# **Building Conversational AI with Amazon Bedrock, LangChain, and LangGraph: A Step-by-Step Guide**

## 

## **Objective**

The objective of this project is to **build an intelligent, stateful conversational AI assistant** powered by **Claude (via Amazon Bedrock)**, using **LangChain**, **LangGraph**, and **AWS-native session storage**. This assistant goes beyond a basic chatbot by maintaining conversation history, dynamically managing execution using graph-based logic, and persisting sessions through checkpointing — enabling real-time, memory-aware interactions across long conversations or even across multiple sessions.

Through this blog, readers will learn how to:

* Integrate **Claude 3** models using **Amazon Bedrock**
* Build a **graph-based conversational pipeline** using **LangGraph**
* Persist and manage **session memory** with **AWS-native checkpointing**
* Stream responses interactively and **maintain multi-turn dialogue**
* Structure modular conversational workflows for real-world AI applications

The goal is to help developers and AI enthusiasts **create smarter, context-aware AI systems** that scale reliably and can handle complex, long-running conversations — just like a human assistant would.

**What Are We Building?**

In this project, we’re creating a smart, context-aware AI assistant that goes far beyond a basic chatbot. By combining LangGraph, LangChain, and Claude 3 via Amazon Bedrock, our assistant is capable of the following:

* **Maintains conversation context** using session-aware memory
* **Supports long-running interactions** that users can resume later
* **Implements graph-based logic** for clean, modular, and maintainable workflows
* **Streams responses in real time** to mimic natural conversation
* **Runs on secure, serverless AWS infrastructure** using Amazon Bedrock

### **Real-World Applications**

This architecture is ideal for building:

* E-commerce chatbots that remember user preferences
* AI agents that manage and recall previous tasks
* Internal tools that blend AI with human decision-making
* Customer support bots that maintain conversation history across sessions

## **Why Are We Building This?**

Most chatbots today are stateless — they respond to each message in isolation and forget everything that came before. This leads to frustrating user experiences, especially in real-world scenarios where continuity matters.

We're building this system to solve that problem.

By combining **LangGraph**, **LangChain**, and **Amazon Bedrock**, we aim to create a **memory-aware, modular, and production-ready AI assistant** that can:

* **Understand the full context** of a conversation, not just the latest input
* **Remember previous interactions**, even across sessions
* **Streamline AI development** using a graph-based design that's easy to debug and extend
* **Run efficiently and securely** using Claude 3 models via Amazon's serverless Bedrock infrastructure

This project isn't just a demo — it's a blueprint for building scalable, real-world AI systems that people can actually rely on.

## 

## 

## **How We Are Doing This**

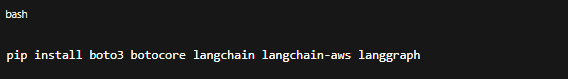
We’re combining several modern AI tools to build a smart, memory-enabled assistant:

* **Amazon Bedrock + Claude 3**: Claude is the core language model, accessed securely and serverlessly via Amazon Bedrock.
* **LangChain**: Acts as a clean interface to Claude, handling the conversation inputs and outputs.
* **LangGraph**: Structures the conversation flow as a state machine, making it modular and easy to manage.
* **Checkpointing with BedrockSessionSaver**: Stores session data so conversations can be resumed later, even after long gaps.
* **Streaming Responses**: Delivers model replies in real time, creating a smooth, live chat experience.
* **Custom Tools**: Adds plug-and-play functions the assistant can use, like searching for shoes or performing tasks.

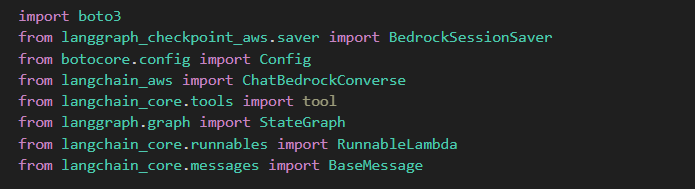
Together, this setup makes the assistant intelligent, persistent, and production-ready.

## **1. Setting Up the Environment**

:- **Install dependency–**



:- **Import dependency–**

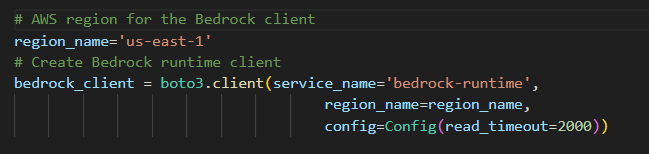


### **Explanation:**

We're importing all necessary libraries:

* boto3: To connect to AWS Bedrock.
* ChatBedrockConverse: Claude 3 wrapper from LangChain.
* StateGraph and RunnableLambda: Core parts of LangGraph — the engine behind our conversational flow.
* BedrockSessionSaver: Stores conversations as **resumable sessions**.

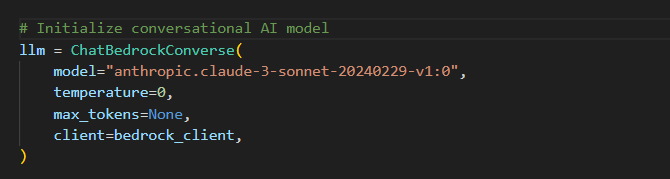
## **2. Connecting to Amazon Bedrock**



### **Explanation:**

We're telling boto3 to use **Bedrock's runtime API** in the us-east-1 region. A longer read\_timeout helps with big responses from Claude.

