Building Cloud Native Applications: Hotel Reservation System

Overview

In this tutorial, we will guide you through the process of building a cloud-native hotel reservation system. This system comprises two primary components:

1. Hotel Reservation Service developed using Java Spring Boot, Python, and Ballerina.
2. Hotel Reservation Web Application developed using ReactJS.

Prerequisites

Before beginning, ensure you have the following prerequisites set up:

- A GitHub account. https://github.com/
- Microsoft Visual Studio (VSCode) with the WSO2 Ballerina plugin. Visual Studio Code Ballerina VS Code extension
- Git installed on your workstation.
- A recent version of Google Chrome or Mozilla Firefox.
- Postman and curl (or any HTTP client) installed on your workstation.
- Ballerina latest version. Ballerina Swan Lake Ballerina VS Code extension
- Python 3.x https://www.python.org/downloads/
- Kafka broker (you may use confluent SaaS based broker free trial) https://confluent.cloud/ or if preferred to run Kafka locally, download 2.13-3.6.0 and setup Kafka server locally
    - *Note* if you prefer to run Kafka locally and to connect Choreo hosted service you may use ngrok for proxy configuration*
    - *Download ngrok https://ngrok.com/download*
- Azure communication service (guidance will be given on how to generate keys, may use a trial account). https://azure.microsoft.com/en-us/products/communication-services
- A Choreo account.

Business Scenario

The objective is to construct a reservation system for a luxury hotel that enables users to search for rooms, make reservations, and manage their bookings.

High-Level Steps

1. Develop the HTTP service using Spring Boot (refer to the code repository), Python services for email communication, and Ballerina for Backend for Frontend (BFF) services implementation using GraphQL.
2. Push the code to your GitHub account.
3. Deploy the cloud-native application on Choreo, including both services and the web application.

Detailed Steps

1. Develop the GraphQL Service for Rooms Search API

- Implement Ballerina graphQL that will be utilized for the room search API.

2. Develop Java Spring Boot HTTP Service

Using Java Spring Boot, develop an HTTP service that manages the backend logic for room searches, reservations, and management. This service should include endpoints for:

- Making reservations: Handle POST requests with user data and return a confirmation with a unique reference number.
- Listing reservations: Allow users to retrieve their booking details.
- Updating reservations: Enable modifications to existing bookings.
- Canceling reservations: Allow users to cancel their bookings.

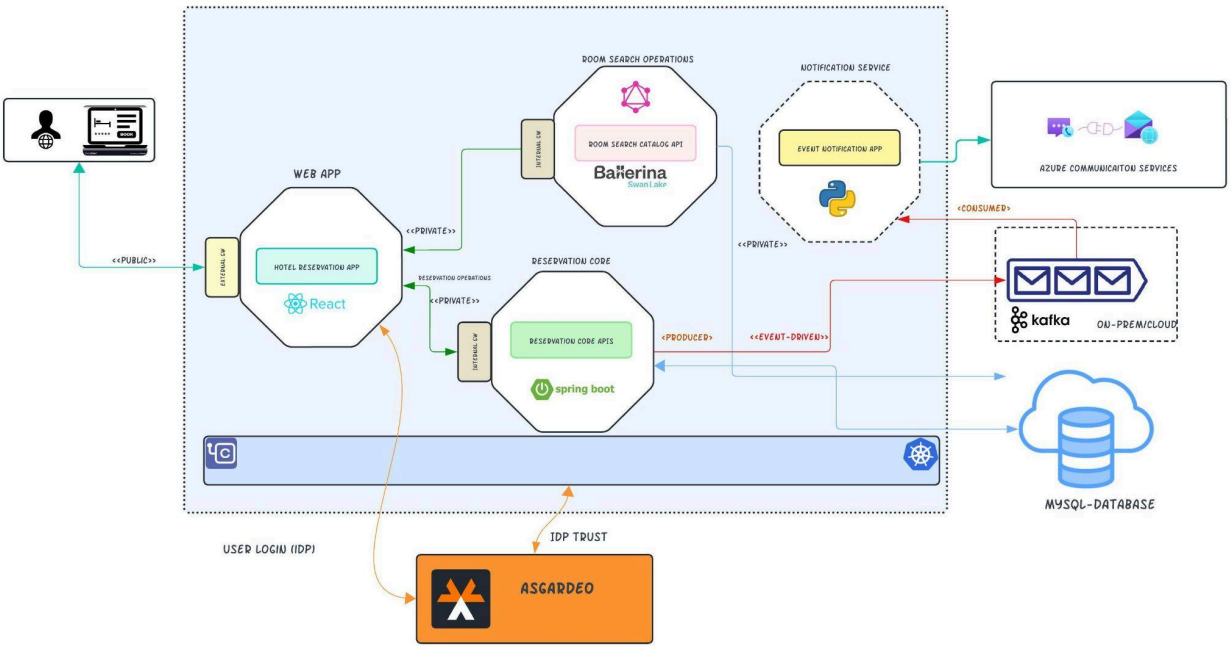
Example of a simple Ballerina service for room reservation:

3. Develop Python HTTP for Email Notification (Event based API)

- The Python service will be designed to send email notifications related to user reservation activities.It'll listen to the reservation update events at the Kafka broker topic. Based on the event. This service will interact with Azure communication services to manage the dispatch of emails.

