for basic support and licensing. Traefik starts at $20,000 per cluster with support included. Finally, Nginx plus starts at $2,500 per year for support and licensing. With pricing in mind, we can narrow our options down to HAProxy and Nginx for further analysis. Both HAProxy and NGINX have containerized and standalone installation options with similar hardware requirements for each making them net even on this playing field. When it comes to the performance of these two options, HAProxy can be a bit faster depending on the scale and use case of the environment when compared to NGINX. HAProxy has a bit of a simpler setup to it, but NGINX has a more detailed and finer tunable setup. For scalability, both of these options can be scaled up or down depending on how much resource is needed to appropriately function. One major standout between HAProxy and NGINX is that NGINX can also operate fully on its own as a web server, proxy, and load balancer- this means in a non-containerized setup, the only software solution technically needed to get started is NGINX. With how similar these two products are to one another in functionality, pricing, and requirements, the choice of what to choose comes down to the application and the desire for tunability. If an organization is looking to stand up a reverse proxy and load balancer without needing much fine-tuning, HAProxy could be a good solution! However, if a company needs that webserver functionality or wants finer tunable configurations, NGINX may be a better option for your application.

**Network Diagram Breakdown**

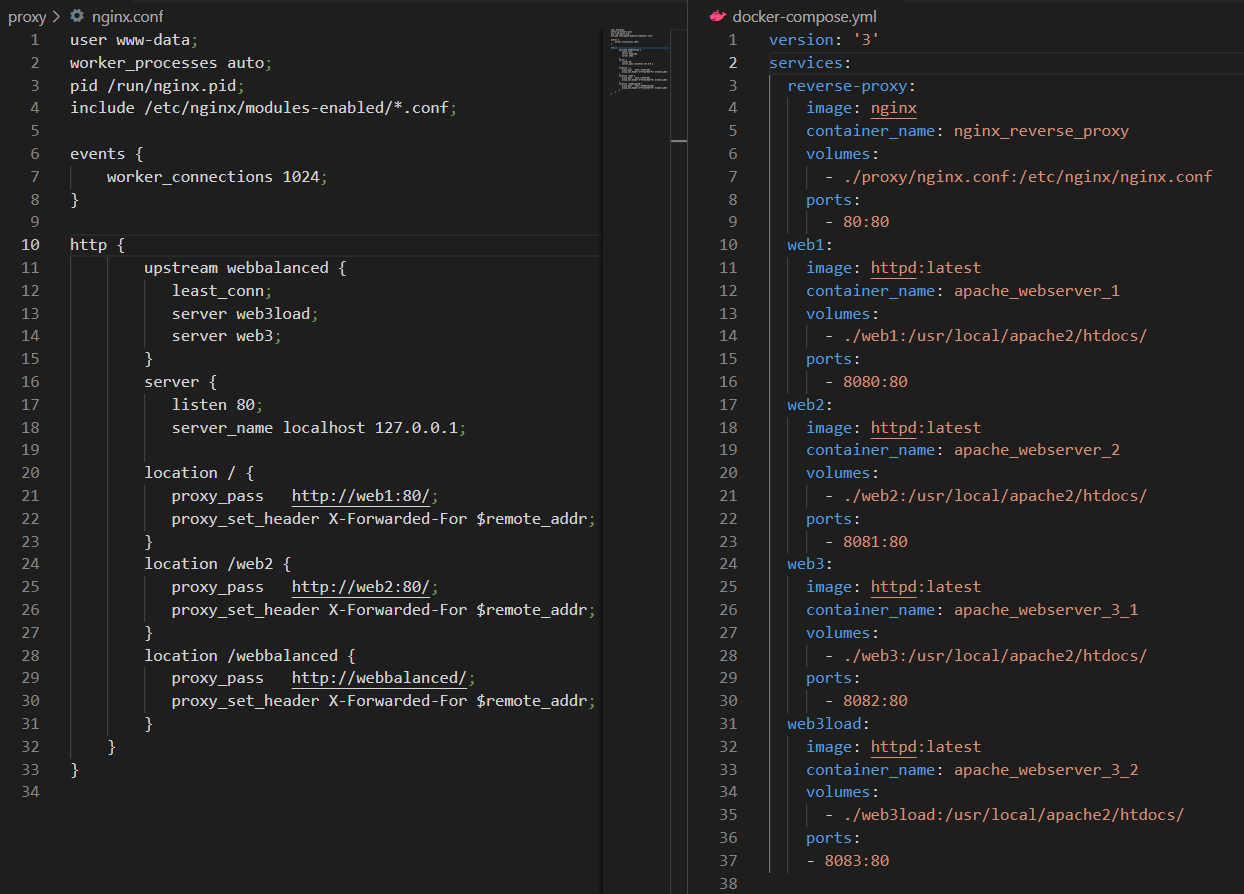


Above is a network diagram of how the information is processed upon receiving a request. When an external device queries our server on port 80 (HTTP) the information is directed to our container titled "reverse-proxy". This container is our Nginx container. The reverse-proxy container then examines the URL and uses that information to determine the next hop. Our reverse proxy is designed to look for the following 3 modifiers:

1. /
2. /web2
3. /webbalanced

f a device attempts to query http://192.168.2.6/, the information is passed to our web1 container where a webpage indicating the container is hosted. If a device attempts to query http://192.168.2.6/web2/, the request is passed to the web2 container where similar to web1, a webpage appears showing the container we are currently in. Finally, if a device attempts to query http://192.168.2.6/webbalanced/, the information is passed to either web3 or the web3load container. These two containers are tied together in NGINX to be a load-balanced setup. This will be explained more in the configuration breakdown. When queried, the client would end up on 1 of 2 pages depending on the container it is dropped into. In a normal production setup, these containers would ideally be identical in function and design to make disruption to the user minimal. This whole setup creates a very efficient and scalable environment for a developer's web application and allows for the organization of applications better.

