

IT490 - Systems Integration

Authors: Gabriel Aquende, Edwin Rodriguez, Artur Serafim,
Franklin Pavon

Project Name : Tales

Version: 2.0 (Final)

About :

The Tales project is a cocktail recommendation application based on a quiz the user take. The quiz will be asking the user 1) If they want a alcoholic drink or not 2) What category of drinks they want 3) If they select an alcoholic drink then it will ask what primary liquor they want 4) The last question will ask what ingredients or flavor notes the user wants to taste in their drink.

Technologies:

The Tales projects utilizes PHP, JavaScript, HTML, BS, and CSS on the Front End. The DMZ is a file of Python functions that query the API and returns results. Specifically the API's that will be utilized The DB uses MySQL in order to store data. Lastly RabbitMQ communicates with the Front End, DMZ, and the Database. RabbitMQ is a messaging system that helps glue all our systems together without communicating directly with each other. All of our environment is running on Google Cloud.

Launch:

First RabbitMQ server must be started on one VM. Next the Database will have the scripts for Registration, Login, User, Friends , and Favorites that will listen to user Queries generated by the Front End. At the same time the Database runs their script the DMZ will be running scripts for recommended drinks, popular drinks, and for custom features like to plan my night. The Front End is then started by getting onto the local host which contains our website. The user is then prompted to register and log in. They are then able to select what

action they would like to do and if any of those options require access to live CocktailDB or Yelp data then the DMZ will go out and get the desired information.

RabbitMQ:

For RabbitMQ, simply turn on service and monitor queues to confirm that messages are being sent to correct queues. Queues to be monitored are APISend, Register, User, alcoholic, alcoholicReply, favorites, friends, friendsReply, login, loginReply, newFriends, popular, quiz and registerReply.

• Firewall Rules

Firewall rules

CREATE FIREWALL RULE

REFRESH

DELETE

Firewall rules control incoming or outgoing traffic to an instance. By default, incoming traffic from outside your network is blocked. [Learn more](#)

Note: App Engine firewalls are managed [here](#).

Filter resources

Columns

<input type="checkbox"/>	Name	Type	Targets	Filters	Protocols / ports	Action	Priority	Network ^
<input type="checkbox"/>	block-http	Egress	nointernet	IP ranges: 0.0.0.0/0	tcp:80,443	Deny	1000	default
<input type="checkbox"/>	allowemail	Ingress	Apply to all	IP ranges: 0.0.0.0/0	tcp:587	Allow	1000	default
<input type="checkbox"/>	allowsmtp	Ingress	Apply to all	IP ranges: 0.0.0.0/0	tcp:25 udp:25	Allow	1000	default
<input type="checkbox"/>	allowtv	Ingress	Apply to all	IP ranges: 0.0.0.0/0	tcp:5938	Allow	1000	default
<input type="checkbox"/>	allowtvudp	Ingress	Apply to all	IP ranges: 0.0.0.0/0	udp:5938	Allow	1000	default
<input type="checkbox"/>	default-allow-http	Ingress	http-server	IP ranges: 0.0.0.0/0	tcp:80	Allow	1000	default
<input type="checkbox"/>	default-allow-https	Ingress	https-server	IP ranges: 0.0.0.0/0	tcp:443	Allow	1000	default
<input type="checkbox"/>	default-allow-icmp	Ingress	Apply to all	IP ranges: 0.0.0.0/0	icmp	Allow	65534	default
<input type="checkbox"/>	default-allow-internal	Ingress	Apply to all	IP ranges: 10.128.0.0/9	tcp:0-65535 udp:0-65535 icmp	Allow	65534	default
<input type="checkbox"/>	default-allow-rdp	Ingress	Apply to all	IP ranges: 0.0.0.0/0	tcp:3389	Allow	65534	default
<input type="checkbox"/>	default-allow-ssh	Ingress	Apply to all	IP ranges: 0.0.0.0/0	tcp:22	Allow	65534	default

Rows per page: 50

1 - 11 of 11

Our firewall rules block internet access on non api and

frontend servers along with allowing TeamViewer remote control.


