Build a Question-answering Bot using Generative AI

**SPL-TF-200-MLAKAL-1 - Version 1.0.14**

© 2024 Amazon Web Services, Inc. or its affiliates. All rights reserved. This work may not be reproduced or redistributed, in whole or in part, without prior written permission from Amazon Web Services, Inc. Commercial copying, lending, or selling is prohibited. All trademarks are the property of their owners.

Note: Do not include any personal, identifying, or confidential information into the lab environment. Information entered may be visible to others.

Corrections, feedback, or other questions? Contact us at [*AWS Training and Certification*](https://support.aws.amazon.com/#/contacts/aws-training).

**Lab Overview**

In this lab, you build a chatbot that answers questions about AWS services. The lab is designed to provide you with hands-on experience deploying a large language model (LLM), integrating it with an Amazon Kendra data source, and building an Amazon Lex V2 chatbot that queries your LLM and uses retrieval augmented generation (RAG) to find answers to users’ questions. This lab will help you understand how to supplement a language model’s native abilities with additional information.

The chatbot you build has three primary components: Flan T5-XL Foundation Model, Langchain, and a Kendra index. The Flan T5-XL is a large language model hosted in Amazon SageMaker. Langchain is a framework used to ingest a Kendra index made up of AWS documentation. These documents, along with questions entered in the chatbot text window, are then passed to the Flan model which generates a response. This lab provides you with a comprehensive understanding of how to build a chatbot interface using Lex V2 and how to use various AWS services to enhance the capabilities of a language model.

OBJECTIVES

By the end of this lab, you should be able to do the following:

* Explain how retrieval augmented generation can be used to improve the output produced by Generative AI applications.
* Deploy a Lex chatbot powered by a large language model.
* Connect *Langchain* to a model launched in *Amazon SageMaker*.

**Technical knowledge prerequisites**

Familiarity with containerized applications and basic machine learning concepts is required for this lab. You should have a basic understanding of a variety of AWS services, including AWS CodeBuild, AWS Lambda, AWS Cloudformation, Amazon Kendra, and Amazon Lex. You should also have completed the *Getting Started with Amazon Kendra* course.

DURATION

This lab requires approximately *75* minutes to complete.

ICON KEY

Various icons are used throughout this lab to call attention to different types of instructions and notes. The following list explains the purpose for each icon:

* **Command:** A command that you must run.
* **Expected output:** A sample output that you can use to verify the output of a command or edited file.
* **Note:** A hint, tip, or important guidance.
* **Learn more:** Where to find more information.
* **Caution:** Information of special interest or importance (not important enough to cause problems with equipment or data if you miss it, but it could result in the need to repeat certain steps).
* **WARNING:** An action that is irreversible and could potentially impact the failure of a command or process (including warnings about configurations that cannot be changed after they are made).
* **Consider:** A moment to pause to consider how you might apply a concept in your own environment or to initiate a conversation about the topic at hand.
* **File contents:** A code block that displays the contents of a script or file that you need to run, which has been pre-created for you.
* **Task complete:** A conclusion or summary point in the lab.

**Start lab**

1. To launch the lab, at the top of the page, choose **Start lab**.

**Caution:** You must wait for the provisioned AWS services to be ready before you can continue.

1. To open the lab, choose **Open Console**.

You are automatically signed in to the AWS Management Console in a new web browser tab.

**WARNING:** **Do not change the Region unless instructed.**

COMMON SIGN-IN ERRORS

**Error: You must first sign out**



If you see the message, **You must first log out before logging into a different AWS account:**

* Choose the **click here** link.
* Close your **Amazon Web Services Sign In** web browser tab and return to your initial lab page.
* Choose **Open Console** again.

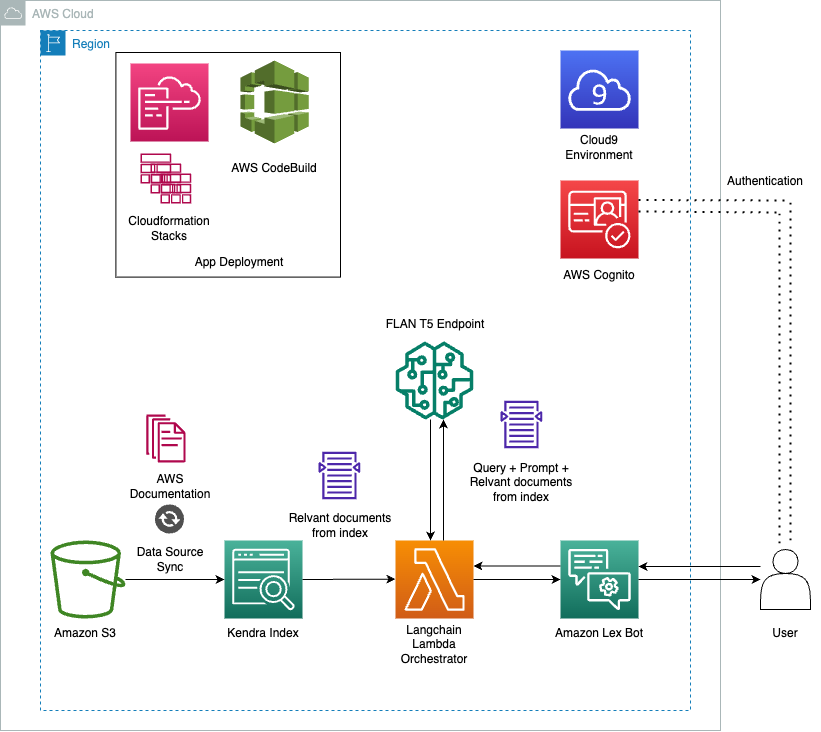
**Error: Choosing Start Lab has no effect**

In some cases, certain pop-up or script blocker web browser extensions might prevent the **Start Lab** button from working as intended. If you experience an issue starting the lab:

* Add the lab domain name to your pop-up or script blocker’s allow list or turn it off.
* Refresh the page and try again.

