Cloud Computing Major Project

MyHotelAssistant: An AI Chatbot for Hotel Bookings

Shelly Kashyap

06/07/2025

---

1. Project Overview & Goal

This project focuses on developing "MyHotelAssistant," an AI-powered chatbot designed to streamline the hotel room booking process. The primary goal is to enable users to interact conversationally to provide booking details, receive dynamic confirmations including pricing, and have their booking information securely stored.

Core Project Requirement: All booking information, including the calculated price and duration of stay, must be conveyed to the user upon successful fulfilment.

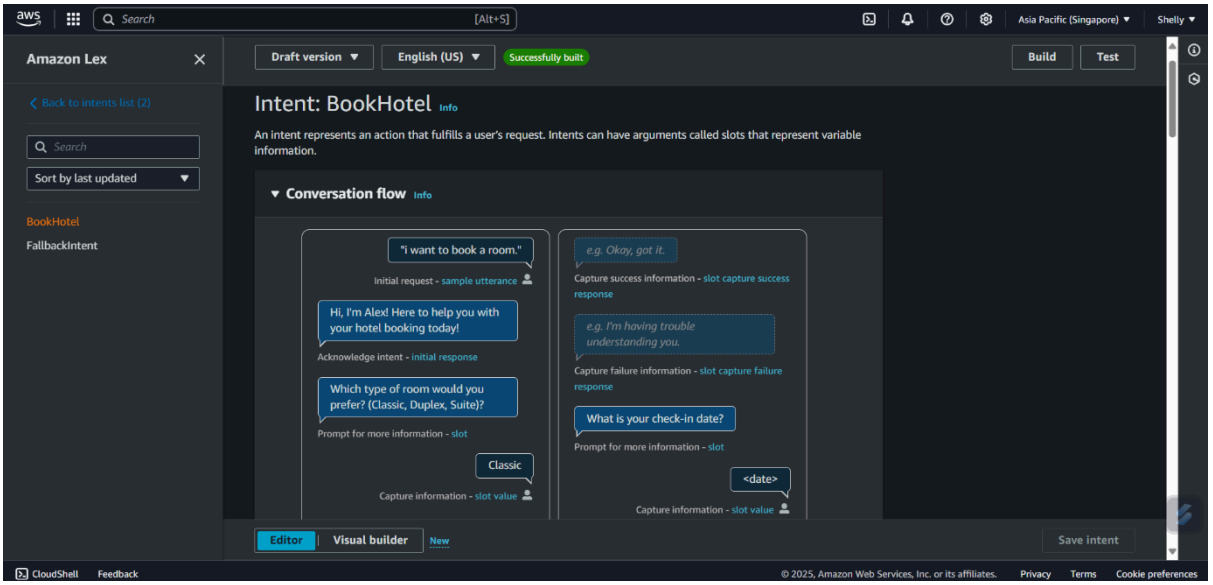
---

2. Amazon Lex: The Conversational Core

Amazon Lex is utilized to build the conversational interface of our chatbot. It handles understanding user input, managing the dialogue flow, and collecting necessary information.

2.1. Intent Definition: BookHotel

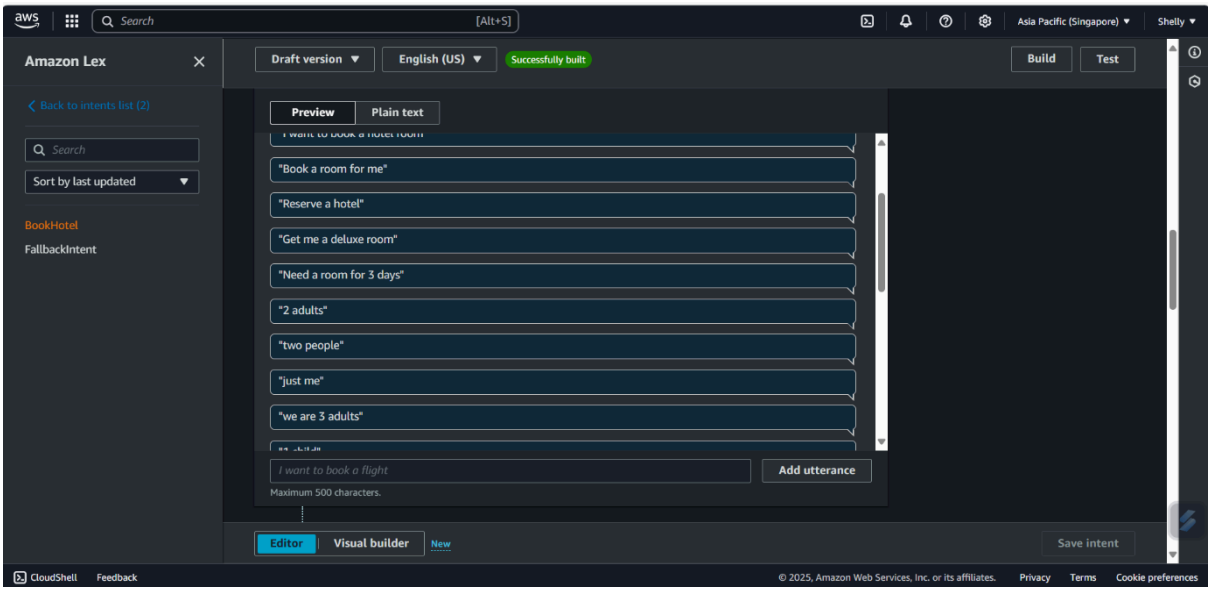
The central purpose of our chatbot is encapsulated in the BookHotel intent. This intent is designed to recognize when a user wishes to reserve a hotel room and then orchestrate the collection of all required booking details.



The BookHotel intent was configured within the Amazon Lex console as the primary conversational goal for the chatbot.

2.2. Utterance Generation

Utterances are sample phrases that train the BookHotel intent to understand various ways users might express their desire to book a hotel.

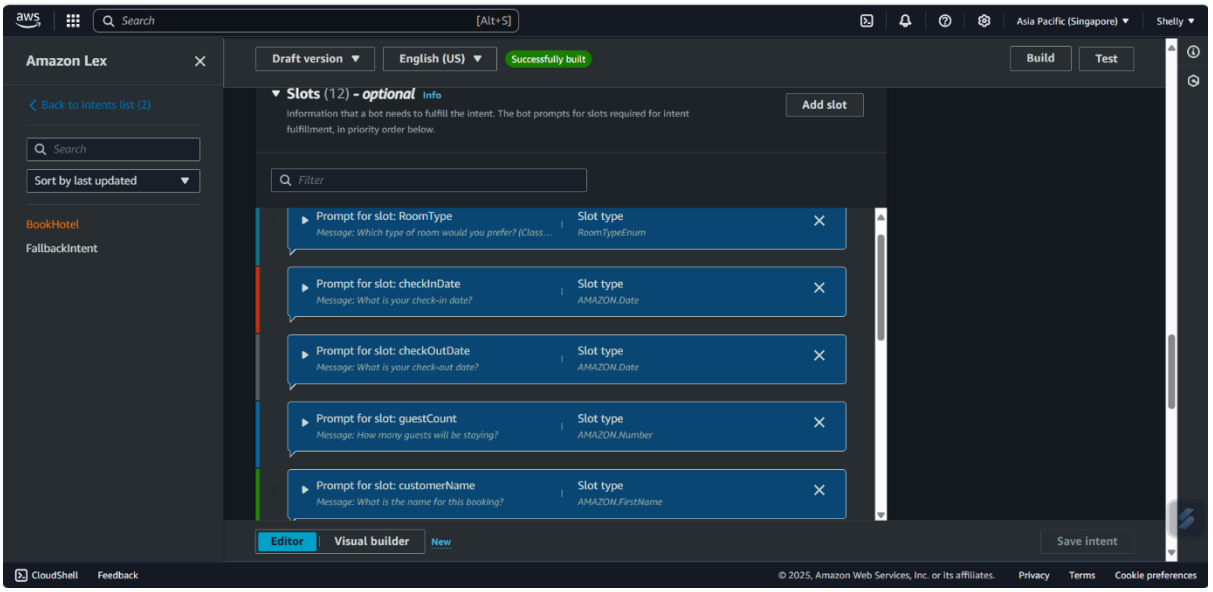


These utterances are typical phrases a user might say to initiate a booking. Examples include "I want to book a hotel", "Book a room for me", "Make a reservation", or more specific requests like "I need a classic room for two guests". By supplying a diverse range of these

phrases in the Amazon Lex console, we train the chatbot to accurately identify and fulfil the user's intention, regardless of how they phrase their request.

