**PYTHON MICROSERVICES: BREAKING MOLOLITH INTO MICROSERVICES.**

[**https://www.udemy.com/course/python-microservices/**](https://www.udemy.com/course/python-microservices/)

**email: chegejohn159@gmail.com**

**Password: YpfKDo#d8qN&**

**Objectives**

* Event Driven Architecture with Apache Kafka
* Multiple Database Connections
* Internal Http Requests
* Run Docker with multiple networks
* Import Data from multiple databases
* Use Kubernetes
* Deploy to Google Cloud Platform

1. **Introduction**

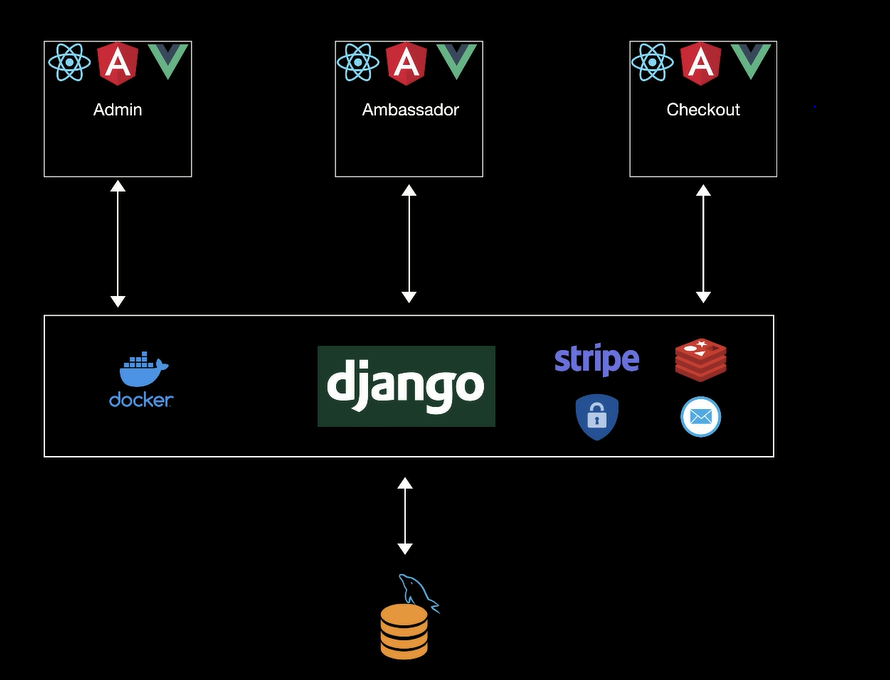


Figure 1: Monolith

The 3 front end apps (Admin, ambassador, checkout) communicating with the Django app.