LAB ENVIRONMENT

The following diagram shows the basic architecture of the lab environment:



*Image description: The preceding diagram depicts a VPC containing a RAG workflow made of an Amazon Lex Bot, a Langchain AWS Lambda function, a Kendra index, and documents in an Amazon S3 bucket. The VPC also includes a deployment mechanism, consisting of Cloudformation stacks and AWS CodeBuild. Additionally, an AWS Cloud9 environment provides access to VPC resources and Amazon Cognito provides user authentication.*

The following list details the major resources in the diagram:

* All resources are contained inside of a signle *Virtual Private Cloud (VPC)*.
* A RAG workflow made of an *Amazon Lex Bot*, a Langchain *AWS Lambda* function, an *Amazon Kendra* index, and documents in an *Amazon S3* bucket. The documents in the *Amazon S3* bucket are continually synced to the index.
* *Amazon Cognito* provides user authentication.
* An *AWS Cloud9* environment provides access to *VPC* resources.
* The web app is deployed using a combination of *AWS Cloudformation* stacks and *AWS CodeBuild*.

SERVICES USED IN THIS LAB

**AWS Cloud9**

AWS Cloud9 allows you to write, run, and debug your code with just a browser. With AWS Cloud9, you have immediate access to a rich code editor, integrated debugger, and built-in terminal with preconfigured AWS CLI. You can get started in minutes and no longer have to spend the time to install local applications or configure your development machine.

**Amazon SageMaker**

Amazon SageMaker is a fully managed machine learning service. With SageMaker, data scientists and developers can quickly and easily build and train machine learning models, and then directly deploy them into a production-ready hosted environment. It provides an integrated Jupyter authoring notebook instance for easy access to your data sources for exploration and analysis, so you don’t have to manage servers.

**Amazon Kendra**

Amazon Kendra is an intelligent search service that uses natural language processing and advanced machine learning algorithms to return specific answers to search questions from your data. Unlike traditional keyword-based search, Amazon Kendra uses its semantic and contextual understanding capabilities to decide whether a document is relevant to a search query.

**Amazon Lex V2**

Amazon Lex V2 is an AWS service for building conversational interfaces for applications using voice and text. Amazon Lex V2 provides the deep functionality and flexibility of natural language understanding (NLU) and automatic speech recognition (ASR) so you can build highly engaging user experiences with lifelike, conversational interactions, and create new categories of products.

**AWS CloudFormation**

AWS CloudFormation is a service that helps you model and set up your AWS resources so that you can spend less time managing those resources and more time focusing on your applications that run in AWS. You create a template that describes all the AWS resources that you want (like Amazon EC2 instances or Amazon RDS DB instances), and CloudFormation takes care of provisioning and configuring those resources for you.

**AWS Lambda**

AWS Lambda is a compute service that lets you run code without provisioning or managing servers. Lambda runs your code on a high-availability compute infrastructure and performs all of the administration of the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, and logging.

**AWS CodeBuild**

AWS CodeBuild is a fully managed build service in the cloud. CodeBuild compiles your source code, runs unit tests, and produces artifacts that are ready to deploy. CodeBuild eliminates the need to provision, manage, and scale your own build servers. It provides prepackaged build environments for popular programming languages and build tools such as Apache Maven, Gradle, and more.

AWS SERVICES NOT USED IN THIS LAB

AWS service capabilities used in this lab are limited to what the lab requires. Expect errors when accessing other services or performing actions beyond those provided in this lab guide.

**Task 1: Locate your large language model (LLM)**

In this task, you confirm that a *Flan T5 XL* large language model has been deployed into your account and that it is ready to receive queries. The model will provide your application with the ability to create a compelling conversational AI experiences.

1. At the top of the page, in the unified search bar, search for and choose

SageMaker

.

1. In the **Amazon SageMaker** panel on the left side of the screen, under **Inference**, choose **Endpoint configurations**.

**Note:** A single endpoint configuration called *sagemaker-flan-EndpointConfig* has been deployed into your account. Endpoint configurations specify the resources and settings used to deploy models hosted at an SageMaker endpoint.

1. Choose **sagemaker-flan-EndpointConfig**.
2. Scroll down the page to the *Variants* section. Note that a production variant with a name similar to *sagemaker-flan-endpoint-XXXXXXXXXXXX-SageMakerModel* has been deployed to the endpoint. This production variant serves inference requests made to the endpoint.

A production variant is the model that is deployed and actively handling prediction requests. An endpoint can also have shadow variants, which are alternative models deployed to the same endpoint but do not receive traffic. Shadow variants allow A/B testing or canary deployments where a new model is deployed as a shadow but does not impact the existing production model. After evaluation, the shadow variant can be promoted to replace the old production model.

**Note:** The configuration includes the model artifacts, instance type, number of instances, and other configurations required to create an endpoint that can receive inference requests and provide low-latency predictions. Endpoint configurations allow you to save a standardized configuration for deploying models and easily update or swap models by creating a new endpoint that points to an existing configuration.

1. Now let’s examine the model that has been deployed into your account. Select **sagemaker-flan-endpoint-XXXXXXXXXXXX-SageMakerModel** and review the information displayed on the subsequent page.

The SageMaker model *sagemaker-flan-endpoint-XXXXXXXXXXXX-SageMakerModel* was created using a Docker image based on *Hugging Face* and *PyTorch* to serve a Transformer text-to-text model stored in S3. The model has network isolation enabled and was created by *Cloudformation* when your lab environment was built.