2.3. Slot Configuration

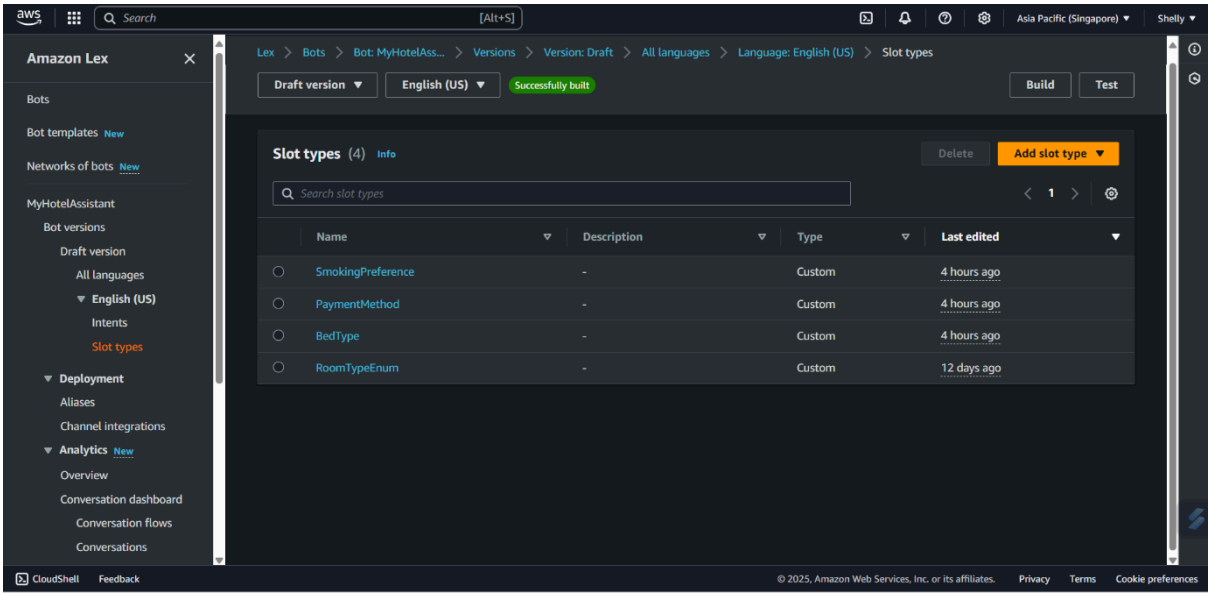
Slots represent the specific pieces of information the bot needs to collect from the user to fulfill the BookHotel intent. Each slot has a defined type and a prompt to guide the user.



Key slots include RoomType, checkInDate, checkOutDate, guestCount, City, HotelName, BedType, SmokingPreference, and PaymentMethod. These slots ensure all necessary details for a hotel booking are systematically gathered.

2.4. Custom Slot Types

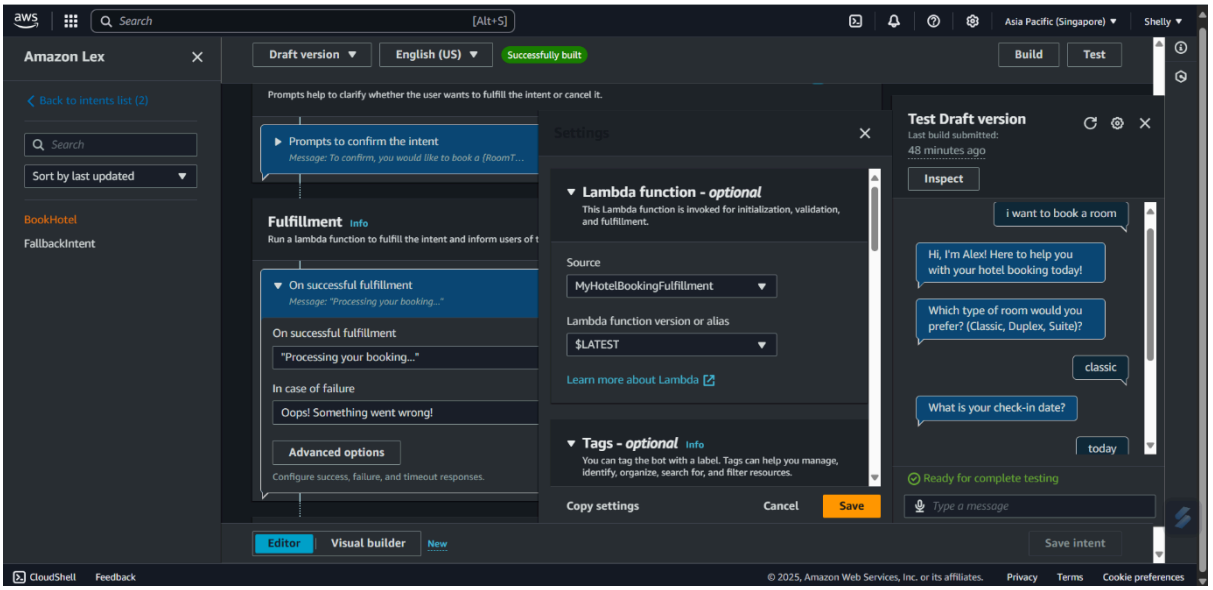
Custom slot types were created to define specific sets of values for certain booking preferences, ensuring the bot accurately recognizes user input for these categories.



For instance, BedType specifies options like 'king bed' and 'Twin bed', while SmokingPreference includes 'non-smoking room' and 'smoking room'. This helps restrict user input to valid choices.

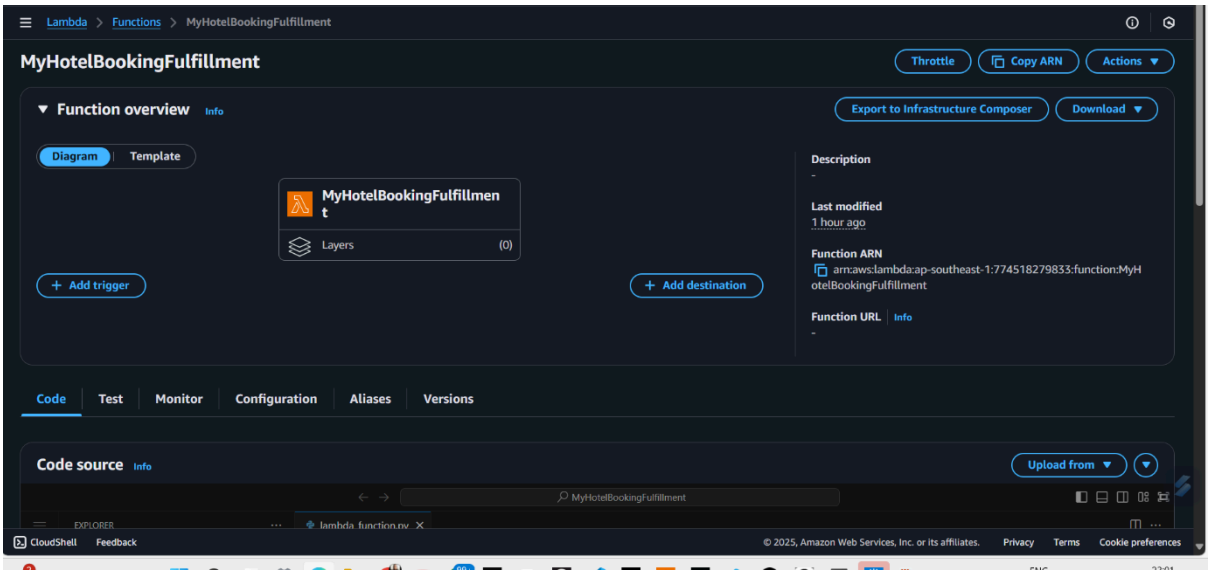
2.5. Fulfillment Configuration

The Fulfillment section configures how the BookHotel intent is completed after all necessary information is collected. Our bot leverages an AWS Lambda function, MyHotelBookingFulfillment, for this crucial step, passing all collected data to it for backend processing and database storage.