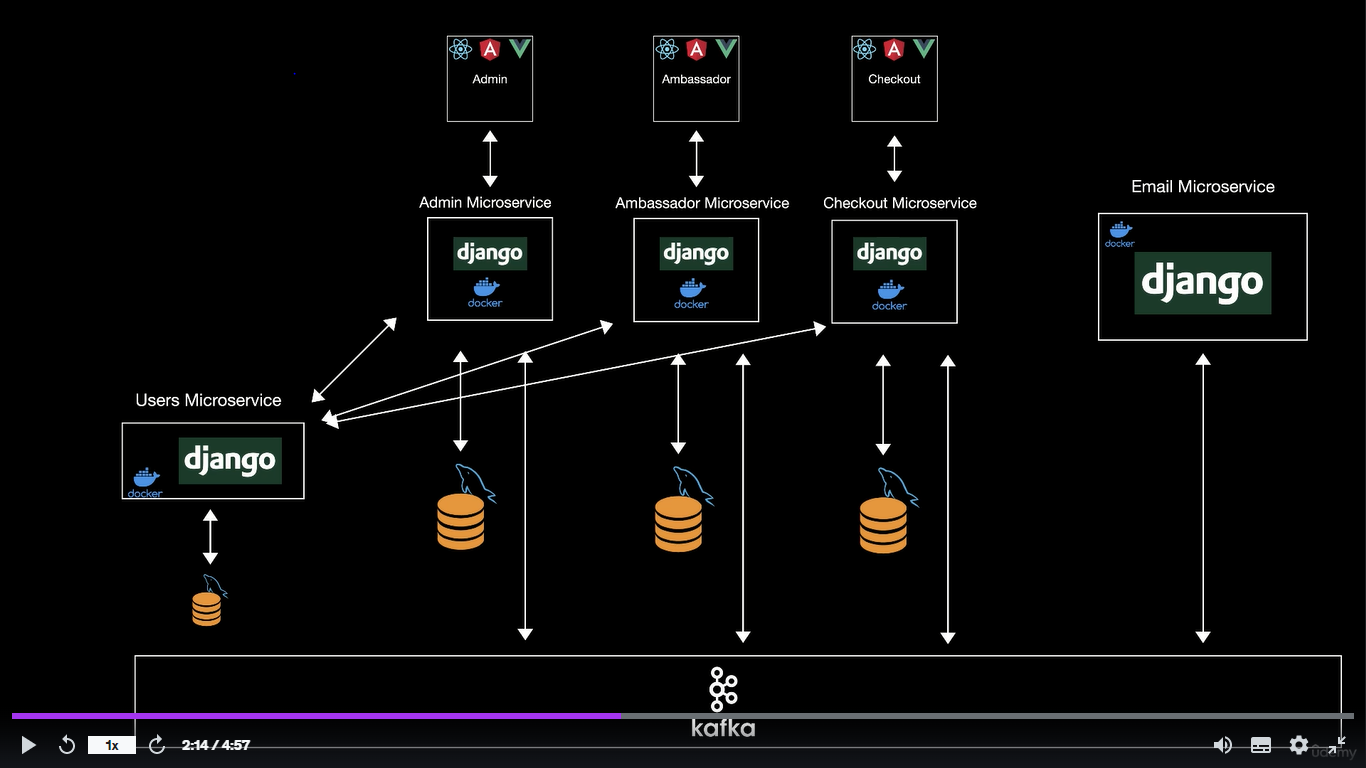
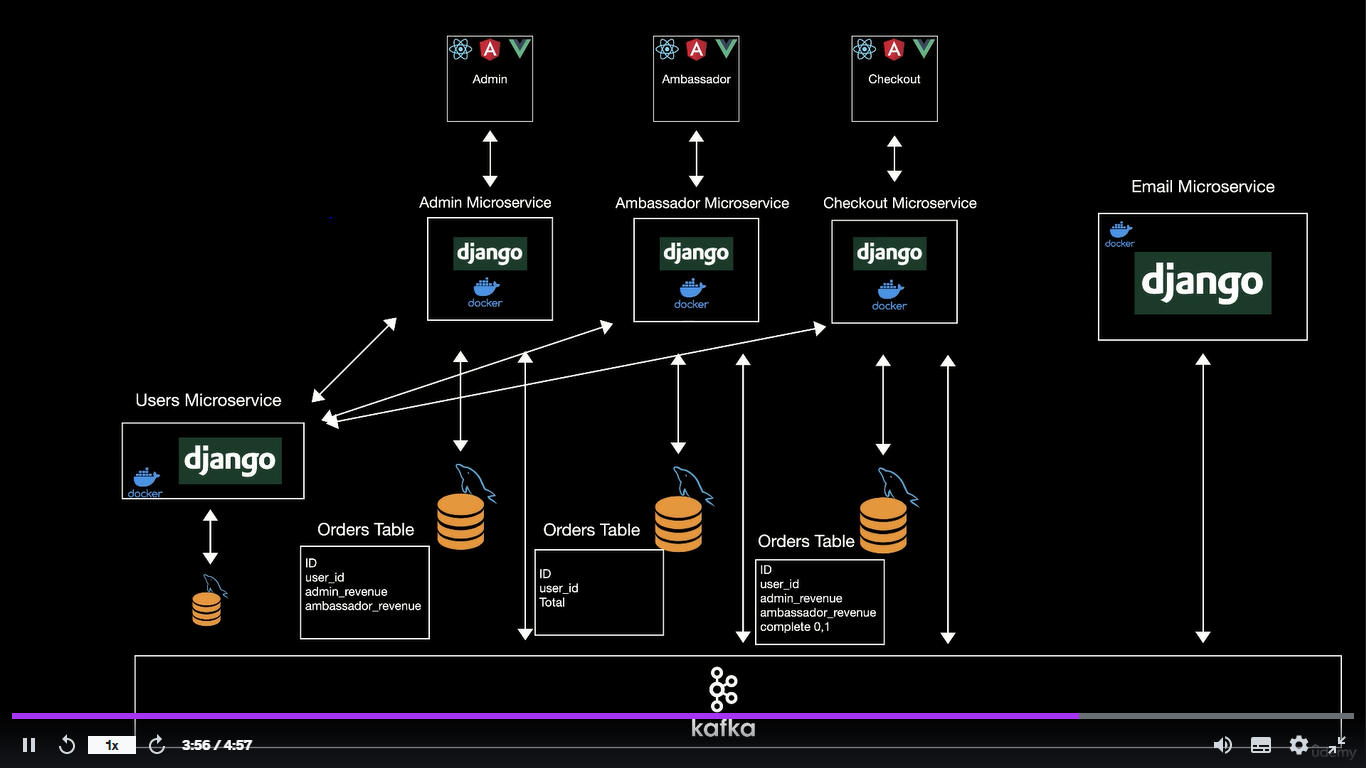