- **Instance Groups**

Our Project consists of 13 VMs with 3 different tiers and a version control machine that pushes updates from one tier to another

development-group

Members Details Monitoring

Zone: **us-east1-b** Network: **default** Subnetwork: **default** In use by:

 Filter group members

 Columns


<input type="checkbox"/>	Name	Creation time	Template	Health check status	Internal IP	External IP	Connect
<input type="checkbox"/>	 actual-api	Nov 21, 2019, 10:37:33 AM			api-ip (10.142.0.5) (nic0)	35.211.126.171 	SSH 
<input type="checkbox"/>	 api-dmz	Nov 30, 2019, 10:21:51 PM			rabbitmq-dev (10.142.0.3) (nic0)	35.211.196.245	SSH 
<input type="checkbox"/>	 database-server	Nov 24, 2019, 6:12:36 PM			ip4 (10.142.0.6) (nic0)	35.227.98.189	SSH 
<input type="checkbox"/>	 frontend-server	Nov 17, 2019, 11:51:30 PM			ip3 (10.142.0.7) (nic0)	35.237.18.128 	SSH 





staging-group

Members Details Monitoring

Zone: **us-central1-a** Network: **default** Subnetwork: **default** In use by:

 Filter group members

 Columns

<input type="checkbox"/>	Name	Creation time	Template	Health check status	Internal IP	External IP	Connect
<input type="checkbox"/>	 api-server-staging	Nov 24, 2019, 6:30:50 PM			ip2 (10.128.0.4) (nic0)	35.208.91.51 	SSH 
<input type="checkbox"/>	 database-server-staging	Nov 24, 2019, 6:28:29 PM			ip1 (10.128.0.3) (nic0)	None	SSH 
<input type="checkbox"/>	 frontend-server-staging	Nov 24, 2019, 6:58:23 PM			ip4 (10.128.0.6) (nic0)	None	SSH 
<input type="checkbox"/>	 rabbitmq-server-staging	Nov 24, 2019, 6:57:14 PM			ip3 (10.128.0.5) (nic0)	None	SSH 

Instance groups





 CREATE INSTANCE GROUP

 REFRESH

 DELETE

 Filter resources

 Columns

<input type="checkbox"/>	Name 	Zone	Instances	Template	Creation time	Recommendation	Autoscaling	In use by
<input type="checkbox"/>	 development-group	us-east1-b	4	—	Nov 24, 2019, 6:20:43 PM			
<input type="checkbox"/>	 production-group	us-east1-d	4	—	Nov 24, 2019, 8:12:48 PM			
<input type="checkbox"/>	 staging-group	us-central1-a	4	—	Nov 24, 2019, 7:01:50 PM			

production-group

[Members](#) [Details](#) [Monitoring](#)

Zone: **us-east1-d** Network: **default** Subnetwork: **default** In use by:

Filter group members

<input type="checkbox"/>	Name	Creation time	Template	Health check status	Internal IP	External IP	Connect
<input type="checkbox"/>	<input checked="" type="radio"/> api-server-production	Nov 24, 2019, 7:39:40 PM			apiserver-ip (10.142.0.18) (nic0)	None	SSH ▾
<input type="checkbox"/>	<input checked="" type="radio"/> database-server-production	Nov 24, 2019, 7:33:06 PM			database-ip (10.142.0.17) (nic0)	None	SSH ▾
<input type="checkbox"/>	<input checked="" type="radio"/> frontend-server-production	Nov 24, 2019, 7:57:06 PM			frontend-ip (10.142.0.20) (nic0)	None	SSH ▾
<input type="checkbox"/>	<input checked="" type="radio"/> rabbitmq-server-production	Nov 24, 2019, 7:55:35 PM			rabbitmq-ip (10.142.0.19) (nic0)	None	SSH ▾

Snapshots [Snapshot schedules](#)

Filter snapshots

?