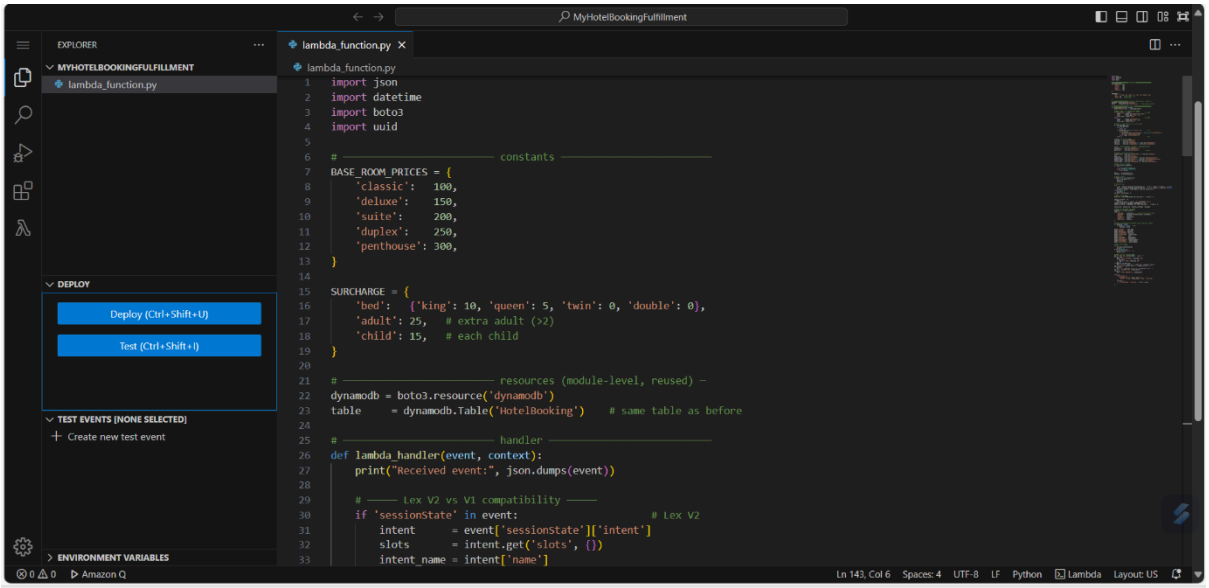
3. AWS Lambda: The Brains of the Operation

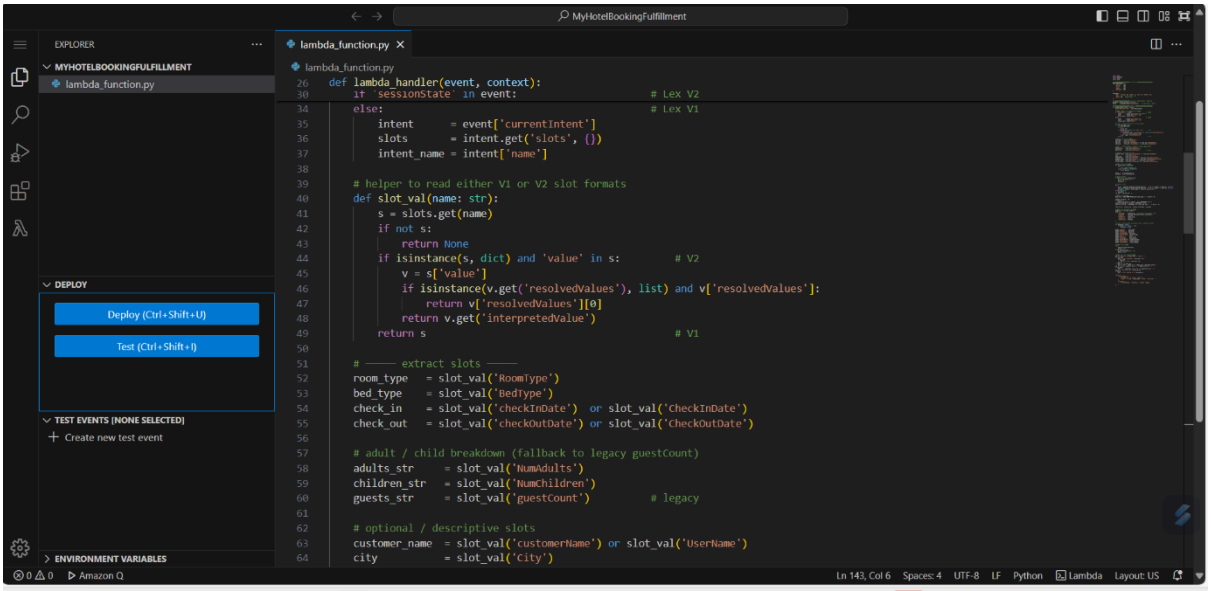
AWS Lambda provides the serverless compute power for our chatbot's backend logic. It processes the information collected by Lex and interacts with our database.