Project Objective:

Develop a reservation platform for a premium hotel establishment.



Proposed Solution:

Create an interactive web application that facilitates room bookings for hotel guests. The application will be equipped with the following functionalities:

Room Search Feature:

- Guests will have the ability to search for available rooms by entering their desired check-in and check-out dates, and they can refine their search based on the number of occupants.
- The search output will display various room categories, such as single, double, and so on.
- Accompanying each room type in the search results will be a "Reserve" button, streamlining the booking process.

Room Reservation Process:

- For room booking, guests are required to provide their personal details, including their full name, contact number, and email address.

- The reservation form will enable the "Reserve" button once all mandatory fields are completed correctly. Following a successful reservation, the system will generate a unique reference number for the guest to note down.

Reservation Management:

- Guests can view their current reservations after signing into their account.
- Within their reservation list, guests will have the option to either amend the details of their booking or proceed with cancellation.

Reservation Modification:

- Guests are granted the autonomy to alter any aspect of their existing reservation.

Reservation Cancellation:

- Guests retain the right to revoke their reservations at their convenience, with a clear and accessible cancellation feature within the booking system

Project Setup Guidance

Open project locally

- Fork the GitHub Repo - https://github.com/wso2con2024/architecture-tutorial. Important: Make sure you untick the option “Copy the main branch only”.

- Open the hotel-reservation-demo directory using Visual Studio Code. First Click on **File > Open** and then select the hotel-reservation-demo folder and click **Open**.

Deploying the Hotel Reservation App using Choreo

Step 1: Sign Up and Login to Choreo

- Sign Up to Choreo by the https://choreo.dev/ URL
- Once you log in to Choreo for the first time, you will be asked to provide an organization
  handle name. Provide a handle name and click Create.

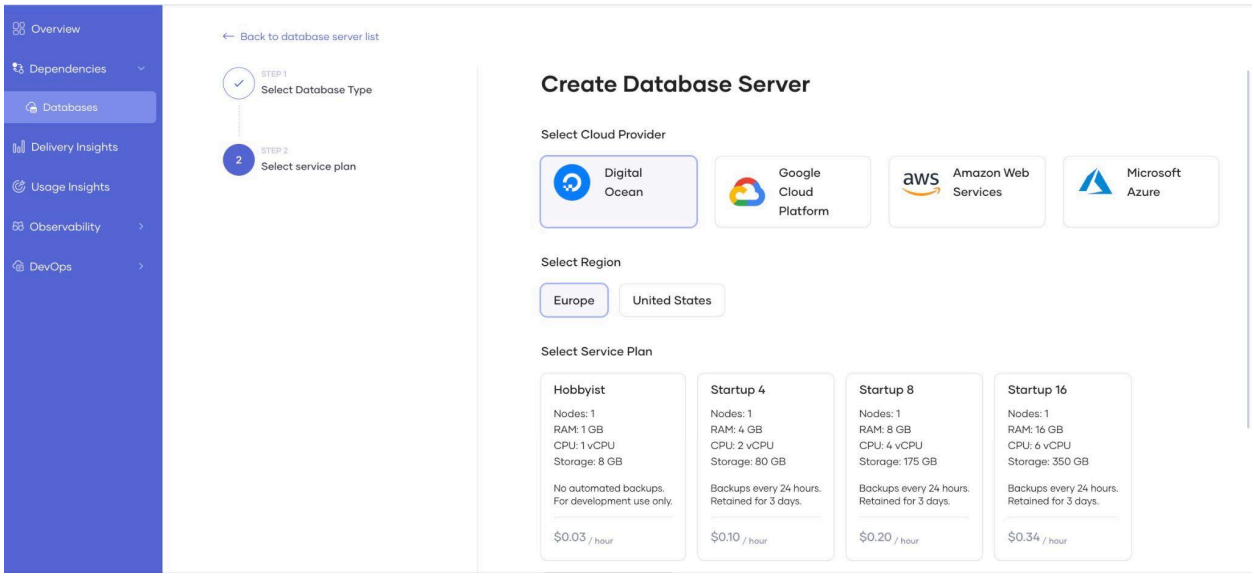
Step 2: Create a new Project

- Click organization card in the top menu
- Click Create Project card
- Add the fields shown in the table

FieldName	Field Value
Name	Luxury Hotel
Project Type	Multi Repository

Step 3: Setup Database

- Click organization
- Navigate to Dependencies tab
- Select MySQL
- Select Digital Ocean or preferred cloud vendor
- Click Create Database



- Once Database create, copy database URL, port, username, password and keep it somewhere
- Using any DB client login to the database.
- Navigate to <<**resources>>** folder.
    - Run schema.sql

- Run data.sql

Step 3: Setup Kafka Broker

Confluent

- You may setup the Kafka broker, this is an independent task, for the tutorial we would recommend you to use Confluent free tire. https://confluent.cloud/ . You may need to obtain following information