Figure 2: Microservices after being broken down.

Each front end app will consume its own microservice and will all communicating the users app which will store all the information regarding each user.



1. **Difference between a monlolith and a microservice.**

Monolith:

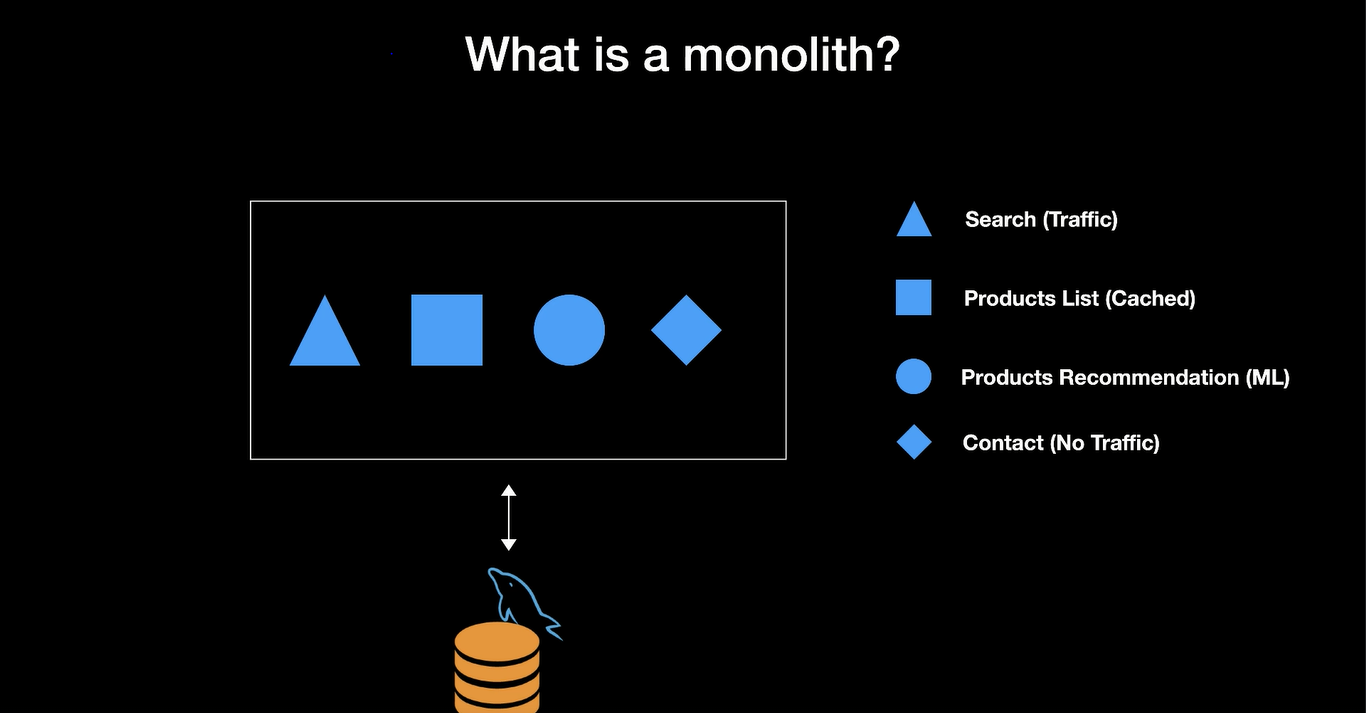


Figure 3: Monolith App

A monolith app is an entire complex app that has all the service and functionality. i.e. In one