Now let’s confirm that your endpoint is active.

1. In the **Amazon SageMaker** panel on the left side of the screen, under **Inference**, choose **Endpoints**.

A endpoint called **QuestionAnswerBotEndpoint** appears in the *Endpoints* card. Confirm that its status is listed as  InService. If your endpoint is still creating, wait for the status to change before proceeding to the next task.

**Task complete:** You have confirmed that a pretrained large language model endpoint has been deployed into your account.

**Task 2: Update an Amazon Kendra index**

Amazon Kendra is a cloud-based enterprise search service that uses natural language processing (NLP) and machine learning to index data sources and enable searching across them. Kendra ingests content from data sources like file systems, databases, websites and applications, to create an index. Using these indexes, developers can add search capabilities to their applications that quickly and efficiently parse large quantities of data and return highly accurate results. When a user queries a Kendra index, the service uses NLP to determine the context and return the most relevant results. In some cases, this could be a precise answer, while in other situations it could be an entire document.

In this task, you examine an existing Kendra index and then add a new data source to it.

1. Open a new browser tab and, at the top of the AWS Management Console, in the search bar, search for and choose

Kendra

. Do not close the tab connected to SageMaker Studio, as you will return to it in subsequent tasks.

You are brought to the *Indexes* page, which shows that a single index called *index* has been created.

**Note:** An index holds the contents of your documents and is structured to optimize their searchability.

1. Choose **index** to view its details.

The *Getting started* section at the top of the screen shows that your index is  Active, but has not been connected to any data sources.

**Note:** A data source is a location, such as an Amazon S3 bucket, where you store the documents for indexing. You can automatically synchronize a data source with an Amazon Kendra index so that newly added, updated, or deleted documents in the data source are also added, updated, or deleted in the index.

Popular data sources that Kendra natively provides connectors for include:

* Amazon S3
* Salesforce
* ServiceNow
* Amazon Relational Database Service (RDS)
* SharePoint
* Microsoft OneDrive

In the following steps, you add a dataset made up of AWS documentation to your index.

1. Choose the **Add data sources** button.

You are brought to the *Add data source* page, which lists a large number of connectors developed by both AWS and its partners.

1. Locate the *Sample AWS documentation* card at the top of the page and choose the **Add dataset** button.

**Note:** The sample documentation is stored in an *Amazon S3* bucket and the Kendra *Amazon S3 connector* is used to crawl and index its contents.

1. In the *Name and description* section, enter the following details:

* For **Data source name**, enter

AwsDocumentation

.

* For **Description**, enter

Sample Documentation for Amazon Elastic Compute Cloud (Amazon EC2), Amazon Kendra, Amazon Simple Storage Service (Amazon S3), and AWS Lambda

.

1. Scroll down the page and confirm that the **Default language** is set to **English (en)**.
2. Choose the **Add data source** button at the bottom of the page.

**Expected output:** A banner at the bottom of the page initially indicates that your IAM role is propagating. You are then brought to the *AwsDocumentation* page while crawling and indexing completes.

1. In the *Data source details* section, locate the **Data source ID**.

The value you are looking for should look similar to this:

bee64774-a109-432a-bd31-19ed0e21143y

**Task complete:** You have successfully used the *Amazon S3 connector* to add documentation to your index.

**Task 3: Create an Amazon Lex V2 chatbot**

Now that you’ve added documentation to your Kendra index, let’s connect it to Lex V2 and create a conversational chatbot that answers questions about AWS services.

Amazon Lex V2 enables you to direct basic conversation flow. Amazon Lex V2 manages the dialog and dynamically adjusts the responses in the conversation. In this case, it will use your Kendra index to inform its responses.

1. At the top of the AWS Management Console, in the search bar, search for and choose

Lex

.

You are brought to the Lex console.

1. At the top of the *Bots* section, choose the **Create bot** button.
2. On the **Configure bot settings** page, in the **Creation method** section, select **Create a blank bot**.
3. In the **Bot configuration** section:

* For **Bot name**, enter

QuestionAnswerBot

.

* For **Description**, enter

A Lex V2 bot that queries a Kendra index to answer questions about AWS services

.

1. In the **IAM permissions** section:

* Select **Use an existing role** and then open the dropdown menu and choose **LabStack-LexServiceRole-XXXXXXXXXXXX**.

1. Continue to the next section. Under **Is use of your bot subject to the Children’s Online Privacy Protection Act (COPPA)**, select **No**.

**Note:** *COPPA* is a United States federal law, regulating the information websites can collect about children.

1. Leave the remaining fields unchanged, scroll to the bottom of the page and choose the **Next** button.
2. Your application will only be text based and operate in English, so leave default settings on the **Add language to bot** page unchanged, scroll to the bottom of the screen, and select the **Done** button.

*Amazon Lex V2* creates your bot along with a default intent called *NewIntent*, and takes you to a page to configure this intent.

**Note:** Intents are the goals that your users want to accomplish. These could be as anything from ordering flowers to booking a hotel.

In this case, your Question and Answer bot needs the following three intents:

* **AMAZON.KendraSearchIntent**: This intent searches documents in your Kendra index.
* **AMAZON.FallbackIntent**: This intent is triggered when your bot is asked something that does not match any of its intents. For example, you are configuring your bot to answer questions about AWS documentation. If a user asks it, “What is the weather in Las Vegas today?”, the bot will not find an answer in its dataset and the fallback intent will be triggered.
* **RequiredIntent**: Since your bot uses the locale *en\_US*, it must also have at least one intent that uses this locale. The *RequiredIntent* will be used to satisfy this requirement, but it will not be used to respond to questions about documentation.