  - KAFKA_USERNAME=xx (api key)
  - KAFKA_PASSWORD=xx (api secret)
  - BOOTSTRAP_SERVERS=xx
  - SECURITY_PROTOCOL=SASL_SSL (default)
  - SASL_MECHANISMS=PLAIN (default)
  - SESSION_TIMEOUT_MS=45000(default)
  - TOPIC_NAME=xx (default : notifications)
  - PREFERRED_BROKER="confluent"

Local setup

- Extract ngrok
- Run ./ngrok tcp localhost:9092
  - Mark down the URL
- Open kafka_home/config/servers.properites
  - Replace ngrok address
    advertised.listeners=PLAINTEXT://<<ngrok-ip>>:<<port>>
    - *e.g advertised.listeners=PLAINTEXT://8.tcp.ngrok.io:14784*
- Open Terminal Tab - 1.
  - Go inside kafka_home
  - Execute `sh bin/zookeeper-server-start.sh config/zookeeper.properties`
- Open Terminal Tab - 2
  - Go inside kafka_home
  - Execute `sh bin/kafka-server-start.sh config/server.properties`
- Open Terminal Tab - 3 (Kafka Producer)
  I am using the default Kafka producer client which comes with Kafka distribution to send sample events.
  - Go inside kafka_home

    - Now we are going to create a Kafka topic for our integration flow. Topic name is **sales**
    - Execute `bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor 1 --partitions 1 --topic` **`notifications`**
- Local Broker
    - export BOOTSTRAP_SERVERS="<<ngrok ip:port>>"
    - export SECURITY_PROTOCOL="PLAINTEXT"
    - export SESSION_TIMEOUT_MS="45000"
    - export TOPIC_NAME="notifications"
    - export PREFERRED_BROKER="local-broker"

Step 4: Create and Deploy a Python notification event handler - Event Notification Service

- Create Project e.g " Notification-Services"
- Create a New Component->Event Handler
- Provide Component name/ Description
- Connect to the git repo
- Select **buildpack Python**
- Select **<<notification-event-consumer>>** directory
- Select Python language version
- Select create
- Navigate to Build card
    - Click build latest which trigger building the image from the latest commit
- Once build is successfully completed under deployment provide following environment variable refer section under ref **Appendix [2]** on how to obtain connection string and azure communication sender address
    - AZURE_COMM_SERVICES_CONNECTION_STRING
    - AZURE_COMM_SERVICES_SENDER_ADDRESS
- Setup Kafka configuration under environment variable (consumer service)

  Local Broker
    - export BOOTSTRAP_SERVERS="<<ngrok ip:port>>"
    - export SECURITY_PROTOCOL="PLAINTEXT"
    - export SESSION_TIMEOUT_MS="45000"

    - export TOPIC_NAME="notifications"
    - export PREFERRED_BROKER="local-broker"

  Confluent
    - KAFKA_USERNAME=xx (api key)
    - KAFKA_PASSWORD=xx (api secret)
    - BOOTSTRAP_SERVERS=xx
    - SECURITY_PROTOCOL=SASL_SSL (default)
    - SASL_MECHANISMS=PLAIN (default)
    - SESSION_TIMEOUT_MS=45000(default)
    - TOPIC_NAME=xx (default : notifications)
    - PREFERRED_BROKER="confluent"

- Deploy the service

Step 5: Create and Deploy a SpringBoot Services (reservation core services)

- Create Project e.g " Reservation Core Services"
- Create a New Component->Service
- Provide Component name/ Description
- Connect to the git repo
- Select **buildpack Java**
- Select **<<service-java>>** project
- **Select JDK 17 from language version (***this program compile jdk 17 and above)**
- Click Create
- Navigate to Build card
    - Click build latest which trigger building the image from the latest commit
- Once build is successfully completed under deployment provide following environment variable
    - DB_HOST
    - DB_NAME
    - DB_PASSWORD
    - DB_PORT
    - DB_USERNAME

//Producer

Local Broker

- export BOOTSTRAP_SERVERS="<<ngrok ip:port>>"
- export SECURITY_PROTOCOL="PLAINTEXT"
- export SESSION_TIMEOUT_MS="45000"
- export TOPIC_NAME="notifications"
- export PREFERRED_BROKER="local-broker"

Confluent

- KAFKA_USERNAME=xx (api key)
- KAFKA_PASSWORD=xx (api secret)
- BOOTSTRAP_SERVERS=xx
- SECURITY_PROTOCOL=SASL_SSL (default)
- SASL_MECHANISMS=PLAIN (default)
- SESSION_TIMEOUT_MS=45000(default)
- TOPIC_NAME=xx (default : notifications)
- PREFERRED_BROKER="confluent"

- Click and deploy service

Step 6: Create and Deploy a Ballerina Services (graphQL based room search service)