**Configuration files**

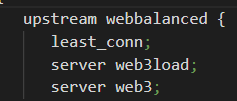


* **Docker-compose.yml**

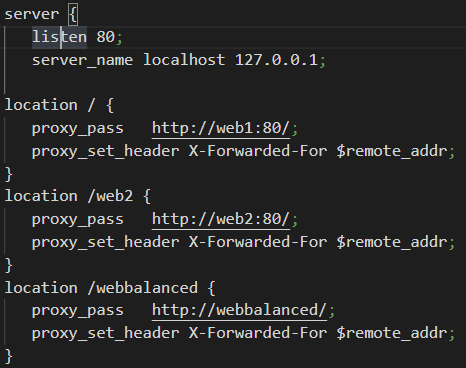
Above are both my docker-compose.yml files as well as my Nginx.conf file. For starts, the docker-compose file orchestrates what containers will be started, the mounting volumes, the ports to expose, and what names to assign the containers. Starting from the top of the docker file, we are using docker-compose version 3 to spin up a total of 5 services. Our first service "reverse-proxy" is the Nginx container and it is copying the Nginx.conf from a directory on the device and running on port 80. Our second service is our first web container using the httpd container with a volume mount that contains a basic webpage. This container is exposed on port 8080. Containers web2, web3, and web3load are all the same, with variations in the mount volume and the ports that are exposed. Web2 is exposed on port 8081, web3 is exposed on port 8082, and web4 is exposed on port 8083. This process stands up all of our containers and makes them accessible on the device. Until the Nginx container is configured, all of these containers are fully independent of one another and can be accessed by visiting http://ip:port.

* **Nginx.conf**

The next file we are going to examine is the Nginx.conf file. This file is responsible for all of the brains of our reverse proxy and load balancing. Like the previous config file, we will start from the top of the config and work down. The first block of the Nginx.conf file is described as the "Main Context". This carries all global directives for the configuration. These details affect the entire application being run. Starting off, the "user www-data" directive sets the running user for the app. Next, the worker process directive determines how many processes can spawn. Since ours is auto, it will scale accordingly. The PID directive determines the process id write location and the next line imports the modules we will be using. In our case, we are just importing all available modules for ease of use. With the global directive explained, we will explore the events directive. The events directive allows us to set global options responsible for how Nginx handles connections. In our config, the only directive set here is the "worker\_connections" directive which is set for 1024. What this means is that each worker process (defined above) can handle a maximum of 1024 connections before a new worker process is spawned. With this explored, we can delve into the meat of our config, the HTTP context! The HTTP context is responsible for handling HTTP and HTTPS connections, the bulk of our configuration lays in here.



The first block of configuration in the HTTP context creates our load-balanced instance. The upstream directive allows us to give the outgoing name for the bundled servers. So in our instance, upstream webbalanced will create a callable variable called webbalanced for the reverse proxy. The least\_conn directive sets the load balancing methodology. In this instance, we are using least conn which sends the connection to the container with the least number of connections to it. Lastly, the server directive allows us to select which of our containers is part of the webbalanced upstream.



The last portion of our Nginx config is our server context. The server context contains all of the locations for the reverse proxy as well as the port and server name that will be queried for information. Starting from the top, we are listening on port 80 as it is the standard HTTP port. Our server\_name is localhost/ 127.0.0.1 as we are referencing a host on the local machine. Further down, we start declaring our locations. The location directive is set up as "location /(web URL)". So, the web URL field represents the / location we are navigating to when hitting the server. For web1 and web2, we are using proxy\_pass and setting it to the container that the connection should be proxied. Because of the setup of Nginx and docker, we can point directly to the container name and port. The second line in each of these, proxy\_set\_header, is responsible for reformatting the header to send it to the appropriate location. The final location we are looking at is the /webbalanced location. This is almost the same, except the proxy\_pass points to our previous declared upstream variable as opposed to a container name.

**Conclusion**

To wrap this report up, I want to give another overview of what this project seeks to accomplish. At its core, this project expands upon the docker knowledge taught in class by exploring docker-compose. On top of this, the project incorporates the integration of nginx into the docker stack to demonstrate the reverse proxy process as well as the ability to load balance containers. For an enterprise, this grants the ability to either host internal or external applications in an easily scalable fashion, with built in proxying, and the ability to load balance applications more efficiently giving customers and staff a more consistent experience.

**References**

Advanced load balancer, web server, & reverse proxy. (2022, April 28). Retrieved April 28, 2022, from https://www.nginx.com/

Anicas, M. (2022, March 28). An introduction to HAProxy and load balancing concepts. Retrieved April 28, 2022, from https://www.digitalocean.com/community/tutorials/an-introduction-to-haproxy-and-load-balancing-concepts

Ellingwood, J. (2014, November 19). Understanding the nginx configuration file structure and configuration contexts. Retrieved April 28, 2022, from https://www.digitalocean.com/community/tutorials/understanding-the-nginx-configuration-file-structure-and-configuration-contexts

HAProxy vs Nginx: What are the differences? (n.d.). Retrieved April 28, 2022, from https://stackshare.io/stackups/haproxy-vs-nginx

Kubernetes vs Docker: Microsoft Azure. (n.d.). Retrieved April 28, 2022, from https://azure.microsoft.com/en-us/topic/kubernetes-vs-docker/

Traefik Enterprise Edition on red hat marketplace - pricing - United States. (n.d.). Retrieved April 28, 2022, from https://marketplace.redhat.com/en-us/products/traefik-enterprise-edition/pricing#pricing-and-plans