Implementation guide

QnABot on AWS

QnABot on AWS: Implementation guide

Table of Contents

Create a custom question and answer chatbot

The QnABot on AWS solution is a generative AI-enabled multi-channel, multi-language conversational chatbot that responds to your customer's questions, answers, and feedback. It is built on Amazon Lex, Amazon Polly, Amazon OpenSearch Service, Amazon Translate, Amazon Comprehend, Amazon Kendra, and Amazon Bedrock. This solution helps you to quickly deploy self-service conversational artificial intelligence (AI) on multiple channels, including your contact centers, websites, social media channels, SMS text messaging, or Amazon Alexa without programming.

This implementation guide provides an overview of the QnABot on AWS solution, its reference architecture and components, considerations for planning the deployment, configuration steps for deploying the solution to the Amazon Web Services (AWS) Cloud. It also includes a user's guide with prescriptive guidance for using QnABot on AWS.

Use this navigation table to quickly find answers to these questions:

If you want to . . .	Read . . .
Know the cost for running this solution	Cost
Understand the security considerations for this solution	Security
Know how to plan for quotas for this solution	Quotas
Know which AWS Regions are supported for this solution	Supported AWS Regions
View or download the AWS CloudFormation template included in this solution to automatically deploy the infrastructure resources (the "stack") for this solution	AWS CloudFormation template
Access the source code and optionally use the AWS Cloud Development Kit (AWS CDK) (AWS CDK) to deploy the solution	GitHub repository

Use cases

Contact centers - How can I help?

Virtual agents to automatically help resolve customer questions or guide customers to the right agent.

Informational bots - Can I answer your question?

Chatbots for everyday requests and frequently asked questions.

Enterprise productivity bots - Can I help you get more done?

Streamline internal enterprise work activities and enhance productivity.

Features and benefits

With the solution's content management environment and the Contact Center Integration wizard, you can set up and customize an environment that provides the following benefits:

- Enhance your customer's experience by providing personalized tutorials and question and answer support with intelligent multi-part interaction.
- Uncover insights and business trends.
- Reduce call center wait times by automating customer support workflows.
- Expand existing channels and grow new ones.
- Implement the latest machine learning (ML) technology to create engaging, human-like interactions for chatbots.
- Reduce customer support costs.

QnABot on AWS provides the following features:

High quality speech recognition and natural language understanding (NLU)

This solution uses automatic speech recognition (ASR) and NLU technologies to create a Speech Language Understanding (SLU) system with Amazon Lex. Amazon Lex uses the same proven technology that powers Alexa. Amazon Lex is able to learn the multiple ways users can express their intent based on a few sample utterances provided by the developer. The SLU system takes

natural language speech and text input, understands the intent behind the input, and fulfills the user intent by invoking the appropriate response.

Context management

As the conversation develops, being able to accurately classify utterances requires managing context across multi-turn conversations. Amazon Lex supports context management natively, so you can manage the context directly without the need for custom code. As the initial prerequisite intents are filled, you can create `"contexts"` to invoke related intents. This simplifies bot design and expedites the creation of conversational experiences.

Generative responses

Integration with the various large language models (LLMs) hosted on Amazon Bedrock allows QnABot to:

- Disambiguate customer questions by considering conversational context
- Dynamically generate answers from relevant FAQs, Amazon Kendra search results, and Amazon Bedrock knowledge bases
- Ask questions and summarize data from a single uploaded document

Generated responses reduce the number of FAQs you must maintain because the solution synthesizes concise answers from existing documents. You can customize responses to be short, concise, and suitable for voice channel contact center bots as well as website text bots. Text generation is fully compatible with this solution's multi-language support, allowing users to interact in their chosen languages and receive generated answers in the same language.

> **Note**
>
> By choosing to use the generative responses features, you acknowledge that QnABot on AWS engages third-party generative AI models that AWS does not own or otherwise has any control over (`"Third-Party Generative AI Models"`). Your use of the Third-Party Generative AI Models is governed by the terms provided to you by the Third-Party Generative AI Model providers when you acquired your license to use them (for example, their terms of service, license agreement, acceptable use policy, and privacy policy). You are responsible for ensuring that your use of the Third-Party Generative AI Models comply with the terms governing them, and any laws, rules, regulations, policies, or standards that apply to you.

You are also responsible for making your own independent assessment of the Third-Party Generative AI Models that you use, including their outputs and how Third-Party Generative AI Model providers use any data that may be transmitted to them based on your deployment configuration.
AWS does not make any representations, warranties, or guarantees regarding the Third-Party Generative AI Models, which are "Third-Party Content" under your agreement with AWS. QnABot on AWS is offered to you as "AWS Content" under your agreement with AWS.

8 kHz telephony audio support

This solution uses high fidelity with telephone speech interactions, such as through a contact center application or helpdesk. This feature leverages the Amazon Lex speech recognition engine, which has been trained on telephony audio (8 kHz sampling rate).

Multi-turn dialog

After the solution identifies an intent, it prompts users for information that is required for the intent to be fulfilled (for example, if "Book hotel" is the intent, then the user is prompted for the location, check-in date, number of nights, etc.). QnABot on AWS gives you an easy way to build multi-turn conversations for your chatbots. You simply list the slots/parameters you want to collect from your bot users, as well as the corresponding prompts, and the Amazon Lex component takes care of orchestrating the dialogue by prompting for the appropriate slot.

Early implementation of intent and slot matching

This new capability supports creating dedicated custom Intents for a QnABot Item ID. You can extend QnABot to support one or more related intents. For example, you might create an intent that makes a car reservation, or assists an agent during a live chat or call (via Amazon Connect). For more details, see the Intent and slot matching section in the GitHub repository.

Custom domain names in QnABot content designer and QnABot client

This solution supports using custom domain names for QnABot content designer and client interfaces. For more details, see the Set up custom domain name for QnABot content designer and client section in the GitHub repository.

Importing and exporting questions and answers using CLI

You can import and export questions and answers using the AWS QnABot Command Line Interface (CLI) command line. For more details, see the AWS QnABot Command Line Interface (CLI) section in the GitHub repository.

Support for the Amazon Kendra Redirect feature

With the Amazon Kendra Redirect feature, you can now include an Amazon Kendra query within an Item ID. For more details, see the Amazon Kendra Redirect section in the GitHub repository.

Enhanced functionality for Excel

This solution supports importing QnABot questions and answers from an Excel file when uploaded to the Amazon Simple Storage Service (Amazon S3) data folder, as well as support for importing session attributes via Excel.

Concepts and definitions

The following terms are specific to this document:

fulfillment

The process of performing actions based on user requests. It involves taking the information gathered during the conversation and performing relevant tasks or providing appropriate responses. For instance, Alexa uses fulfillment processes to run tasks such as setting reminders, playing music, or providing weather updates.

intent

An action in response to user input in natural language. An intent represents the main purpose or goal behind a user's query. It captures what the user is trying to accomplish when interacting with a chatbot. Intents are the building blocks that empower chatbots to understand and respond effectively to user queries. For instance, if a user asks, *"Show me the weather in Houston, Texas,"* the intent behind their query is to "get the weather information". If the user says, *"Is it going to rain today?"* or *"What's the weather like today?"* , the chatbot should be able to understand that both these *utterances* have the same *intent*, which is to get the weather information.

slot

Slots are placeholders for specific pieces of information. For example, in the query *"Book a flight from New York to Los Angeles,"* the slots are *"departure city"* (New York)

and *"destination city"* (Los Angeles). The slots are the specific input data extracted from the user's utterance and needed to fulfill the intent. Slot filling helps extract relevant entities from the user's input.

token

A token is the smallest unit into which text data can be broken down for an AI model to process. It is similar to how we might break sentences into words or characters. For AI, especially in the context of language models, tokens can represent a character, a word, or even larger chunks of text, such as phrases, depending on the model and its configuration. Tokens are used to characterize different models. For example, the more questions a user asks the chat bot, the more it will cost because more tokens are processed.

utterance

Utterances are simply anything a user says to a chatbot or virtual assistant. These could be in the form of text input, voice commands, or any other form of user input. For instance, if a user types *"Show me the weather in Houston, Texas"* the entire sentence is the utterance.

> **Note**
>
> For a general reference of AWS terms, see the AWS Glossary.

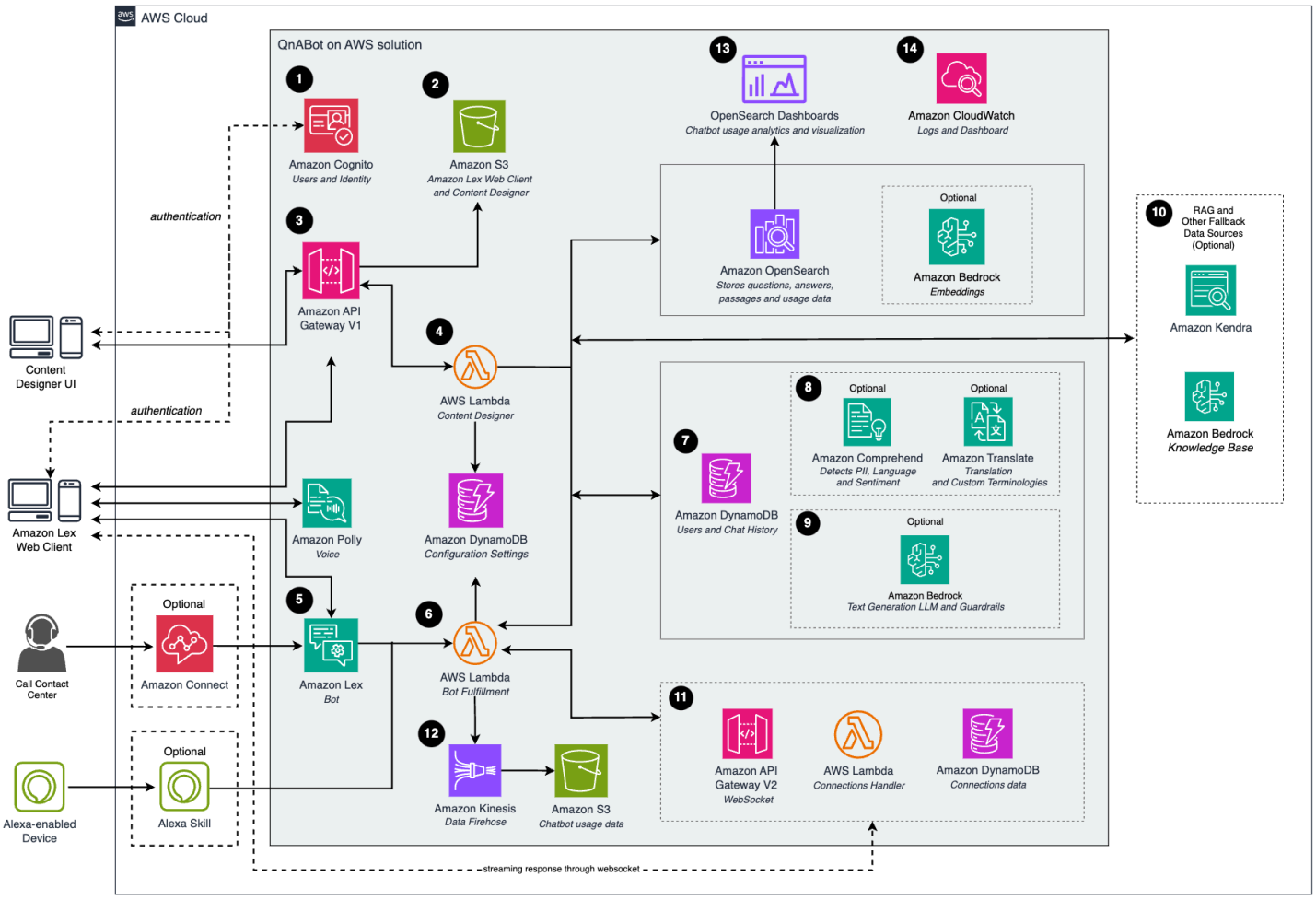
Architecture overview

This section provides a reference implementation architecture diagram for the components deployed with this solution.

Architecture diagram

Deploying this solution with the default parameters deploys the following components in your AWS account (components with dotted line border are optional).

QnABot on AWS architecture on AWS



The high-level process flow for the solution components deployed with the AWS CloudFormation template is as follows:

1. The admin deploys the solution into their AWS account, opens the Content Designer UI or Amazon Lex web client, and uses Amazon Cognito to authenticate.
2. After authentication, Amazon API Gateway and Amazon S3 deliver the contents of the Content Designer UI.
3. The admin configures questions and answers in the Content Designer and the UI sends requests to Amazon API Gateway to save the questions and answers.
4. The `Content Designer` AWS Lambda function saves the input in Amazon OpenSearch Service in a questions bank index. If using text embeddings, these requests will first pass through a LLM model hosted on Amazon Bedrock to generate embeddings before being saved into the question bank on OpenSearch. In addition, the `Content Designer` saves default and custom configuration settings in Amazon DynamoDB.
5. Users of the chatbot interact with Amazon Lex via the web client UI, Amazon Alexa or Amazon Connect.
6. Amazon Lex forwards requests to the `Bot Fulfillment` AWS Lambda function. Users can also send requests to this Lambda function via Amazon Alexa devices. ***NOTE:*** When streaming is enabled, the chat client uses Amazon Lex sessionId to establish WebSocket connections through API Gateway V2.
7. The user and chat information is stored in Amazon DynamoDB to disambiguate follow up questions from previous question and answer context.
8. Amazon Comprehend and Amazon Translate (if necessary) are used by the `Bot Fulfillment` AWS Lambda function to translate non-native Language requests to the native Language selected by the user during the deployment and look up the answer in Amazon OpenSearch Service.
9. If using LLM features such as text generation and text embeddings, these requests will first pass through various foundational models hosted on Amazon Bedrock to generate the search query and embeddings to compare with those saved in the question bank on OpenSearch.
    a. If pre-processing guardrails are enabled, they scan and block potentially harmful user inputs before they reach the QnABot application. This acts as the first line of defense to prevent malicious or inappropriate queries from being processed.
    b. If using Bedrock guardrails for LLMs or Knowledge Base, it can apply contextual guarding and safety controls during LLM inference to ensure appropriate answer generation.
    c. If post-processing guardrails are enabled, they scan, mask, or block potentially harmful content in the final responses before they are sent to the client through the fulfillment

Lambda. This serves as the last line of defense to ensure that sensitive information (like PII) is properly masked and inappropriate content is blocked.

10. If no match is returned from the OpenSearch question bank or text passages, then the Bot fulfillment Lambda function forwards the request as follows:

    a. If an Amazon Kendra index is configured for fallback, then the `Bot Fulfillment` AWS Lambda function forwards the request to Kendra if no match is returned from the OpenSearch question bank. The text generation LLM can optionally be used to create the search query and to synthesize a response from the returned document excerpts.

    b. If a Bedrock Knowledge Base ID is configured, then the `Bot Fulfillment` AWS Lambda function forwards the request to the Bedrock Knowledge Base. The `Bot Fulfillment` AWS Lambda function leverages the RetrieveAndGenerate or RetrieveAndGenerateStream APIs to fetch the relevant results for an user's query, augment the foundational model's prompt and return the response.

11. When streaming is enabled, RAG-enhanced LLM responses from text passages or external data sources is streamed via WebSocket connection using same Lex sessionId, while the final response is processed through the fulfillment Lambda.

12. User interactions with the `Bot Fulfillment` function generate logs and metrics data, which is sent to Amazon Kinesis DataFirehose then to Amazon S3 for later data analysis. The OpenSearch Dashboards can be used to view usage history, logged utterances, no hits utterances, positive user feedback, and negative user feedback and also provides the ability to create custom reports.

13. The OpenSearch Dashboards can be used to view usage history, logged utterances, no hits utterances, positive user feedback, and negative user feedback, and also provides the ability to create custom reports.

14. Using Amazon CloudWatch, the admins can monitor service logs and use the CloudWatch dashboard created by QnABot to monitor deployment's operational health.

