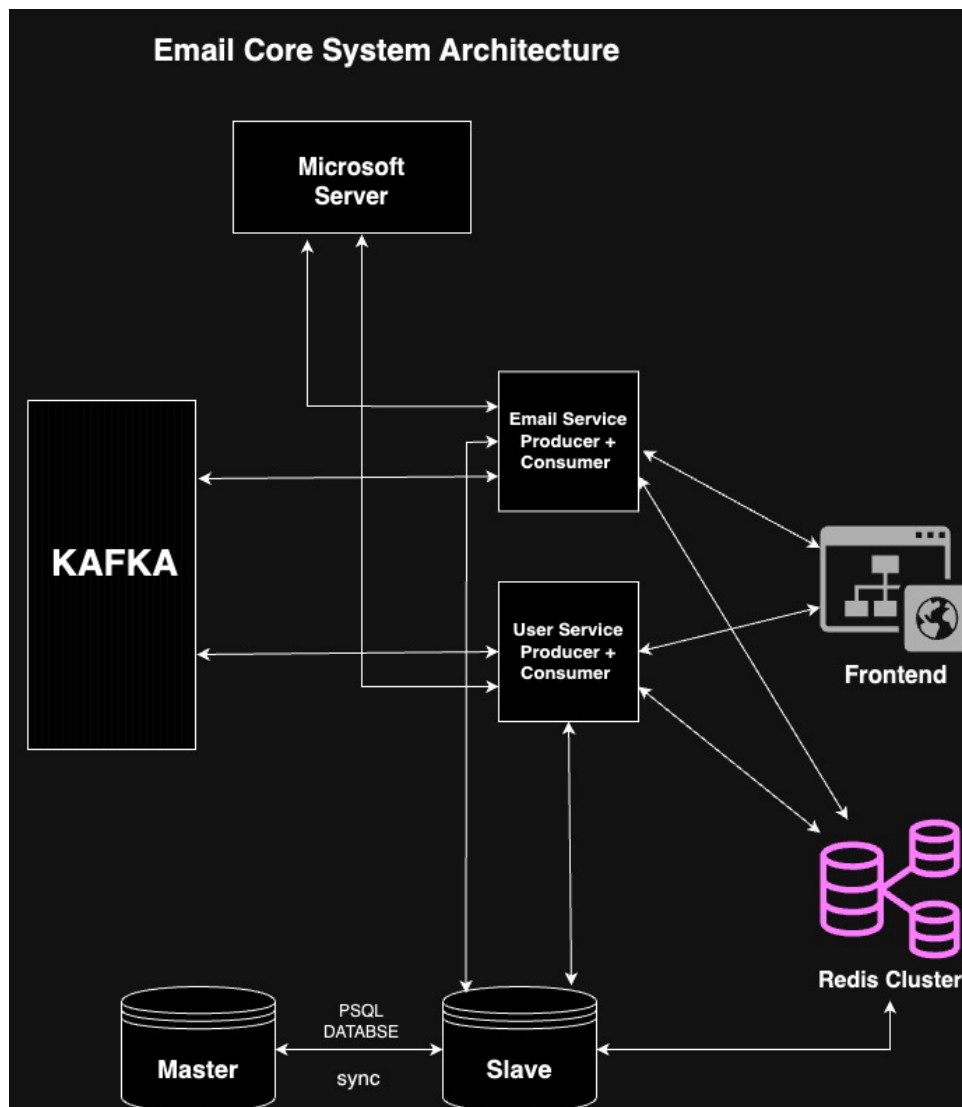


Email Core System Documentation

Architecture Diagram:



Database Schema:

Email Core System Schema

Users		Email	
id	(primary Key)	id	(primary Key)
user_name	varchar	email_id	varchar
password_hash	varchar	userId	int
email	varchar	user_name	varchar
created_at	timestamp	sender	varchar
last_login	timestamp	recipient	varchar
oAuth_email_token	varchar	subject	varchar
oAuth_email_refresh_token	varchar	body	text
oAuth_expires_in	int	importance	varchar
oAuthExtExpiresIn	int	received_date_time	timestamp
token_secret_key	varchar	time_stamp	timestamp
email_connected	boolean	read	boolean
		hasAttachments	boolean

Technologies:

Node Js, Typescript, Nest Js, React Js, Kafka, PostgreSQL, Docker, Azure

Project Overview:

This core system is designed to serve as an email client, offering essential functionalities. It facilitates connectivity with user email accounts, with a primary focus on Outlook, and efficiently manages email data.

Architecture Overview:

- I've opted for Node.js as the runtime environment, leveraging TypeScript as the programming language within the NestJS framework. Node.js is chosen for its efficiency in handling I/O operations, making effective use of the event loop mechanism.
- I've implemented the CQRS pattern along with the repository pattern in my project. The reason for choosing CQRS was to separate read and write operations; *ideally, I wanted to utilize a master-slave database setup where read operations would use the slave database and write operations the*

master. However, due to time constraints, I ended up solely implementing the CQRS pattern with the repository pattern handling all database logic.

Additionally, NestJS inherently follows the singleton pattern, which was also leveraged in my project

- I've developed a custom JWT mechanism for authentication in my application. Additionally, for Outlook verification, I've integrated OAuth Outlook authentication, allowing me to obtain access and refresh tokens. These tokens are then used to fetch emails from the Microsoft server.
- I've implemented a consumer-producer model along with Kafka to streamline processes. By scaling up the number of consumers, tasks can be processed in parallel, resulting in faster completion compared to a single consumer setup. Kafka facilitates this efficient processing.
- I've opted for a Redis cluster to handle large volumes of data efficiently. Redis cluster enables me to manage millions of data points effectively. In this application, Redis is utilized for storing user information, access tokens, refresh tokens, and authentication tokens. Additionally, Redis cluster is used to cache database queries for future use
- I've incorporated an event-driven mechanism in my API to expedite processing and ensure faster response times. Throughout the code, whenever events are required, they are leveraged to optimize performance.