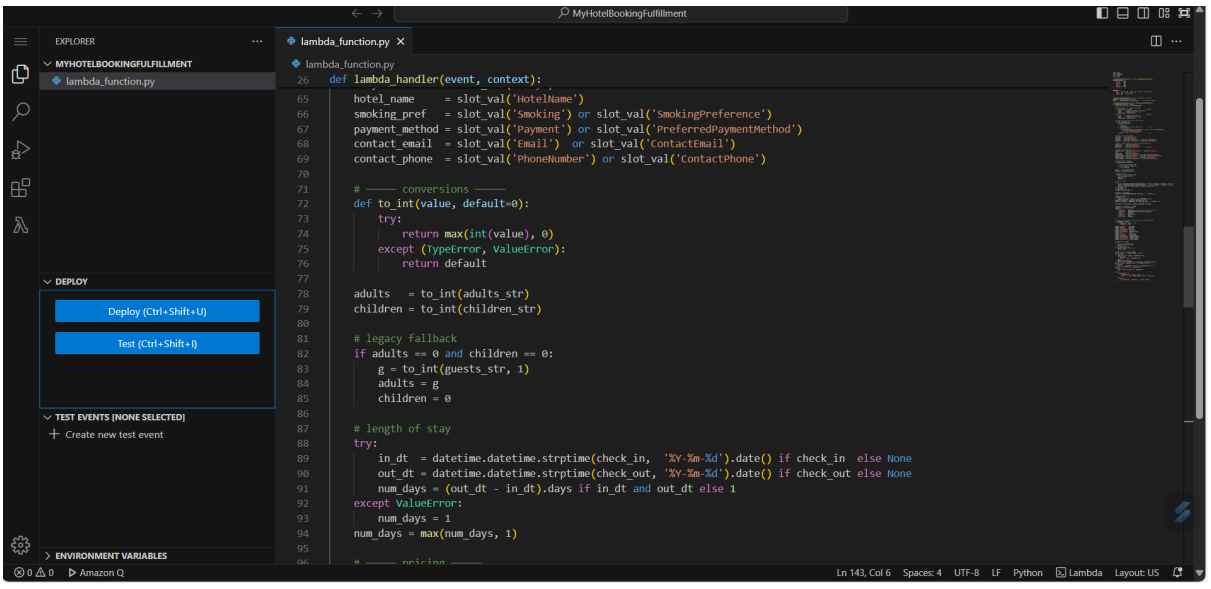


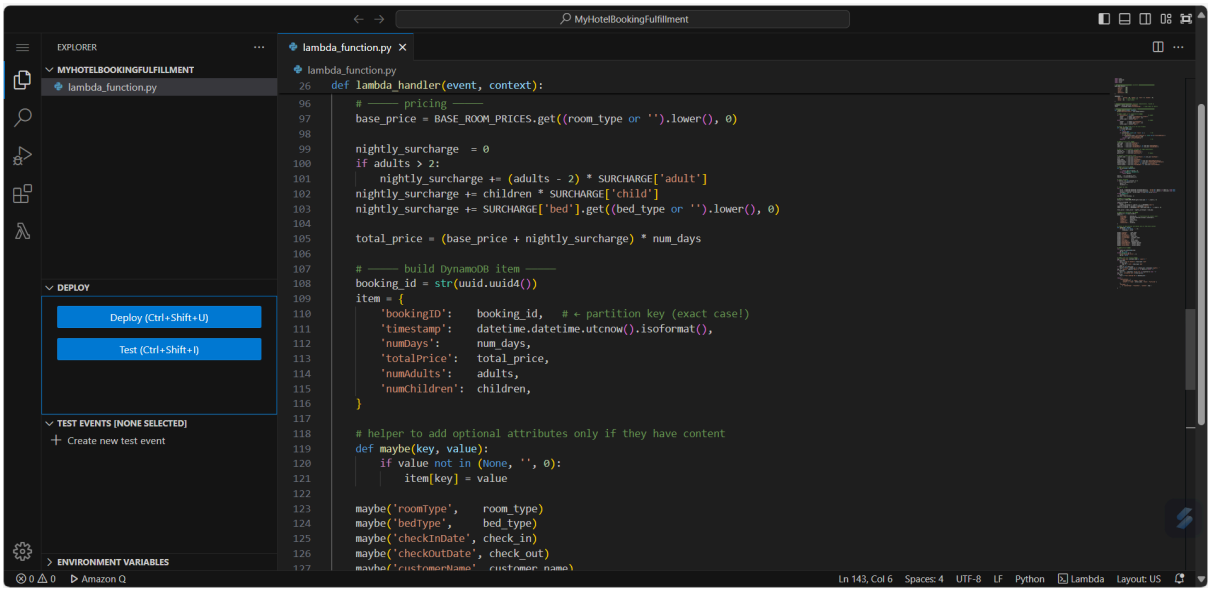
3.1. Lambda Function Code

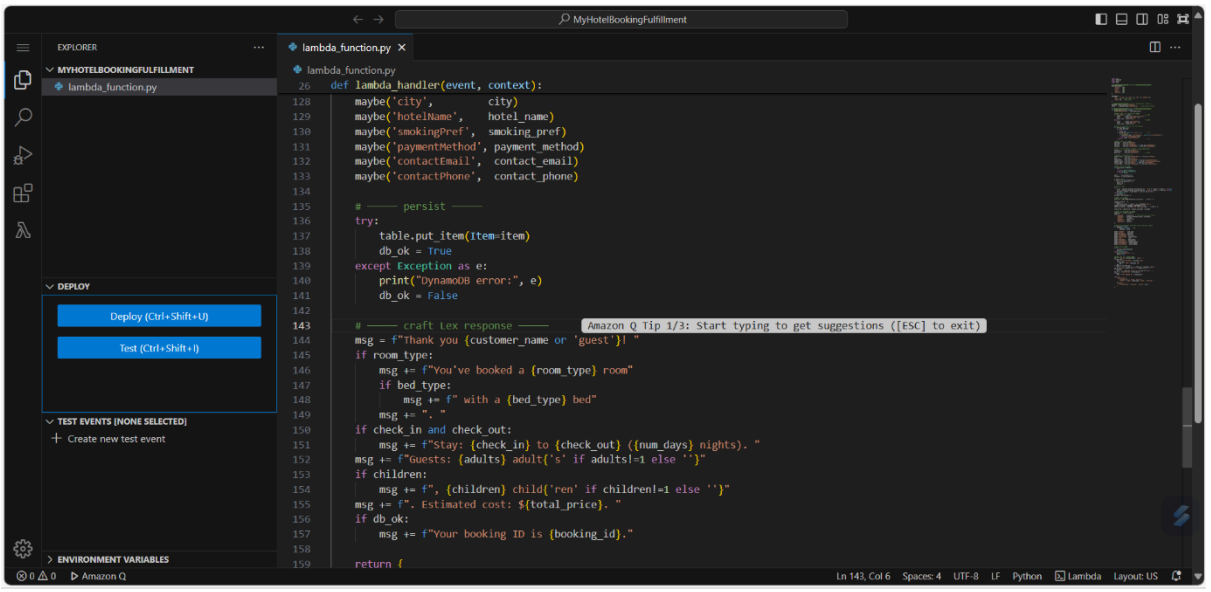
The MyHotelBookingFulfillment Lambda function receives the collected slot values from Lex. It extracts details like room type, dates, guest count, city, hotel name, and preferences. It then calculates the total price and duration of stay. Most importantly, it generates a unique bookingId and stores all these booking details in the DynamoDB table.

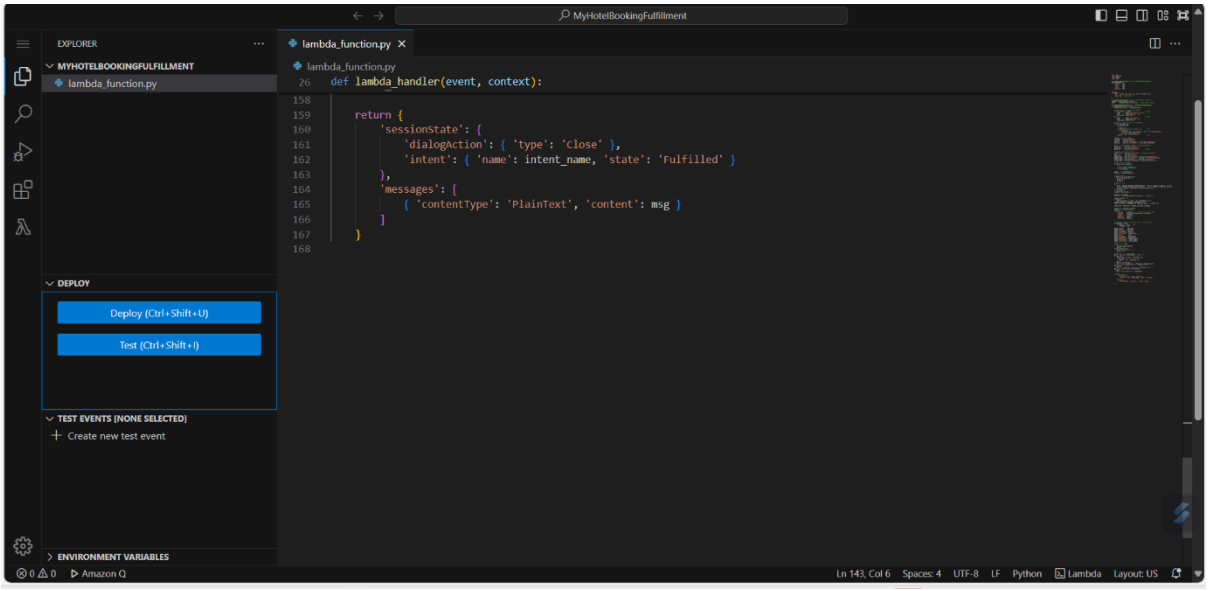

MyHotelBookingFulfillment
EXPLORER
MYHOTELBOOKINGFULFILLMENT
lambda_function.py
DEPLOY
Deploy (Ctrl+Shift+U)
Test (Ctrl+Shift+I)
TEST EVENTS [NONE SELECTED]
Create new test event
ENVIRONMENT VARIABLES
lambda_function.py
import json
import datetime
import boto3
import uuid
# ──────────── constants ────────────
BASE_ROOM_PRICES = {
'classic': 100,
'deluxe': 150,
'suite': 200,
'duplex': 250,
'penthouse': 300,
}
SURCHARGE = {
'bed': {'king': 10, 'queen': 5, 'twin': 0, 'double': 0},
'adult': 25, # extra adult (>2)
'child': 15, # each child
}
# ──────────── resources (module-level, reused) ─
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('HotelBooking') # same table as before
# ──────────── handler ────────────
def lambda_handler(event, context):
print("Received event:", json.dumps(event))
# ─── Lex V2 vs V1 compatibility ───
if 'sessionState' in event: # Lex V2
intent = event['sessionState']['intent']
slots = intent.get('slots', {})
intent_name = intent['name']
Amazon Q
Ln 143, Col 6
Spaces: 4
UTF-8
LF
Python
Lambda
Layout: US


MyHotelBookingFulfillment
EXPLORER
MYHOTELBOOKINGFULFILLMENT
lambda_function.py
DEPLOY
Deploy (Ctrl+Shift+U)
Test (Ctrl+Shift+I)
TEST EVENTS [NONE SELECTED]
Create new test event
ENVIRONMENT VARIABLES
lambda_function.py
def lambda_handler(event, context):
if 'sessionState' in event: # Lex V2
else: # Lex V1
intent = event['currentIntent']
slots = intent.get('slots', {})
intent_name = intent['name']
# helper to read either V1 or V2 slot formats
def slot_val(name: str):
s = slots.get(name)
if not s:
return None
if isinstance(s, dict) and 'value' in s: # V2
v = s['value']
if isinstance(v.get('resolvedValues'), list) and v['resolvedValues']:
return v['resolvedValues'][0]
return v.get('interpretedValue')
return s # V1
# ─── extract slots ───
room_type = slot_val('RoomType')
bed_type = slot_val('BedType')
check_in = slot_val('checkInDate') or slot_val('CheckInDate')
check_out = slot_val('checkOutDate') or slot_val('CheckOutDate')
# adult / child breakdown (fallback to legacy guestCount)
adults_str = slot_val('NumAdults')
children_str = slot_val('NumChildren')
guests_str = slot_val('guestCount') # legacy
# optional / descriptive slots
customer_name = slot_val('customerName') or slot_val('UserName')
city = slot_val('City')
Amazon Q
Ln 143, Col 6
Spaces: 4
UTF-8
LF
Python
Lambda
Layout: US


MyHotelBookingFulfillment
EXPLORER
MYHOTELBOOKINGFULFILLMENT
lambda_function.py
DEPLOY
Deploy (Ctrl+Shift+U)
Test (Ctrl+Shift+I)
TEST EVENTS [NONE SELECTED]
Create new test event
ENVIRONMENT VARIABLES
lambda_function.py
def lambda_handler(event, context):
hotel_name = slot_val('HotelName')
smoking_pref = slot_val('Smoking') or slot_val('SmokingPreference')
payment_method = slot_val('Payment') or slot_val('PreferredPaymentMethod')
contact_email = slot_val('Email') or slot_val('ContactEmail')
contact_phone = slot_val('PhoneNumber') or slot_val('ContactPhone')
# ─── conversions ───
def to_int(value, default=0):
try:
return max(int(value), 0)
except (TypeError, ValueError):
return default
adults = to_int(adults_str)
children = to_int(children_str)
# legacy fallback
if adults == 0 and children == 0:
g = to_int(guests_str, 1)
adults = g
children = 0
# length of stay
try:
in_dt = datetime.datetime.strptime(check_in, '%Y-%m-%d').date() if check_in else None
out_dt = datetime.datetime.strptime(check_out, '%Y-%m-%d').date() if check_out else None
num_days = (out_dt - in_dt).days if in_dt and out_dt else 1
except ValueError:
num_days = 1
num_days = max(num_days, 1)
Amazon Q
Ln 143, Col 6
Spaces: 4
UTF-8
LF
Python
Lambda
Layout: US