app we have Search, products, products recommendation and contact.

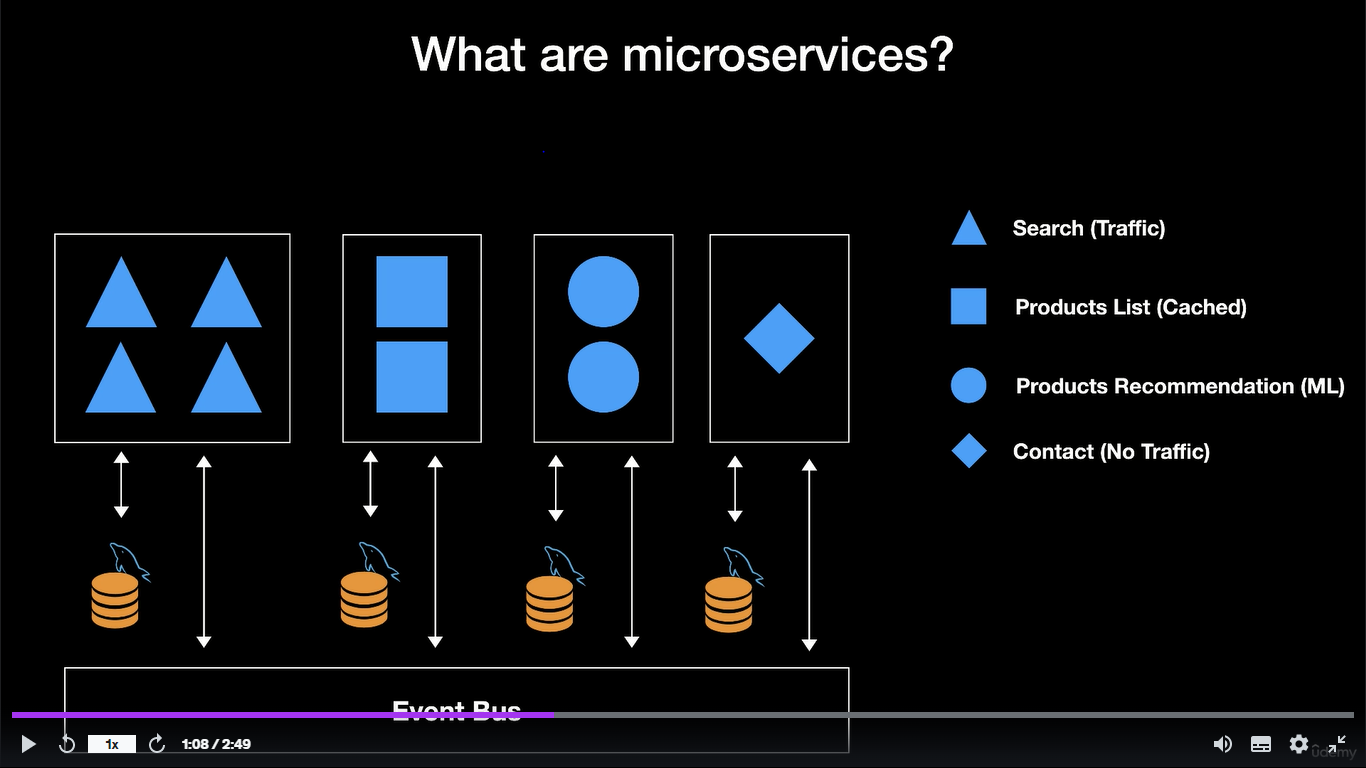


Figure 4: Microservices App

A microservice are smaller apps that communicate with each other using internal API calls or via an event bus( i.e. Kafka event bus) The entire app is broken down into smaller services e.g. Search Microservice, product microservice, product recommendation microservice and contact microservice.