Application Workflow with API:

Step # 1 (Sign Up):

Initially, users will register by providing their email address and password.

Step # 2 (Login):

After entering their email and password, users will log in, and upon successful authentication, they will receive an authentication token to navigate the platform.

Step # 3 (Authenticate Email With Outlook) :

Upon clicking the button to link their local account with their Outlook account, the system will authenticate the user's email. Subsequently, the Outlook

authentication page will open for the user to complete the authentication process. After successful authentication, the user will be redirected back to our local website.

Step # 4 (Email Sync Process):

Upon redirection to our page, a sync email call will be initiated, triggering the synchronization of all emails from the user's Outlook account to our local database. Once the synchronization is complete, a confirmation message will be received, and all emails will be displayed on the user's dashboard.

Step # 5 (Mechanism for Real-Time Email Updates and Creation.)

Microsoft provides an API for real-time mechanism where new or updated emails are directed to the API endpoint we've set up.

```
const subscriptionRequest = {
  changeType: status,
  notificationUrl: `${EMAIL_ENVIRONMENT.LOCAL_APP
  resource: EMAIL_ENVIRONMENT.MSURL.EMAIL_RESOURCE,
  expirationDateTime: new Date(Date.now() + 3600000),
  clientState: JSON.stringify(+userId),
};
```

The crucial part lies in configuring the notification URL correctly. Microsoft requires HTTPS URLs, so to achieve this, I utilize ngrok to generate HTTPS URLs. If this setup isn't implemented correctly, the real-time mechanism won't function as expected.

SETUP APPLICATION

GITHUB REPO URL: [git@github.com:tahakhan-dev/core-email-system.git](https://github.com/tahakhan-dev/core-email-system.git)

Clone this repository and ensure you switch to the "master" branch, not the main branch.

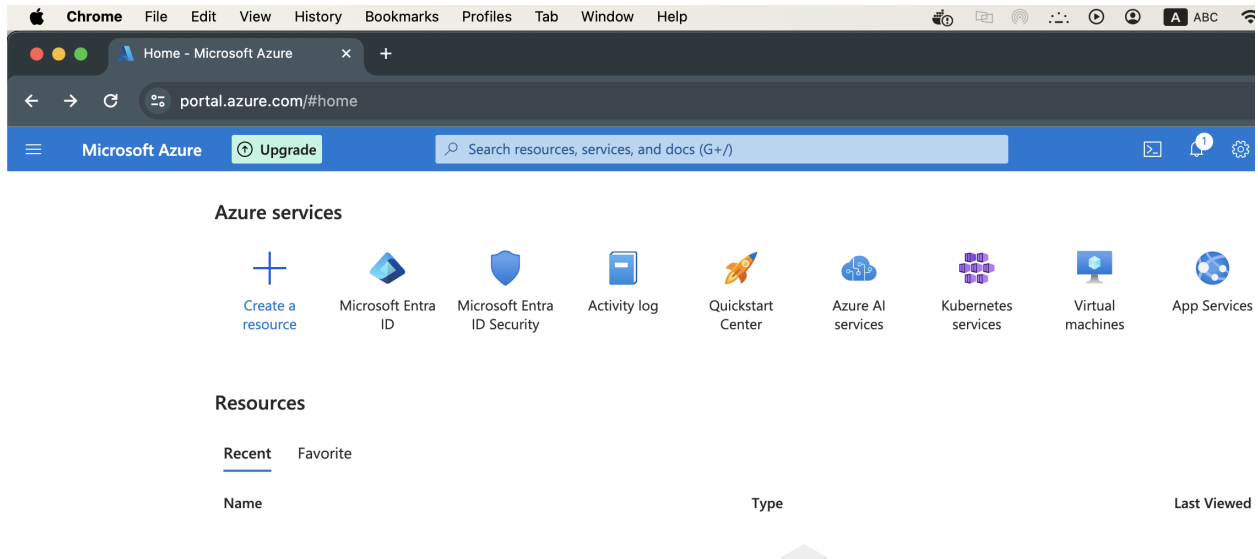
Project Structure :

```
> core-email-consumer
> core-email-producer
> frontend
  docker-compose.kafka.yml
  docker-compose.nest.yml
  docker-compose.pg.yml
  docker-compose.redis.yml
  Dockerfile.pg
  update_pg_hba.sh
```

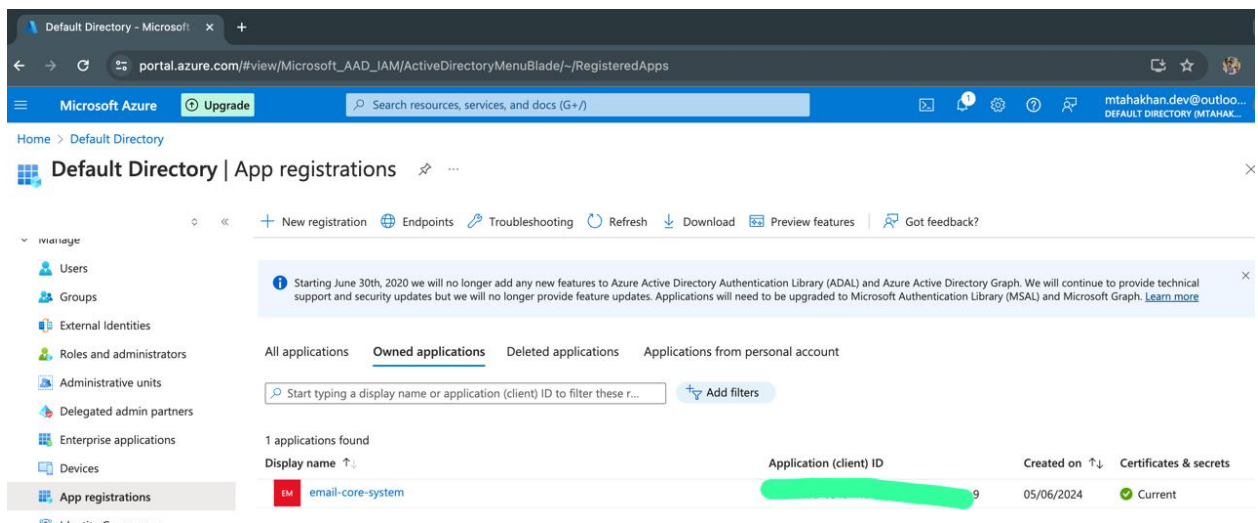
As evident, there are three directories: one for the consumer backend, another for the producer backend, and the last for the frontend. Additionally, multiple docker-compose.yml files are available to facilitate running the application within containers. These compose files handle the installation of dependencies such as PostgreSQL, Kafka, and Redis. With Docker installed, there's no need for manual installation of these dependencies