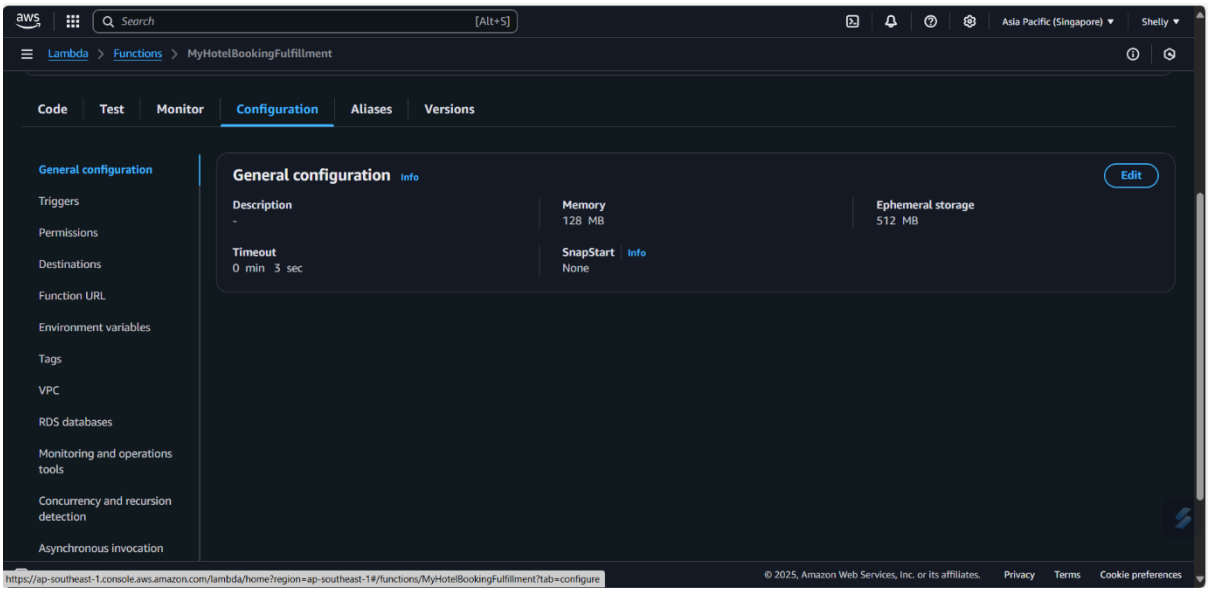

MyHotelBookingFulfillment
EXPLORER
MYHOTELBOOKINGFULFILLMENT
lambda_function.py
DEPLOY
Deploy (Ctrl+Shift+U)
Test (Ctrl+Shift+I)
TEST EVENTS [NONE SELECTED]
Create new test event
ENVIRONMENT VARIABLES
lambda_function.py
def lambda_handler(event, context):
# ——— pricing ———
base_price = BASE_ROOM_PRICES.get((room_type or '').lower(), 0)
nightly_surcharge = 0
if adults > 2:
nightly_surcharge += (adults - 2) * SURCHARGE['adult']
nightly_surcharge += children * SURCHARGE['child']
nightly_surcharge += SURCHARGE['bed'].get((bed_type or '').lower(), 0)
total_price = (base_price + nightly_surcharge) * num_days
# ——— build DynamoDB item ———
booking_id = str(uuid.uuid4())
item = {
'bookingID': booking_id, # ← partition key (exact case!)
'timestamp': datetime.datetime.utcnow().isoformat(),
'numDays': num_days,
'totalPrice': total_price,
'numAdults': adults,
'numChildren': children,
}
# helper to add optional attributes only if they have content
def maybe(key, value):
if value not in (None, '', 0):
item[key] = value
maybe('roomType', room_type)
maybe('bedType', bed_type)
maybe('checkInDate', check_in)
maybe('checkOutDate', check_out)
Ln 143, Col 6 Spaces: 4 UTF-8 LF Python Lambda Layout: US

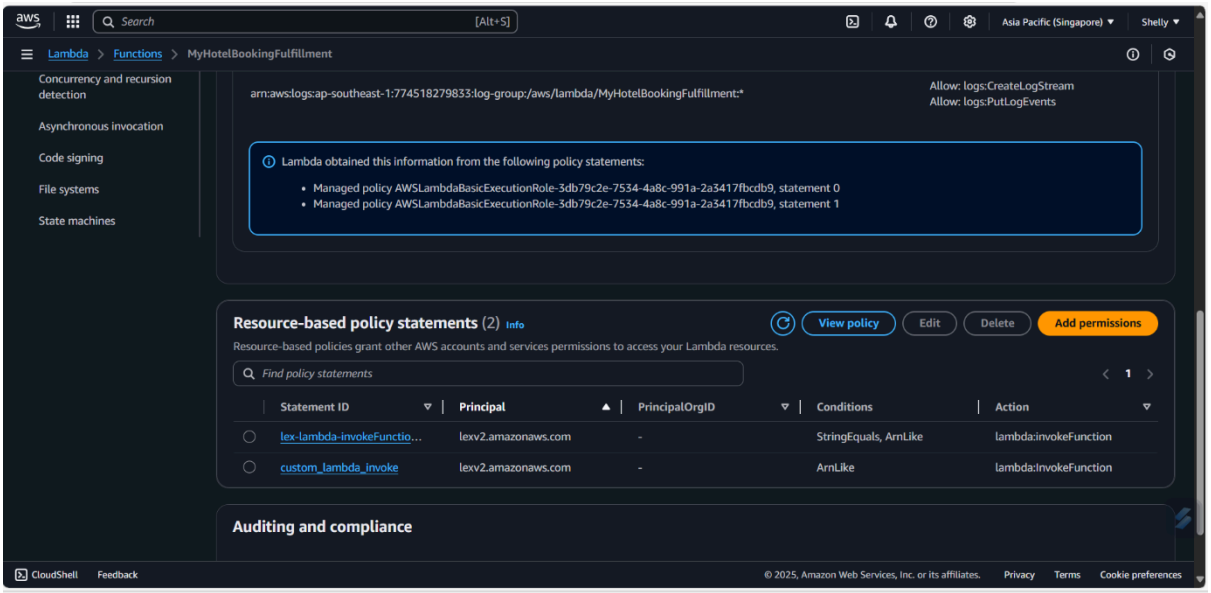

MyHotelBookingFulfillment
EXPLORER
MYHOTELBOOKINGFULFILLMENT
lambda_function.py
DEPLOY
Deploy (Ctrl+Shift+U)
Test (Ctrl+Shift+I)
TEST EVENTS [NONE SELECTED]
Create new test event
ENVIRONMENT VARIABLES
lambda_function.py
def lambda_handler(event, context):
maybe('city', city)
maybe('hotelName', hotel_name)
maybe('smokingPref', smoking_pref)
maybe('paymentMethod', payment_method)
maybe('contactEmail', contact_email)
maybe('contactPhone', contact_phone)
# ——— persist ———
try:
table.put_item(Item=item)
db_ok = True
except Exception as e:
print("DynamoDB error:", e)
db_ok = False
# ——— craft Lex response ———
Amazon Q Tip 1/3: Start typing to get suggestions ([ESC] to exit)
msg = f"Thank you {customer_name or 'guest'}! "
if room_type:
msg += f"You've booked a {room_type} room"
if bed_type:
msg += f" with a {bed_type} bed"
msg += ". "
if check_in and check_out:
msg += f"Stay: {check_in} to {check_out} ({num_days} nights). "
msg += f"Guests: {adults} adult{'s' if adults!=1 else ''}"
if children:
msg += f", {children} child{'ren' if children!=1 else ''}"
msg += f". Estimated cost: ${total_price}. "
if db_ok:
msg += f"Your booking ID is {booking_id}."
return {
Ln 143, Col 6 Spaces: 4 UTF-8 LF Python Lambda Layout: US


MyHotelBookingFulfillment
EXPLORER
MYHOTELBOOKINGFULFILLMENT
lambda_function.py
DEPLOY
Deploy (Ctrl+Shift+U)
Test (Ctrl+Shift+I)
TEST EVENTS [NONE SELECTED]
Create new test event
ENVIRONMENT VARIABLES
lambda_function.py
def lambda_handler(event, context):
return {
'sessionState': {
'dialogAction': { 'type': 'Close' },
'intent': { 'name': intent_name, 'state': 'Fulfilled' }
},
'messages': [
{ 'contentType': 'PlainText', 'content': msg }
]
}
Ln 143, Col 6 Spaces: 4 UTF-8 LF Python Lambda Layout: US

3.2. Lambda Function Permissions

The Lambda function requires specific IAM (Identity and Access Management) permissions to interact with other AWS services, particularly DynamoDB. The attached IAM role grants the Lambda function the necessary permissions, such as AmazonDynamoDBFullAccess, to perform put_item operations and write booking data to the HotelBooking table.