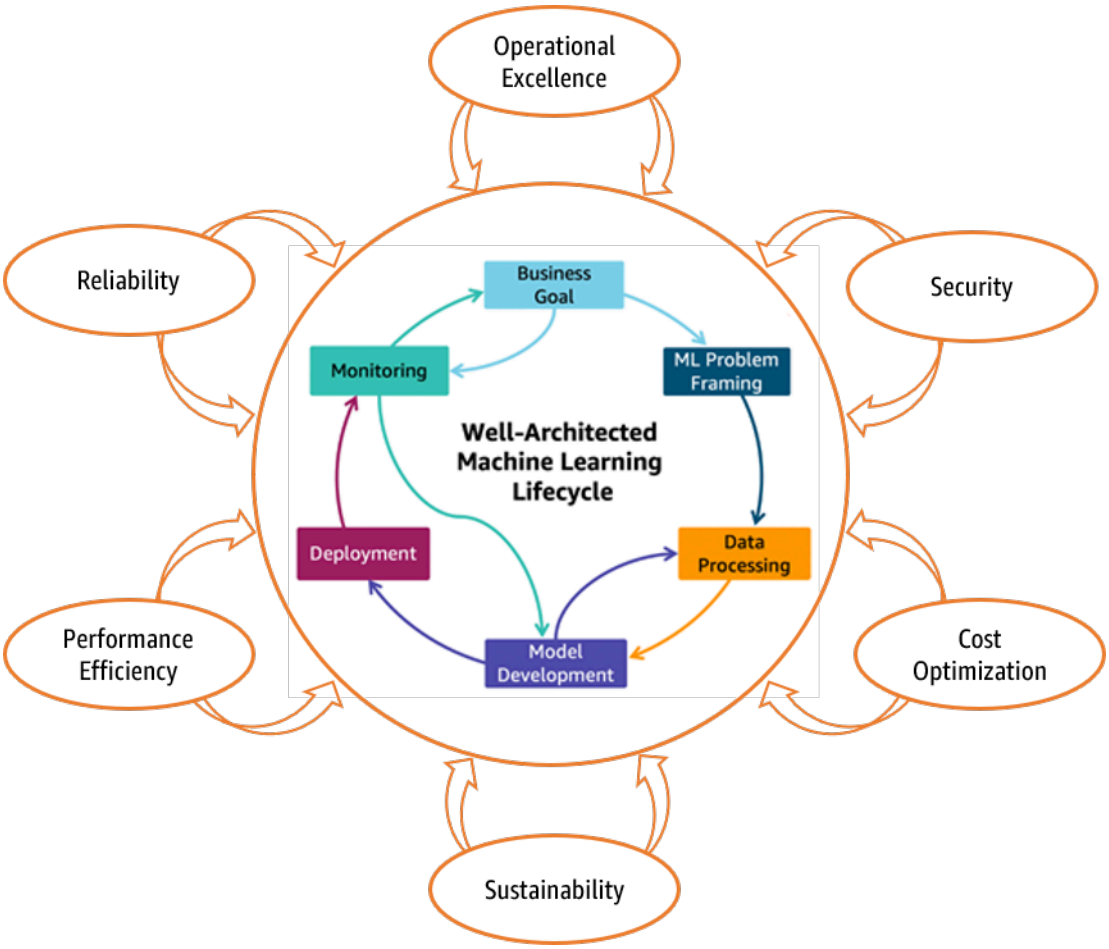
AWS Well-Architected pillars

This solution uses the best practices from the AWS Well-Architected Framework, which helps customers design and operate reliable, secure, efficient, and cost-effective workloads in the cloud.

This section describes how the design principles and best practices of the Well-Architected Framework benefit this solution.

The machine-learning lifecycle is the iterative process, with instructions and best practices, to use across defined phases while developing an ML workload. It adds clarity and structure for making a machine learning project successful. The Well-Architected machine learning lifecycle superimposes the Well-Architected Framework pillars to each of the machine learning lifecycle phases illustrated in the center of the following figure.

The Well-Architected machine learning lifecycle



Operational Excellence

This section describes how we architected this solution using the principles and best practices of the operational excellence pillar.

The QnABot on AWS solution pushes metrics to Amazon CloudWatch at various stages to provide observability into the infrastructure; Lambda functions, AI services, Amazon S3 buckets, and the rest of the solution components. Continuous integration and continuous delivery (CI/CD) and infrastructure deployment are managed in code through AWS Amplify. Data processing errors are

added to the Amazon Simple Queue Service (Amazon SQS) queue and displayed in the application layer for user response.

Security

This section describes how we architected this solution using the principles and best practices of the security pillar.

- Content designer UI app users and the Amazon Lex client are authenticated and authorized with Amazon Cognito.
- User permissions to app accounts are managed in the Amazon DynamoDB.
- All inter-service communications use AWS Identity and Access Management (IAM) roles.
- All multi-account communications use IAM roles.
- All roles used by the solution follows least-privilege access. That is, it only contains minimum permissions required so the service can function properly.
- Communication end user and Amazon API Gateway uses Bearer token generated and handed by Amazon Cognito.
- All data storage including Amazon S3 buckets have encryption at rest.

Reliability

This section describes how we architected this solution using the principles and best practices of the reliability pillar.

- The solution uses AWS Serverless Services wherever possible (examples Lambda, API Gateway, Amazon S3, and Amazon Lex) to ensure high availability and recovery from service failure.
- The solution protects against state machine definition errors by having automated tests performed on the solution.
- Data processing uses AWS Lambda functions. Data is stored in DynamoDB and Amazon S3, so it persists in multiple Availability Zones by default.

Performance Efficiency

This section describes how we architected this solution using the principles and best practices of the performance efficiency pillar.

- The solution as mentioned earlier uses serverless architecture throughout this solution.
- The solution can be launched in any Region that supports AWS services in this solution such as: AWS Lambda, Amazon API Gateway, AWS S3, Amazon Lex, Amazon Kendra, and Amazon Comprehend.
- The solution is automatically tested and deployed every day. As well as reviewed by solutions architects and subject matter experts for areas to experiment and improve.
- The QnABot on AWS CLI supports the capability to import and export questions and answers from your QnABot setup are designed to reduce IT overhead for maintenance and upkeep.

Cost Optimization

This section describes how we architected this solution using the principles and best practices of the cost optimization.

- The solution uses serverless architecture therefore, customers only get charged for what they use.
- The compute layer defaults to AWS Lambda, so it provides pay per use. DynamoDB indexes are selected to reduce throughput cost for queries.
- The solution provides an option to the user to use more advanced AI/ML services. Services such as Amazon Kendra are optional and can be turned on or off to reduce the cost for users who don't intend to use these features.

Sustainability

This section describes how we architected this solution using the principles and best practices of the sustainability pillar.

- The solution utilizes managed and serverless services, to minimize the environmental impact of the backend services. A critical component for sustainability provided by the solution is maximizing the usage of the AWS AI services. The solution Serverless design (using Lambda and DynamoDB) and the use of managed services (such as AWS Amplify) are aimed at reducing carbon footprint compared to the footprint of continually operating on-premises servers.

Architecture details

This section describes the components and AWS services that make up this solution and the architecture details on how these components work together.

AWS services in this solution

The following AWS services are included in this solution:

AWS service	Description
Amazon API Gateway	**Core.** Used for internal API management.
AWS CloudFormation	**Core.** Used to deploy the solution.
Amazon CloudWatch	**Core.** Used for monitoring and logs.
Amazon Cognito	**Core.** Used for user management.
AWS Identity and Access Management	**Core.** Used for user role and permissions management.
AWS Key Management Service	**Core.** Used for encryption.
AWS Lambda	**Core.** Provides logic for chatbot interacti ons and provides extension capabilities for Amazon Translate before and after interaction with Amazon Lex.
Amazon Lex	**Core.** Provides the advanced deep learning functionalities of ASR for converting speech to text, and NLU to recognize the intent of the text.
Amazon OpenSearch Service	**Core.** Provides question bank, metrics, feedback indices, and provides OpenSearch Dashboards for chatbot usage.
Amazon SNS	**Core.** Used for notifications, such as feedback.

AWS service	Description
Amazon Data Firehose	**Supporting.** Delivers logs and metrics data to an Amazon S3 bucket.
Amazon Polly	**Supporting.** Used for Interactive Voice Response systems. It provides text to speech capabilities to relay the response back in the voice of choice.
Amazon S3	**Supporting.** Provides object storage for content designer UI data and logs and metrics data.
AWS Systems Manager Parameter Store	**Supporting.** Provides secure, hierarchical storage for configuration data management and secrets management.
Amazon Translate	**Supporting.** Provides multi-language support to your customer's bot interactions. You can maintain question and answer banks in a single language while still offering support to customers who interact with the bot in other languages through the use of Amazon Translate.
Amazon Bedrock	**Optional.** This solution utilizes Bedrock for embedding models, LLM models, knowledge base, and guardrails.

AWS service	Description
Amazon Connect	**Optional.** Provides an omnichannel cloud contact center. If you implement this component, you can create personalized experiences for your customers. For example, you can dynamically offer chat and voice contact, based on such factors as customer preference and estimated wait times. Agents, meanwhile, conveniently handle all customers from just one interface. For example, they can chat with customers, and create or respond to tasks as they are routed to them.
Amazon Kendra	**Optional.** Hosts unstructured datasets hosted in an index. You can also use Amazon Kendra to provide semantic search capabilities to your question bank through the use of Amazon Kendra FAQs.

Amazon Lex web client

Amazon Lex allows conversational interfaces to be integrated into applications such as the Amazon Lex web client. An Amazon Lex chatbot uses *intents* to encapsulate the purpose of an interaction, and *slots* to capture elements of information from the interaction. Since QnABot on AWS has a single purpose, to answer a user's question, it defines just one intent. This intent has a single slot which is trained to capture the text of the question. QnABot on AWS also uses `AMAZON.FallBackIntent` to ensure that all user input is processed. To learn more about how Amazon Lex bots work, and to understand the concepts of intents, slots, sample values, fulfillment functions, see the Amazon Lex Developer Guide.

The QnABot on AWS Amazon Lex web client is deployed to an Amazon S3 bucket in your account, and accessed via Amazon API Gateway.

Amazon Alexa devices

Amazon Alexa devices interact with QnABot on AWS using an Alexa skill. Like an Amazon Lex chatbot, an Alexa skill also uses *intents* to encapsulate the purpose of an interaction, and *slots* to capture elements of information from the interaction.

The Alexa QnABot on AWS skill uses the same `Bot fulfillment` Lambda function as the Amazon Lex chatbot. When you ask a question, for example, *`"Alexa, ask Q and A, How can I include pictures in Q and A Bot answers?"`* , your Alexa device interacts with the skill you created, which in turn invokes the `Bot fulfillment` Lambda function in your AWS account, passing the transcribed question as a parameter.

Content designer UI

The QnABot on AWS content designer UI, like the Amazon Lex web client, is also deployed to an Amazon S3 bucket and accessed via Amazon API Gateway, and it too retrieves configuration from an API Gateway endpoint. The content designer UI requires the user to sign in with credentials defined in a Cognito user pool.

Using temporary AWS credentials from Cognito, the content designer UI interacts with secure API Gateway endpoints backed by the content designer Lambda functions. All interactions with Amazon OpenSearch Service and Amazon Lex are handled by these Lambda functions.

How QnABot on AWS works

This solution is powered by the same technology as Alexa. The Amazon Lex component provides the tools that you need to tackle challenging deep learning problems, such as speech recognition and language understanding, through an easy-to-use fully managed service. Amazon Lex integrates with AWS Lambda, which you can use to initiate functions for running your backend business logic for data retrieval and updates. Once built, your bot can be deployed directly to chat platforms, mobile clients, and IoT devices. You can also use the reports provided to track metrics for your bot. This solution provides a scalable, secure, easy to use, end-to-end solution to build, publish, and monitor your bots.

Intelligent contact centers leverage conversational UX engines like Amazon Lex in order to provide proactive service to customers. Amazon Lex uses a deep learning engine that combines ASR and NLU to manage the customer experience. This enables it to be natural and adaptable to customer needs.

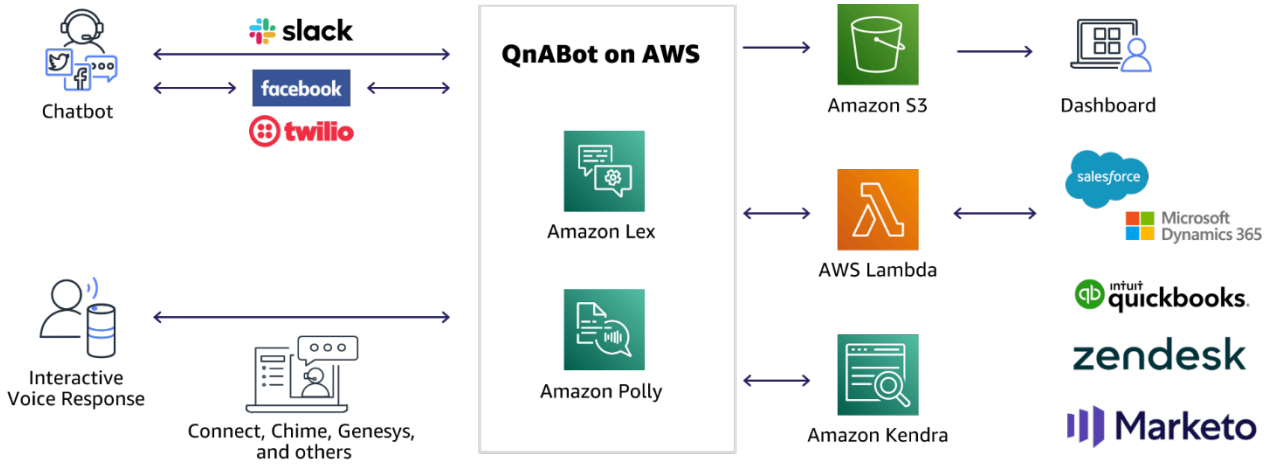
Chatbots are the starting point for many organizations. Amazon Lex comes with both voice and text. Amazon Lex has many application integrations for popular messaging platforms such as Slack and Facebook.

For Interactive Voice Response systems you can utilize the Text to speech capabilities of Amazon Polly to relay the response back in the voice of your choice.

To help fulfill many self-service requests, you can integrate Amazon Lex with your data or applications to retrieve information or use Amazon Kendra to search for the most accurate answers from your unstructured data sets.

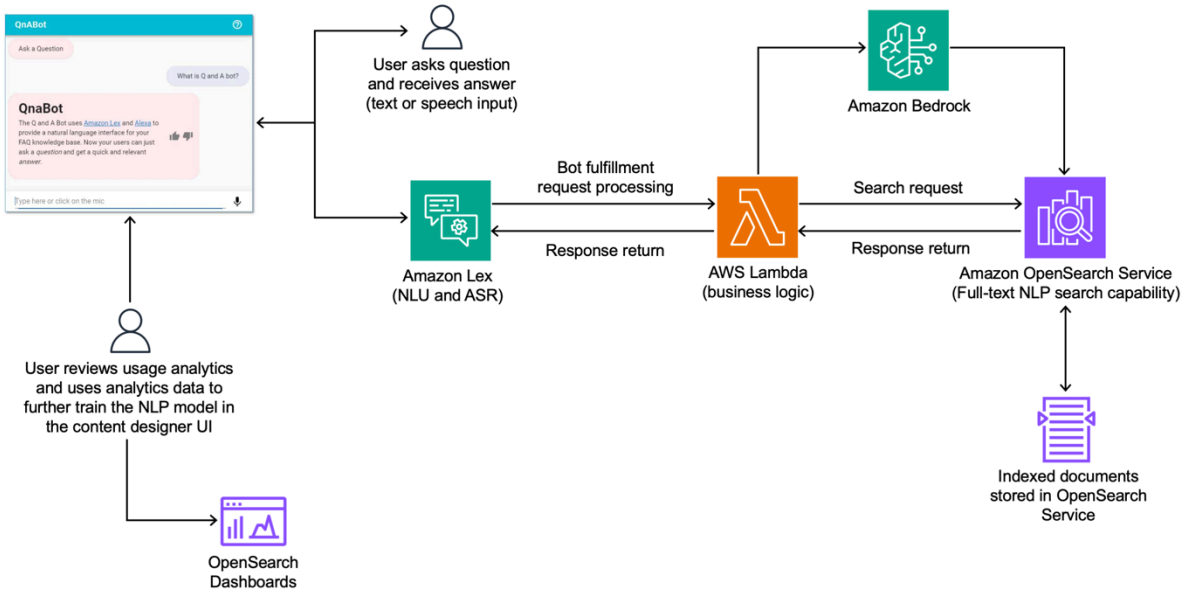
The following figure illustrates a reference architecture for how QnABot on AWS integrates with external components.

Reference architecture for QnABot on AWS integrations with external components



The following figure illustrates how Amazon Lex and Amazon OpenSearch Service help power the QnABot on AWS solution.

How Amazon Lex and Amazon OpenSearch Service help power the QnABot on AWS solution.



Asking QnABot on AWS questions initiates the following processes:

1. The question gets processed and transcribed by Amazon Lex using NLU and Natural Language Processing (NLP) engines.
2. The solution initially trains the NLP engine to match a wide variety of possible questions and statements so that the Amazon Lex chatbot can accept almost any question a user asks. The Amazon Lex interaction model is set up with the following:
   - **intents** - An intent represents an action that fulfills a user's spoken request. Intents can optionally have arguments called *slots*. The solution uses *slots* to capture user input and fulfill the intent via a Lambda function.
   - **sample utterances** - A set of likely spoken phrases mapped to the intents. This should include as many representative phrases as possible. The sample utterances specify the words and phrases users can say to invoke your intents. The solution updates the **sample utterances** with the various questions to train the chatbot to understand different end user's input.
3. This question is then sent to Amazon OpenSearch Service. The solution attempts to match an end user's request to the list of questions and answers stored in Amazon OpenSearch Service.
   - The QnABot on AWS uses full-text search to find the most relevant ranked document from the searchable index. Relevancy ranking is based on a few properties:
     - **count** - How many search terms appear in a document.