- Create Project e.g " Room Search Operations"
- Create a New Component->Service
- Provide Component name/ Description
- Connect to the git repo
- Select **buildpack Ballerina**
- Select <<**service-graphql**>> directory
- Select Ballerina language version
- Select create
- Navigate to Build card
  - Click build latest which trigger building the image from the latest commit
- Once build is successfully completed under deployment provide following environment variable as in follows (obtained from step-2)
  - DB_HOST
  - DB_NAME
  - DB_PASSWORD
  - DB_PORT

    - DB_USERNAME
- Deploy API

Step 7 - Deploy the Hotel Reservation Web Application

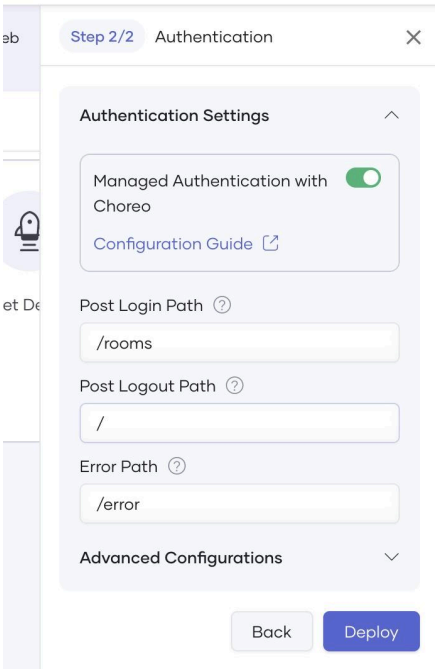
- Create Project e.g " Hotel Reservation Front End"
- Create a New Component->Web Application
- Provide Component name/ Description
- Connect to the git repo
- Select <<**webapp**>> directory
- Select React as the buildpack and set the following parameters

Field	Field Value
BuildPack	Ract
Project Director	/webapp
Build Command	/npm run build
Build Path	/build
Node Version	20.11.0

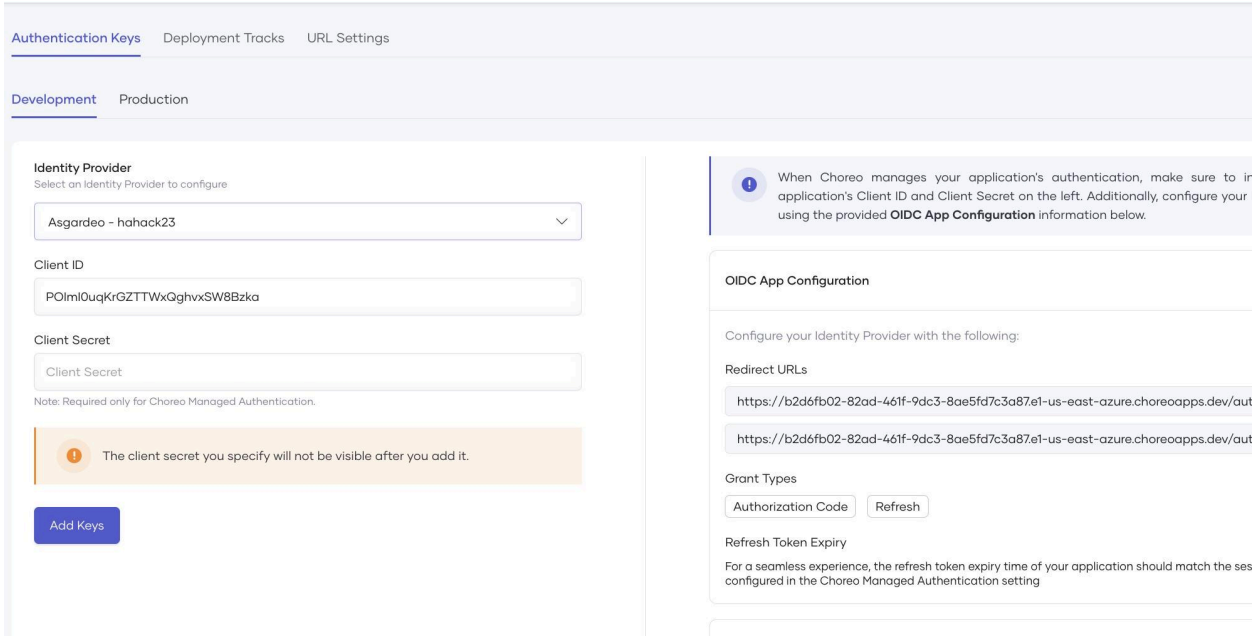
- Click and expand the Dependencies on the left navigation menu and click Connection tab
- Select the Hotel Reservation Service that was created at <<step-5 and step-6>>
- Provide a name and the description
- Then click create
- Copy the serverURL for later usage
- Click Deploy on the left navigation menu. Click configure Deploy in the deployment page

```
window.configs = {
  apiUrl: '<<spring boot reservation core service url>>',
  catalogUrl :'<<ballerina graphql url>>',
```