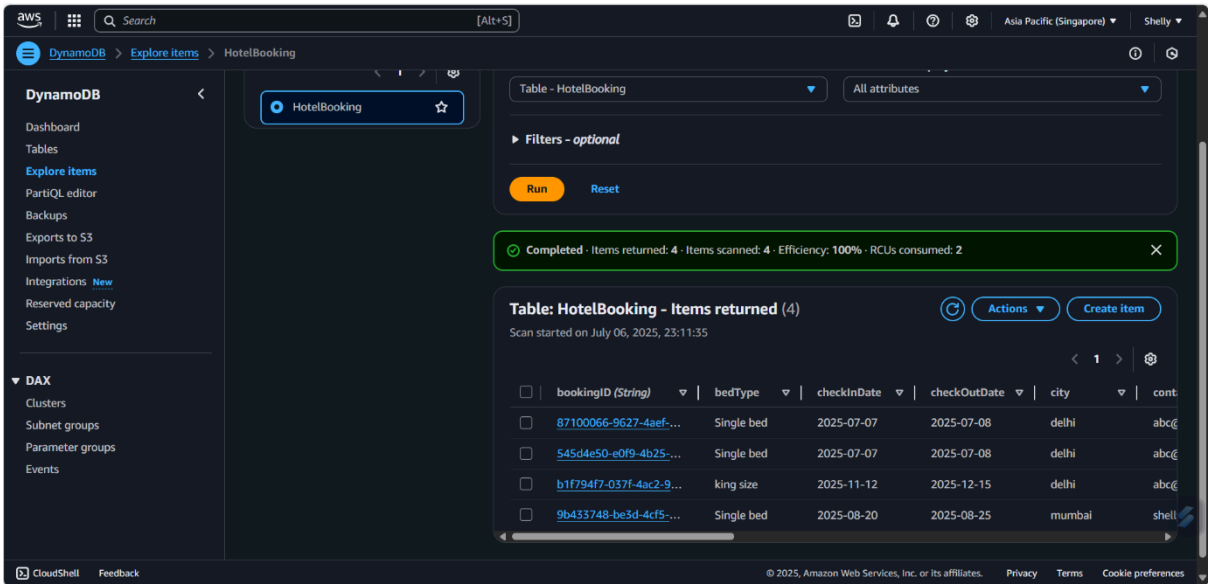
---

4. Amazon DynamoDB: Storing the Bookings

Amazon DynamoDB is used as our NoSQL database to persistently store all the hotel booking records processed by the chatbot.

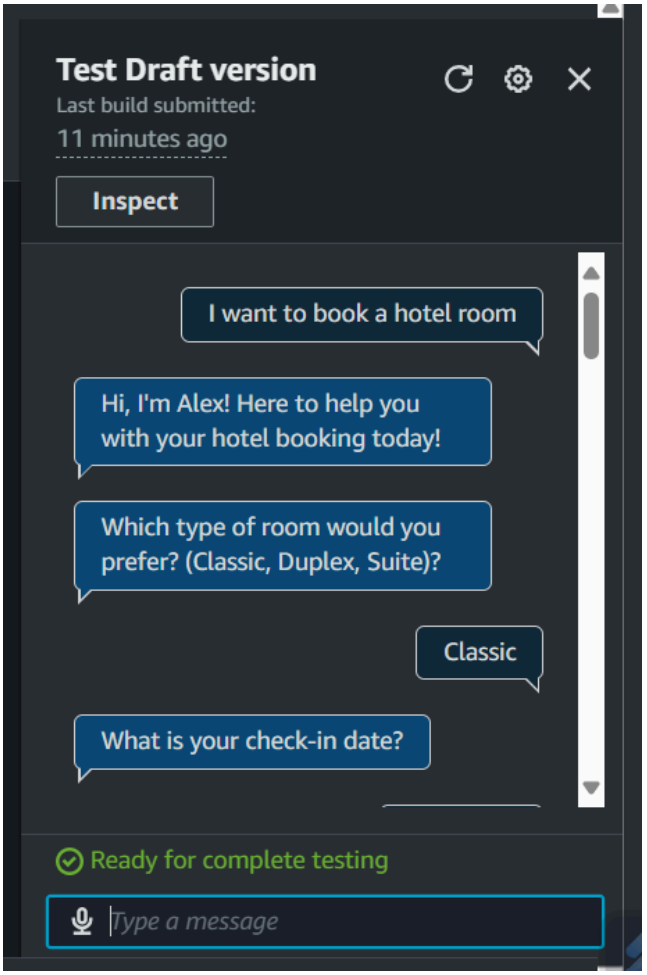
4.1. HotelBooking Table Items

Each successful booking conversation results in a new item (record) being added to the HotelBooking table in DynamoDB. The bookingId serves as the unique partition key for each record, ensuring efficient storage and retrieval. This demonstrates that our Lambda function successfully saves data to the database.