STEP # 1 (Create Azure account and get credentials)

- To create your Active Directory, navigate to portal.azure.com, sign up or log in, and proceed to the Microsoft Azure portal. From there, access the Microsoft Entra ID section, which is now known as Active Directory.



- Navigate to the application registration section to register your application. Under the authentication settings, add the following redirect URL: <http://localhost:3000/api/user/outlook/redirect>. This URL will be used to redirect the user after successful authentication. Additionally, select the option for multitenant integration instead of single tenant. Ensure to enable "Allow direct integration with the Microsoft account service (login.live.com)", as it's necessary for integrating with Microsoft account SDKs like Xbox or Bing Ads.



Microsoft Azure | Upgrade | Search resources, services, and docs (G+)

Home > Default Directory | App registrations > email-core-system

email-core-system | Authentication

Search | Got feedback?

Overview
Quickstart
Integration assistant
Manage
Branding & properties
Authentication
Certificates & secrets
Token configuration
API permissions
Expose an API
App roles
Owners
Roles and administrators
Manifest
Support + Troubleshooting

Platform configurations

Depending on the platform or device this application is targeting, additional configuration may be required such as redirect URIs, specific authentication settings, or fields specific to the platform.

+ Add a platform

Mobile and desktop applications

Redirect URIs

The URIs we will accept as destinations when returning authentication responses (tokens) after successfully authenticating users. The redirect URI you send in the request to the login server should match one listed here. Also referred to as reply URLs. [Learn more about Redirect URIs and their restrictions](#)

- ☐ <https://login.microsoftonline.com/common/oauth2/nativeclient>
- ☐ https://login.live.com/oauth20_desktop.srf (LiveSDK)
- ☐ [msal46bcb3d-0549-4f8d-9238-a4fb3fd6f8b9://auth](https://login.live.com/oauth20_desktop.srf) (MSAL only)

+ Add URI

Live SDK support ⓘ

Allow direct integration with the Microsoft account service (login.live.com). Yes No

Required for integration with Microsoft account SDKs such as Xbox or Bing Ads

Allow public client flows ⓘ

Enable the following mobile and desktop flows: Yes No

- No keyboard (Device Code Flow) [Learn more](#)

- Navigate to the API permission section to grant the necessary permissions to your API for it to function properly.

email-core-system | API permissions

Search Refresh Got feedback?

Applications are authorized to call APIs when they are granted permissions by users/admins as part of the consent process. The list of configured permissions should include all the permissions the application needs. [Learn more about permissions and consent](#)

+ Add a permission ✓ Grant admin consent for Default Directory

API / Permissions name	Type	Description	Admin consent requ...	Status
▼ Microsoft Graph (14)				
DelegatedAdminRelationship.Re	Delegated	Read Delegated Admin relationships with customers	Yes	✓ Granted for Default Dire... ...
DelegatedAdminRelationship.Re	Application	Read Delegated Admin relationships with customers	Yes	✓ Granted for Default Dire... ...
DelegatedPermissionGrant.Read	Delegated	Read delegated permission grants	Yes	✓ Granted for Default Dire... ...
DelegatedPermissionGrant.Read	Application	Read all delegated permission grants	Yes	✓ Granted for Default Dire... ...
email	Delegated	View users' email address	No	✓ Granted for Default Dire... ...
Mail.Read	Delegated	Read user mail	No	✓ Granted for Default Dire... ...
Mail.Read	Application	Read mail in all mailboxes	Yes	✓ Granted for Default Dire... ...
MailboxSettings.Read	Delegated	Read user mailbox settings	No	✓ Granted for Default Dire... ...
offline_access	Delegated	Maintain access to data you have given it access to	No	✓ Granted for Default Dire... ...
openid	Delegated	Sign users in	No	✓ Granted for Default Dire... ...
profile	Delegated	View users' basic profile	No	✓ Granted for Default Dire... ...
User.Read	Delegated	Sign in and read user profile	No	✓ Granted for Default Dire... ...
User.Read.All	Application	Read all users' full profiles	Yes	✓ Granted for Default Dire... ...
UserAuthenticationMethod.Reac	Delegated	Read user authentication methods.	Yes	✓ Granted for Default Dire... ...

- You will receive the following credentials as depicted in the image.

Delete Endpoints Preview features

Got a second? We would love your feedback on Microsoft identity platform (previously Azure AD for developer). →

^ Essentials

Display name	: email-core-system	Client credentials	: 0 certificate, 1 secret
Application (client) ID	: [REDACTED]	Redirect URIs	: 0 web, 0 spa, 1 public client
Object ID	: [REDACTED]	Application ID URI	: Add an Application ID URI
Directory (tenant) ID	: [REDACTED]	Managed application in I...	: email-core-system
Supported account types	: All Microsoft account users		

Additionally, you will obtain the following API to facilitate email functionality or to acquire your access token.