- **frequency** - How often the specified keywords occur in a given document.
- **importance** - How rare or new the specified keywords are and how closely the keywords occur together in a phrase.

- The closer the alignment between a question associated with an item and a question asked by the user, the greater the probability that the solution will choose that item as the most relevant answer. Noise words such as articles and prepositions in sentence construction have lower weighting than unique keywords.
- The keyword filter feature helps the solution to be more accurate when answering questions, and to admit more readily when it doesn't know the answer. The keyword filter feature works by using Amazon Comprehend to determine the part of speech that applies to each word you say to QnABot on AWS. By default, nouns (including proper nouns), verbs, and interjections are used as keywords. Any answer returned by QnABot on AWS must have questions that match these keywords, using the following (default) rule:
  - If there are one or two keywords, then all keywords must match.
  - If there are three or more keywords, then 75% of the keywords must match.
  - If QnABot on AWS can't find any answers that match these keyword filter rules, then it will admit that it doesn't know the answer rather than guessing an answer that doesn't match the keywords. QnABot on AWS logs every question that it can't answer so you can see them in the included Kibana Dashboard.
  - The Bot fulfillment Lambda function generates an Amazon OpenSearch Service query containing the transcribed question. The query attempts to find the best match from all the questions and answers you've previously provided, filtering items to apply the keyword filters and using Amazon OpenSearch Service relevance scoring to rank the results. Scoring is based on 1) matching the words in the end user's question against the unique set of words used in the stored questions (`quniqueterms`), 2) matching the phrasing of the user's question to the text of stored questions (nested field `questions.q`), and 3) matching the topic value assigned to the previous answer (if any) to increase the overall relevance score when the topic value (field `t`) matches. The following example code shows an Amazon OpenSearch query:

```
"query":{
    "bool": {
        "filter": {
            "match": {
                "quniqueterms": {
                    "query": "<LIST_OF_IDENTIFIED_KEYWORDS>",
```

```
                    "minimum_should_match":
                          "<ES_MINIMUM_SHOULD_MATCH SETTING>",
                    "zero_terms_query": "all"
                }
            }
        },
        "should": [
            {
                "match": {
                    "quniqueterms": {
                        "query": "<USER QUESTION>",
                        "boost": 2
                    }
                }
            },
            {
                "nested": {
                    "score_mode": "max",
                    "boost": "<ES_PHRASE_BOOST SETTING>",
                    "path": "questions",
                    "query": {
                        "match_phrase": {
                            "questions.q": "<USER QUESTION>"
                        }
                    }
                }
            },
            {
                "match": {
                    "t": "<PREVIOUS_TOPIC>"
                }
            }
        ]
    }
}
```

Plan your deployment

This section describes the cost, network security, quotas, and other considerations prior to deploying the solution.

Supported AWS Regions

This solution uses AWS services that are not currently available in all AWS Regions. You must launch this solution in an AWS Region where Amazon Lex is available. See the services implemented in this solution for more details on core services needed for the solution. Note that the solution is not supported in AWS GovCloud (US) or China Regions. For the most current availability by Region, see the AWS Services by Region list.

Cost

You are responsible for the cost of the AWS services used while running this solution. As of this latest revision, the cost for running the default basic implementation of this solution in the US East (N. Virginia) Region is approximately **$547.33 per month**.

> **Note**
>
> Amazon Kendra and Amazon Connect are **not** part of the default solution implementation, but the solution does provide the capability to integrate with them. Because the solution does not create resources for Amazon Kendra or Amazon Connect automatically, they are not included in the example cost table. If you intend to integrate Amazon Kendra and Amazon Connect, review the Amazon Kendra pricing and Amazon Connect pricing to adjust your cost estimate accordingly.

We recommend creating a budget through AWS Cost Explorer to help manage costs. Prices are subject to change. For full details, see the pricing webpage for each AWS service used in this solution. For additional information, see Creating a cost budget in the *AWS Cost Management User Guide*.

Option 1: Default basic deployment

The following table provides a sample cost breakdown for deploying this solution with the default parameters in the US East (N. Virginia) Region for one month.

AWS service	Dimensions	Cost [$USD]
Amazon API Gateway	1,000,000 REST API calls per month	$3.50
Amazon Cognito	1,000 active users per month without the advanced security feature	$0.00
Amazon S3	100 GB data transfer + 1,000,000 requests (100 records x 100 KB from Amazon Kinesis)	$3.27
AWS Lambda	2,000,000 requests with 200 ms duration	$1.23
Systems Manager Parameter Store	2,000,000 requests with 10 standard parameters	$0.00
Amazon Lex	100,000 text requests per month	$75.00
Amazon Data Firehose	100,000 records per month with 100 KB per record	$0.28
Amazon DynamoDB	2 GB storage + 2 reads and 2 writes per second + 20 hours peak read/write per month	$16.14
Amazon Polly	10,000 requests + 50 characters per request	$4.00

AWS service	Dimensions	Cost [$USD]
Amazon Translate	100,000 requests + 50 characters per request (OPTIONAL for non-English)	$75.00
Amazon Comprehend	100,000 requests + 50 characters per request	$5.00
Amazon OpenSearch Service	m6g.large.search instance running all hours in a month for 4 nodes	$368.64
Total for a default basic deployment:		**$547.33**

Option 2: Amazon Bedrock embeddings only

AWS service	Dimensions	Cost [$USD]
Amazon Bedrock for text embeddings (optional)	Daily average of 8,000 requests of 2,000 input tokens estimated using Amazon Titan Embeddings Text	$48.00
Total with Amazon Bedrock embeddings only ($547.33 + $48.00):		**$595.33**

Option 3: Amazon Bedrock embeddings and LLMs

AWS service	Dimensions	Cost [$USD]
Amazon Bedrock for LLM question answering (optional)	Daily average of 8,000 requests each made of 2,000	$180.00 (Haiku) to $2,160.00 (Sonnet)

AWS service	Dimensions	Cost [$USD]
	input tokens and 200 output tokens estimated using Anthropic Claude 3 Haiku (lower cost LLM option) or Anthropic Claude 3 Sonnet (higher cost LLM option)	
Total with Amazon Bedrock embeddings and LLMs ($595.33 + $180.00 to $2,160.00):		**$775.33 to $2,755.33**

Option 4a: Amazon Bedrock embeddings and LLM and Amazon Kendra

AWS service	Dimensions	Cost [$USD]
Amazon Kendra index	0-8,000 queries a day and up to 100,000 documents with Amazon Kendra Enterpris e Edition with 0-50 data sources	$1,008.00
Total with Amazon Bedrock embeddings and LLM and Amazon Kendra ($775.33 to $2,755.33 + $1,008.00):		**$1,783.33 to $3,763.33**

Option 4b: Amazon Bedrock embeddings and LLM and RAG using Amazon Bedrock knowledge base

AWS service	Dimensions	Cost [$USD]
Amazon Bedrock knowledge base (optional)	8,000 questions a day with 5 GB of data stored in Amazon OpenSearch Service Serverless vector store and using Anthropic Claude 3 Haiku (lower cost LLM option) or Anthropic Claude 3 Sonnet (higher cost LLM option)	$733.00 (Haiku) to $2,713.00 (Sonnet)
Total with Amazon Bedrock embeddings and LLM and RAG using Amazon Bedrock knowledge base ($775.33 to $2,755.33 + $733.00 to $2,713.00):		**$1,508.33 to $5,468.33**

Option 5a: Amazon Bedrock Guardrails Integration (Optional)

AWS service	Dimensions	Cost [$USD]
Content Filters	8,000 requests/day (1 text unit (400 characters) for both query and FM response)	$14.40
Denied Topics	8,000 requests/day (1 text unit (400 characters) for both query and FM response)	$14.40
Sensitive Information Filter (PII)	8,000 requests/day (1 text unit (400 characters) for both query and FM response)	$9.60

AWS service	Dimensions	Cost [$USD]
Contextual grounding check	8,000 requests/day (1 text unit (600 characters) for both query and FM response)	$14.40
Sensitive Information filter (Regex)	Free	-
Word Filters	Free	-
Total with Amazon Bedrock embeddings and LLM and RAG using Amazon Bedrock knowledge base ($2,084.77 to $5,468.33 + $52.80):		**$2,137.57 to $5,491.13**

Option 5b: Amazon Bedrock Pre-process Guardrails Integration (Optional)

AWS service	Dimensions	Cost [$USD]
Content Filters	8,000 requests/day (1 text unit (100 characters - questions only) for both query and FM response)	$3.60
Denied Topics	8,000 requests/day (1 text unit (100 characters - questions only) for both query and FM response)	$3.60
Sensitive Information Filter (PII)	8,000 requests/day (1 text unit (100 characters - questions only) for both query and FM response)	$2.40

AWS service	Dimensions	Cost [$USD]
Contextual grounding check	N/A	
Sensitive Information filter (Regex)	Free	-
Word Filters	Free	-
Total with Amazon Bedrock embeddings and LLM and RAG using Amazon Bedrock knowledge base ($2,084.77 to $5,468.33 + $9.60):		**$2,094.37 to $5,477.93**

Option 5c: Amazon Bedrock Post-process Guardrails Integration (Optional)

AWS service	Dimensions	Cost [$USD]
Content Filters	8,000 requests/day (1 text unit (300 characters - answers only) for both query and FM response)	$10.80
Denied Topics	8,000 requests/day (1 text unit (300 characters - answers only) for both query and FM response)	$10.80
Sensitive Information Filter (PII)	8,000 requests/day (1 text unit (300 characters - answers only) for both query and FM response)	$7.20
Contextual grounding check	N/A	

AWS service	Dimensions	Cost [$USD]
Sensitive Information filter (Regex)	Free	-
Word Filters	Free	-
Total with Amazon Bedrock embeddings and LLM and RAG using Amazon Bedrock knowledge base ($2,084.77 to $5,468.33 + $28.80):		**$2,113.57 to $5,497.13**

Option 6: Streaming Responses for QnABot

AWS service	Dimensions	Cost [$USD]
DynamoDB Table (optional)	1 GB storage + 1 read and 1 write per second + 20 hours peak read/write per month	$11.41
Lambda (optional)	2,000,000 requests with 200 ms duration	$1.23
WebSocket API (optional)	100,000 messages per day with 25,000 connections for 10 minutes	$4.94
Total with Streaming Responses for QnABot only ($547.33 + $17.58):		**$564.91**

Option 7: QnABot with OpenSearch Dedicated Master Nodes

AWS service	Dimensions	Cost [$USD]
Amazon OpenSearch Service	m6g.large.search instance running all hours in a month for 3 dedicated master nodes	$276.48
Total with Streaming Responses for QnABot only ($547.33 + $276.48):		**$823.81**

Security

When you build systems on AWS infrastructure, security responsibilities are shared between you and AWS. This shared responsibility model reduces your operational burden because AWS operates, manages, and controls the components including the host operating system, the virtualization layer, and the physical security of the facilities in which the services operate. For more information about AWS security, visit AWS Cloud Security.

Security best practices

QnABot on AWS is designed with security best practices in mind. However, the security of a solution differs based on your specific use case. Adding additional security measures can add to the cost of the solution. The following are additional recommendations to enhance the security posture of QnABot on AWS in production environments.

Amazon S3 access logging bucket configuration

We recommend having a central access logging Amazon S3 bucket, and updating the S3 buckets that this solution creates to allowing access logging. QnABot on AWS by default configures a central access logging Amazon S3 bucket to store access logging. For more information about Amazon S3 access logging see Enabling Amazon S3 server access logging in the *Amazon Simple Storage Service User Guide*.

Multi-factor authentication (MFA) in Amazon Cognito user pools

This solution creates only one user in its Cognito user pools. MFA is not activated by default; however, we recommend using MFA for users in Cognito for a stronger security posture in production workloads. For more information about setting up MFA in Cognito, see Adding MFA to a user pool and Adding advanced security to a user pool in the *Amazon Cognito Developer Guide.*

Single sign-on with AWS IAM Identity Center

Solution administrators can also federate into the content designer UI and OpenSearch Dashboards using single sign-on with AWS IAM Identity Center. In this case, IAM Identity Center serves as the identity provider for the Cognito user pool. Additionally, using Cognito, you can configure a SAML or OpenID Connect identity provider to federate with as well.

When users federate into Cognito, a user profile is dynamically provisioned for them, but they will not be granted access to QnABot on AWS until they are added to the `Admins` group. For more information about automating using a Lambda trigger see Customizing User Pool Workflows with Lambda in the *Amazon Cognito Developer Guide*.

AWS WAF for Amazon API Gateway

When the chatbot application is open to public access in production, we recommend allowing AWS WAF for API Gateway. For guidance about setting up AWS WAF, see Using AWS WAF to protect your APIs in the *Amazon API Gateway Developer Guide*. We also recommend reviewing the AWS Best Practices for DDoS Resiliency whitepaper for information about protecting your AWS applications from Distributed Denial of Service (DDoS) attacks.

For best security practices, we recommend adding rules/rule groups when creating your web access control list (ACL) in AWS WAF. AWS WAF provides the ability to set AWS managed rules and custom rule groups which the customer creates and maintains. We recommend adding Core rule set and Known bad inputs managed rule groups when setting up your web ACL. See AWS WAF rule groups in the *AWS WAF, AWS Firewall Manager, and AWS Shield Advanced Guide* for more information on setting up managed and created rule groups.

Creating a custom domain in Amazon API Gateway

By default, QnABot deploys the default domain in API Gateway. The default domain uses a TLS version 1.0 security policy, which uses outdated encryption protocols and weak encryption cyphers.

We recommend that the customer sets up a custom domain name and uses a TLS version 1.2 security policy. See Choosing a security policy for your custom domain in API Gateway in the *Amazon API Gateway Guide.*

Children Online Privacy Protection Act (COPPA) settings for Amazon Lex

When using this solution to create or update an Amazon Lex chatbot, set the Amazon Lex API **childDirected** parameter to `true` if the bot's users are subject to COPPA. For more information, see DataPrivacy in the *Amazon Lex API Reference*.

AWS CloudFormation parameters

Before deployment, we recommend reviewing the **PublicOrPrivate** parameter. It has two possible values: `Public` or `Private`. We recommend choosing `Private` unless the use case for this solution dictates having the chatbot open to the public without needing to sign up or register. If you select `Public`, we recommend enabling AWS WAF for Amazon API Gateway.

Amazon Cognito

The solution uses a Cognito user pool for controlling administrative access to the QnABot on AWS content designer UI, Amazon Lex web client, and OpenSearch Dashboards. Users are also required to be members of the `Admins` group in the Cognito user pool.

The content designer UI requires that you sign in with credentials defined in an Amazon Cognito user pool. Using temporary AWS credentials from Cognito, the content designer UI interacts with secure API Gateway endpoints backed by the content designer's Lambda functions.

The Amazon Lex web client is deployed to an Amazon S3 bucket in your account, and accessed via API Gateway. An API Gateway endpoint provides run time configuration. Using this configuration, the web client connects to Cognito to obtain temporary AWS credentials, and then connects with the Amazon Lex service.

AWS Lambda

The solution uses Lambda functions. Depending on your use case, we recommend that you configure Lambda function-level concurrency run limits. Adding concurrency limits can prevent a rapid spike in usage and costs, while also increasing or lowering the default concurrency limit.

IAM roles

IAM roles allow customers to assign granular access policies and permissions to services and users on the AWS Cloud. This solution creates IAM roles with least privileges that grant the solution's resources with needed permissions.

CloudWatch Logs

For QnABot on AWS, CloudWatch Logs are set by default to never expire. You can Export log data to Amazon S3.

Data storage and protection

The solution uses multiple services to store and protect your data. This solution defaults to the following when storing and protecting the customer's data:

Service/Resource	Default
CloudWatch Logs	- Default CloudWatch Logs set to **Never Expire**.
DynamoDB	- User table stores chat message history (per user) - never expires. - Data fully encrypted at rest (managed by DynamoDB). - Point-in-time recovery enabled by default. - Continuous backups disabled.
OpenSearch Dashboards index	- Default expiry set to 30 days.
Amazon S3	- Default **Never Expire** for Metrics bucket and Export bucket. - All buckets are enabled with server-side encryption (SSE) by default. See Setting default server-side encryption behavior for Amazon S3 buckets for additional guidance.