Start by configuring the *RequiredIntent*.

1. On the **Intent: NewIntent** page, locate the **Intent details** section and change the name to

RequiredIntent

.

1. Scroll down the page to the **Sample utterances** section. Choose the **Plain text** button.

**Note:** Utterances are variations of phrases that you expect users to enter when they interact with your bot. Lex V2 uses these sample utterances to identify users’ input and match it to the correct intent. For example, if you were building an intent that books a flight, you might include sample utterances such as “I want to book a flight”, “help me get a flight”, or “flight from {DepartureCity} to {DestinationCity}”.

1. In the **text box**, enter

Required utterance

.

**Note:** Normally, you should provide at least 10 sample utterances for each intent, using different words and sentence structures. However, since the *RequiredIntent* won’t actually be used to respond to questions about documentation, one utterance is sufficient.

1. Choose the **Save intent** button at the bottom of the screen.
2. In the navigation panel at the left of the page, choose **Back to intents list (2)**.
3. Now, create a third intent that searches the Kendra index and returns a response. Open the **Add intent** menu button and choose **Use built-in intent**.
4. In the **Use built-in intent** popup window:

* For **Built-in intent**, choose **AMAZON.KendraSearchIntent**.
* For **Intent name**, enter

KendraSearchIntent

.

* For **Amazon Kendra index**, choose **index**.

1. Choose the **Add** button.
2. Scroll down the page and locate the **Closing response** section. Enter the following settings to configure what your bot does with the data it retrieves from your Kendra index:

* Expand the  **Response sent to the user after the intent is fulfilled** menu
* Under  **Message group**, locate the **Message** text box, and enter

((x-amz-lex:kendra-search-response-answer-1))

.

1. Expand the  **Variations** menu and enter the following strings in the text boxes that appear:

* I found an excerpt from a helpful document: ((x-amz-lex:kendra-search-response-document-1))
* I think the answer to your questions is ((x-amz-lex:kendra-search-response-answer-1))

1. At the bottom of the screen, choose the **Save intent** button.

Well done! Now that your bot has been configured with intents and responses, it’s time to build and test it.

1. At the top of the screen, choose the **Build** button.

**Expected output:** A banner at the bottom of the page indicates that the bot is building.

1. Wait for the bot to finish building and then choose the **Test** button.

A chat box appears at the side of the screen.

1. Enter the following question in the chat window to test your bot:

Explain the difference between EC2 Instance Savings Plans and Amazon EC2 Reserved Instances.

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

I found an excerpt from a helpful document: ...Reserved Instances provide you with significant savings on your Amazon EC2 costs compared to On-Demand Instance pricing. Reserved Instances are not physical instances, but rather a billing discount...

**Note:** The output from your Lex V2 bot may be different from the sample shown above.

Although the bot returned accurate information about Reserved Instances, the response was truncated and didn’t include a comparison to EC2 Savings Plans. Let’s try again with a more direct question.

1. Enter the following question in the chat window to test your bot:

What runtimes does Lambda support?

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

I think the answer to your questions is Name, Identifier, AWS SDK for Ruby, Operating system | Ruby 2.7, ruby2.7, 3.0.1, Amazon Linux 2 | Ruby 2.5, ruby2.5, 3.0.1, Amazon Linux

Based on our testing, we can see that the bot is able to retrieve information from your Kendra index. However, the bot’s ability to generate flexible responses that combine information from multiple documents is limited. More careful configuration would improve the quality of the bot’s responses, but perhaps there’s another solution.

Maybe your large language model will be better suited to these tasks and negate the need for further configuration in Lex V2?

**Task complete:** Great work! You have successfully built an Amazon Lex V2 bot that uses your Amazon Kendra index to answer questions about AWS services.

**Task 4: Implement a RAG workflow**

The predictions, inferences, and text generation produced by GenAI applications can often be improved using a technique called Retrieval Augmented Generation (RAG). RAG retrieves data from outside the language model and augments the prompts by adding the relevant retrieved data in context.

In this task, you connect to a *Cloud9* environment which you use to build a Docker container that runs *Langchain*, a framework for simplifying the creation of applications that use large language models (LLMs). This container will be used to orchestrate interactions among *Amazon Kendra*, *Amazon Lex*, and your LLM endpoint. You then push the container to *Amazon Elastic Container Registry (ECR)* and create a *Lambda* function that runs the container.

TASK 4.1: CREATE THE MODEL

Build the Docker image and push it to ECR.

1. To connect to your **AWS Cloud9 environment**, retrieve the **Cloud9Environment** link located in the panel to the left of the instructions and paste it into a new browser tab.

*AWS Cloud9* is a cloud-based integrated development environment (IDE) that you can use to write, run, and debug your code within your browser. It comes prepackaged with many tools that are commonly used in application development, including Docker, Python, and the AWS Command Line Interface (AWS CLI).

1. You do not need the **Cloud9 Welcome screen** or any of the other default tabs that appear when you first launch *AWS Cloud9*, so choose the **x** next to each tab to close them.
2. At the top of the IDE, choose the  plus icon and select **New Terminal**.

 Take a moment to familiarize yourself with the *AWS Cloud9* IDE interface.

* In the middle of the screen, a single terminal session is open in the editor. You can open multiple tabs in this window to edit files and run terminal commands.
* The file navigator appears on the left side of the screen.
* A gear icon appears on the right side of the screen. Choosing this icon opens the AWS Cloud9 Settings panel.

Your *Cloud9* environment has been bootstrapped with the files you’ll use to build a container running *Langchain*. Start by exploring the files.