Endpoints



OAuth 2.0 authorization endpoint (v2)

<https://login.microsoftonline.com/common/oauth2/authorize>

Copy to clipboard

<https://login.microsoftonline.com/common/oauth2/v2.0/authorize>

OAuth 2.0 token endpoint (v2)

<https://login.microsoftonline.com/common/oauth2/v2.0/token>

OAuth 2.0 authorization endpoint (v1)

<https://login.microsoftonline.com/common/oauth2/authorize>

OAuth 2.0 token endpoint (v1)

<https://login.microsoftonline.com/common/oauth2/token>

OpenID Connect metadata document

<https://login.microsoftonline.com/common/v2.0/.well-known/openid-configuration>

Microsoft Graph API endpoint

<https://graph.microsoft.com>

Federation metadata document

<https://login.microsoftonline.com/d91e1996-06ec-4186-a621-94ddb3201a49/federationmetadata/2007-06/federationmetadata.xml>

WS-Federation sign-on endpoint

<https://login.microsoftonline.com/d91e1996-06ec-4186-a621-94ddb3201a49/wsfed>

SAML-P sign-on endpoint

<https://login.microsoftonline.com/d91e1996-06ec-4186-a621-94ddb3201a49/saml2>

SAML-P sign-out endpoint

<https://login.microsoftonline.com/d91e1996-06ec-4186-a621-94ddb3201a49/saml2>

Step # 2: (NGROK or Use Domain);

To establish a subscription for real-time functionality, you require an HTTPS URL. Therefore, you'll need to host your API or purchase a domain to enable HTTPS. As a temporary workaround, we utilize ngrok to address this issue.

```
const subscriptionRequest = {
  changeType: status,
  notificationUrl: `${EMAIL_ENVIRONMENT.LOCAL_APP
  resource: EMAIL_ENVIRONMENT.MSURL.EMAIL_RESOURCE
  expirationDateTime: new Date(Date.now() + 3600000)
```

```
        clientState: JSON.stringify(+userId),
    };
};
```

Visit the ngrok website at <https://ngrok.com/> and create an account. Once signed up, navigate to the token page at <https://dashboard.ngrok.com/get-started/your-auth-token> to obtain your authentication token. After retrieving your token, if you're using Linux or macOS, execute the following command:

```
ngrok config add-auth-token $YOUR_AUTH_TOKEN
```

Your token will remain secure. Subsequently, run the following command to host your port.

```
ngrok http <port_number>
```

replace `<port_number>` with the port you want to expose. This command will create a secure tunnel to your localhost server, allowing external access via a unique ngrok URL.

To run ngrok in the background, you need to install pm2 and execute the ngrok URL as a background process.

```
pm2 start "ngrok http 3001" --name "tunnel"
```

```
tahakhan@Muhammads-MacBook-Pro core-email-consumer % pm2 list
```

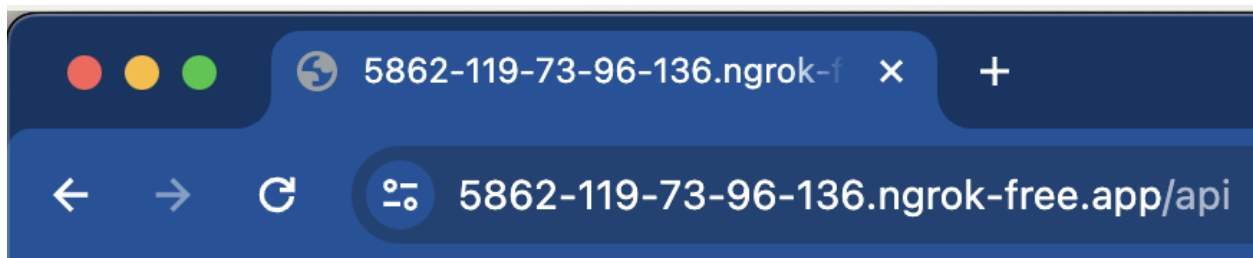
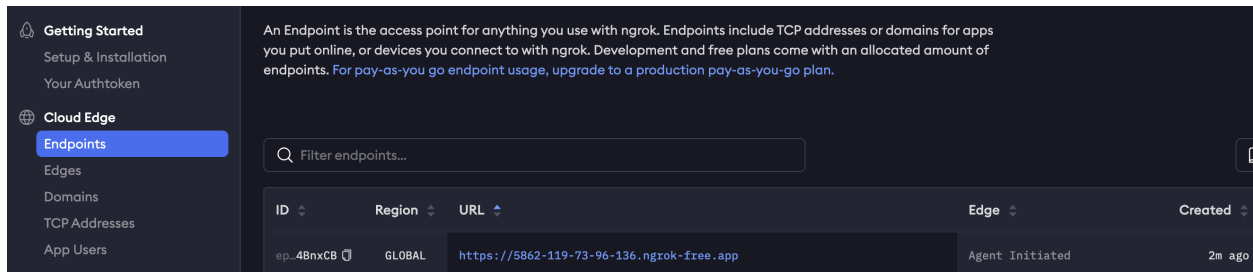
id	name	namespace	version	mode	pid	uptime	⌵	status	cpu	mem	user	watching
0	tunnel	default	N/A	fork	36450	0s	0	online	0%	844.0kb	tahakhan	disabled

```
[PM2][WARN] Current process list is not synchronized with saved list. App tunnel-http 3001 differs. Type 'pm2 save' to synchronize.
tahakhan@Muhammads-MacBook-Pro core-email-consumer % pm2 start "ngrok http 3001" --name "tunnel"

[PM2] Starting /bin/bash in fork_mode (1 instance)
[PM2] Done.
```

Navigate to <https://dashboard.ngrok.com/cloud-edge/endpoints> to find the endpoint URL. This URL is tunneling your port, enabling you to utilize it as the

notification URL in Microsoft for real-time API integration.