Service/Resource	Default
	- Access logging is disabled, customer can configure. For additional guidance, see Setting default server-side encryption for Amazon S3 buckets in the *Amazon Simple Storage Service User Guide*.
Amazon Lex	- Default, logs not enabled. For additiona l guidance, see Conversation Logs in the *Amazon Lex V2 Developer Guide*. - Encrypting conversation logs is optional, but can be implemented if needed. For additional guidance, see Encrypting Conversation Logs in the *Amazon Lex V2 Developer Guide*. - Audio logs are stored in Amazon S3 (default encryption). - The **childDirected** parameter for COPPA defaults to `false`. For additional guidance, see DataPrivacy in the *Amazon Lex API Reference*. - PII redaction capability is implemented on logs.
AWS Key Management Service	- The solution can store PII data. By default, DynamoDB is encrypted, but we recommend using Customer Managed Keys (CMK) if you intend to store sensitive data. For additional guidance, see the utility_scripts section in the GitHub repository.
Amazon Data Firehose	- SSE enabled via AWS KMS key.

Quotas

Service quotas, also referred to as limits, are the maximum number of service resources or operations for your AWS account.

Quotas for AWS services in this solution

Make sure you have sufficient quota for each of the services implemented in this solution. For more information, see AWS service quotas.

Select one of the following links to go to the page for that service. To view the service quotas for all AWS services in the documentation without switching pages, view the information in the Service endpoints and quotas page in the PDF instead.

AWS CloudFormation quotas

Your AWS account has AWS CloudFormation quotas that you should be aware of when launching the stack in this solution. By understanding these quotas, you can avoid limitation errors that would prevent you from deploying this solution successfully. For more information, see AWS CloudFormation quotas in the *AWS CloudFormation User's Guide*.

Amazon Lex quotas

Your AWS account has Amazon Lex quotas, which you can view by following these steps:

1. Sign in to the AWS Service Quotas console.
2. Choose **AWS services** from the left navigation menu.
3. Enter `Amazon Lex` in the **Find services** field.
4. Choose **Amazon Lex**.

Amazon Lex V2 requires the fulfillment Lambda's maximum output size to be set to 50 KB. You cannot adjust this setting through the AWS account's Service endpoints and quotas. You might reach this quota when you are trying to return very large responses by increasing the number of words or context in the response. Additionally, when you use RAG with Amazon Kendra or Knowledge Bases for Amazon Bedrock, you might want to limit your output by customizing the settings such as prompt templates, max retrieved results, or documents.

Amazon DynamoDB backups

Backups for Amazon DynamoDB Tables are not set up by default. If you require backups for the data that is stored in DynamoDB Tables, see Backing Up a DynamoDB Table in the *Amazon DynamoDB Developer Guide*.

For recovery of backed up data, see Restoring a DynamoDB table from a backup in the *Amazon DynamoDB Developer Guide*. Alternatively, you can use Point-in-time recovery for DynamoDB as your backup and recovery method.

Deploy the solution

This solution uses AWS CloudFormation templates and stacks to automate its deployment. The CloudFormation templates to describe the AWS resources included in this solution and their properties. The CloudFormation stack provisions the resources that are described in the template.

Deployment process overview

Before you launch the solution, review the cost, architecture, security, and other considerations discussed in this guide. Follow the step-by-step instructions in this section to configure and deploy the solution into your account.

Time to deploy: Approximately 30-45 minutes

Step 1: Launch the stack

- Launch the AWS CloudFormation template into your AWS account.
- Enter values for the required parameters.
- Review the template parameters, and adjust if necessary.

Step 2. Launch the chatbot content designer

- Update password and sign in to the content designer.

Step 3: Populate the chatbot with your questions and answers

- Enter question and answer pairs.

Step 4: Interact with the chatbot

- Interact with the chatbot through voice or text.

> **Important**
>
> This solution includes an option to send anonymized operational metrics to AWS. We use this data to better understand how customers use this solution and related services and

products. AWS owns the data gathered though this survey. Data collection is subject to the AWS Privacy Policy.
To opt out of this feature, download the template, modify the AWS CloudFormation mapping section, and then use the AWS CloudFormation console to upload your updated template and deploy the solution. For more information, see the Anonymized data collection section of this guide.

AWS CloudFormation templates

You can download the AWS CloudFormation templates for this solution before deploying it.

Deploy via main template

View template

qnabot-on-aws-main.template - Use this template to launch the solution and all associated components. The default configuration deploys the core and supporting services found in the AWS services in this solution section but you can customize the template to meet your specific needs.

Deploy via VPC template

View template

qnabot-on-aws-vpc.template - Use this template to launch the solution and all associated components. The default configuration deploys the core and supporting services found in the AWS services in this solution section but you can customize the template to meet your specific needs.

This template is made available for use as a separate installation mechanism. It is not the default template utilized in the public distribution. Take care in deploying QnABot in VPC. The OpenSearch Cluster becomes private to the VPC. In addition, the QnABot Lambda functions installed by the stack will be attached to subnets in the VPC. The OpenSearch cluster is no longer available outside of the VPC. The Lambda functions attached to the VPC allow communication with the cluster.

Two additional parameters are required by this template.

- **VPCSubnetIdList**

> **Important**
>
> You should specify two private subnets, spread over two Availability Zones.

- **VPCSecurityGroupIdList**

More information on how to deploy can be read in the VPC Support section of the GitHub repository. Additionally, we recommend following the best practices for securing the VPC.

> **Note**
>
> If you have previously deployed this solution, see Update the stack for update instructions.

Step 1: Launch the stack

This automated AWS CloudFormation template deploys the QnABot on AWS solution in the AWS Cloud. You must set up an AWS account before launching the stack.

> **Note**
>
> You are responsible for the cost of the AWS services used while running this solution. For more details, see the Cost section in this guide, and reference to the pricing webpage for each AWS service used in this solution.

1. Sign in to the AWS Management Console and select the button to launch the `qnabot-on-aws-main.template` AWS CloudFormation template.

   Launch solution

   Launch solution

The template launches in the US East (N. Virginia) Region by default. To launch the solution in a different AWS Region, use the Region selector in the console navigation bar.

> **Note**
>
> This solution uses Amazon Lex, which is not currently available in all AWS Regions. You must launch this solution in an AWS Region where Amazon Lex is available. For the most current availability by Region, see the AWS Services by Region list.

2. On the **Create stack** page, verify that the correct template URL is in the **Amazon S3 URL** text box and choose **Next**.
3. On the **Specify stack details** page, assign a name to your solution stack. For information about naming character limitations, see IAM and AWS STS quotas in the *AWS Identity and Access Management User Guide*.
4. Under **Parameters**, review the parameters for this solution template and modify them as necessary. This solution uses the following default values.

> **Note**
>
> Amazon Lex V1 has been deprecated and removed from QnABot v6.1.0. Amazon Lex V2 is used by default.

Parameter	Default	Description
Authentication		
Email	*<Requires input>*	Email address for the admin user. This email address will receive a temporary password to access the QnABot on AWS content designer.
Username	*<Requires input>*	This username will be used to sign in to the QnABot

Parameter	Default	Description
		on AWS content designer console and client if the client is private.
PublicorPrivate	`PUBLIC`	Choose whether access to the QnABot on AWS client should be publicly available or restricted to users in the QnABot in the Cognito user pool.
Language	`English`	The primary language for your QnABot on AWS deployment. NOTE: Selecting `non-English` might correspond with limited functionalities.
Amazon Kendra Integration		
Amazon KendraWeb PageIndexId	*<Optional input>*	ID of the Amazon Kendra index to use for the web crawler. A custom data source will automatically be added to the specified index.
Amazon KendraFaqIndexId	*<Optional input>*	ID of the Amazon Kendra index to use for syncing OpenSearch questions and answers.
AltSearchAmazon KendraIndexes	*<Optional input>*	A comma separated string value specifying IDs of one or more Amazon Kendra indexes to be used for Amazon Kendra fallback.

Parameter	Default	Description
AltSearchAmazonKendraIndexAuth	FALSE	Choosing TRUE enables the solution to send an OpenID token to Amazon Kendra index(es) to limit results to which the user is entitled.
Amazon OpenSearch Service		
OpenSearchDedicatedMasterNodes	DISABLED	Enable OpenSearch add dedicated master nodes to increase cluster stability. Please note that deploying additional nodes will increase cost, see - https://aws.amazon.com/opensearch-service/pricing/.
OpenSearchMasterNodeInstanceType	m6g.large.search	Required when OpenSearchDedicatedMasterNodes is ENABLED. OpenSearch instance type for master nodes in the domain. Default recommendation for production deployments is m6g.large.search (see https://docs.aws.amazon.com/opensearch-service/latest/developerguide/supported-instance-types.html for other options).

Parameter	Default	Description
OpenSearchMasterNodeCount	3	Required when OpenSearchDedicatedMasterNodes is ENABLED. Number of dedicated master nodes to add in your Amazon OpenSearch Service domain. 3 is the minimum default value. See - https://docs.aws.amazon.com/opensearch-service/latest/developerguide/managedomains-dedicatedmasternodes.html#dedicatedmasternodes-number.
OpenSearchNodeInstanceType	`m6g.large.search`	OpenSearch instance type for data nodes in the domain. Default recommendation for production deployments is `m6g.large.search` . For details, see Supported instance types in Amazon OpenSearch Service in the *Amazon OpenSearch Service Developer Guide*.
OpenSearchNodeCount	4	Number of data nodes in Amazon OpenSearch Service domain. We recommend 4 for fault-tolerant production deployments.

Parameter	Default	Description
OpenSearchEBSVolumeSize	`10`	The size in GB of the OpenSearch node instances . `10` is the minimum default volume size.
OpenSearchDashboardsRetentionMinutes	`43200`	To conserve storage in Amazon OpenSearch Service, metrics and feedback data used to populate the OpenSearch Dashboards are automatically deleted after this period (default `43200` minutes = 30 days). Monitor the free storage space for your OpenSearch Service domain to ensure that you have sufficient space available to store data for the desired retention period.

Parameter	Default	Description
OpenSearchFineGrainAccessControl	TRUE	Set to FALSE if fine-grained access control does not need to be enabled by default. Once fine-grained access control is enabled, it cannot be disabled. Note that it may take an additional 30-60 minutes for OpenSearch Service to apply these settings to the OpenSearch domain after the stack has been deployed. For details, see Fine-grained access control in Amazon OpenSearch Service in the *Amazon OpenSearch Service Developer Guide*.
Amazon LexV2		
LexV2BotLocaleIds	`en_US,es_US,fr_CA`	Languages for QnABot on AWS voice interaction using LexV2. Specify as a comma-separated list of valid locale IDs without empty spaces. For details, see the Supported languages section in the GitHub repository.
Semantic Search and Embeddings		

Parameter	Default	Description
EmbeddingsApi	DISABLED	Enable QnABot semantics search using embeddings from a pre-trained LLM. Selecting LAMBDA allows for configuration with other models. Disabled by default.
EmbeddingsLambdaArn	*<Requires input>*	Required when **EmbeddingsApi** is set to LAMBDA. Provide the ARN for a Lambda function that takes JSON `{"inputtext":"string"}` , and returns JSON `{"embedding":[…]}`.
EmbeddingsLambdaDimensions	1536	Required when **EmbeddingsApi** is set to LAMBDA. Provides the number of dimensions for embeddings returned from the Lambda function.

Parameter	Default	Description
EmbeddingsBedrockModelId	`amazon.titan-embed-text-v1`	Required when **EmbeddingsApi** is set to BEDROCK. Select the embeddings model from the list of available models. Check account and Region availability and ensure that the model is enabled in the Amazon Bedrock console before deploying. For details, see Model support by AWS Region in the *Amazon Bedrock User Guide*.
LLM Integration		
LLMApi	`DISABLED`	Enable question disambiguation and generative responses using an LLM model. Selecting the LAMBDA option allows for configuration with other LLMs.
LLMBedrockModelId	`anthropic.claude-instant-v1`	Required when **LLMApi** is set to BEDROCK. Select the LLM model from the list of available models. Check account and Region availability and ensure that the model is enabled in the Amazon Bedrock console before deploying.

Parameter	Default	Description
LLMLambdaArn	*<Requires input>*	Required if **LLMApi** is set to LAMBDA. Provide the ARN for a Lambda function that takes JSON `{"prompt":"string" , "settings": {key:value,..}}` ,and returns JSON `{"generated_text":"string"}` .
BedrockKnowledgeBaseId	*<Optional input>*	ID of an existing Amazon Bedrock knowledge base. This setting enables the use of Amazon Bedrock knowledge bases as a fallback mechanism when a match is not found in OpenSearch.
BedrockKnowledgeBaseModel	`anthropic.claude-instant-v1`	Required if **BedrockKnowledgeBaseId** is not empty. Sets the preferred LLM model to use with the Amazon Bedrock knowledge base.
Other parameters		

Parameter	Default	Description
InstallLexResponseBots	TRUE	Configures your chatbot to ask questions and process your end user's answers for surveys and quizzes. If the Elicit Response feature is not needed, choose FALSE to skip the installation of the sample Lex response bots. For details, see Configuring the chatbot to ask the questions and use response bots.
FulfillmentConcurrency	`0`	The amount of provisioned concurrency for the `Fulfillment` Lambda function. For details, see Configuring reserved concurrency.
VPCSubnetIdList	*<Optional input>*	Set to a comma delimited list of subnet IDs belonging to the target VPC you want to deploy QnABot on AWS in.
VPCSecurityGroupIdList	*<Optional input>*	Set to a comma delimited list of security group IDs used by QnABot when deployed within a VPC.
XraySetting	FALSE	Configure Lambda functions with AWS X-Ray enabled.
LogRetentionPeriod	`0`	The number of days that logs are kept before expiring. By default, logs never expire.

Parameter	Default	Description
EnableStreaming	FALSE	Configures your QnABot to use streaming responses . For details, see Enabling Streaming Responses from QnABot

5. Choose **Next**.
6. On the **Configure stack options** page, keep the default settings.
7. On the **Review and create** page, review and confirm the settings. Check the box acknowledging that the template might create IAM resources with custom names, and the box acknowledging that AWS CloudFormation might require the CAPABILITY_AUTO_EXPAND capability.
8. Choose **Submit** to deploy the stack.

You can view the status of the stack in the AWS CloudFormation console in the **Status** column. You should receive a CREATE_COMPLETE status in approximately 30-45 minutes.

When the stack deployment is complete, the **Output** tab displays the following information:

- **ContentDesignerURL** - URL to launch the content designer UI
- **ClientURL** - URL to launch the end user client webpage
- **CloudWatchDashboardURL** - URL to launch the CloudWatch dashboard for monitoring
- **FeedbackSNSTopic** - Topic name to allow feedback notifications
- **LexV2 bot information** - Data for configuring integration with contact centers and web clients.

> **Note**
>
> In addition to the primary AWS Lambda functions,this solution includes the `solution-helper` Lambda function, which runs only during initial configuration or when resources are updated or deleted.
> When you run this solution, the `solution-helper` Lambda function is not regularly active; however, you must not delete it because it is necessary to manage associated resources.

Step 2: Launch the chatbot content designer

After successfully deploying the stack, you will receive an email at the email address listed in the deployment parameters with the subject *QnABot on AWS Signup Verification Code*. This email contains a generated temporary password that you can use to sign in to the content designer and create your own password.

Use the following procedure to launch the content designer, reset your password, and sign in to the content designer UI.

1. Open the verification email and select the link. Alternatively, sign in to the CloudFormation console, choose this solution's stack, select the **Outputs** tab, then select the **ContentDesignerURL** link. The content designer opens in a separate browser tab.
2. Sign in with your username and temporary password.
   a. Enter the **username** that you specified in the deployment parameters.
   b. Enter the temporary **password** from the verification email.
3. Follow the prompts to change your password and sign in. Your new password must have a length of at least eight characters, and contain at least one of each of the following: upper-case and lower-case characters, numbers, and special characters.
4. Sign in with your username and new password.

To reset the user password using the **Forgot your password** option on the sign in page, verify the user email.

AWS Management Console method for verifying user email

1. Sign in to the Amazon Cognito console.
2. Choose **User Pools** and select the user pool belonging to the QnABot stack.
3. Choose **Users** and select the user for which the password needs to be reset.
4. Choose **Edit User attributes**, select **Mark email address as verified**.
5. Choose **Save**.