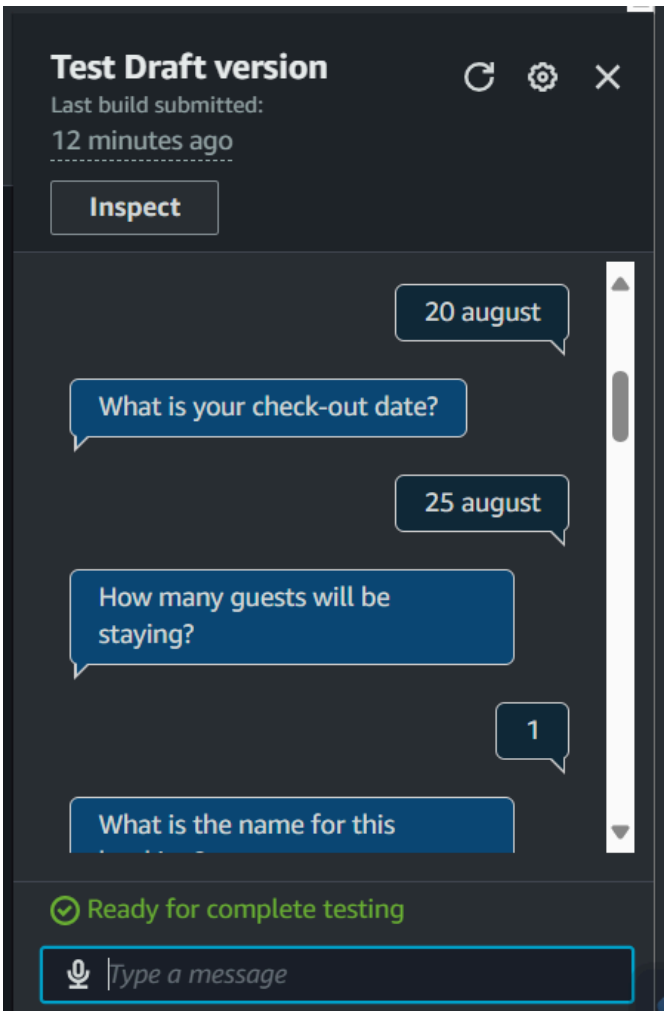


5. Demonstration / Testing

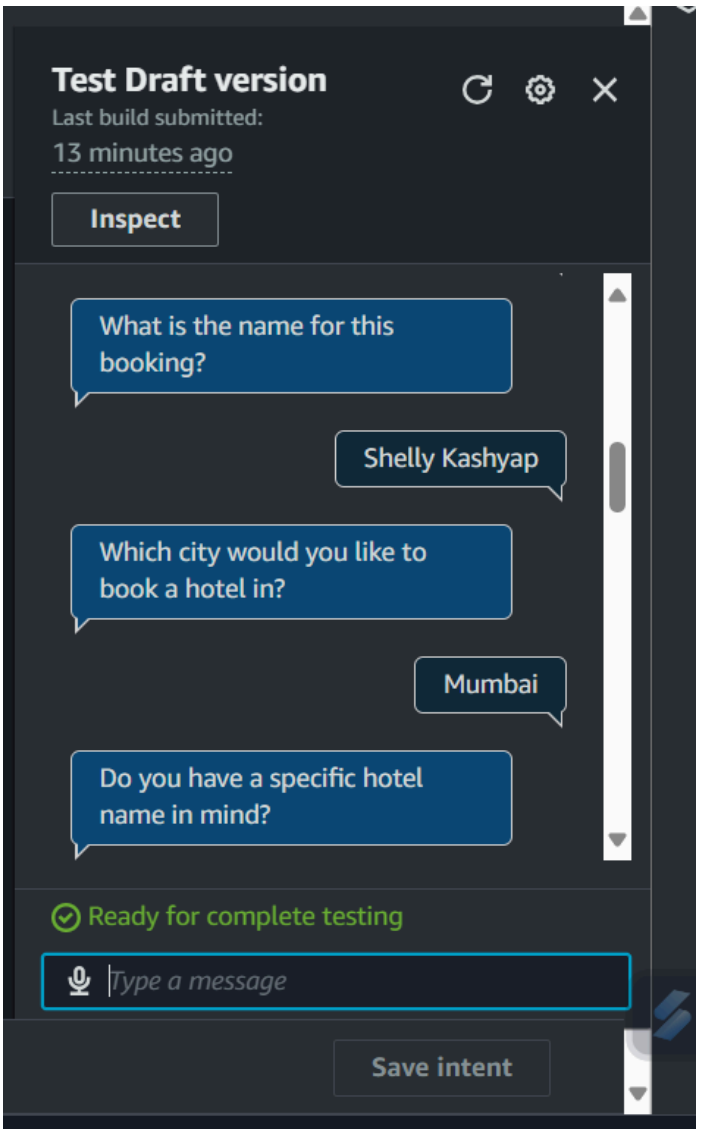
The chatbot was extensively tested within the Amazon Lex console to ensure the end-to-end conversational flow and backend integration are functional.



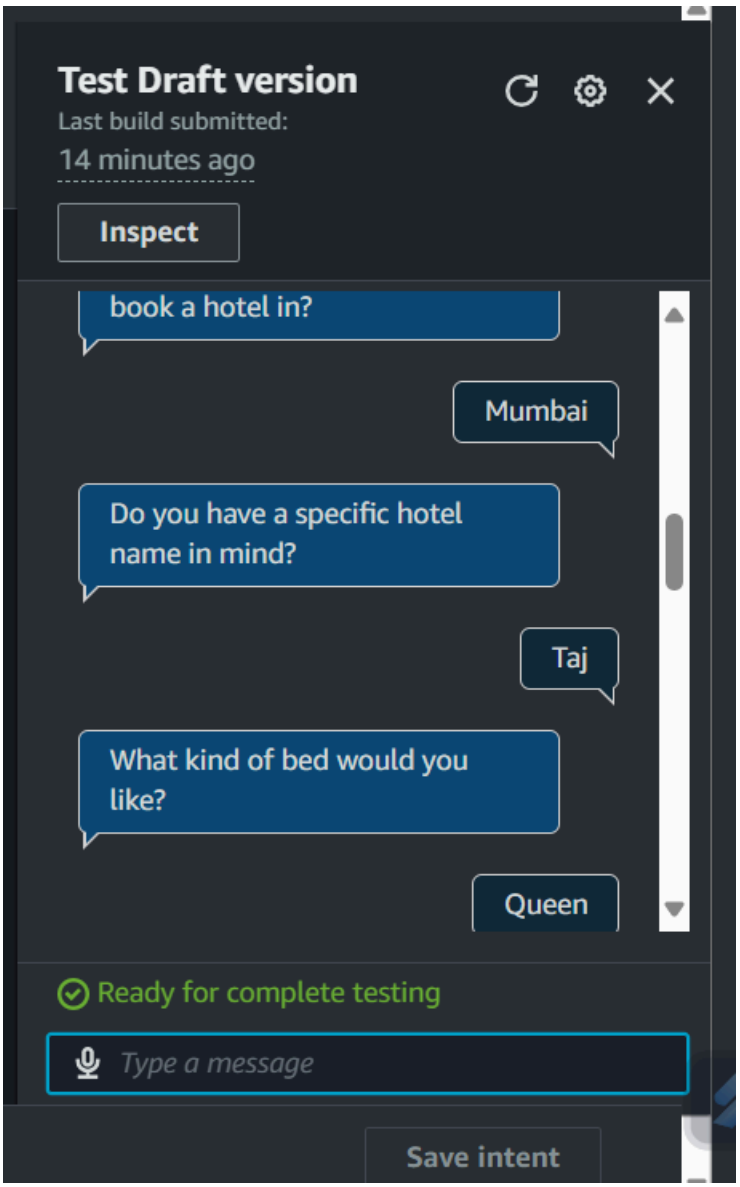
This is the initial part of a conversation in the Lex "Test Draft version" pane. The user starts by expressing intent ("I want to book a hotel room"), and the bot introduces itself before asking for the desired room type ("Classic, Duplex, Suite?") and then the check-in date.



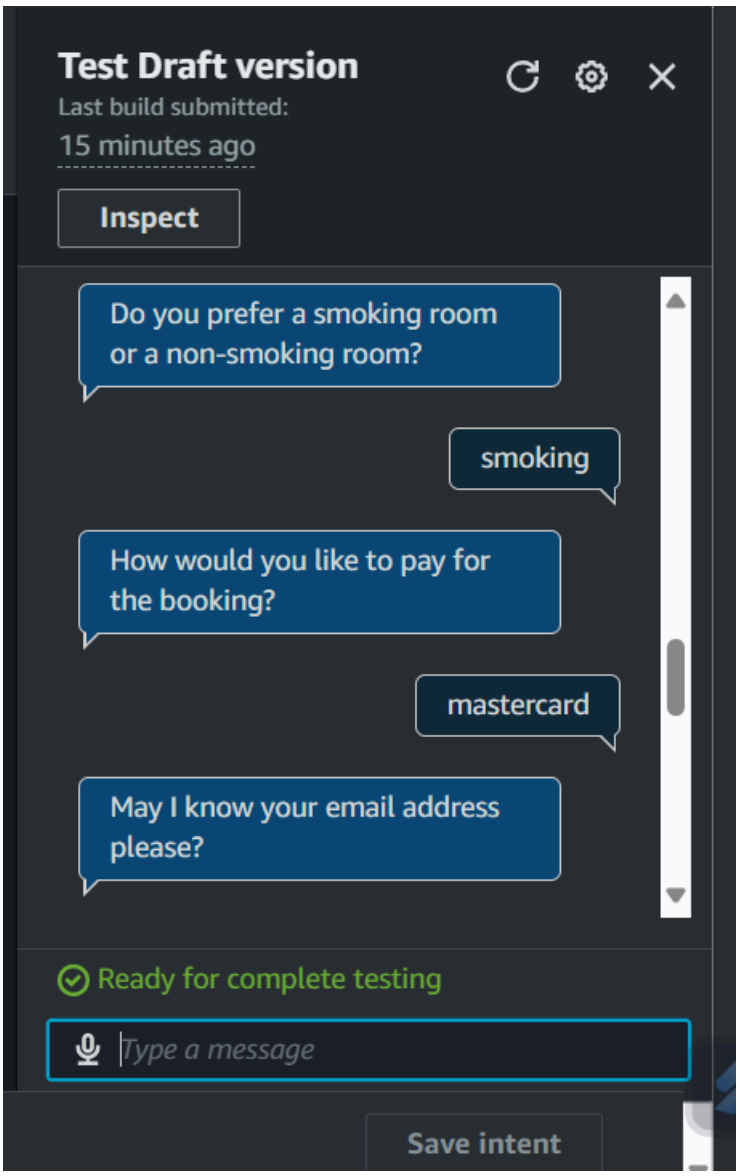
Continuing the conversation, the user provides the check-in date ("20 august") and then the check-out date ("25 august"). The bot proceeds to ask for the number of guests staying, to which the user responds "1".