Hello World!

Step # 3 (Clone project):

```
git clone https://github.com/tahakhan-dev/core-email-system.git
```

Step # 4 (Install Docker):

Here's an article detailing the installation process for Docker and Docker Compose.

<https://medium.com/@tomerklein/step-by-step-tutorial-installing-docker-and-docker-compose-on-ubuntu-a98a1b7aaed0>

Step # 5 (Run multiple docker-compose.yml file):

To execute the multiple Docker files, you'll require the following command. Remember, refrain from running all docker-compose.yml files simultaneously.

Instead, execute them step by step, following the provided instructions.

Step # 6 (Run docker-compose.pg.yml)

This docker-compose.pg.yml file will set up a PostgreSQL container and automatically create the necessary databases. To automate this process, I've implemented a shell script.

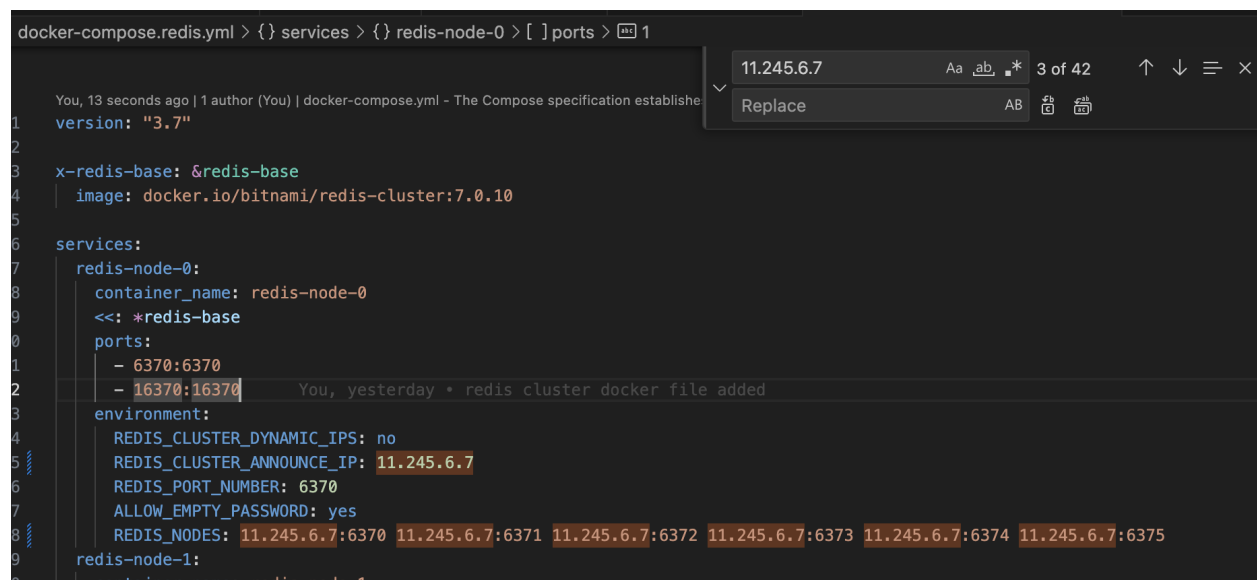
```
docker-compose -f docker-compose.pg.yml up -d
```

Step # 7 (Run docker-compose.redis.yml)

For MacBook users, execute the following command to obtain your IP address:

```
echo "$ (route get uninterrupted.tech | grep interface | sed -  
e 's/.*: //' | xargs ipconfig getifaddr)"
```

Now, substitute this IP address in the docker-compose.redis.yml file.



The screenshot shows a code editor with the `docker-compose.redis.yml` file open. A search and replace dialog is visible, showing the IP address `11.245.6.7` being replaced. The file content includes the following sections:

```
version: "3.7"

x-redis-base: &redis-base
  image: docker.io/bitnami/redis-cluster:7.0.10

services:
  redis-node-0:
    container_name: redis-node-0
    <<: *redis-base
    ports:
      - 6370:6370
      - 16370:16370
    environment:
      REDIS_CLUSTER_DYNAMIC_IPS: no
      REDIS_CLUSTER_ANNOUNCE_IP: 11.245.6.7
      REDIS_PORT_NUMBER: 6370
      ALLOW_EMPTY_PASSWORD: yes
      REDIS_NODES: 11.245.6.7:6370 11.245.6.7:6371 11.245.6.7:6372 11.245.6.7:6373 11.245.6.7:6374 11.245.6.7:6375
  redis-node-1:
    container_name: redis-node-1
```

Now, update the IP address with your local IP to enable the cluster to run properly.

For Linux users, simply use "localhost" or "127.0.0.1" without the need to add your local IP. This step is only necessary for macOS users and run the following below

command.

```
docker-compose -f docker-compose.redis.yml up -d
```

After executing the following command, perform the following steps to verify whether the cluster is running correctly or not.

```
redis-cli -c -h 127.0.0.1 -p 6375
127.0.0.1:6375> CLUSTER NODES
275c337ffe6073071088bb0706a2536608c03fe9 192.168.1.184:6372@16372
0e779e30ce3879ee37b470b299a97d41c0e16779 192.168.1.184:6370@16370
3e7dc06681938425f71a39a2a7d189b336132e4a 192.168.1.184:6374@16374
841b05c2890454b345da8380ee63fa41c03fae6b 192.168.1.184:6371@16371
44444520eb2bbc346ab4dba9b1c495973399a227 192.168.1.184:6375@16375
8f14d7d1faac0e131d315ee34c0c7fc3836716c9 192.168.1.184:6373@16373

127.0.0.1:6375> SET key "value"
-> Redirected to slot [12539] located at 192.168.1.184:6372
OK
```

STEP # 8 (Run docker-compose.kafka.yml)

```
docker-compose -f docker-compose.kafka.yml up -d
```

Once your Kafka container is up, access the Kafka UI by entering <http://localhost:8080/ui/clusters/local/all-topics> into your browser. Within the UI, **create the specified topic with three partitions. Remember, creating a topic with three partitions is essential.**

Create the specified topics required to run the application.