AWS CLI method for verifying user email

To verify email, run:

```
aws cognito-idp admin-update-user-attributes \
 --user-pool-id <qnabot user pool id> \
 --username <username> \
 --user-attributes Name="email_verified",Value="true"
```

To get the user pool ID, run:

```
aws cognito-idp list-user-pools --max-results 10
```

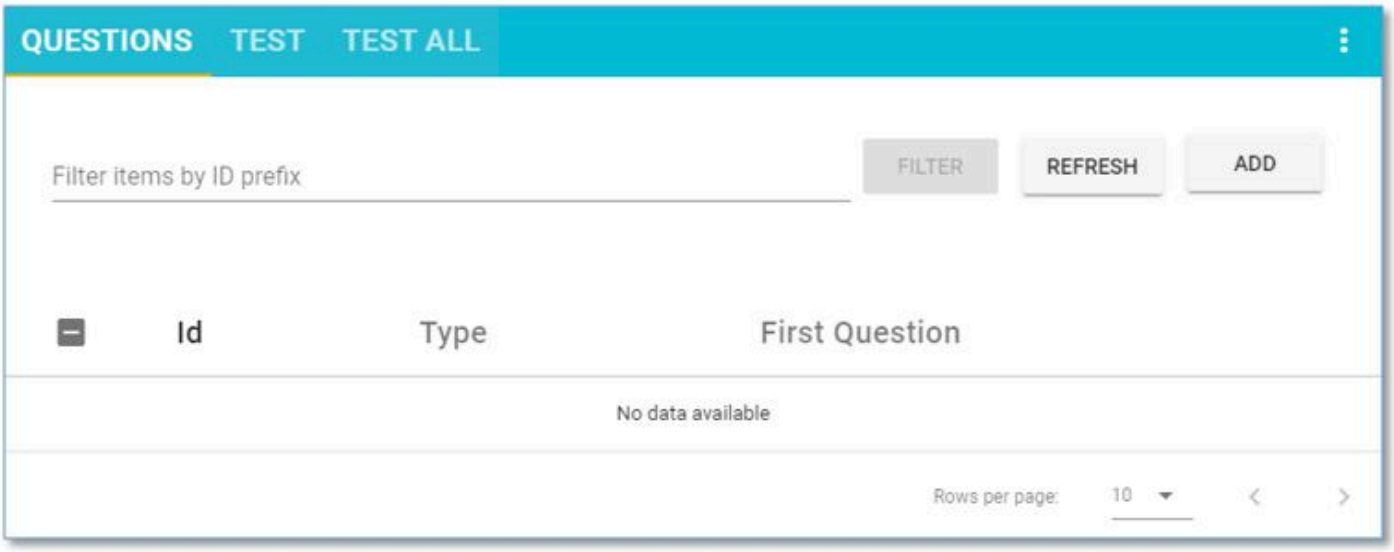
Step 3: Populate the chatbot with your questions and answers

Create or upload question and answer data through the content designer before sharing the QnABot on AWS with your end users. Your data is stored in Amazon OpenSearch Service, which allows the data to be crawled when end users ask questions using either an Amazon Lex client UI or an Amazon Alexa hands-free device.

Use the following procedure to get started with customizing your chatbot using the solution's sample questions. You can edit the sample questions to customize the data to meet your needs.

1. From the AWS CloudFormation console, launch the content designer user interface by selecting the **ContentDesignerURL** link from the **Outputs** tab of the primary CloudFormation stack.
2. Enter the administrator username you provided when you launched the stack and your new password.

QnABot on AWS content designer web user interface — QUESTIONS tab



3. Choose **Add.**
4. Enter the ID: `AWS-QnABot.001 `

> **Note**
>
> Use a naming convention to identify your items within categories.

5. Enter the question: `What is Q and A bot?`
6. Enter the answer: `The Q and A Bot uses Amazon Lex and Alexa to provide a natural language interface for your FAQ knowledge base, so your users can just ask a question and get a quick and relevant answer. `
7. Select **CREATE**.
8. Repeat steps 3-7, entering the items from the following table (**Table 1: Sample Q and A data**).

   Alternatively, you can import the items directly from a file. Select **Import from the top left tools menu (☰)**, then choose **Examples/Extensions**, find the package called **blog-samples**, and choose **LOAD.**

> **Note**
>
> We recommend that you always import the **QnaUtility** example of questions set because it enables support of `no_hits`, `no_verified_identity`, `help`, `repeat`, and thumbs up and down feedback.

9. When the import is complete, choose **Edit** from the top left tools menu (☰), and then choose **LEX REBUILD** from the top right edit card menu (⋮).

Table 1: Sample Q and A data

Id	Question	Answer
`AWS-QnABot.002`	How do I use Q and A Bot?	Create and administer your questions and answers using the **QnABot** content designer UI. End users ask questions using the Amazon Lex web UI

Id	Question	Answer
		that supports voice or chat, or using Alexa devices for hands free voice interaction.
Admin.001	How do I modify Q and A Bot content?	Use the content designer Question and Test tools to find your existing documents and edit them directly in the console. You can also export existing documents as a JSON file, make changes to the file, and re-import.
Admin.002	Can I back up Q and A Bot content?	Yes. Use the content designer to export your content as a JSON file. Maintain this file in your version control system or in an S3 bucket. Use the content designer UI import feature to restore content from the JSON file.
Admin.003	Can I import Q and A Bot content from a file?	Yes, the content designer has an import function that lets you load items from a formatted JSON file. You can create JSON files using the export feature, or you can write your own tools to create JSON files from existing content such as a website FAQ page.

Id	Question	Answer
Admin.004	How do I troubleshoot and fix problems with Q and A Bot?	Use the content designer test tool to test a question, and check what items are returned, ranked in order of score. If the desired item does not have the highest score, then add the question to the item and run the test again. The desired item should now have the highest score. Ensure that you aren't creating items with duplicate questions to avoid unpredict able responses.
Admin.005	How can I find specific Q and A items in the Designer UI?	Use the filter feature in the **Questions** tab to filter the items list based on the ID field. Or use the **Test** tab to list all the items that match a question.
Media.001	How can I include pictures in Q and A Bot answers?	Add an image attachment to the item using the content designer.

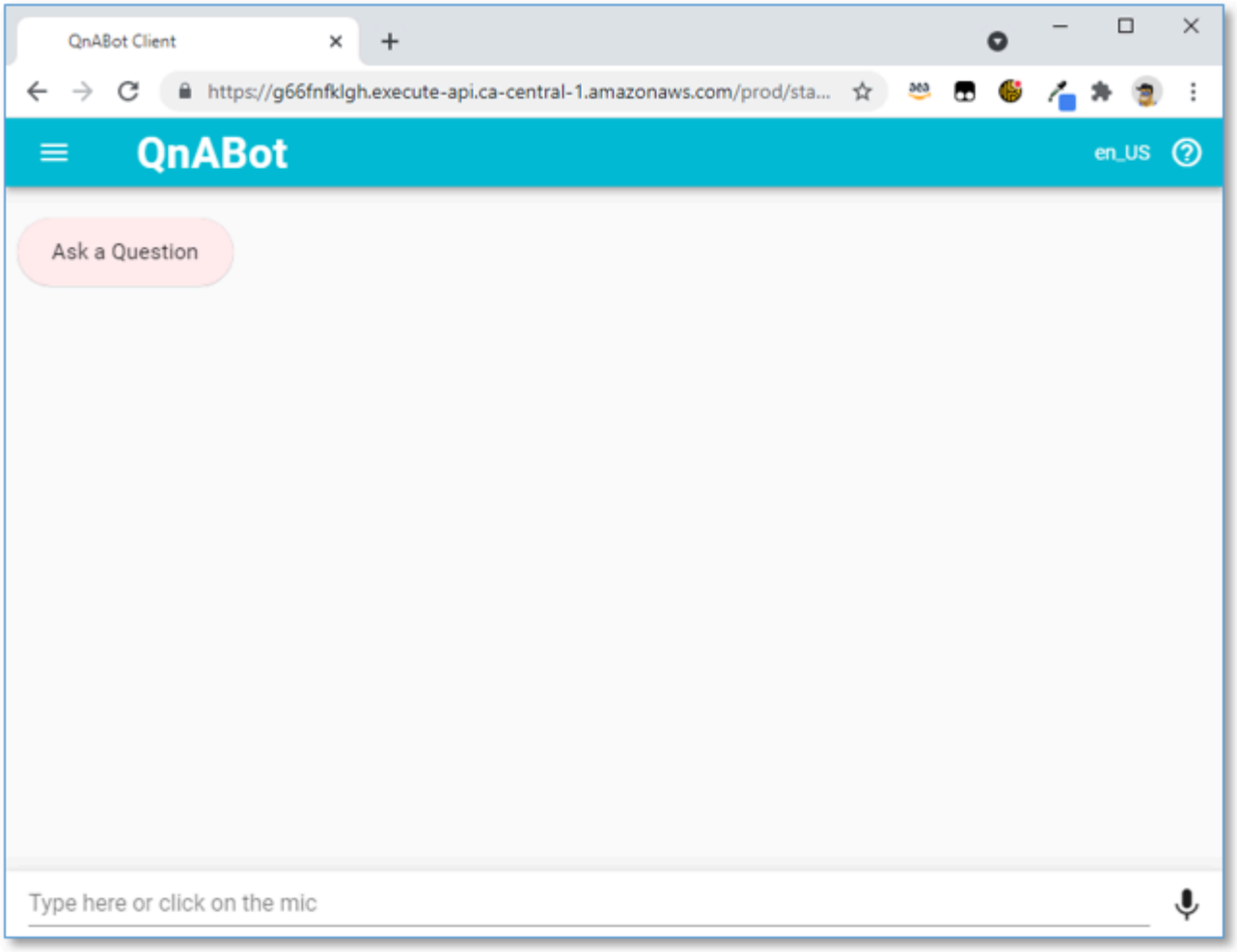
Step 4: Interact with the chatbot

Getting answers using an Amazon Lex web client user interface

You can launch QnABot on AWS from a Chrome, Firefox, or Microsoft Edge browser on your PC, Mac OS, Chromebook, or Android tablet.

1. From the AWS CloudFormation console, select the main QnABot on AWS stack, choose **Output**, and then select the link to the **ClientURL**. Alternatively, launch the client by choosing **QnABot on AWS Client** from the content designer tools menu (☰).
2. When your browser requests access to the microphone on behalf of the web application, allow it. The QnABot on AWS chat window opens.

   QnABot on AWS web user interface chat window



3. Interact with the chatbot through the chat window. You can communicate through voice or text.

Select the microphone icon (bottom right) and say, *"What is Q and A Bot?"*

The chatbot responds with the answer you programmed in Step 3: Create chatbot content and load sample Q and A data.

Getting answers using Amazon Alexa

The QnABot on AWS solution also works with Amazon Alexa, allowing your end users to get answers from your programmed content via any Amazon Alexa device, including Amazon FireTV, and any of the Amazon Echo family of devices.

> **Note**
>
> To integrate with Amazon Alexa, you must first use the Amazon Developer Console to create an Alexa skill for QnABot on AWS. This solution doesn't automatically create Alexa skills. You can use the content designer to launch a walkthrough for creating an Alexa skill.

Use the following procedure to create an Alexa skill.

1. Sign in to the QnABot on AWS content designer, open the tools menu (☰), and choose **Alexa**.
2. Follow the instructions in the console.
3. (Optional) Test your new skill in the Amazon Developer Console, even if you don't have an Alexa device nearby.

When testing the skill, invoke the skill with the invocation name before asking questions and answers. For example, if your invocation name is *my qnabot*, in the Alexa skill test console, first say, *"Open my Q and A Bot."* Alexa will reply with *"Hello, please ask a question,"* then you can ask your QnABot a question.

> **Note**
>
> If you want to publish your new QnABot on AWS skill to the Alexa skills store so that other users can access it, see Submitting an Alexa Skill for Certification. Unpublished skills are accessible only to Alexa devices registered to your Amazon account; published skills are available to anyone.

Monitor the solution with Service Catalog AppRegistry

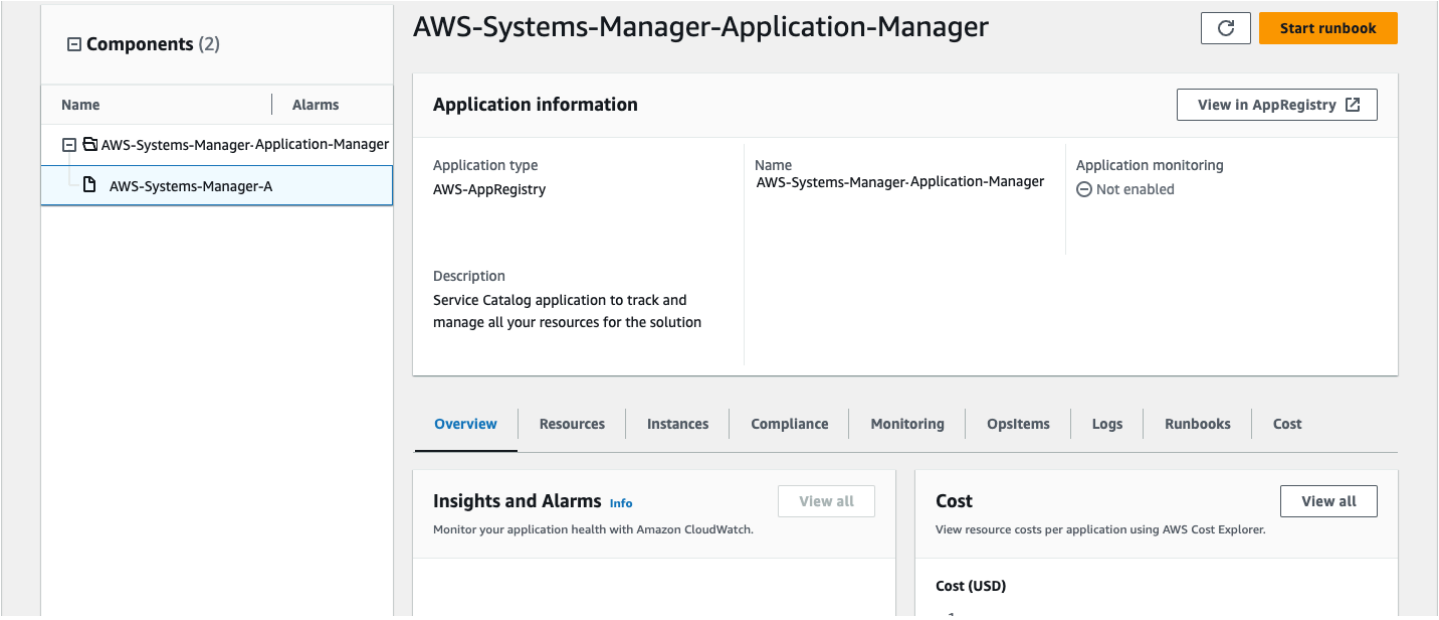
This solution includes a Service Catalog AppRegistry resource to register the CloudFormation template and underlying resources as an application in both Service Catalog AppRegistry and AWS Systems Manager Application Manager.

AWS Systems Manager Application Manager gives you an application-level view into this solution and its resources so that you can:

- Monitor its resources, costs for the deployed resources across stacks and AWS accounts, and logs associated with this solution from a central location.
- View operations data for the resources of this solution (such as deployment status, CloudWatch alarms, resource configurations, and operational issues) in the context of an application.

The following figure depicts an example of the application view for the solution stack in Application Manager.

Depicts an AWS Solution stack in Application Manager



Activate CloudWatch Application Insights

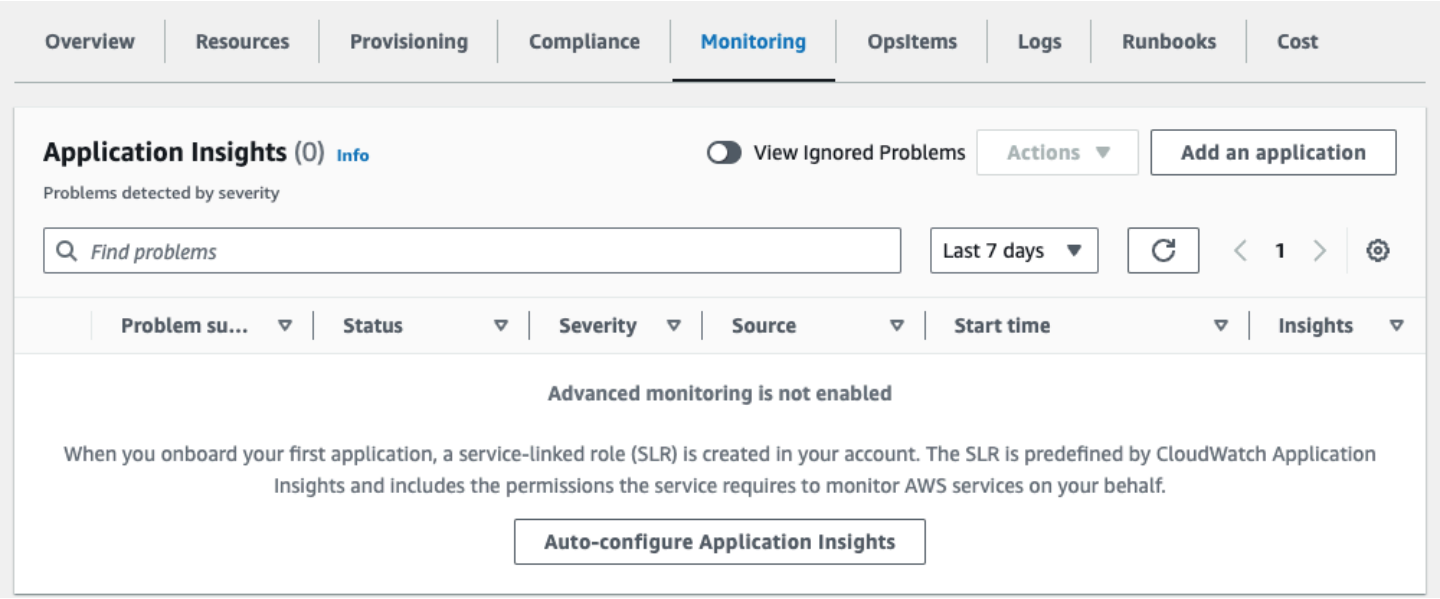
1. Sign in to the Systems Manager console.
2. In the navigation pane, choose **Application Manager**.