Columns

<input type="checkbox"/>	Name ^	Location	Snapshot size	Creation time	Creation type	Source disk	Disk size
<input type="checkbox"/>	<input checked="" type="checkbox"/> dev-api	us-east1	2.43 GB	Dec 16, 2019, 11:47:31 PM	Manual	actual-api	15 GB
<input type="checkbox"/>	<input checked="" type="checkbox"/> dev-frontend	us-east1	2.42 GB	Dec 16, 2019, 11:48:21 PM	Manual	frontend-server	15 GB
<input type="checkbox"/>	<input checked="" type="checkbox"/> dev-rabbitmq	us-east1	2.63 GB	Dec 16, 2019, 11:46:35 PM	Manual	api-dmz	25 GB
<input type="checkbox"/>	<input checked="" type="checkbox"/> dev-sql	us-east1	2.45 GB	Dec 16, 2019, 11:47:01 PM	Manual	database-server	25 GB
<input type="checkbox"/>	<input checked="" type="checkbox"/> prod-api	us-east1	2.12 GB	Dec 16, 2019, 11:54:15 PM	Manual	api-server-production	20 GB
<input type="checkbox"/>	<input checked="" type="checkbox"/> prod-frontend	us-east1	2.99 GB	Dec 16, 2019, 11:54:38 PM	Manual	frontend-server-production	20 GB
<input type="checkbox"/>	<input checked="" type="checkbox"/> prod-rabbitmq	us-east1	2.83 GB	Dec 16, 2019, 11:55:03 PM	Manual	rabbitmq-server-production	20 GB
<input type="checkbox"/>	<input checked="" type="checkbox"/> prod-sql	us-east1	2.57 GB	Dec 16, 2019, 11:55:31 PM	Manual	database-server-production	20 GB
<input type="checkbox"/>	<input checked="" type="checkbox"/> staging-api	us-central1	2.13 GB	Dec 16, 2019, 11:52:36 PM	Manual	api-server-staging	15 GB
<input type="checkbox"/>	<input checked="" type="checkbox"/> staging-frontend	us-central1	2.8 GB	Dec 16, 2019, 11:51:39 PM	Manual	frontend-server-staging	15 GB
<input type="checkbox"/>	<input checked="" type="checkbox"/> staging-rabbitmq	us-central1	2.37 GB	Dec 16, 2019, 11:51:09 PM	Manual	rabbitmq-server-staging	15 GB
<input type="checkbox"/>	<input checked="" type="checkbox"/> staging-sql	us-central1	2.35 GB	Dec 16, 2019, 11:52:13 PM	Manual	database-server-staging	15 GB
<input type="checkbox"/>	<input checked="" type="checkbox"/> versioncontrol-snapshot	us-central1	2.36 GB	Dec 16, 2019, 11:57:13 PM	Manual	version-control	18 GB

Snapshots

We regularly update snapshots of servers before and after completing work on the VMs

Front End:

The front-end portion of Tales where the user interacts with the web app. Tales is hosted on our Ubuntu machine using apache2.

The web pages UI were built using HTML, CSS and JavaScript, while the functionality of the pages and communication with RabbitMQ was programmed in PHP.

1. Login Page

- a. Built using HTML and CSS libraries like Bootstrap.
- b. The user inputs their login credentials within the form and then hits the submit button, triggering a php action.
- c. The php action page takes the user input, assigns them to variables, and inputs the values into a session array, then it sends the values through a message queue through RabbitMQ. The message queue sends it to the database for authentication.
- d. The php action page also sends a log message through RabbitMQ to the database stating that somebody attempted to login.
- e. After the messages have been sent, it redirects you to the php receive page, where we receive one of two messages from RabbitMQ coming from the database. Either the user was authenticated, and we receive their First name, last name, and zip code to add to the session array. Or we receive a "User does not exist" message and then the user gets redirected to the login page again with an error message.

2. User Dashboard

- a. Once the user is authenticated, they are placed in the dashboard page. The page verifies that they have a valid session. The page uses the same web form layout as the login but customized to fit 4 buttons. The four buttons are:
 - i. Take or Retake Quiz
 - ii. Plan my night
 - iii. Popular Drinks
 - iv. User Profile
 - v. Logout
- b. The page is built using HTML and CSS libraries like Bootstrap.
- c. If a user clicks on "Take or Retake Quiz" they are just redirected to the quiz page.
- d. If a user clicks on "Popular Drinks" they are just redirected to the popular drinks page.
- e. If a user clicks on "Logout", their session is destroyed, and they are redirected to the login page.
- f. If a user clicks on "Plan my night", the php action page sends the zip code in the session array through

RabbitMQ to the api. A log sentence is also sent to through RabbitMQ to the database. Once the messages have been sent, the php listening page listens for RabbitMQ to deliver a message from the api containing a list of local bars and their information to help the user plan their evening.

g. If a user clicks on "User Profile", the php action page sends the email in the session array through RabbitMQ to the database and a log sentence. Once the messages have been sent, the php listening page listens for RabbitMQ to deliver a message from the database containing all user information, including the info on their favorite drink as well as what they rated them.

3. Take or Retake the Quiz

a. Built using HTML and CSS libraries like Bootstrap.

b. The page uses the same web form layout from the dashboard.

c. The web form has input fields for users to answer a few questions regarding what they are looking in a drink.

d. Once the form is filled and submitted, the php action page sends the user input through RabbitMQ to the api and a log entry to the database. Once the messages have been sent, the php action page waits in the loading screen until the api sends a reply with the results through RabbitMQ.

e. The results are received in an array and displayed on the page with a favorite button built next to the drinks. The buttons are assigned values in case the user decides to add the drink to their favorites list.

f. If the user decides to add to their favorites list, the button triggers the php action page and the button value is sent through RabbitMQ to the db to add the drink name and ID into the user's info.