1. Enter the following commands to navigate to the **builddir** directory and view the files that have been saved to it:

cd ~/environment/builddir

ls

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

AIHandler.py helpers.py ragfunctions.py requirements.txt

**Note:** The directory contains the following files:

* *AIhandler.py* is a Python script that manages interactions with an AI chatbot, handling different user intents and dispatching them to the appropriate handler functions.
* The *ragfunctions.py* script sets up and runs a question-answering system using Amazon Kendra and your FLAN endpoint. It defines functions to build the system, run it with a given prompt, and format the answer and source documents into a JSON object.
* The *helpers.py* file is a Python script that contains various helper functions to assist in the operation of a larger program, specifically for interacting with AWS services like Kendra and Sagemaker, handling user requests, and managing session attributes. It includes functions for closing sessions, incrementing counters, transforming model inputs, getting predictions from a Sagemaker model, retrieving answers from AWS Kendra, and handling AI operations.
* The *requirements.txt* file lists libraries that need to be imported.

**File contents:** For a detailed explanation of each file and its functions, expand the sections below:

**AIHandler.py**

**ragfunctions.py**

**helpers.py**

**requirements.txt**

In earlier tasks, you saw that neither the *AMAZON.KendraSearchIntent* intent nor querying your LLM endpoint directly provided the accurate, flexible, and conversational responses your application needs. Building this functionality will require enabling custom logic using a Lambda function.

Once fully deployed, when your Lex V2 bot receives a question, your Lambda function will perform the following actions:

* Retrieve relevant information from your Kendra index
* Package that information along with the original question into a prompt.
* Send the prompt to your LLM endpoint.
* Return the LLM’s response to your Lex V2 bot.

This technique known as zero-shot prompting.

**Learn more:** For more information about *zero-shot prompting*, refer to *Simplify access to internal information using Retrieval Augmented Generation and LangChain Agents* in the *Additional resources* section at the end of the lab.

1. **Command:** To set environment variables for your AWS Region and Account ID, run the following command:

TOKEN=$(curl --request PUT "http://169.254.169.254/latest/api/token" --header "X-aws-ec2-metadata-token-ttl-seconds: 3600")

export AWS\_REGION=$(curl -s http://169.254.169.254/latest/meta-data/placement/region --header "X-aws-ec2-metadata-token: $TOKEN")

export ACCOUNT\_ID=$(aws sts get-caller-identity --query Account --output text)

echo "Your AWS REGION is $AWS\_REGION and your ACCOUNT ID is $ACCOUNT\_ID"

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* EXAMPLE OUTPUT \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Your AWS REGION is us-east-1 and your ACCOUNT ID is 123456789101

1. **Command:** Enter the following command to save the ID of your Kendra index to an environment variable:

export KENDRA\_INDEX=$(aws kendra list-indices | jq -r '.IndexConfigurationSummaryItems[].Id')

echo "Your Kendra index ID is $KENDRA\_INDEX"

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Your Kendra index ID is d558e4e5-0c8a-4ab7-ad86-5c62b6de3809

1. **Command:** Enter the following command to save the name of your FLAN T5 XL endpoint to an environment variable:

export FLAN\_ENDPOINT=$(aws sagemaker list-endpoints | jq -r '.Endpoints[].EndpointName')

echo "Your Large Language Model endpoint name is $FLAN\_ENDPOINT"

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Your Large Language Model endpoint name is jumpstart-dft-QuestionAnswerBotEndpoint

1. **Command:** Enter the following command to create a Dockerfile for the Lambda function:

cat << EOF > ~/environment/builddir/Dockerfile

FROM public.ecr.aws/lambda/python:3.11

LABEL maintainer="dev@example.com"

# Copy function code

COPY ragfunctions.py \${LAMBDA\_TASK\_ROOT}

COPY AIHandler.py \${LAMBDA\_TASK\_ROOT}

COPY helpers.py \${LAMBDA\_TASK\_ROOT}

COPY requirements.txt \${LAMBDA\_TASK\_ROOT}

# Install the function's dependencies using file requirements.txt

# from your project folder.

RUN yum install -y gcc-c++

RUN pip3 install -r requirements.txt -t "\${LAMBDA\_TASK\_ROOT}"

ENV AWS\_REGION=$AWS\_REGION

ENV KENDRA\_INDEX\_ID=$KENDRA\_INDEX

ENV FLAN\_XL\_ENDPOINT=$FLAN\_ENDPOINT

# Set the CMD to your handler (could also be done as a parameter override outside of the Dockerfile)

CMD [ "AIHandler.lambda\_handler" ]

EOF

**Expected output:**

*None, unless there is an error.*

**File contents:** The following list provides a high-level overview of instructions included in the Dockerfile:

* Copies *ragfunctions.py*, *AIHandler.py*, *helpers.py*, and *requirements.txt* from the current directory to the *${LAMBDA\_TASK\_ROOT}*, which is */var/task* by default.
* Runs the command

pip3 install -r requirements.txt -t "${LAMBDA\_TASK\_ROOT}"

 inside the image, which installs the dependencies listed in *requirements.txt* to the Lambda root directory.

* Uses the *$AWS\_REGION*, *$KENDRA\_INDEX*, *$FLAN\_ENDPOINT* variables that you set to define a environment variables.
* *CMD [ “AIHandler.lambda\_handler” ]* sets the default command to run when the container is started, which is to invoke the *lambda\_handler* function in the module named *AIHandler*.

1. **Command:** Enter the following command to build your Docker image:

docker build . -t rag-kendra-llm-lex

**Caution:** You must be in the **/home/ec2-user/environment/builddir** directory when you build the Docker image.

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

...