3. In **Applications**, search for the application name for this solution and select it.

   The application name will have App Registry in the **Application Source** column, and will have a combination of the solution name, Region, account ID, or stack name.

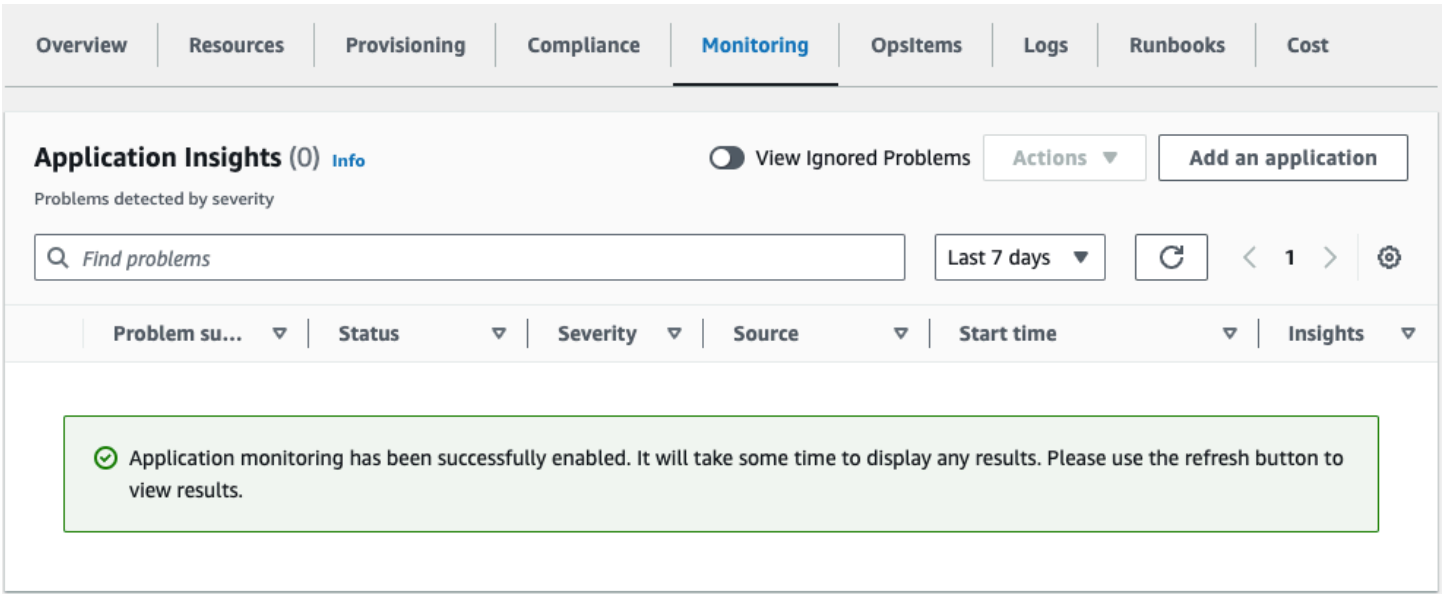
4. In the **Components** tree, choose the application stack you want to activate.
5. In the **Monitoring** tab, in **Application Insights**, select **Auto-configure Application Insights**.

Application Insights dashboard showing no detected problems and advanced monitoring not enabled.



Monitoring for your applications is now activated and the following status box appears:

Application Insights dashboard showing successful monitoring activation message.

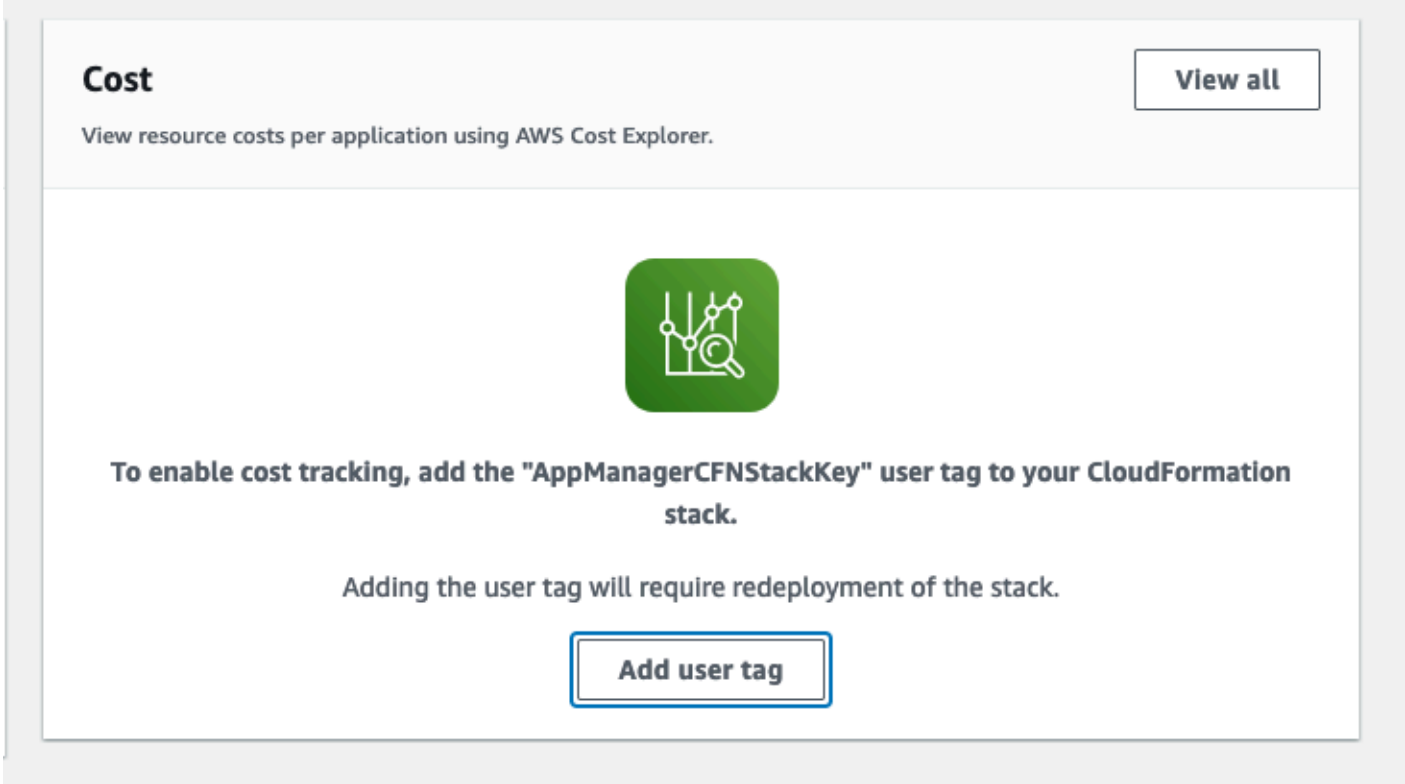


Confirm cost tags associated with the solution

After you activate cost allocation tags associated with the solution, you must confirm the cost allocation tags to see the costs for this solution. To confirm cost allocation tags:

1. Sign in to the Systems Manager console.
2. In the navigation pane, choose **Application Manager**.
3. In **Applications**, choose the application name for this solution and select it.
4. In the **Overview** tab, in **Cost**, select **Add user tag**.

Screenshot depicting the Application Cost add user tag screen



5. On the **Add user tag** page, enter `confirm`, then select **Add user tag**.

The activation process can take up to 24 hours to complete and the tag data to appear.

Activate cost allocation tags associated with the solution

After you confirm the cost tags associated with this solution, you must activate the cost allocation tags to see the costs for this solution. The cost allocation tags can only be activated from the management account for the organization.

To activate cost allocation tags:

1. Sign in to the AWS Billing and Cost Management and Cost Management console.
2. In the navigation pane, select **Cost Allocation Tags**.
3. On the **Cost allocation tags** page, filter for the `AppManagerCFNStackKey` tag, then select the tag from the results shown.
4. Choose **Activate**.

AWS Cost Explorer

You can see the overview of the costs associated with the application and application components within the Application Manager console through integration with AWS Cost Explorer. Cost Explorer helps you manage costs by providing a view of your AWS resource costs and usage over time.

1. Sign in to the AWS Cost Management console.
2. In the navigation menu, select **Cost Explorer** to view the solution's costs and usage over time.

Update the solution

If you have previously deployed the solution, follow this procedure to update the QnABot on AWS CloudFormation stack to get the latest version of the solution's framework.

1. Sign in to the AWS CloudFormation console, select your existing QnABot on AWS CloudFormation stack, and choose **Update**.
2. Select **Replace current template**.
3. Enter the appropriate Amazon S3 URL:
   - If using the default main template: link:https://solutions-reference.s3.amazonaws.com/qnabot-on-aws/latest/qnabot-on-aws-main.template
   - If using the VPC template: link:https://solutions-reference.s3.amazonaws.com/qnabot-on-aws/latest/qnabot-on-aws-vpc.template
4. Under **Parameters**, review the parameters for the template and modify them as necessary. Refer to Step 1: Launch the stack for details about the parameters.
5. Choose **Next**.
6. On the **Configure stack options** page, choose **Next**.
7. On the **Review** page, review and confirm the settings. Be sure to check the box acknowledging that the template might create AWS Identity and Access Management (IAM) resources.
8. Choose **View change set** and verify the changes.
9. Choose **Update stack** to deploy the stack.

You can view the status of the stack in the AWS CloudFormation console in the **Status** column. You should receive a status in approximately 30 minutes.

> **Note**
>
> If you have previously deployed the solution but do not want to perform in-place upgrade or encountered issues during in-place upgrade or in case of a breaking change which does not allow you to upgrade in-place, please refer to migration section in this README.md.

Note

For those upgrading to v6.1.X and above

You might not see your previous executions on the Import, Export, and Test All pages disappear. To restore them, go to the respective S3 buckets and copy all the folders containing the data or status for each function (Import, Export, Test All) to the `ContentDesignerOutputBucket`. Rename them as `data-{function}` or `status-{function}`. Omitting this step doesn't effect your QandAs and only impacts these specific pages.

Troubleshooting

If you need help with this solution, contact AWS Support to open a support case for this solution.

Contact AWS Support

If you have AWS Developer Support, AWS Business Support, or AWS Enterprise Support, you can use the Support Center to get expert assistance with this solution. The following sections provide instructions.

Create case

1. Sign in to Support Center.
2. Choose **Create case**.

How can we help?

1. Choose **Technical**.
2. For **Service**, select **Solutions**.
3. For **Category**, select **Other Solutions**.
4. For **Severity**, select the option that best matches your use case.
5. When you enter the **Service**, **Category**, and **Severity**, the interface populates links to common troubleshooting questions. If you can't resolve your question with these links, choose **Next step: Additional information**.

Additional information

1. For **Subject**, enter text summarizing your question or issue.
2. For **Description**, describe the issue in detail.
3. Choose **Attach files**.
4. Attach the information that AWS Support needs to process the request.

Help us resolve your case faster

1. Enter the requested information.
2. Choose **Next step: Solve now or contact us**.

Solve now or contact us

1. Review the **Solve now** solutions.
2. If you can't resolve your issue with these solutions, choose **Contact us**, enter the requested information, and choose **Submit**.

Uninstall the solution

You can uninstall the QnABot on AWS solution from the AWS Management Console or by using the AWS Command Line Interface.

Using the AWS Management Console

1. Sign in to the AWS CloudFormation console.
2. Select this solution's installation stack.
3. Choose **Delete**.

> **Note**
>
> Some IAM Roles are retained after stack deletion. You can find and delete them by searching `AdminRole`, `OpenSearchDashboardsRole`, `UnauthenticatedRole`, and `UserRole`. You can also find all the roles by taking the first portion of your deleted **Stack ID** found in CloudFormation.

Using AWS Command Line Interface

Determine whether the AWS Command Line Interface (AWS CLI) is available in your environment. For installation instructions, see What Is the AWS Command Line Interface in the *AWS CLI User Guide*. Optionally, you can use the AWS CloudShell service to run AWS CLI commands. After confirming that the AWS CLI is available, run the following command:

```
$ aws cloudformation delete-stack --stack-name <installation-stack-name>
```

Advanced setup

This section provides detailed instructions on how to set up QnABot to perform the following tasks:

- Adding images to your answers
- Displaying rich text answers
- Using SSML to control speech synthesis
- Using topics to support follow-up questions and contextual user journeys
- Adding buttons to the web UI
- Integrating Handlebars templates
- Quizzes
- Setting Amazon Lex session attributes
- Specifying Lambda hook functions
- Using keyword filters for more accurate answers and customizing "don't know" answers
- Configuring intent and slot matching
- Configuring the chatbot to ask the questions and use response bots
- Bot routing
- Connecting QnABot on AWS to an Amazon Connect call center
- Connecting QnABot on AWS to Genesys Cloud
- Tuning testing and troubleshooting unexpected answers
- Importing and exporting chatbot answers
- Modifying configuration settings
- Integrating Amazon Kendra
- Embeddings Model
- Text generation and query disambiguation using LLMs
- Setting up a custom domain name for QnABot content designer and client
- Using QnABot on AWS Command Line Interface (CLI)
- Enabling Streaming Responses from QnABot

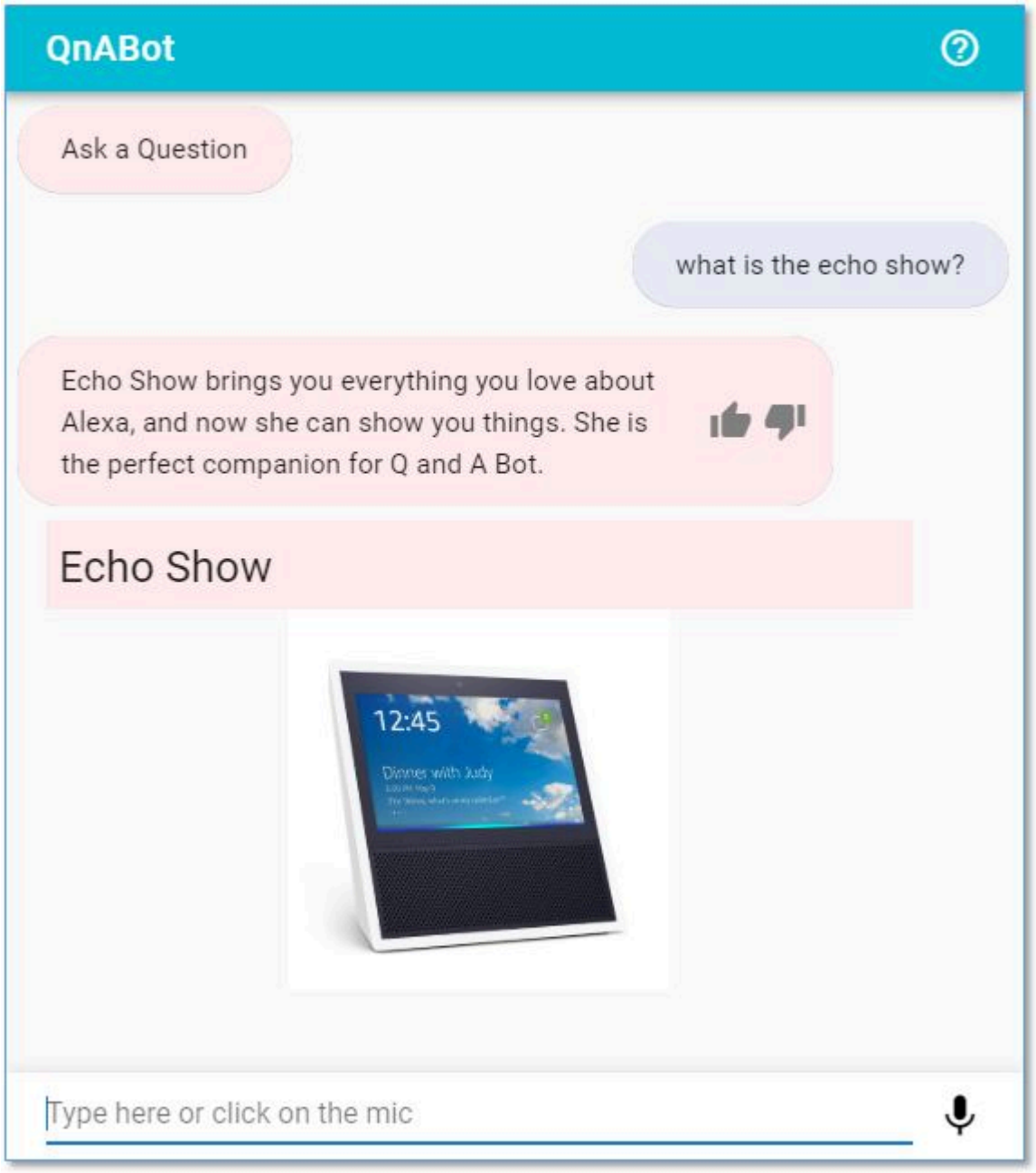
Adding images to your answers

You can augment your answers with image attachments that can be displayed on an end user's Amazon Lex web client user interface, Alexa smartphone app, or Amazon Echo Show device touch screen. For example, you can use images to display maps, diagrams, or photographs to depict places and products relevant to a question.