- email_ack (3 partition)
- notify_user_mutate_email (3 partition)
- sync_email (3 partition)
- create_email_subscription_cron (3 partition)
- create_email_subscripton (3 partition)
- update_email_subscripton (3 partition)

STEP # 9 (Run docker-compose.nest.yml)

Execute the provided command to run the application, including your consumer and producer within containers. Ensure that your environment file matches the specifications outlined in my provided env file. Failure to do so may result in the containers stopping or throwing errors. Monitor the container logs to confirm that the application is running successfully.

core-email-consumer (env)

```
PORT=3001
GLOABAL_API_PREFIX=api/
APP_VERSION=1
NODE_ENV=development

#----- Database Credential-----
DB_HOST=host.docker.internal
DB_PORT=5433
DB_USER=tahakhan
DB_PASSWORD=
DB_DATABASE=email
DB_TYPE=postgres
ENABLE_AUTOMATIC_CREATION=true
AUTO_LOAD_ENTITIES=true

#----- ACTIVITY KEY -----

ACTIVITY_CASE=activity_case

# ----- Redis Credentials-----
CACHE_TYPE=ioredis/cluster
CACHE_CLUSTER_NODE1_HOST=host.docker.internal
CACHE_CLUSTER_NODE1_PORT=6370

CACHE_CLUSTER_NODE2_HOST=host.docker.internal
CACHE_CLUSTER_NODE2_PORT=6371

CACHE_CLUSTER_NODE3_HOST=host.docker.internal
CACHE_CLUSTER_NODE3_PORT=6373

CACHE_TTL=345600
CACHE_PASSWORD= ''
```

```

CACHE_CLOSE_CLIENT=true
CACHE_READY_LOG=true
CACHE_ERROR_LOG=true

CACHE_ENABLE_AUTO_PIPELINING=false
CACHE_ENABLE_OFFLINE_QUEUE=true
CACHE_READY_CHECK=true

CACHE_CLUSTER_SCALE_READ=slave

#----- Microsoft Credentials -----

CLIENT_ID=you microsoft azure client id
OBJECT_ID=you microsoft azure object id
TENANT_ID=you microsoft azure tenant id
CLIENT_SECRET=you microsoft azure client secret
AUTHORIZATION_URL=https://login.microsoftonline.com/common/oauth2/authorize
REDIRECT_URI=http://localhost:3000/api/user/outlook/redirect
TOKEN_URL=https://login.microsoftonline.com/common/oauth2/v2.0/tokens
CALLBACK_URL=http://localhost:3000/api/user/outlook/redirect
SCOPE=offline_access User.Read Mail.Read openid email profile https://outlook.officeapps.in/
FETCH_EMAIL_API_URL=https://graph.microsoft.com/v1.0/me/messages
MS_EMAIL_SUBSCRIPTION_URL=https://graph.microsoft.com/v1.0/subscriptions
MS_GET_EMAIL_BY_ID=https://graph.microsoft.com/v1.0/me/messages
MS_EMAIL_RESOURCE=me/mailFolders/inbox/messages

#-----Throttler Keys -----

THROTTLER_USER_TTL=600
THROTTLER_USER_LIMIT=1000

THROTTLER_TTL=600
THROTTLER_LIMIT=1000

#----- Kafka Credentials -----

```



```

KAFKA_NAME=ACK_KAFKA
KAFKA_CLIENT_ID=consumer_email
KAFKA_GROUP_ID=my_kafka_group
KAFKA_BROKER_IP=<your local ip>:39092
KAFKA_TOPIC_SYNC_EMAIL=sync_email
KAFKA_SYNC_EMAIL_ACK_TOPIC=email_ack
KAFKA_EMAIL_SUBSCRIPTION_TOPIC=create_email_subscription_cron
KAFKA_CREATE_EMAIL_SUBSCRIPTION_TOPIC=create_email_subscripton
KAFKA_UPDATE_EMAIL_SUBSCRIPTION_TOPIC=update_email_subscripton
KAFKA_NOTIFY_MUTATION_EMAIL_TOPIC=notify_user_mutate_email

# ===== LOACAL APP URL =====
NGROK_URL=https://2a74-119-73-96-136.ngrok-free.app # your doma:
REFRESH_TOKEN_URL=http://localhost:3000/api/user/outlook/refresl

```

To run this application, you need to make the following changes:

- Replace the Microsoft credentials with your own.
- Update the Kafka broker IP with your local IP address.

core-email-producer (env):

```

PORT=3000
GLOABAL_API_PREFIX=api/
APP_VERSION=1
NODE_ENV=development

#----- Database Credential-----
DB_HOST=host.docker.internal
DB_PORT=5433
DB_USER=tahakhan
DB_PASSWORD=
DB_DATABASE=email