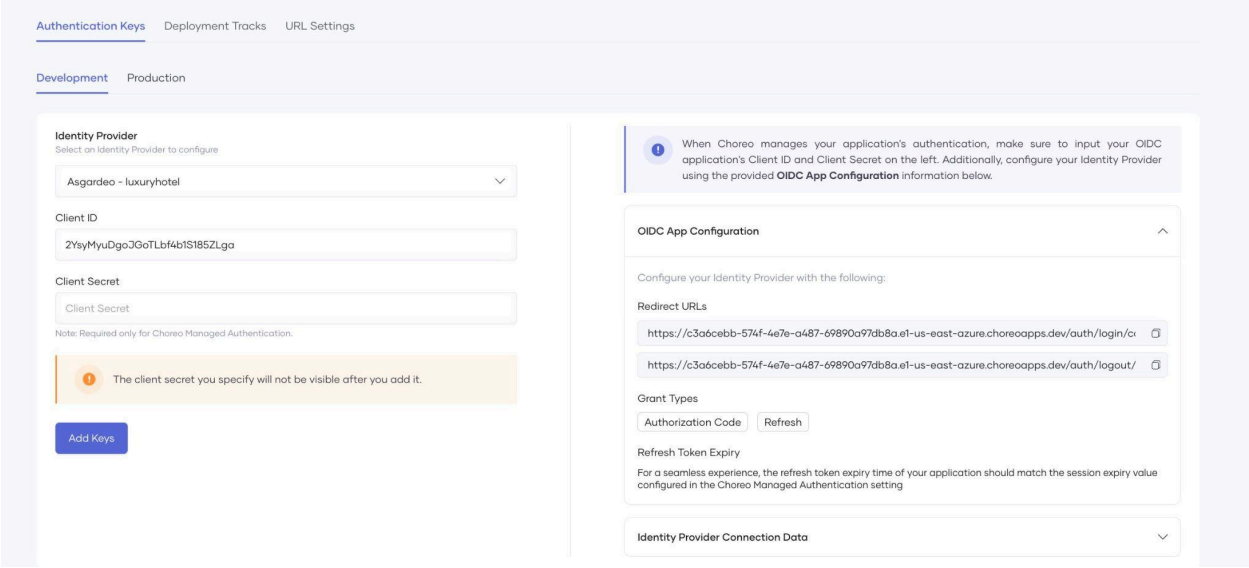
};

