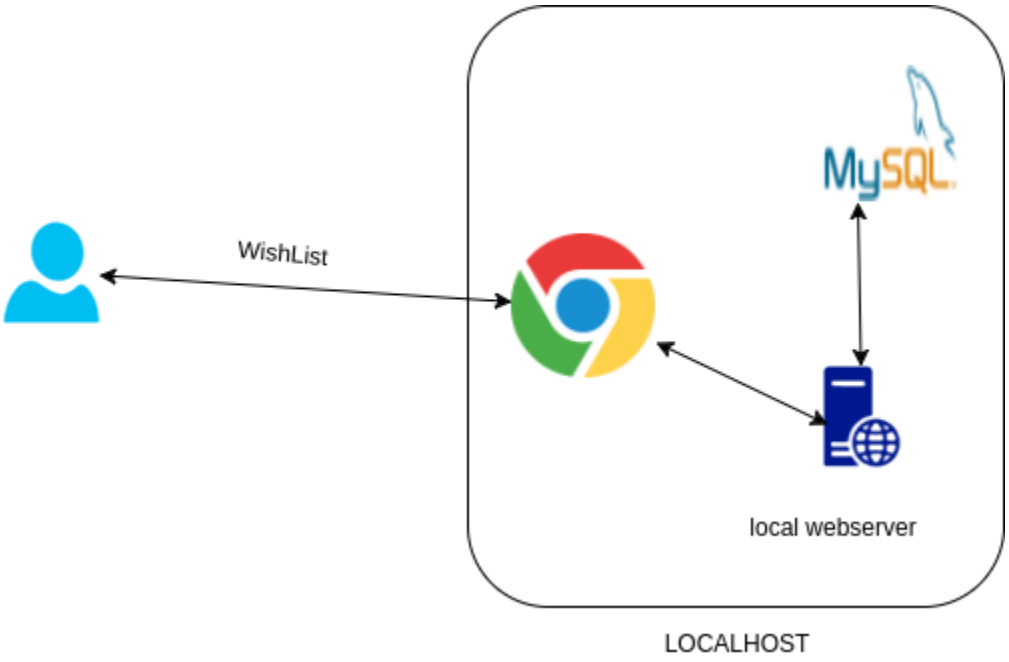
Chandrashekar Dashudu

Web Application documentation

1) Create a Web App

Requirements-
Framework used: Python Flask
Front End: HTML, CSS, JS, AJAX.
Database: MYSQL.
OS: Ubuntu 16.04

Here the initial setup is done on localhost using the above mentioned technologies. The model on localhost is shown below.



The web app is “WishList” which gives a simple UI to the user to enter any wishes and that is stored in the database.

The user can login to the website using any standard browser and Create/Edit/Delete wishes

This is just a basic web app and this can be further extended to add more features like uploading images, add a “LIKE” button etc.

Implementation

As shown in the model, the database is a mysql server which stores all use data like credentials, the entries the user makes etc.

The backend is written using python flask which has the main logic for validating the user, adding wishes/ToDO's editing and deleting them.

The backend has a few pages like HomePage, Login, Logout, AddWish, and an Error page.

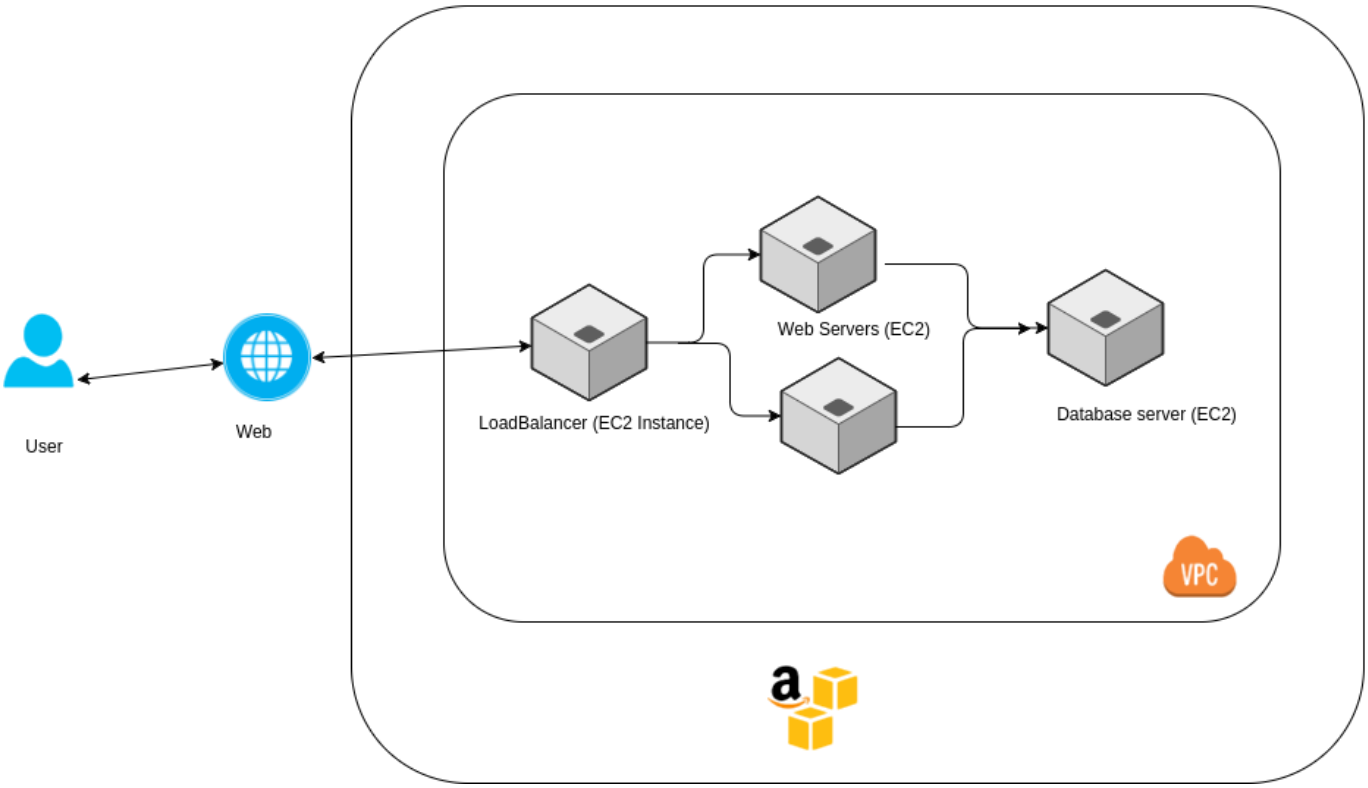
Any request that is made through the browser is redirected to the backend using ajax and the function written in flask handles and manages the request made.