```

```
DB_TYPE=postgres
ENABLE_AUTOMATIC_CREATION=true
AUTO_LOAD_ENTITIES=true

#----- ACTIVITY KEY -----

ACTIVITY_CASE=activity_case

# ----- Redis Credentials-----
CACHE_TYPE=ioredis/cluster
CACHE_CLUSTER_NODE1_HOST=host.docker.internal
CACHE_CLUSTER_NODE1_PORT=6370

CACHE_CLUSTER_NODE2_HOST=host.docker.internal
CACHE_CLUSTER_NODE2_PORT=6371

CACHE_CLUSTER_NODE3_HOST=host.docker.internal
CACHE_CLUSTER_NODE3_PORT=6373

CACHE_TTL=345600
CACHE_PASSWORD=''

CACHE_CLOSE_CLIENT=true
CACHE_READY_LOG=true
CACHE_ERROR_LOG=true

CACHE_ENABLE_AUTO_PIPELINING=false
CACHE_ENABLE_OFFLINE_QUEUE=true
CACHE_READY_CHECK=true

CACHE_CLUSTER_SCALE_READ=slave

#----- Microsoft Credentials -----

CLIENT_ID=you microsoft azure client id
```

```
OBJECT_ID=you microsoft azure object id
TENANT_ID=you microsoft azure tenant id
CLIENT_SECRET=you microsoft azure client secret
AUTHORIZATION_URL=https://login.microsoftonline.com/common/oauth2/authorize
REDIRECT_URI=http://localhost:3000/api/user/outlook/redirect
TOKEN_URL=https://login.microsoftonline.com/common/oauth2/v2.0/tokens
CALLBACK_URL=http://localhost:3000/api/user/outlook/redirect
SCOPE=offline_access User.Read Mail.Read openid email profile https://outlook.officeapps.net/owa/
FETCH_EMAIL_API_URL=https://graph.microsoft.com/v1.0/me/messages
```

```
# -----Throttler Keys -----
```

```
THROTTLER_USER_TTL=600
THROTTLER_USER_LIMIT=1000
```

```
THROTTLER_TTL=600
THROTTLER_LIMIT=1000
```

```
# ----- Kafka Credentials -----
```

```
KAFKA_CLIENT_ID=my_email_sync
KAFKA_GROUP_ID=my_kafka_group
KAFKA_CONSUMER_1_GROUP_ID=email_ack_group
KAFKA_CONSUMER_2_GROUP_ID=email_ack_group_2
KAFKA_TOPIC_CONSUMER_1=email_ack
KAFKA_TOPIC_CONSUMER_2=notify_user_mutate_email
KAFKA_TOPIC_SYNC_EMAIL=sync_email
KAFKA_BROKER_IP=<YOUR_LOCAL_IP_ADDRESS>:39092
```

```
docker-compose -f docker-compose.nest.yml up -d docker-compose
```

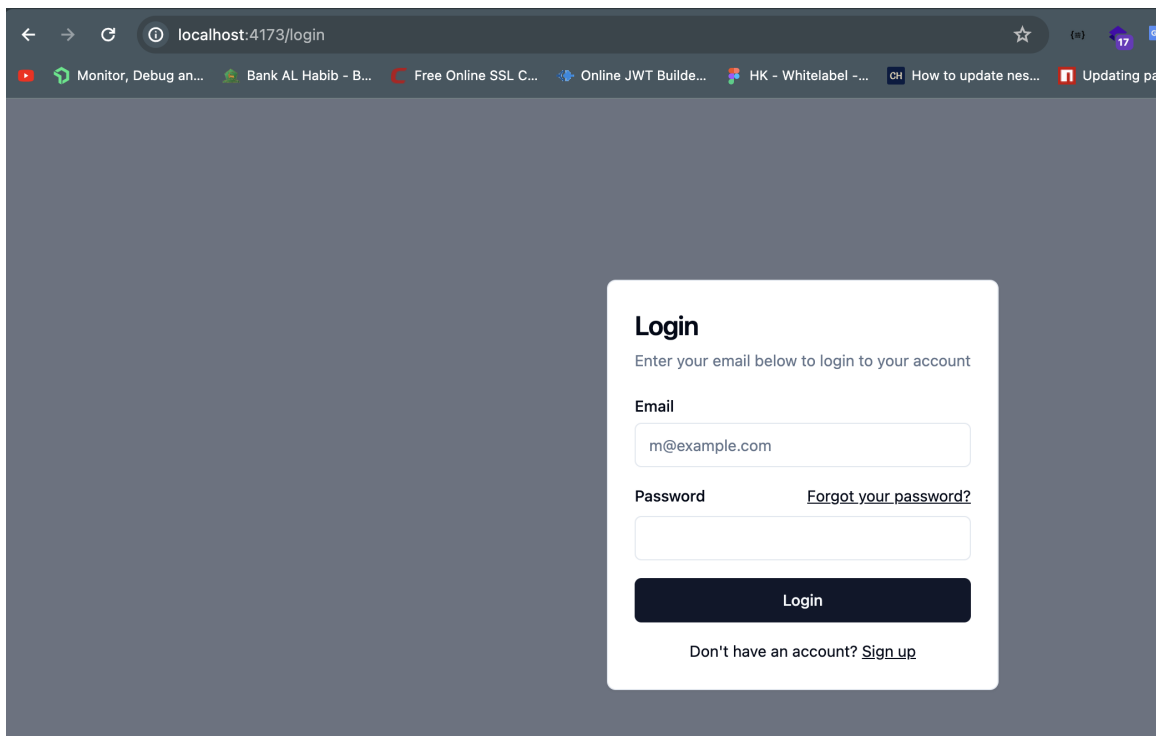
After executing the command above, your application will be running within a container. You can monitor the application's logs accordingly.