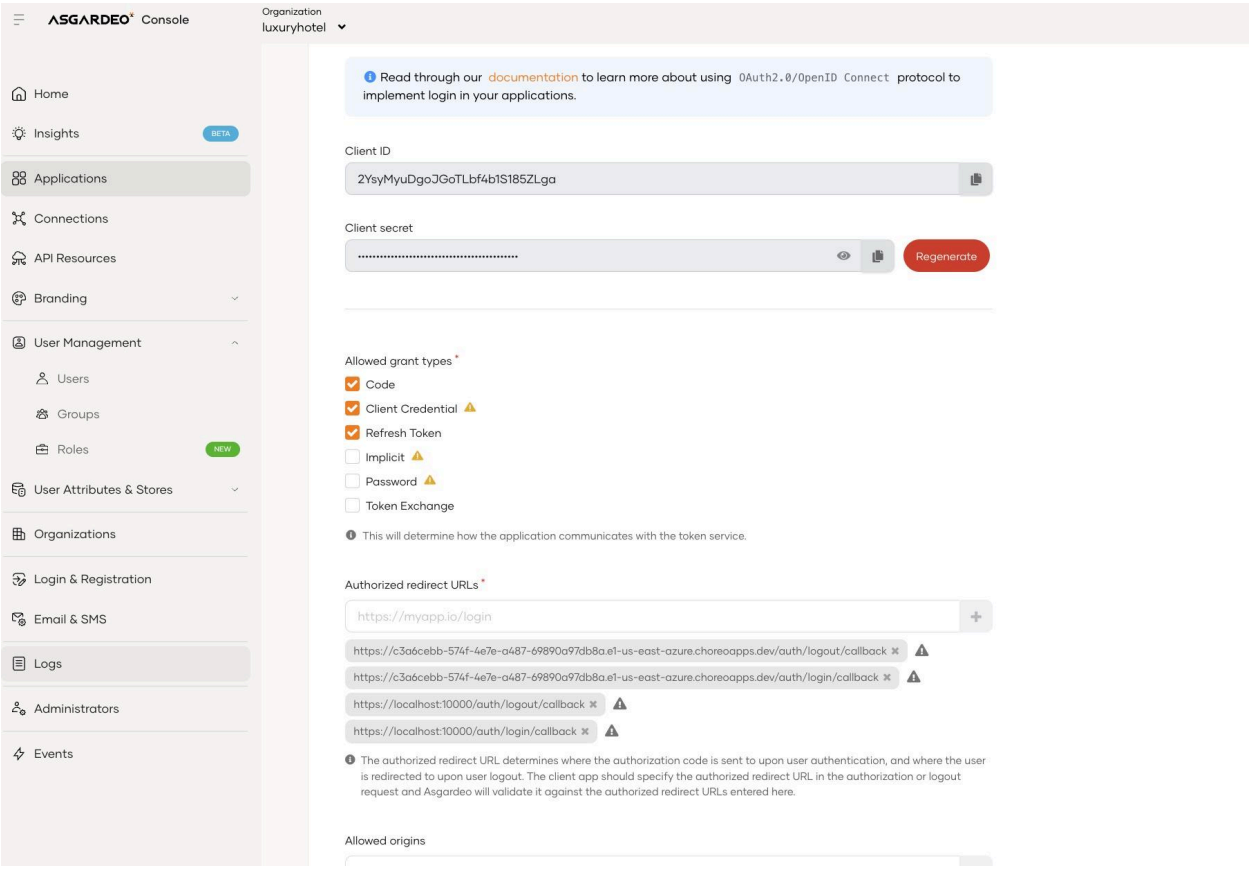


- Under the development tab, select Asgardeo as IDP.

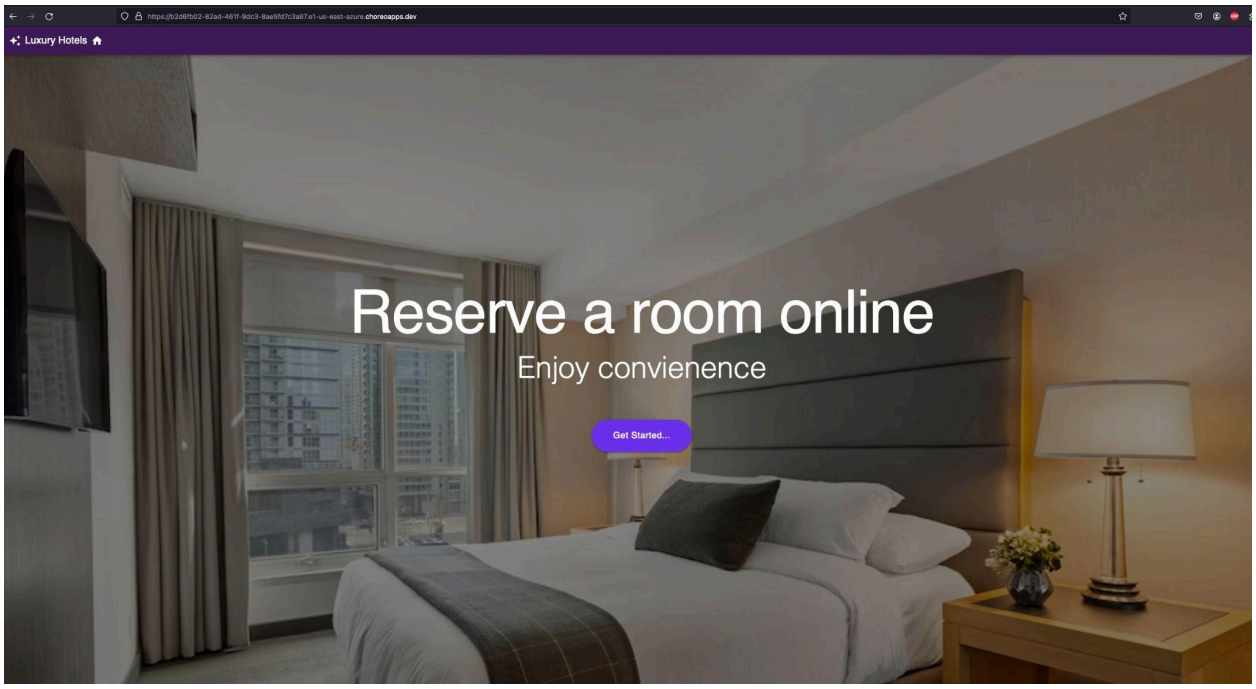


- Navigate to Asgardeo and configure the Application add callback URLs obtained from Choreo Authentication Keys





- Deploy the web application. If app deployed successfully, you may able to to obtain the URL to access the homepage of the web application