Step 8/11 : ARG build\_AWS\_region="us-east-1"

---> Running in 73d8ba0e24a4

Removing intermediate container 73d8ba0e24a4

---> 4f24a28015b5

Step 9/11 : ARG build\_Kendra\_Index\_ID="d558e4e5-0c8a-4ab7-ad86-5c62b6de3809"

---> Running in 82d208ee02f5

Removing intermediate container 82d208ee02f5

---> 6aa705ec08d6

Step 10/11 : ARG build\_FLAN\_XL\_ENDPOINT="jumpstart-dft-QuestionAnswerBotEndpoint"

---> Running in 42a9fd9988d6

Removing intermediate container 42a9fd9988d6

---> d2c8c50a956c

Step 11/11 : CMD [ "AIHandler.lambda\_handler" ]

---> Running in 658ee09f0c0c

Removing intermediate container 658ee09f0c0c

---> 427ad1ba5983

Successfully built 427ad1ba5983

Successfully tagged rag-kendra-llm-lex:latest

Now that you’ve built your image, it needs to be saved to ECR.

1. **Command:** Enter the following command to create a new ECR repository for your image:

aws ecr create-repository --repository-name rag-kendra-llm-lex --image-scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

{

"repository": {

"repositoryArn": "arn:aws:ecr:us-east-1:547029689126:repository/rag-kendra-llm-lex",

"registryId": "547029689126",

"repositoryName": "rag-kendra-llm-lex",

"repositoryUri": "547029689126.dkr.ecr.us-east-1.amazonaws.com/rag-kendra-llm-lex",

"createdAt": "2023-09-18T19:53:36+00:00",

"imageTagMutability": "MUTABLE",

"imageScanningConfiguration": {

"scanOnPush": true

},

"encryptionConfiguration": {

"encryptionType": "AES256"

}

}

}

1. **Command:** Now, tag your image and push it to the newly created repository:

aws ecr get-login-password --region $AWS\_REGION | docker login --username AWS --password-stdin $ACCOUNT\_ID.dkr.ecr.$AWS\_REGION.amazonaws.com

docker tag rag-kendra-llm-lex:latest $ACCOUNT\_ID.dkr.ecr.$AWS\_REGION.amazonaws.com/rag-kendra-llm-lex:latest

docker push $ACCOUNT\_ID.dkr.ecr.$AWS\_REGION.amazonaws.com/rag-kendra-llm-lex:latest

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

...

The push refers to repository [547029689126.dkr.ecr.us-east-1.amazonaws.com/rag-kendra-llm-lex]

ffbd120e3b80: Pushed

93f89337569c: Pushed

5e2478cbb8a9: Pushed

e231c9a36876: Pushed

e9ef755f7689: Pushed

dbe68323edb6: Pushed

18a5b8654b12: Pushed

15dd6c63f3a2: Pushed

8c6bb6395809: Pushed

39cc5c97da73: Pushed

02c24900bf38: Pushed

latest: digest: sha256:927f9f9cbc113f02e11732f8cd2a73a250e0e236cb6e05f757799504eefd1c31 size: 2624

TASK 4.2: DEPLOY THE IMAGE AS A LAMBDA FUNCTION

Now that you’ve pushed the image to ECR, it’s time to use it in a *Lambda* function.

1. Return to the the AWS Management Console and in the search bar at the top of the screen, search for and choose

Lambda

.

1. Choose the **Create function** button.
2. At the top of the **Create function** screen, select **Container image**.
3. In the **Basic information** section, configure the following settings:

* For **Function name**, enter

RAGKendraLLMLexOrchestrator

.

* For **Container image URI**, choose the **Browse images** button. In the popup window, open the **Amazon ECR image repository** menu and choose **rag-kendra-llm-lex**. Then select the **radio button** next to **latest** and choose the **Select image** button.

1. Leave the remaining options unchanged, scroll to the bottom of the page and choose **Create function**.

You are brought to the **RAGKendraLLMLexOrchestrator** page.

1. Scroll down the screen and choose the **Configuration** tab.
2. In the **General configuration** section, select the **Edit** button.
3. Scroll down the page and locate the Lambda function **Timeout**. Increase this value to

1

 minute.

**Note:** Performing text generation is a compute-intensive operation and it may take a while for your endpoint to process and respond to some queries. In a production-grade deployment, it may make sense to select a larger instance type to host your LLM.

1. At the bottom of the screen, select the **Save** button.

The default role created for your Lambda function do not include all of the permissions it requires. In the following steps, you enhance these permissions by attaching an additional managed policy to the role.

1. At the top of the AWS Management Console, in the search bar, search for and choose

IAM

.

You are brought to the **IAM Dashboard**.

1. In the navigation pane at the left of the page, under **Identity and Access Management (IAM)**, choose **Access management**, **Roles**.
2. In the **search bar** enter

RAGKendraLLMLexOrchestrator

 to locate the role created for your function.

1. Choose the role.
2. Scroll down the page to the **Permissions policies** section, open the **Add permissions** button, and select **Attach policies**.
3. In the **search bar** enter

WebAppPolicy

.

The search results return a single policy whose name is similar to *LabStack-awsstudent-tA9Mv9UyvgkfJeTxbGDUmG-0-WebAppPolicy-kxY1QvEJeSH7*.

1. Select the checkbox next to the policy and then choose the **Add permissions** button at the bottom of the screen.

**Task complete:** Nice job! You have successfully deployed Langchain as a Lambda function. Now it’s time to build a web app that deploys your LLM-powered Lex V2 bot.

**Task 5: Deploy a web app with Cloudformation**

In this task, you deploy a second Lex V2 bot that has been pre-configured for use with a web user interface. You then use Cloudformation to deploy a CodeBuild project that builds and deploys the web application.