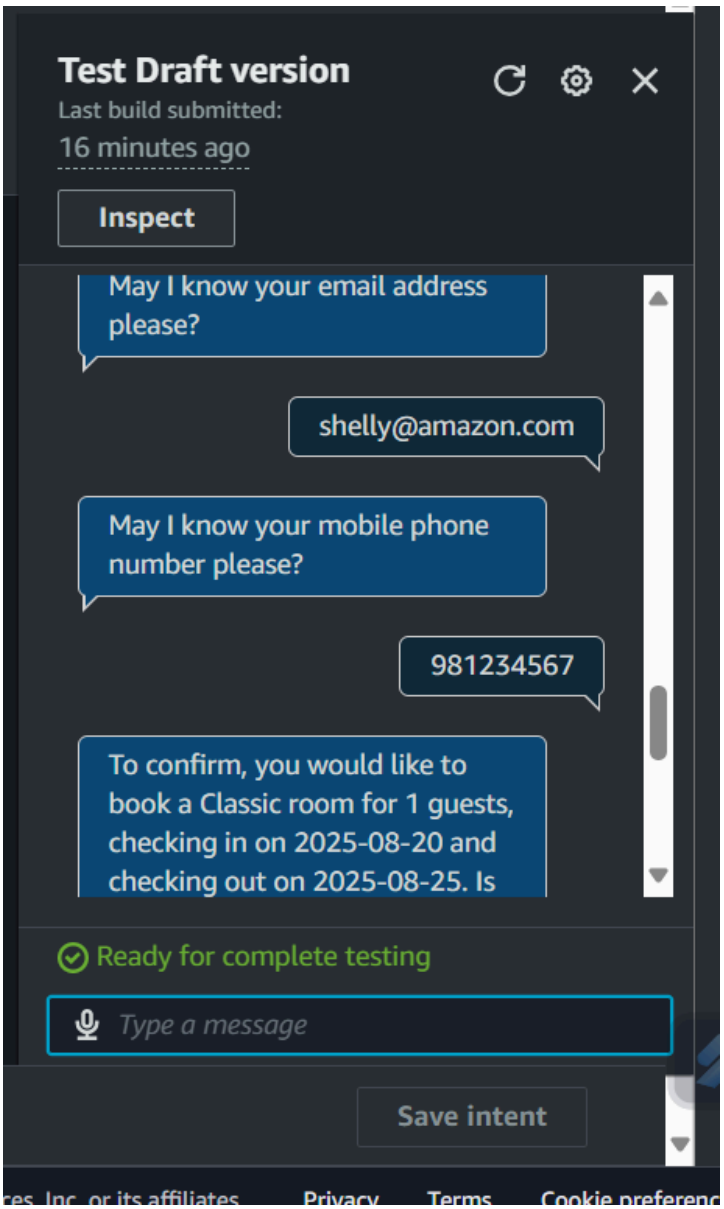
In this segment, the bot asks for the name associated with the booking ("Shelly Kashyap") and then prompts the user for the city where they'd like to book a hotel ("Mumbai"). The conversation then moves to ask for a specific hotel.



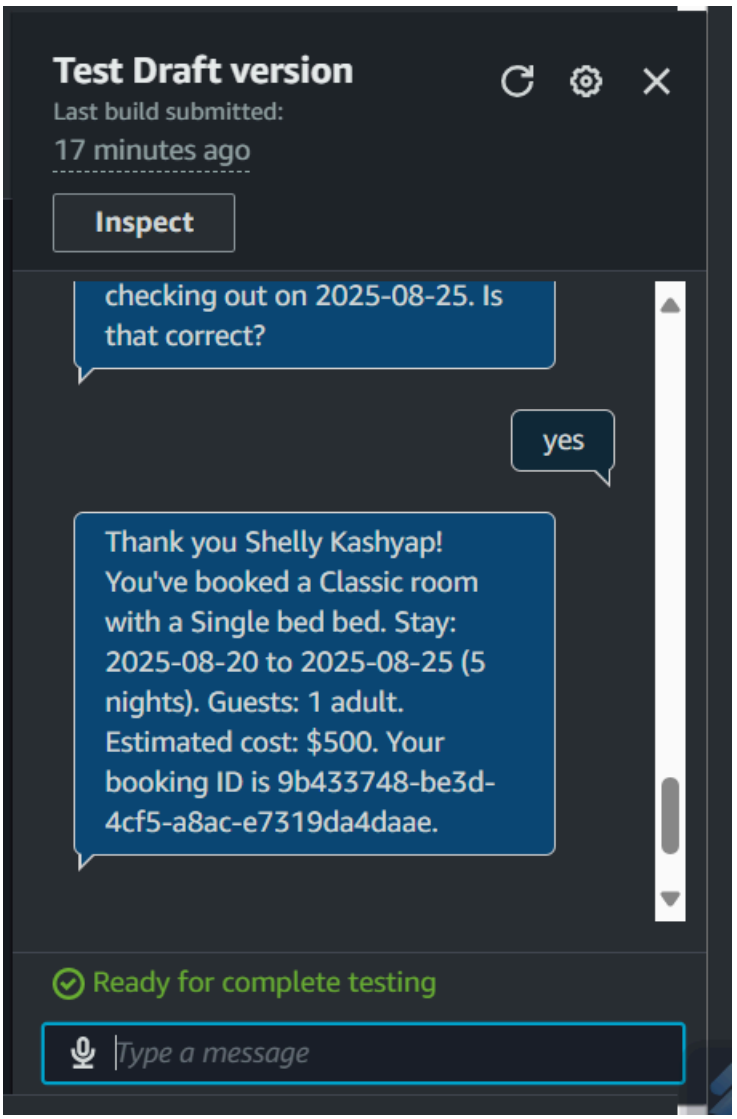
The conversation progresses with the user providing the city ("Mumbai") and then a hotel name ("Taj"). The bot then asks for the preferred bed type, demonstrating its ability to collect specific preferences.



This part of the conversation shows the bot asking about smoking preference ("Do you prefer a smoking room..?), the payment method ("How would you like to pay for the booking?"), and finally, requesting the user's email.



The user provides their email address and then their mobile phone number. The bot then begins the confirmation process, summarized.



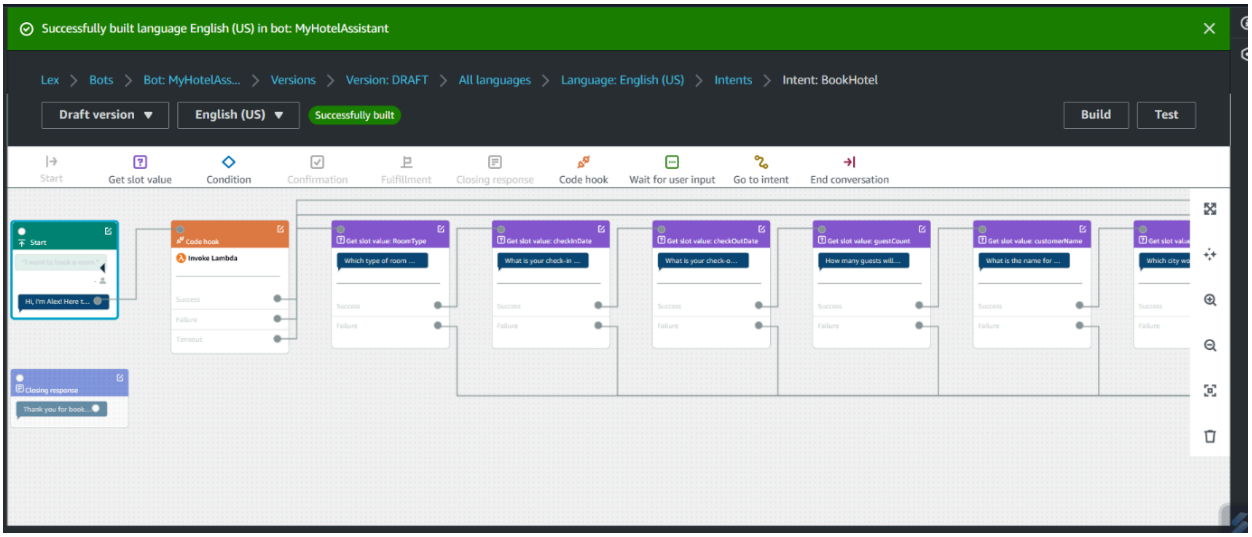
This final screenshot captures the successful fulfilment message from the bot. It confirms the booking details, including the room type ("Classic room"), bed type ("Single bed"), guest count ("1 adult"), stay dates ("2025-08-20 to 2025-08-25"), duration ("5 nights"), estimated cost ("$500"), and provides a unique booking ID. This confirms the end-to-end functionality of the chatbot and the successful return of processed information from Lambda.

---

6. Future Enhancements

While the core booking functionality is complete, several enhancements can further improve the chatbot:

- **Expanded User Preferences:** Integrate more detailed user preferences (e.g., number of adults/children, contact email/phone) into the conversation and DynamoDB storage.
- **Web Application Integration:** Deploy the chatbot to a user-friendly web interface. This would involve using Amazon Cognito for secure user authentication and an EC2 instance to host the web application, allowing broader access beyond the Lex console.
- **Advanced Booking Features:** Implement functionalities like booking modification, cancellation, or real-time room availability checks.



2.6. Visual Dialogue Flow

This screenshot displays the Amazon Lex V2 "Visual builder" interface for the BookHotel intent. The Visual builder provides a clear, graphical representation of the chatbot's conversational flow. It illustrates the sequence from the "Start" of the intent, through

various "Get slot value" nodes (for collecting information like room type, dates, guest count, customer name), and crucially, the "Call code hook" block which represents the invocation of the Lambda function for fulfillment. This visual aid helps in understanding and designing the step-by-step interaction of the bot.