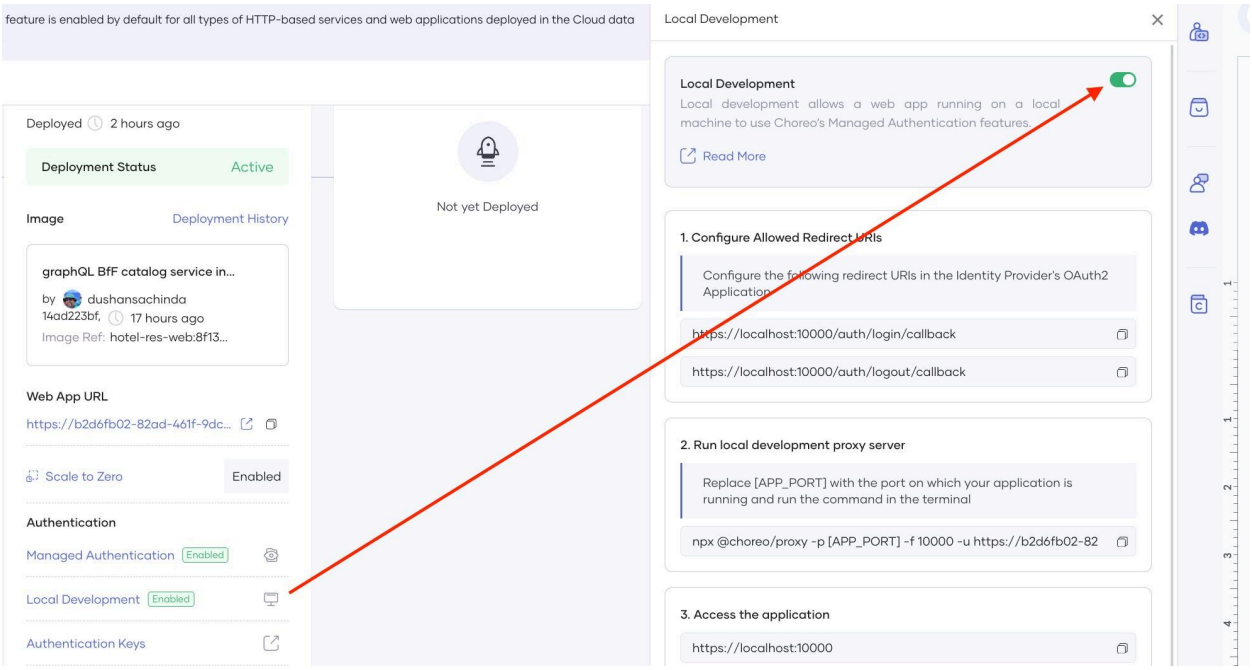
Extra: Run local (development mode)

Prerequisites: Java 17 , Maven, Python 3.x or above with Kafka_consumer, and Azure communication services installed via PIP3, Ballerina latest

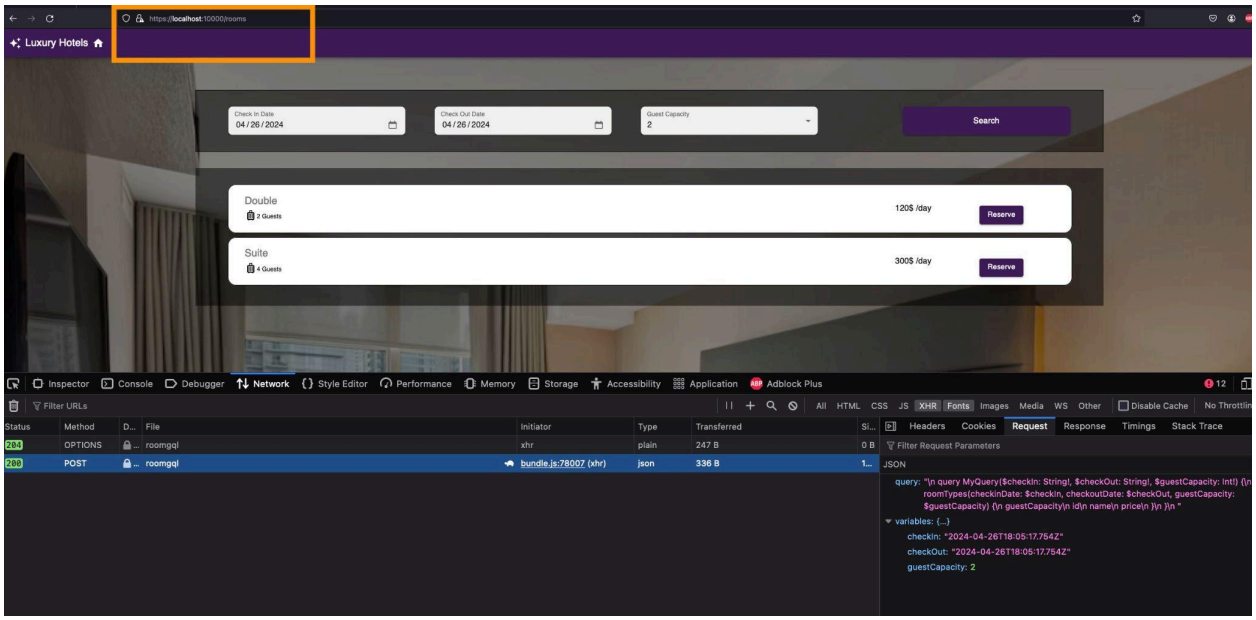
- Set Environment variables (should set up at each terminal you planning to run the backend services>
    - export DB_HOST=xx
    - export DB_PORT=xx
    - export DB_NAME=xx
    - export DB_USERNAME=xx
    - export DB_PASSWORD=xx
    - export KAFKA_USERNAME=xx
    - export KAFKA_PASSWORD=xx+x
    - export BOOTSTRAP_SERVERS="xx-lzvrd.us-west4.gcp.confluent.cloud:9092"

    - export SECURITY_PROTOCOL="SASL_SSL"
    - export SASL_MECHANISMS="PLAIN"
    - export SESSION_TIMEOUT_MS="45000"
    - export TOPIC_NAME="notifications"