TASK 5.1: IMPORT A LEX V2 BOT

In order for your Lex V2 bot to connect to the Lambda function you created, it needs to be configured with dialog code hooks. Rather than updating your existing bot, in the following steps, you import a bot that has been pre-configured with code hooks.

**Learn more:** For more information about using *dialog code hooks* with Lex V2 bots, refer to *Invoke dialog code hook* in the *Additional resources* section at the end of the lab.

1. At the top of the AWS Management Console, in the search bar, search for and choose

S3

.

1. Locate and select the bucket whose name starts with **lex-json-config**.

The bucket contains a single file called *LexConfig.zip*. This file defines the configuration for the Lex V2 bot that will connect to your web app.

1. Select the **checkbox** next to **LexConfig.zip**, choose the **Download** button, and save the configuration file to your local machine.
2. Now that you have the configuration file, return to the **Lex** console. At the top of the AWS Management Console, in the search bar, search for and choose

Lex

.

1. In the **Bots** section on the Lex console, open the **Action** dropdown menu and select **Import**.
2. In the **Input file** section:

* For **Bot name**, enter

WebAppBot

.

* Choose the **Browse file** button, navigate to the **LexConfig.zip** file, and upload it.

1. In the **IAM permissions** section, choose **Create a role with basic Amazon Lex permissions**.

**WARNING:** Do not select the role you used for the bot you built earlier in this lab. That role does not include permissions enabling dialog code hooks.

1. Continue to the next section. Under **Is use of your bot subject to the Children’s Online Privacy Protection Act (COPPA)**, select **No**.
2. Leave the remaining fields unchanged, scroll to the bottom of the page and choose the **Import** button.
3. Wait for the bot to finish importing and then choose **WebAppBot**.
4. In the **Bot details** page, locate and copy the value next to **ID**. You will use this value to map the bot you just imported to your web app.

The value you are looking for should look similar to this:

V5R1NOLHRM

TASK 5.2: BUILD THE WEB APP IN CLOUDFORMATION

Now that the bot has been imported, it’s time to build the web app.

1. At the top of the AWS Management Console, in the search bar, search for and choose

Cloudformation

.

1. On the **Stacks** page, open the **Create stack** button and select **With new resources (standard)**.

The web app is defined in a series of nested stacks that have been saved to an S3 bucket. In the following steps, you deploy the root stack, which is configured to deploy the nested stacks as resources.

1. In the panel to the left of these instructions, you will find a series of values. Locate and copy the URL labeled **RootStack** and paste it into the **Amazon S3 URL** field in the **Specify template** section of the screen.
2. Choose the **Next** button.

You are brought to the the **Specify stack details** page.

1. Enter

LexWebApp

 in the **Stack name** field.

The *Parameters* section of the screen is divided into two section, the first of which is labeled **Update the following parameters**. Only update the parameters in this section.

**Note:** *Cloudformation* Parameters are used to declare values that can be passed to the template when you create a stack. A parameter is an effective way to specify sensitive or dynamic information, such as user names and resources identifiers, that cannot be stored in the template itself.

1. In the **Parameters** section of the screen, locate the **LexV2BotId** field and enter the value you saved when you imported your bot. If you did not save the ID:

* Navigate to the **Lex** console.
* Select **WebAppBot**.
* In the **Bot details** page, locate and copy the value next to **ID**.

1. Copy the **AccountID** value that is listed to the left of these instructions, and then paste it into the **AccountID parameter field**.
2. Your bot needs an initial question that it will send to the the LLM to verify that it is working. In the **WebAppConfBotInitialUtterance** field, enter:

What is AWS Lambda?

**WARNING:** Do not change any other parameters, as these may cause the app deployment to fail.

1. Scroll to the bottom of the page and select the **Next** button.
2. Leave the default settings on the **Configure stack options** page and choose the **Next** button at the bottom of the screen.
3. On the **Review LexWebApp** page, scroll to the bottom of the screen and select the **two checkboxes** in the **Capabilities** section. These checkboxes note your acknowledgement that Cloudformation may create new IAM resources and that the service requires *CAPABILITY\_AUTO\_EXPAND* permissions.
4. Choose the **Submit** button to build your web application.

**Note:** The following stacks are deployed in *Cloudformation*:

* *LexWebApp*: This stack deploys a web user interface for your Amazon Lex web app. The stack creates resources such as S3 buckets to host the web application, and calls nested stacks that manage a CodeBuild project, Amazon Cognito resources, CloudWatch Logs groups, and associated IAM roles.
* *LexWebApp-CognitoIdentityPool*: This stack builds a Cognito User Pool and Identity Pool for user authentication and identity management. It also sets up IAM roles for authenticated and unauthenticated users, with policies that allow or deny access to Amazon Lex.
* *LexWebApp-CodeBuildDeploy*: This stack builds a CodeBuild project that configures and deploys the chatbot user interface (UI) for Amazon Lex. The stack defines parameters for the CodeBuild project, S3 buckets for source code and logs, and various settings for the chatbot UI.
* *LexWebApp-CognitoIdentityPoolConfig*: This stack builds a system that updates a Cognito user pool client with a domain and app configuration. It uses AWS Lambda functions and also sets up IAM roles for them.

1. Wait for the root stack as well as the 3 nested stacks to finished building. This process should take approximately 10 minutes.

Next, you confirm that the **CodeBuild** project deployed in your **Cloudformation** templates has finished building.

1. At the top of the AWS Management Console, in the search bar, search for and choose

CodeBuild

.

You are brought to the **Build projects** page.

1. Choose **lex-web-ui-kk-RAG-Test**.
2. Scroll to the bottom of the page and choose the **Build history** tab.
3. Confirm that two build runs are listed and that the status of both of them is  Succeeded.

