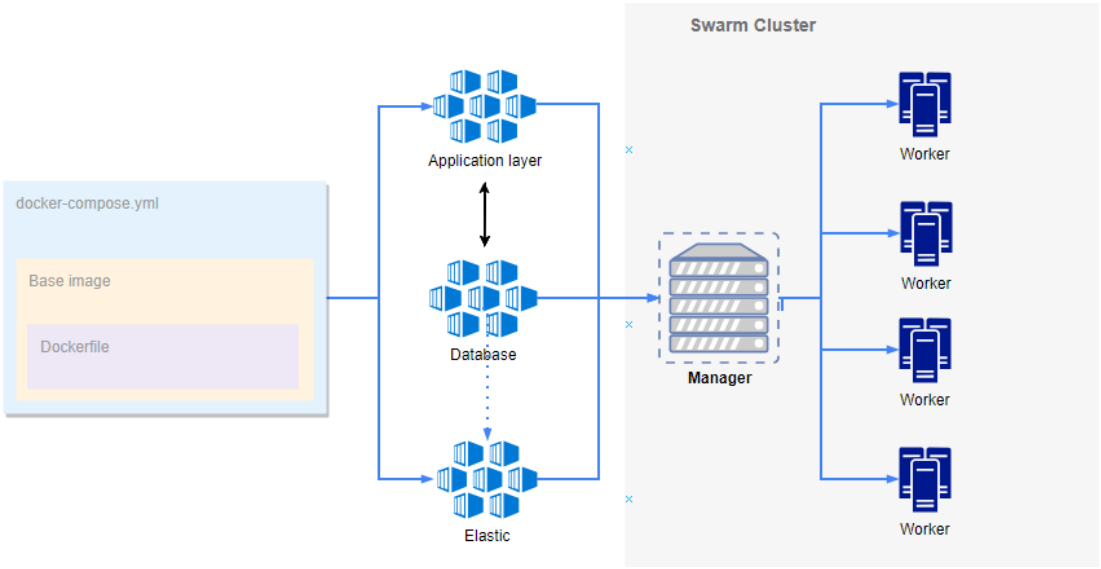
Intro

For the needs of this project have been set up two Apache CentosOS web servers and a proxy server (Nginx) for load balancing on the application layer. A Dockerfile pushed in Github has been used in order to build a custom image hosted in Docker hub. This image will be used into a docker-compose file including other services in order to build and deploy our application in a Swarm cluster.

The above architecture along with the improvement ideas will ensure high availability to the website, performance stability and excited users. (:

Implementation Overview



Implementation Steps

For the environment set up have been used 2 cloud VMs CentOS7 (Core) with Docker installed (Docker version 18.06.0-ce). Having a 2 nodes cluster, 1 swarm manager and 1 worker.

1. Using a Dockerfile created an image to host the application layer of the website and pushed on Docker Hub.

2. Created a docker-compose file to build all the services for the multi-container Docker application:
    a. 2 Web servers (Apache)
    b. Load balancer (Nginx)
    c. Database (mysql)
    d. Monitoring (Elasticsearch)

3. Swarm initialization. Turn the host to Manager with static IP:
    `docker swarm init --advertise-addr <static_IP>`

4. Add Worker node in the cluster. Connect to the second VM and execute:
```
docker swarm join --token SWMTKN-1-
302ydp0vgtz1w2zio1qg7l9b9n6orl6kxlrxjae61foh8qisvu-
cur9mmjmhx2o2v77h80fk14k1 <IP>
```

5. Deploy the stack of services to Swarm cluster. Execute from Management node:
```
docker stack deploy --compose-file docker-compose.yml <name>
```

6. Inspect containers and services across the Swarm cluster.

Automated Build and Deployment

A Jenkins job will automate the deployment of the application. The minimum requirements to achieve it consist of a job that downloads the docker-compose file(s) from an artifact repository and transfer it over ssh to the manager(s). The execution of the below command in each management node will deploy only the updated Docker services including configuration across the worker nodes of the Swarm cluster.

```
docker stack deploy --compose-file docker-compose.yml webstack
```

Since the application code is directly embedded in the base image, the Jenkins job could also easily update a concrete artifact inside the web server layer. The process can be automated even more according to the needs. For example, trigger a build process in this repository whenever a developer commits the code.

High Availability

High Availability for the whole service can be assured with the below solutions:

- Implementation of a self-healing environment with automated processes to replace/reboot broken containers or failed managers based on system health checks at any time. Force rebalance required.
- Automated scaling of the resources in response to real-time data metrics to ensure stability and performance even with massive network traffic.
- Distribute Management nodes to multiple datacenters or racks.
- Run Manager-only nodes.
- Spread the deployment of applications over multiple servers.
- Backup and recovery plan for the Swarm cluster.

Improvement steps

Below you may find some improvement steps for the provided solution and a few matters to be considered.

Short term:

- Point domain name to Docker Swarm manager.
- Scale up and incorporate multiple managers and workers in the cluster.
- Implement ELK Stack for centralized logging, monitoring and easier troubleshooting.
- Consider application dependencies.
- Networking limitations and timeouts.
- Improve configuration using variables where possible.

Long term:

- Storage and sizing requirements.
- Automate the Swarm cluster creation process through Jenkins (and maybe docker-machine).
- Automate housekeeping processes.