STEP # 10 (Run docker-compose.frontend.yml)

To launch the frontend project in a Docker container, use this command:

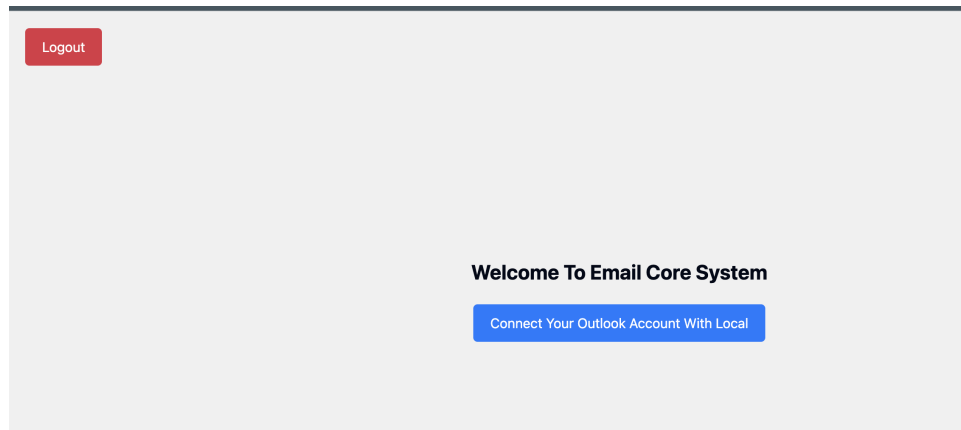
```
docker compose -f docker-compose.frontend.yml up --build -d
```

After executing the command, access the project by visiting this URL:
<http://localhost:4173>

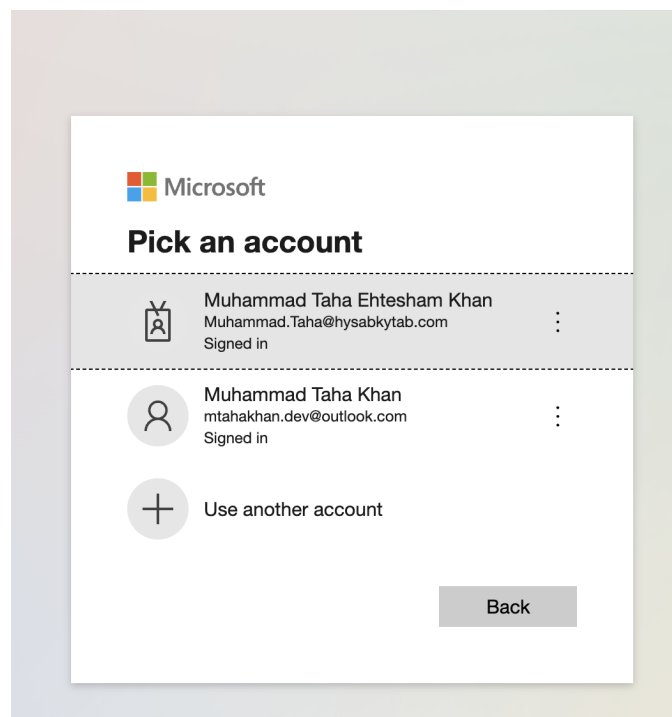


STEP # 11 Application Workflow via User Interface

If you haven't registered previously, click the "Sign Up" button and complete the registration process. After registering, enter your credentials to log in to the application. Upon logging in, you will see the following screen:



If your local account isn't linked to a Microsoft account, you will see a button to initiate the connection. Clicking this button will redirect you to the Microsoft login page, where you can link your Microsoft account to this local one.



After successfully authenticating with your Microsoft account, you will be redirected to a success page that confirms your account has been successfully linked with your Microsoft account.

Successfully Connected

Congratulations! You have successfully connected your local account to your Outlook account. This integration allows seamless synchronization and better management of your emails.

[Back to the Email Core System](#)

Click the button to return to the application, and you will notice that your email is beginning to sync.

Welcome To Email Core System

Your account is connected with Microsoft.

Your Email Is Syncing With Local ...

Once your email has synced, you will see the following screen displaying all your synchronized emails.

Welcome To Email Core System								
<div>Logout</div> <div>Your account is connected with Microsoft. Your all emails are synced.</div>								
User Name	Sender	Recipient	Subject	Body	Importance	Has Attachments	Read	Received Date
mtahakhan.dev	Muhammad.Taha@hysabkytab.com	mtahakhan.dev@outlook.com	New testing 123 new testing 123 Taha	New testing 123 new testing 123 Taha	normal	No	No	15/06/2024, 19:14:59
mtahakhan.dev	Muhammad.Taha@hysabkytab.com	mtahakhan.dev@outlook.com	new testing 2314 new testing	new testing 2314 new testing	normal	No	No	15/06/2024, 19:13:05
mtahakhan.dev	Muhammad.Taha@hysabkytab.com	mtahakhan.dev@outlook.com	New testing 1234 testing 234	Testing 314 gtesting 3214	normal	No	No	15/06/2024, 19:10:30
mtahakhan.dev	Muhammad.Taha@hysabkytab.com	mtahakhan.dev@outlook.com	New testing 234 new testing 2134 testing 234	New gtesing 234 stesing 234 in2. Rfdf sd	normal	No	No	15/06/2024, 19:07:39
mtahakhan.dev	Muhammad.Taha@hysabkytab.com	mtahakhan.dev@outlook.com	Testing 123 testing 123 testing 123	New testing 1234 new testing 1324	normal	No	No	15/06/2024, 19:03:38
mtahakhan.dev	Muhammad.Taha@hysabkytab.com	mtahakhan.dev@outlook.com	Sdfsdf testing 1234 testing 3256	Sdfsdf testing 2345 tewing 2345	normal	No	Yes	15/06/2024, 18:57:43
mtahakhan.dev	Muhammad.Taha@hysabkytab.com	mtahakhan.dev@outlook.com	New testing 134 I am testing 235	fsdfsdfsdf	normal	No	Yes	15/06/2024, 18:55:27
mtahakhan.dev	Muhammad.Taha@hysabkytab.com	mtahakhan.dev@outlook.com	Teething 3423423 testing 234	Vvjksldfjlskfjkl2 4l23 Dsjfksljfksjdf Sjdfklsjfs Evedfokidf Sjdfklsjdf	normal	No	Yes	15/06/2024, 18:48:24

This describes the process for setting up your email core system application. By following these steps, you will successfully configure the application.