- Navigate to graph **service-graphql**
    - Type **bal run**
- Open another terminal navigate to **<<notification-api-event>>** set environment variable
    - export AZURE_COMM_SERVICES_CONNECTION_STRING=”xxx"
    - export AZURE_COMM_SERVICES_SENDER_ADDRESS=xxx
    - export KAFKA_USERNAME=xx
    - export KAFKA_PASSWORD=xx+xx
    - export BOOTSTRAP_SERVERS="xx.gcp.confluent.cloud:9092"
    - export SECURITY_PROTOCOL="SASL_SSL"
    - export SASL_MECHANISMS="PLAIN"
    - export SESSION_TIMEOUT_MS="45000"
    - export TOPIC_NAME="notifications"
    -
    - Run command **flask run --host=0.0.0.0 --port=8081**
- Open another terminal navigate to <**<java-services>>**
    - Setup environment variable #1
    - Build maven project, navigate to target folder
    - Then execute following command
        - java -jar luxury-hotels-1.0.1.jar
- Open another terminal navigate to webapp folder (run following command)
    - npm install
    - npm run
- IMPORTANT: we should use the Choreo configured IDP for the webapp to authenticate successfully. Therefore first navigate to the choreo hosted web application deploy section
    - Enable local deployment
    - Add Callback URLs to the Asgardeo application created for user authentication
    - Copy “npx @choreo/proxy -p [APP_PORT] -f 10000 -u https://xxxx.e1-us-east-azure.choreoapps.dev”
    - Replace port with 3000
    - Run npx command above



- Access the web application via https://localhost:10000
- Continue your development efforts



Appendix

[1]

- Reservation core services (graphQL)

  http://localhost:9090/roomgql

Resource	Path	Action	Query Param	Path Param	Request	Respons

Get all available room types	/roomgql		string checkingDate
			
			String checkoutDate
			
			Int guestCapacity
					query RoomTypes{
					roomTypes(
					checkinDate: "2024-04-26T15:00:38.122Z"
					checkoutDate: "2024-04-26T16:00:38.122Z"
					
					guestCapacity: 1
					){
					id
					guestCapacity
					price
					name
					}
					}
						"data": {
						
						"roomTypes": [
						{
						"id": 0,
						
						"guestCapacity": 1,
						
						"price": 80,
						
						"name": "Single"
						},
						{
						"id": 1,
						
						"guestCapacity": 2,
						
						"price": 120,
						
						"name": "Double"
						},

						{
						"id": 3,
						"guestCapacity": 4,
						"price": 300,
						"name": "Suite"
						}
						]
						}
						}

- Reservation core services (spring boot services)

  http://localhost:8080/reservations

Resource	Path	Action	Query Param	Path Param	Request	Respons

Create new reservation		POST			{
					"checkinDate": "2024-02-19T14:00:00Z",
					"checkoutDate": "2024-02-20T10:00:00Z",
					"rate": 100,
					"user": {
					"id": "123",
						{
						"id": "1",
						"checkinDate": "2024-02-19T14:00:00Z",
						"checkoutDate": "2024-02-20T10:00:00Z",
						"user": {
						"id": "123",
						"name":

					"name": "waruna", "email": "waruna@someemail.com", "mobileNumber": "987" }, "roomType": "Family" }	"waruna", "email": "waruna@someemail.com ", "mobileNumber": "987" }, "room": { "number" : 201, "type": { "id": 0, "name": "Double", "guestCapacity": 2, "price": 100 }}
Update existing reservation		PUT		reservation_id	{ "checkinDate": "2024-02-20T14:00:00Z", "checkoutDate": "2024-02-21T10:00:00Z" }	{ "id": "1", "checkinDate": "2024-02-19T14:00:00Z", "checkoutDate": "2024-02-21T10:00:00Z", "user": { "id": "123", "name": "waruna", "email": "waruna@someemail.com ", "mobileNumber": "987" }, "room": {

						"number" : 201,
						"type": {
						"id": 0,
						"name": "Double",
						"guestCapacity": 2,
						"price": 100
						}}
						}

Remove a reservation		DELETE		reservation_id		

Retrieve all reservation for user	/users/	GET		userID		[
						{
						"checkinDate": "2024-02-19T14:00:00Z",
						"checkoutDate": "2024-02-20T10:00:00Z",
						"rate": 120,
						"user": {
						"id": "123",
						"name": "waruna",
						"email": "waruna@someemail.com ",
						"mobileNumber": "987"
						},
						"roomType": "Family"
						},

						{ "checkinDate": "2024-02-23T14:00:00Z", "checkoutDate": "2024-02-24T10:00:00Z", "rate": 100, "user": { "id": "123", "name": "waruna", "email": "waruna@someemail.com ", "mobileNumber": "987" }, "roomType": "Double" }]

[2] Obtaining Azure communication
https://learn.microsoft.com/en-us/azure/communication-services/quickstarts/email/add-azure-managed-domains

https://learn.microsoft.com/en-us/azure/communication-services/quickstarts/email/send-email?tabs=windows%2Cconnection-string&pivots=programming-language-python