Benefits of microservices:

* We can define the number of instance running on each event. i.e search ms has 4 instances running while contact ms has only one event running.
* Each microservice can have a different programming language.

**3. Set up**

**Git clone** [**https://github.com/antoniopapa/django-ambassador**](https://github.com/antoniopapa/django-ambassador)

**Git clone** [**https://github.com/antoniopapa/react-ambassador.git**](https://github.com/antoniopapa/react-ambassador.git)

**git clone** [**https://github.com/antoniopapa/angular-ambassador.git**](https://github.com/antoniopapa/angular-ambassador.git)

Change port numbers in :

C:\Users\Administrator\Desktop\PythonMicroservices\react-ambassador\react-admin\src\index.tsx

C:\Users\Administrator\Desktop\PythonMicroservices\vue-ambassador\vue-admin\src\main.ts

Install docker:

<https://docs.docker.com/desktop/install/windows-install/>

Intall WSL <https://learn.microsoft.com/en-gb/windows/wsl/install-manual#step-4---download-the-linux-kernel-update-package>

Linux

username: martin

Psw: M@rtin

Pull all the images and create the files.

CD C:\Users\Administrator\Desktop\PythonMicroservices\django-ambassador>

*docker-compose up*

Navigate to react-ambassador through

cd C:\Users\Administrator\Desktop\PythonMicroservices\react-ambassador>

and install npm in react-admin, react-ambassador, react-checkout

*npm i*

in both the in react-admin, react-ambassador run

*npm start*

in react-checkout run

*npm run dev*

**Populating data:**

To enter the backend container, type the following in cmd while inside Django-ambassador dir

*docker-compose exec backend sh*

Make migrations while inside **“#”**

*python manage.py makemigrations*

*python manage.py migrate*

Populate the tables

*python manage.py populate\_ambassadors*

*python manage.py populate\_products*

*python manage.py populate\_orders*

**3. Sending Message from the terminal the confluent cloud.**

confluent Home Page:[**https://confluent.cloud/environments**](https://confluent.cloud/environments)

[**https://confluent.cloud/environments/env-doxw5y/clusters/lkc-v7qo15/integrations/cli**](https://confluent.cloud/environments/env-doxw5y/clusters/lkc-v7qo15/integrations/cli)

Installing CLI using Gitbash (or linux)

*curl -sL --http1.1 https://cnfl.io/cli | sh -s -- latest*

Log in to your Confluent Cloud organization using the Confluent CLI

*confluent login –save*

 To view the available environments, use the

*confluent environment list*

Select your environment

*confluent environment use env-doxw5y*

To view available clusters, use the

*confluent kafka cluster list*

To set your cluster.

*confluent kafka cluster use lkc-v7qo15*

Use an API key and secret in the CLI

 create a new API key and secret pair using this command

*confluent api-key create --resource lkc-v7qo15*

API Key: 5DJHBO5Y3NEMNT2F

Secret: T15dxOg6P9OL8z5ulytkOMPBS9W/FzTnwz7f7A6o41CEeCrRzcRNtt8TRiolPQJ3

After you have created or added your API key pair, copy the API key and paste it at the end of this command: *confluent api-key use <API\_Key> --resource lkc-v7qo15*

e.g.

*confluent api-key use* 5DJHBO5Y3NEMNT2F *--resource lkc-v7qo15*

~~If you have an existing API key that you'd like to use, add it to the CLI with this command:~~

*~~confluent api-key store --resource lkc-v7qo15~~*

to create a topic named **test-topic** in your cluster:

*confluent kafka topic create test-topic*

verify that your topic has been created:

*confluent kafka topic list*

 To start a console producer, which you can use to manually produce messages to test-topic

*confluent kafka topic produce test-topic*

Then type

*Hello*

Open another Gitbash terminal

To consume all of the messages in **test-topic** and print them to the console, run this command:

*confluent kafka topic consume -b test-topic*