4. User Profile

a. Built using HTML and CSS libraries like Bootstrap.

b. The page uses the same web form layout from the dashboard but modified to fit the content being presented.

c. The user page displays the users name and zip code from the session array as well as the users favorite drink information.

d. Here the user may add a rating to their drink from 1 - 5 stars. The dropdown that is used in selecting the drink is populated using JavaScript. The user may select a drink and a rating from 1-5 and submit it. If this is the case, the php action page sends over the user's email, the drink name, and

rating number through RabbitMQ to the database and a log sentence. Then the page refreshes it back to the user page with the updated information.

e. The user can also decide to view the recipes for their favorite drinks. If the user decides to click on the recipe button for their favorite drink, the php action page sends the button value through RabbitMQ to the api and then the listening php page receives the responding message and displays the drinks recipe.

Database:

The database is where we hold most of the data outside of drinks and their descriptions.

The database holds, the users table, registrations table, favorite drinks table and friends table. We use the database as a way to store important data and we have a file called sqlCommands.py that holds most of the important queries that we need to insert or select data from these tables. This part of the project will also be responsible for our resilient Database.

- Database:
- Tables
 - o Users
 - 📖 Holds Users and their information such as name, zip code and email.
 - 📖 Primary Key: Email
 - o Friends
 - 📖 Holds user friends to see who is friends with who.
 - 📖 Primary Key: Entry ID
 - o Logs
 - 📖 Holds logs from API, Front End and RMQ.
 - 📖 Primary Key: Entry ID
 - o Login
 - 📖 Holds the registration information such as username and password.
 - 📖 Primary Key: Email
 - o Favorites
 - 📖 Holds users favorites, used to add and save users favorite cocktails. Has images, names of drinks and their rating for each drink from 1-5.
 - 📖 Primary Key: Entry ID
 - o Password

📖 Holds passwords and their hash, we use it to match hashes for authentication.

📖 Primary Key: Password

- Queries and Functions

- o addFriends.py

- 📖 Script that adds friends into the Friends Table.

- o clogs.py

- 📖 Script that adds logs into the logs table.

- o favorite.py

- 📖 Script that adds favorite drinks to the favorites table.

- o getFavorite.py

- 📖 Script that requests all favorite drinks by a given user.

- o getFriendProfile.py

- 📖 Script that shows friends profile.

- o getFriends.py

- 📖 Script that gives out all the friends a user has.

- o getRate.py

- 📖 Script that adds a rating to the favorite table given a drink.

- o login.py

- 📖 Login script, enables users to log in or get rejected based on credentials.

- o register.py

- 📖 Enables people to register for new users.

- o runAll.py

- 📖 Runs all these scripts in one go.

- o sqlCommands.py

- 📖 Holds all the SQL commands needed by these scripts in order to run

- o userListerner.py

- 📖 Listens for user creation and adds to the User Table.

- Database Resilience

- o Database resilience or a failover is separated into two parts. One is the listener residing in the backup database in this case we used our RMQ VM machines as our backup DB.

- The second part is the DB Update where the main DB continuously dumps the SQL tables and puts them in a .SQL file in this case we call it uLogDB.sql. In our bin folder, we have

a script called updateDB.sh which is run every 10 seconds that dumps the current table and we use SCP in order to send that file over to the backup DB.

o Once our main DB VM fails, we have a listener in our backup DB that continuously listens/pings the main DB and once a ping fails it triggers a series of events.

Once the ping fails, first we use a mysql command(mysql logDB < ulogDB.sql) in order to override our existing DB and update the backup DB. Once that is done the script then runs

runAll.py becoming our backup DB.

- Overall Instructions
- In Main DB
- Run runAll.py
- Run this command in bin folder run `sudo watch -n10 -x bash updateDB`
- In BackUp DB
- Run Based on Which Tier You are in pingDev.py or pingStaging.py or pingProd.py

DMZ:

The API's being used are TheCocktailDB and Yelp. TheCocktailDB provides various attributes about cocktails which includes ID, name, picture, Ingredients, How to guide on making it, whether it's alcoholic or not, type of glass served, etc. I currently have a python file with all the functions needed, the DMZ will be listening to which queue the request comes in and based on that I will run a certain function.

Deployment:

The source code, when ready, will be pushed using a bash script created to push and zip all files to version control in a dedicated folder (DB,API,FE). Once you want to push that code to staging there is a bash script within Version Control that will run and push and unzip files into dedicated folder. Once you want to push that code to production there is a bash script within Version Control that will run and push and unzip files into dedicated folder. This script can rollover to another version number based on the number entered.