1. Sign in to the content designer, and choose **Add**.
2. Enter ID: `Alexa.001`.
3. Enter question: `What is an Amazon Echo Show?`
4. Enter answer: `Echo Show brings you everything you love about Alexa, and now she can show you things. She is the perfect companion for Q and A Bot.`
5. Choose **Advanced.**
6. Under **Response Card**, enter the following:
   a. Card Title: `Echo Show`
   b. Card ImageUrl: https://images-na.ssl-images-amazon.com/images/I/61OddH8ddDL.*SL1000*.jpg
7. Choose **CREATE** to save the new item.
8. Use the web UI chat window to ask: *`"What is an Echo Show?"`*

   The photograph is displayed in the web UI chat.

   Example image response in the web UI chat window



9. Optionally, you can use an Amazon Echo or Echo Dot to say: *"Ask Q and A, What is an Echo Show?"*

   The card shown in the Alexa smartphone app shows the photo attachment.

10. Optionally, you can use an Amazon Echo Show to say: *"Ask Q and A, What is an Echo Show?"*

    The photo attachment is displayed on the Echo Show's touch screen.

Displaying rich text answers

QnABot on AWS supports Markdown, allowing you to create rich text versions of your answers for displaying on the web user interface, or on Slack. To use this feature, populate the **Alternate Markdown answer** field in the content designer.

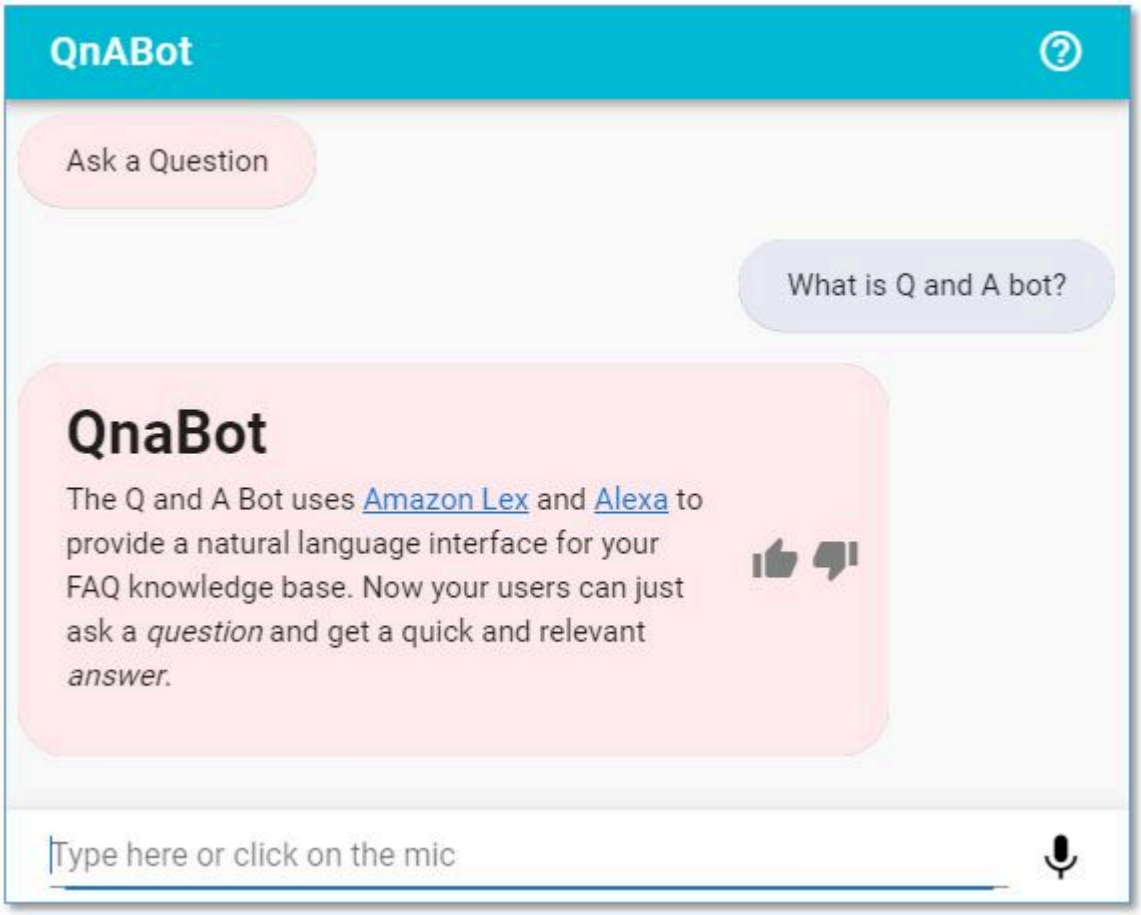
1. From the content designer, edit item `AWS-QnABot001` (What is Q and A bot?) by opening the **Advanced** section and entering the following text in the **Markdown Answer** field:

```
# AWS QnABot
The Q and A Bot uses [Amazon Lex](https://aws.amazon.com/lex) and [Alexa](https://
developer.amazon.com/alexa) to provide a natural language interface for your FAQ
 knowledge base.
Now your users can just ask a *question* and get a quick and relevant *answer*.
```

2. Choose **UPDATE** to save the modification.
3. Use the web user interface to ask: *"What is Q and A bot?"*.

   The answer now displays the heading, links, and emphasis specified in your markdown text.

   Sample Markdown text



QnABot on AWS also supports inline HTML in the markdown field:

4. Choose **ADD** to create a new item in HTML:
   a. Enter ID: FireTV.001
   b. Enter question: What is Amazon Fire TV?
   c. Enter answer:

```
Fire TV brings all the live TV and streaming content you love, and Alexa, onto
 the big screen. Use Alexa on the Fire TV to bring QnABot on AWS into your living
 room!
```

   d. Enter markdown answer:

```
 **Fire TV** brings all the live TV and streaming content you love, and Alexa,
 onto the big screen. Use Alexa on the Fire TV to bring QnABot on AWS into your
 living room!
<iframe src="https://www.youtube.com/embed/OE4MrFx2XCs"></iframe>
```

5. Choose **CREATE** to save the item.

Using SSML to control speech synthesis

The solution supports Speech Synthesis Markup Language (SSML) reference—providing additional control over the speech generation for your response. To use this feature, populate the **SSML answer** field in the content designer.

1. From the content designer, edit item `AWS-QnABot001` *`"What is Q and A Bot"`* by selecting the **Advanced** section and entering the following text in the SSML Answer field:

```
<speak>AWS <sub alias="Q and A">QnA</sub> Bot is <amazon:effect name="drc">great</
amazon:effect>. <sub alias="Q and A">QnA</sub> Bot supports <sub alias="Speech
 Synthesis Markup Language ">SSML</sub> using Polly's neural voice. <prosody
 rate="150%">I can speak very fast</prosody>, <prosody rate="75%">or very
 slowly</prosody>. <prosody volume="-16dB">I can speak quietly</prosody>,
 <amazon:effect name="drc">or speak loud and clear</amazon:effect>. I can say
 <phoneme alphabet="ipa" ph="tə#m##tə#">tomato</phoneme> and tomato. Visit
 docs.aws.amazon.com/polly/latest/dg/supportedtags for more information.</speak>
```

2. Choose **UPDATE** to save the modification.
3. Use the web UI to ask, using voice: *`"What is Q and A bot?"`* , and listen to the whispered response.
4. Choose **UPDATE** to save the item.
5. Choose **ADD** to create a new item for our first follow-up question:
   a. Enter ID: `Alexa.Cost`
   b. Enter question: `How much does it cost?`
   c. Enter answer: `For latest prices on the Echo Show, see the Amazon retail site or`
   d. Enter topic: `EchoShow`

Using topics to support follow-up questions and contextual user journeys

The solution remembers the topic from the last question you asked, which allows you to ask follow-up questions, for example: *`"How much does it cost?"`* The correct answer depends

on the context set by the previous question. To use this feature, you must assign a value to the **Topic** field in the content designer.

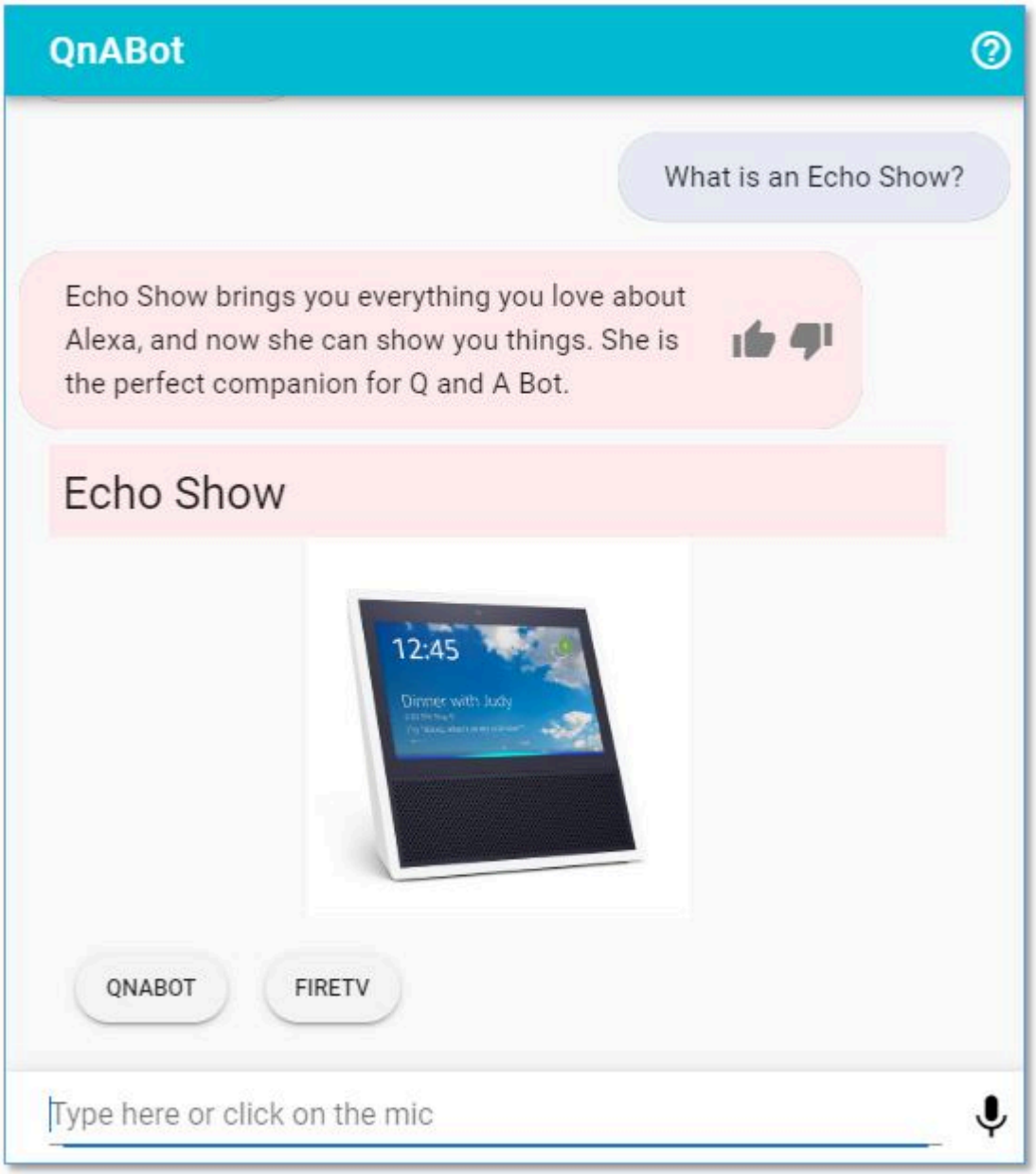
1. From the content designer, edit item `Alexa.001` (*"What is an Amazon Echo Show?"*), and enter `EchoShow` as the topic in the **Advanced section**.
2. Choose **UPDATE** to save the item.
3. Edit item `AWS-QnABot.001` (*"What is Q and A bot?"*), and enter `AWS-QnABot` as the topic.
4. Choose **UPDATE** to save the item.
5. Choose **ADD** to create a new item for our first follow-up question:
   a. Enter ID: `Alexa.Cost`
   b. Enter question: `How much does it cost?`
   c. Enter answer: `For latest prices on the Echo Show, see the Amazon retail site or shopping app.`
   d. Enter topic: `EchoShow`
6. Choose **CREATE** to save the item.
7. Choose **ADD** to create a new item for our next follow-up question. Enter the following values:
   a. Enter ID: `QnABot on AWS.Cost`
   b. Enter question: `How much does it cost?`
   c. Enter answer: `Q and A Bot is priceless`
   d. Enter topic: `QnABot on AWS`
8. Choose **CREATE** to save the item.
9. Use the web UI to ask the following questions and observe the context appropriate answers:
   a. *"What is an Echo Show?"*
   b. The answer to this question now sets the conversation topic to: `'EchoShow'`.
   c. *"How much does it cost?"*
   d. The topic disambiguates this question, so it responds with the answer for the Echo Show.
   e. *"What is the Q and A bot?"*
   f. This question changes the Topic to: `'QnABot on AWS'`.
   g. *"How much does it cost?"*
   h. The new topic allows the QnABot on AWS to respond with the correct answer.

Adding buttons to the web UI

You can add buttons to your chatbot's answers to help guide your end user by suggesting what they might want to do next.

1. From the content designer, edit item `Alexa.001` (*"What is an Amazon Echo Show?"*)
2. From the **Advanced** section, under **Response** card, enter a card title.
3. Under **Lex Buttons** enter the following:
   a. Display text: `AWS-QnABot`
   b. Button value: `What is Q and A bot?`
4. Select **ADD LEX BUTTON** to add another button.
   a. Display text: `FireTV`
   b. Button value: `What is Amazon Fire TV?`
5. Select **UPDATE** to save the item with your new buttons.
6. Use the web UI to ask: *"What is an Echo Show?"*

   Choose a button to send the next question.



7. Choose one of the buttons to automatically send the next question to QnABot on AWS.

> **Note**
>
> When integrating with Connect, QnABot on AWS maps to the Connect List Picker Template. The client sets limits on the number of characters in a field and enforces formatting using text from the QnABot on AWS plaintext response. You might need to modify the QnABot on AWS plaintext response to accommodate these limitations with the Connect chat client.

Integrating Handlebars templates

This solution supports the Handlebars simple templating language in your answers (including in the markdown and SSML fields) which allows you to include variable substitution and conditional elements. Use the following procedure to integrate Handlebars.

1. From the content designer, choose **Add**.
   a. Enter ID: `Handlebars.001`
   b. Enter question: `What is my interaction count?`
   c. Enter answer: `So far, you have interacted with me {{UserInfo.InteractionCount}} times.`
2. Save the new item.
3. Use the web UI, or any Alexa device to say, *"What is my interaction count?"* to your chatbot, and listen to it respond.
4. Ask a few more questions, and then ask *"What is my interaction count?"* again. Notice that the value has increased.
5. From the content designer, edit item `Handlebars.001`
6. Modify the answer to:

```
So far, you have interacted with me {{UserInfo.InteractionCount}} times.
{{#ifCond UserInfo.TimeSinceLastInteraction '>' 60}}
It's over a minute since I heard from you last.. I almost fell asleep!
{{else}}
Keep those questions coming fast.. It's been {{UserInfo.TimeSinceLastInteraction}}
 seconds since your last interaction.
{{/ifCond}}
```

7. Use the web UI, or Alexa, to interact with the chatbot again. Wait over a minute between interactions and observe the conditional answer in action.

There's a lot more that you can do with Handlebars, such as randomly selecting content from a list, setting and accessing session attributes, and generating Amazon S3 presigned URLs. For more information see the Handlebars section in the GitHub repository.