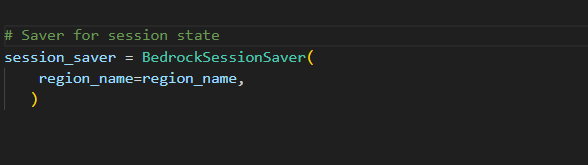
## **3. Initializing the Claude Model**



### **Explanation:**

We create a Claude 3 instance using ChatBedrockConverse. The context gives the model a system-level instruction ("You're a helpful assistant"). The temperature is 0, meaning deterministic (non-random) responses.

## **4. Saving Sessions with LangGraph Checkpointing**



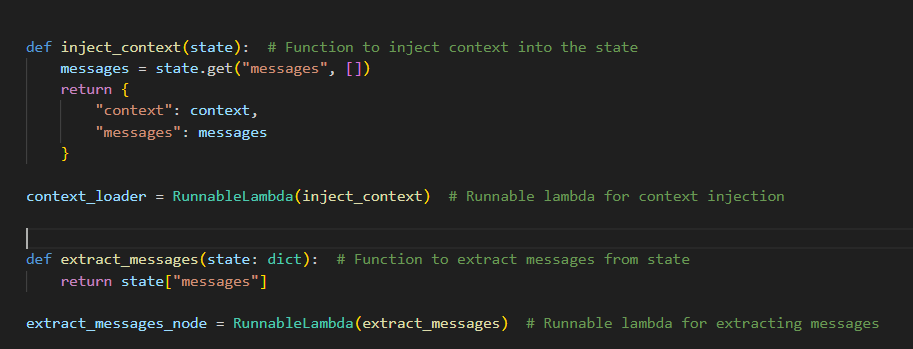
### **Explanation:**

This creates a checkpointing mechanism that stores:

* The conversation messages
* The Claude model's responses
* All graph state transitions

You’ll be able to resume a previous conversation even after a system crash or refresh.

**5. Context Injection and State Extraction**

****

### **Explanation:**

The inject\_context function adds a predefined system context along with existing chat messages to the state dictionary.

RunnableLambda(inject\_context) wraps that function so it can be used as a node in a LangGraph flow.

The extract\_messages function simply pulls out the message history from the state.

RunnableLambda(extract\_messages) also wraps this extractor function for graph use.

These functions help manage and prepare conversational data as it flows through the AI pipeline.

## 

## **6. Building the LangGraph**

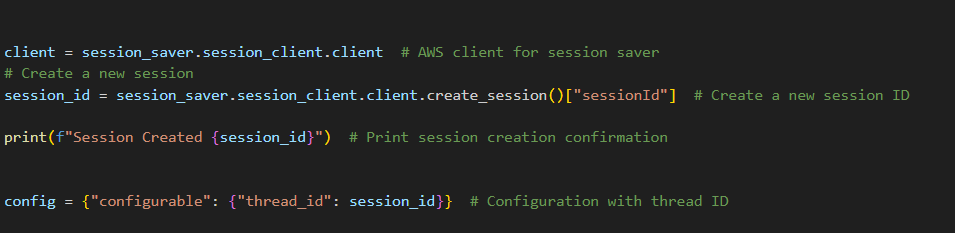
### **Explanation:**

This is where we build the **actual flow** of the chatbot:

1. load\_context: Adds the system prompt
2. extract\_messages: Pulls user messages
3. llm: Sends them to Claude

Every transition is **checkpointed** automatically.

## **7. Creating a Session**



### **Explanation:**

This creates a new **conversation session** on AWS and generates a session\_id. This ID is passed to LangGraph so that everything — messages, checkpoints — is tracked under a single thread.

## **8. Running the Conversational Loop**

### **Explanation:**

This infinite loop waits for the user’s input. If they type "quit", it ends the program.

* The user's message is added to the chat history.
* We run LangGraph, passing in both the chat\_history and config.
* The graph returns an event stream with all state changes.
* If we get a Claude message (BaseMessage), we print it and store it in the chat.

## **9. Output**

## **How Does It All Work?**

### **Session Handling**

Sessions are managed by BedrockSessionSaver, which:

* Saves conversation checkpoints at each node
* Assigns each conversation a sessionId
* Lets you **resume a conversation** later from the same point

### **Chunking (Graph Execution)**

* The LangGraph breaks down your chatbot’s logic into **nodes** and **edges**
* Each message goes through a small pipeline:  
  + Add context → Extract user message → Ask Claude → Save output
* These steps are **modular and chunked**, making debugging and replaying easy

### **Summary:**

This project implements a smart, memory-enabled conversational assistant using Amazon Bedrock, LangChain, and LangGraph. It integrates Claude 3 (Sonnet) via Bedrock to generate natural, contextual responses. The assistant maintains full conversation history using LangGraph’s state machine and session checkpointing. Nodes in the graph handle specific tasks like loading context, extracting messages, and generating replies. Sessions are saved and resumed through BedrockSessionSaver, making the assistant capable of long-running conversations. The conversation is streamed in real time, providing an interactive user experience. Overall, the code builds a scalable and extensible chatbot that remembers and evolves with each session.