2) **Python client to access (attached)**

   Due to limited time because of simultaneous college projects and assignment, I have implemented only login, logout, signup and add wish feature using the python client and I did not get a chance to provide error handling where I just need to add some checks for user inputs.

3) **Scale -up the application**

After successfully testing the web application on localhost I deployed the web app on AWS and the model is shown below.



In the above model, we see that all the server's and the load balancer is deployed on AWS on EC2 Instances.

The Load balancer used here is NGINX which in this case uses round robin algorithm to distribute the load between the webservers. The Load balancer is configured with the IP address of the the 2 web servers and this is how the any hit on the load balancer is directed to one of the web servers.

The web app is deployed on the Web servers which are also EC2 instances. The web app is always running as a daemon until asked to stop. The web server is configured with the IP address of the Database server to access the data stored there.

Both the Web Server is connected to a DB server which stores all the data for the web app.

4) Secure your servers

Security here is implemented using the following methods.

1. Only load balancer is open to the wild internet. Any request to the web server directly is discarded with the help of IPTABLES config.

2. Only the Load Balancer has an public IP address to access the web app

3. Since no other machine or EC2 (Web server and DB server) has an public IP, No one can login or send any http request to the web server directly.

4. There is IPTABLES configured on the web servers, which do not allow any traffic on any port from the wild internet except for the Load Balancer. This ensures that we can SSH to the web servers and the DB server using the Load Balancer.

5. Thus, the Load Balancer acts as a Bastion host

6. We can login to this Load Balancer using a private key without which SSH is not possible. Also to SSH to the Load Balancer, an IPTABLE is entry is made to allow access from only the ADMIN's machine (which is mine's in this case)
7. There is IPTABLE rules setup on the DB server which accepts connection the port 3306 from the IP address of the Web Servers.

8. The Web Servers are in the private IP block 10.0.0.0/16 and all the incoming requests to the DB server from this block of IP address is allowed.

9. DB server has another IP table entry which allows SSH from only the Load Balancer to make any configuration changes.

10. Now since the Web servers and the Database server does not have a Public IP, the Load Balancer can be made as a NAT Instance to install any update on the Web and the Database server.

5) Automate the deployment

1) In order to automate the deployment of the web servers and the DB server, I have created Amazon Machine Image (AMI) for the web server, DB server and the load balancer. AMI is nothing but a OS image with all the configuration files preloaded which reduces the time to setup and deploy the web app

2) Having the AMI images makes it very quick to deploy the web servers in case we want to add more of the Web Server and DB server.

3) I have an Ansible Playbook which can deploys the Web Server, DB server and the Load Balancer as and when required.

Automation here is done in 2 levels.

1) Startup and Shutdown services.

I have 2 services running one of which runs during the startup and the other during the shutdown.

i) Startup Service:

Whenever the ansible script is run to bring up a web server, there is a service runs which performs the following tasks.

a) Configure the IPTABLES rules
b) Join the Load Balancer so that the it can also serve traffic for it user
c) Start the Web app as a Daemon service

ii) Shutdown script:

When the web server is no longer required in case of low load on the web server, it is shut down and the shut down service does the following task.

a) Shut down the web app
b) Leave the Load Balancer

2) Ansible Script

As mentioned earlier, I have a play book which connects to AWS, deploys the Servers in a Virtual Private cloud and assigns storage and attaches the EC2 instance to the security group.

No machine except for the Load balancer has a Public IP.