**WARNING:** Do not proceed to the next step both build runs have completed.

TASK 5.3: CONNECT THE WEB APP TO A LEX BOT ALIAS

Now that the web app has been deployed, it needs to be connected to your Lex V2 bot. However, rather than pointing it at the primary version of your bot, you point it at its alias.

**Note:** An alias is a pointer to a specific version of a bot or bot network. With an alias, you can easily update the version that your client applications are using. For example, you can point an alias to version 1 of your bot. When you are ready to update the bot, you publish version 2 and change the alias to point to the new version. Because your applications use the alias instead of a specific version, all of your clients get the new functionality without needing to be updated.

1. At the top of the AWS Management Console, in the search bar, search for and choose

Lex

.

1. From the **Bots** section, choose **WebAppBot**.
2. Scroll down the **WebAppBot** page and locate the **Create versions and aliases for deployment** section. Choose the **View aliases** button.

One alias called **TestBotAlias** has already been created.

1. Select **TestBotAlias**.
2. Scroll down the page to the **Languages** section and choose **English (US)**.

You are brought to the **Alias language support: English (US)** page. This page enables you to map your alias to the *RAGKendraLLMLexOrchestrator* that you created in the previous task. The Lambda function will effectively replace your bot’s default intent.

1. Open the **Source** menu and select **RAGKendraLLMLexOrchestrator**.
2. Verify that the **Lambda function version or alias** is set to **$LATEST** and choose the **Save** button.

TASK 5.4: CONNECT TO THE LLM-POWERED CHATBOT

Now that you’ve finished configuring it, it’s time to build and test your bot.

1. In the **Amazon Lex** navigation panel on the left side of the screen, in the **WebAppBot** section, choose **English**.
2. Choose the **Build** button at the top of the screen.
3. Wait for the bot to finish building and then use the search bar to navigate back to the

Cloudformation

 console.

1. Choose **LexWebApp**.
2. Open the **Outputs** tab, scroll to the bottom of the screen, and choose the **URL** next to **WebAppUrl** to launch your web app.

The web app launches and a prompt is immediately sent to your LLM endpoint asking “What is AWS Lambda?” The bot responds with “AWS Lambda is a compute service that lets you run code without provisioning or managing servers.” This looks promising!

**Note:** If the web app fails to answer the question, wait a couple of minutes and then refresh the page. Alternatively, you may choose to proceed to the next step and ask a different question.

1. Let’s see how your web app handles a more complicated question. In the text field at the bottom of the screen, enter:

Explain the difference between EC2 Instance Savings Plans and Amazon EC2 Reserved Instances.

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Reserved Instances are not physical instances, but rather a billing discount applied to the use of On-Demand Instances in your account. These On-Demand Instances must match certain attributes, such as instance type and Region, in order to benefit from the billing discount. Savings Plans also offer significant savings on your Amazon EC2 costs compared to On-Demand Instance pricing. With Savings Plans, you make a commitment to a consistent usage amount, measured in USD per hour. This provides you with the flexibility to use the instance configurations that best meet your needs and continue to save money, instead of making a commitment to a specific instance configuration.

**Note:** The web app generates a text response that accurately compares EC2 Instance Savings Plans and Amazon EC2 Reserved Instances.

1. Recall that when you asked Lex what runtimes AWS Lambda supports, its response was incomplete. Let’s see how the LLM handles the same question now that your RAG function has supplemented its knowledgebase. Enter:

What runtimes does Lambda support?

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

AWS Lambda supports multiple languages through the use of runtimes.

**Note:** This time, the output is accurate - demonstrating that the app has used the documentation in the Kendra index to generate its response.

1. Let’s run one final test. When you created the *WebAppBot* in *Lex V2*, you only configured it in English. Similarly, you only added English documentation to your *Kendra* index. Despite these constraints, let’s see if your bot bot can answer a question in another language. Enter the following text to ask it what Amazon S3 is in French:

Qu’est-ce qu’Amazon S3 ?

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Amazon Simple Storage Service

**Note:** The app correctly interpreted the question, even though it was asked in a different language, and output a correct answer.

**Task complete:** You have successfully built and deployed a chatbot that uses an LLM to respond to questions about AWS services.

**Conclusion**

 Congratulations! You now have successfully:

* Explained how retrieval augmented generation can be used to improve the output produced by Generative AI applications.
* Deployed a Lex chatbot powered by a large language model.
* Connected *Langchain* to a model launched in *Amazon SageMaker*.

**End lab**

Follow these steps to close the console and end your lab.

1. Return to the **AWS Management Console**.
2. At the upper-right corner of the page, choose **AWSLabsUser**, and then choose **Sign out**.
3. Choose **End lab** and then confirm that you want to end your lab.

**Additional Resources**

* [Easily get started with machine learning using Amazon SageMaker JumpStart](https://pages.awscloud.com/Easily-get-started-with-machine-learning-using-Amazon-SageMaker-JumpStart_2022_VW_s44e03-MCL_OD.html)
* [FLAN-T5](https://huggingface.co/docs/transformers/model_doc/flan-t5)
* [Simplify access to internal information using Retrieval Augmented Generation and LangChain Agents](https://aws.amazon.com/blogs/machine-learning/simplify-access-to-internal-information-using-retrieval-augmented-generation-and-langchain-agents/)
* [Invoke dialog code hook](https://docs.aws.amazon.com/lexv2/latest/dg/paths-code-hook.html)

For more information about AWS Training and Certification, see [*https://aws.amazon.com/training/*](https://aws.amazon.com/training/).

*Your feedback is welcome and appreciated.*  
If you would like to share any feedback, suggestions, or corrections, please provide the details in our [*AWS Training and Certification Contact Form*](https://support.aws.amazon.com/#/contacts/aws-training).