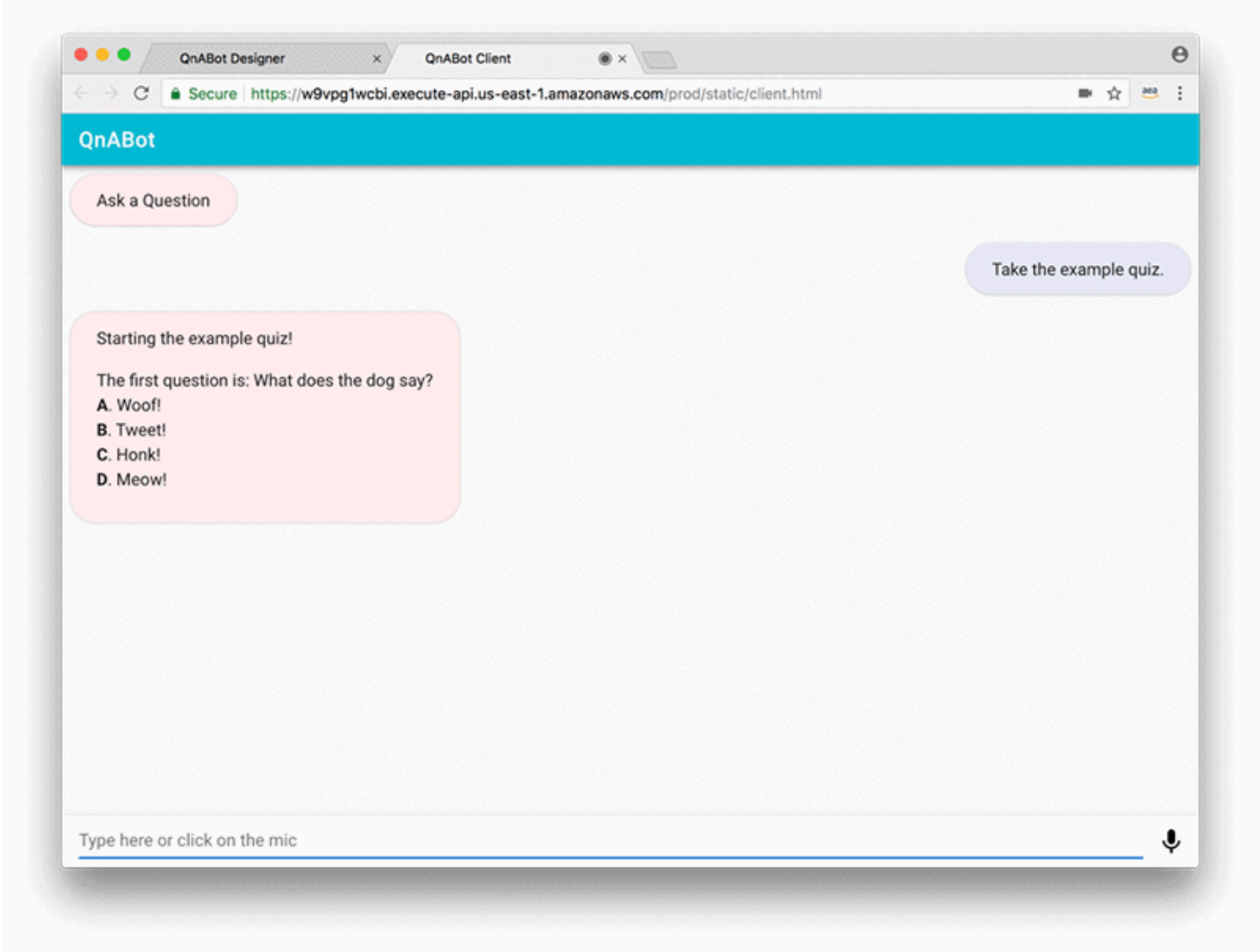
Quizzes

The solution's content designer allows you to set up QnABot on AWS to use its Questionnaire Bot feature to create simple quizzes for your users.

QnABot on AWS comes with a simple quiz example that you can customize:

1. From the content designer, choose **Import** from the tools menu (☰).
2. Select **Examples/Extensions**, and then choose **LOAD** from the **Quiz** example.
3. After the import job has completed, return to the edit page, and examine the items **ExampleQuiz & QuizEntry**.
4. Use the web UI to say, *"Start the example quiz."* or *"Take the example quiz."* to begin the quiz.

   Example Quiz



For more detailed information about how you can create and customize quizzes, see the QnABot Workshop.

Setting Amazon Lex session attributes

The QnABot on AWS solution provides support for a question in the content designer UI to set an Amazon Lex session attribute.

In early versions (v5.0.0 and earlier), using Handlebars in an answer would set a session attribute. For example, the following code can set an attribute called `attributeName` to the value `attributeValue`.

```
"{{setSessionAttr 'attributeName' 'attributeValue'}}"
```

Now, you can optionally use a question in the content designer UI to define a set of name/value pairs as session attributes when the answer is returned. There is a field to set a name/value pair, an **Add** button, and a **Delete** button.

The attribute name can be a simple name, such as `myAttribute` or a complex name, such as `myAttribute.subAttribute`. You can also use the *dot* notation to set an attribute several levels deep.

> **Note**
>
> Avoid using `appContext` or `qnabotcontext` as attribute names. Setting these might have adverse effects on the system.

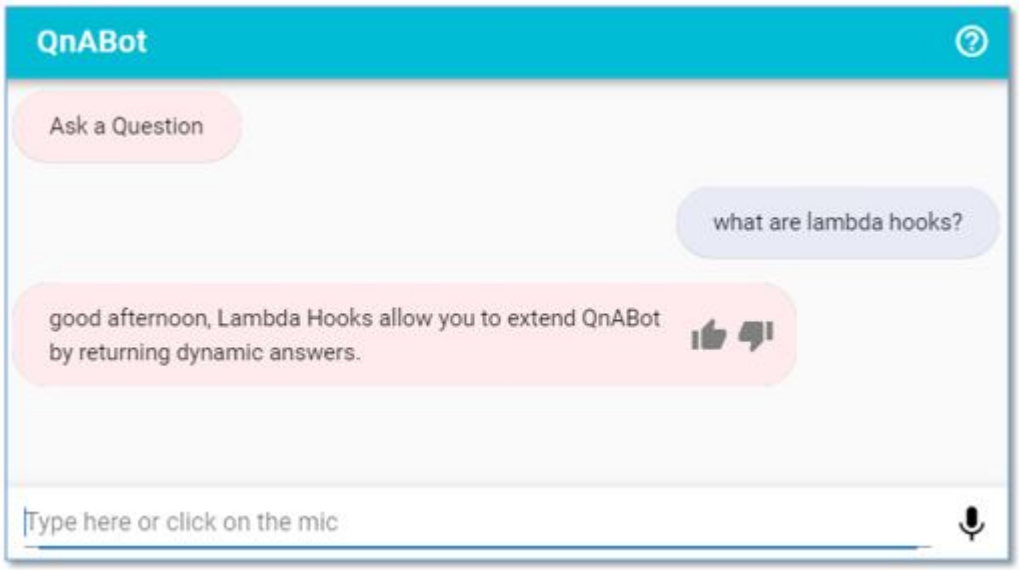
Specifying Lambda hook functions

The solution's content designer allows you to dynamically generate answers by specifying your own Lambda hook function for any item. When you specify the name, or ARN, of a Lambda function in the **Lambda hook** field for an item, QnABot on AWS will call your function any time that item is matched to an end user's question. Your Lambda function can run code to integrate with other services, perform actions, and generate dynamic answers.

QnABot on AWS comes with a simple Lambda hook function example that you can customize:

1. From the content designer, choose **Import** from the tools menu (☰).
2. Select **Examples/Extensions**, and then choose **LOAD** from the **GreetingHook** example.
3. After the import job has completed, return to the edit page, and examine the item **GreetingHookExample**. The Lambda hook field is populated with a Lambda function name.
4. Use the web UI to say, *`"What are Lambda hooks?"`* . Note that the answer is prepended with a dynamic greeting based on the current time of day - in this case *`good afternoon`* .

   Example Lambda hook function.



5. Inspect the `ExampleJSLambdahook` Lambda function using the AWS Lambda console.
6. Choose **Lambda Hooks** from the content designer tools menu (☰) to display additional information to help you create your own Lambda hook functions.

For more information about how you can package Lambda hooks, see the Extending QnABot with Lambda hook functions section in the GitHub repository.

Using keyword filters for more accurate answers and customizing "don't know" answers

The keyword filter feature helps the solution to be more accurate when answering questions with OpenSearch Service, and to admit more readily when it doesn't know the answer.

Keyword filters

The keyword filter feature works by using Amazon Comprehend to determine the part of speech that applies to each word you say to QnABot on AWS. By default, nouns (including proper nouns), verbs, and interjections are used as keywords. Any answer returned by QnABot on AWS must have questions that match these keywords, using the following (default) rule:

- If there are one or two keywords, then all keywords must match.
- If there are three or more keywords, then 75% of the keywords must match.

If you have selected a non-English language for you deployment, then it will use Amazon Translate to translate these keywords back to the language your user is using for interaction. If QnABot on AWS can't find any answers that match these keyword filter rules, then it will admit that it doesn't know the answer rather than guessing an answer that doesn't match the keywords. The solution logs every question that it can't answer so you can see them in OpenSearch Dashboards.

Custom "Don't Know" answers

When QnABot on AWS can't find an answer, by default you'll see or hear the response, *"You stumped me! Sadly, I don't know how to answer your question"* . You can customize this answer by creating a new item in the content designer, called the no_hits item:

1. From the content designer, choose **ADD** to create a new item:
   a. Enter ID: CustomNoMatches
   b. Enter question: no_hits
   c. Enter answer: Terribly sorry, but I don't know that one. Ask me another.
2. Choose **CREATE** to save the item.
3. Use the web UI to ask: *"What are Echo Buds?"*

Configuring intent and slot matching

The solution supports different types of question and answer workflows. For example:

- You can create a question and answer experience to help answer frequently asked questions. In this model, the user asks a question and QnABot on AWS responds with the most relevant answer to the question (from the list of created Item IDs). For more information, see Step 3. Populate the chatbot with your questions and answers.
- Build a diagnostic or questionnaire-based workflow, where a question from a user can result with QnABot on AWS asking follow-up questions. If you are creating a survey or building a diagnostic workflow where you may require inputs to different questions, you can use the ResponseBots and Document Chaining capabilities of QnABot on AWS. For more information, see Configuring the chatbot to ask the questions and use response bots.

Both of these options provide flexibility in creating an interactive chat experience. For example:

- Accepting dynamic user input in a question.

- Automatically asking a question for a given input without needing to setup document chaining.
- Validating user input against an available list of options.

With this early implementation of the intent and slot matching capability in QnABot on AWS, you can now build a richer conversational experiences. For example, you might create an intent that makes a car reservation, or assists an agent during a live chat or call (via Amazon Connect). You can use intent and slot matching also for cases where you might want better intent matching via Amazon Lex NLU engine, as an alternative to QnABot on AWS default OpenSearch Service queries.

> **Note**
>
> The intent and slot matching capability in QnABot on AWS was initially implemented in version 5.2.0. The content and step-by-step procedures in this section apply to QnABot on AWS versions 5.2.0 and later.

Item ID setup

The **Item ID** setup is made of the following attributes:

- **Intent** Represents an action that the user wants to perform. For each intent, provide the following required information:
  - **Intent name** Descriptive name for the intent by providing an Item ID, for example, `IntentSlotMatching.Example.Q1`.
  - **Sample utterances** The intent a user might convey. For example, a user might say, *"book a car"* or *"make a car reservation"*.
- **Slot** An intent can require zero or more slots, or parameters. You add slots as part of the Item ID configuration. At runtime, Amazon Lex V2 prompts the user for specific slot values. The user must provide values for all required slots before Amazon Lex V2 can fulfill the intent.
- **Slot type** Define the values that users can supply for your intent slots. Each slot has a type. You can create your own slot type, or you can use built-in slot types.

Creating custom intent with slots and slot types

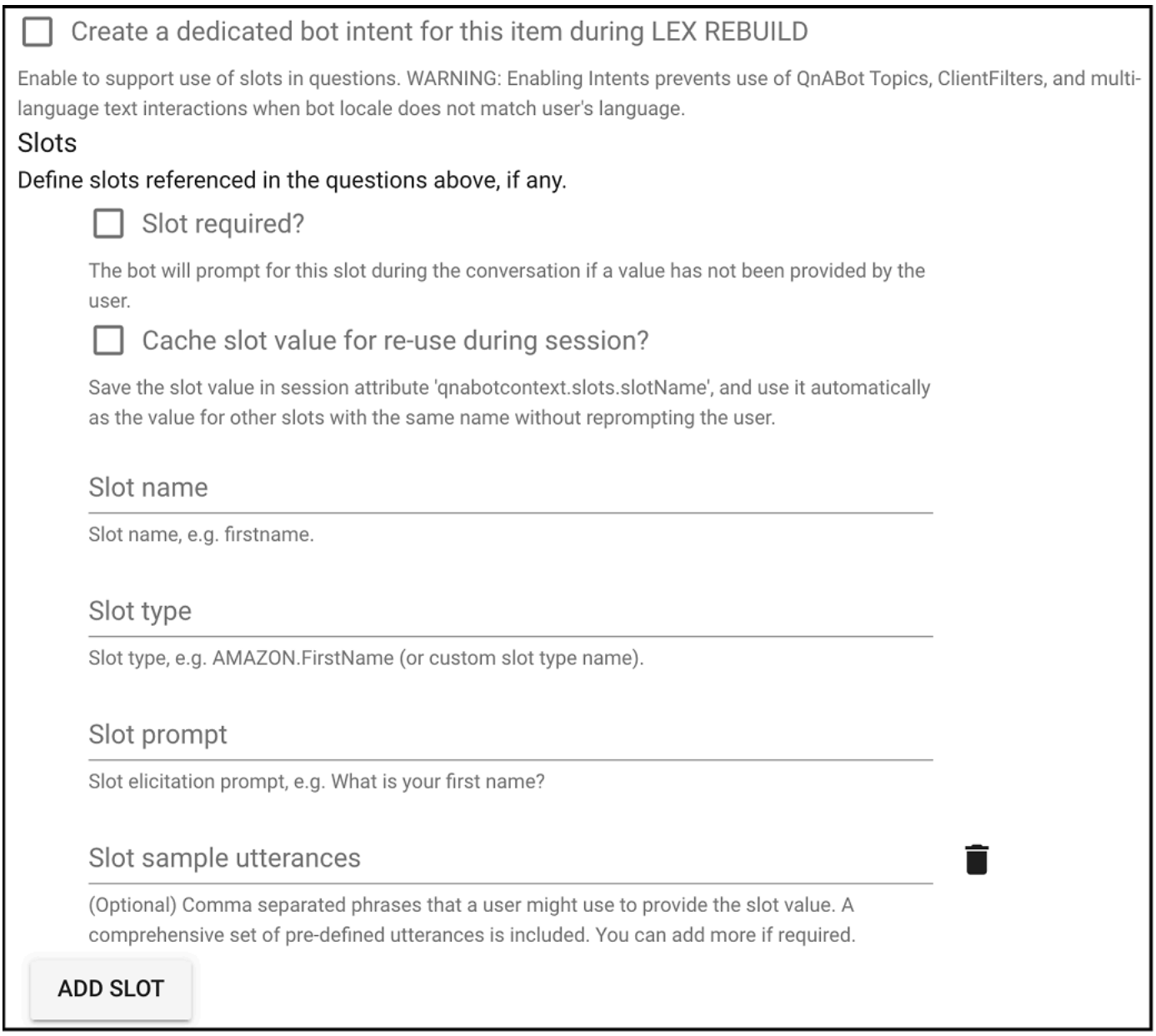
To create a custom intend with slots and slot types:

1. Create a QnABot question as you would normally do by providing an Item ID and questions/utterances.
2. Expand the **Advanced** option.
3. Select the option for **Create a dedicated bot intent for this item** during LEX REBUILD.

Slots can be configured to be either required or optional. If a conversation flow requires user input, choose the **Slot required** option.

For each slot, provide the slot type and one or more prompts that Amazon Lex V2 sends to the client to elicit values from the user. A user can reply with a slot value when input might be needed. You can create your own custom slot type, or you can use built-in slot types.

Intent and slot configuration



Additional **Slots** attributes:

- **Cache slot value for re-use during a session**? The slot value can be stored in session variables and accessed via **qnabotcontext.slots.slotName**. When a slot value is stored in a session attribute, it is used automatically as the value for other slots with the same name without reprompting the user. This can be beneficial when you are capturing a user's profile information to support different conversational workflows, and don't want to ask the same profile information again from the user.
- **Slot sample utterances** A slot can also include optional sample utterances. These are phrases that a user might use to provide the slot value. A comprehensive set of pre-defined utterances

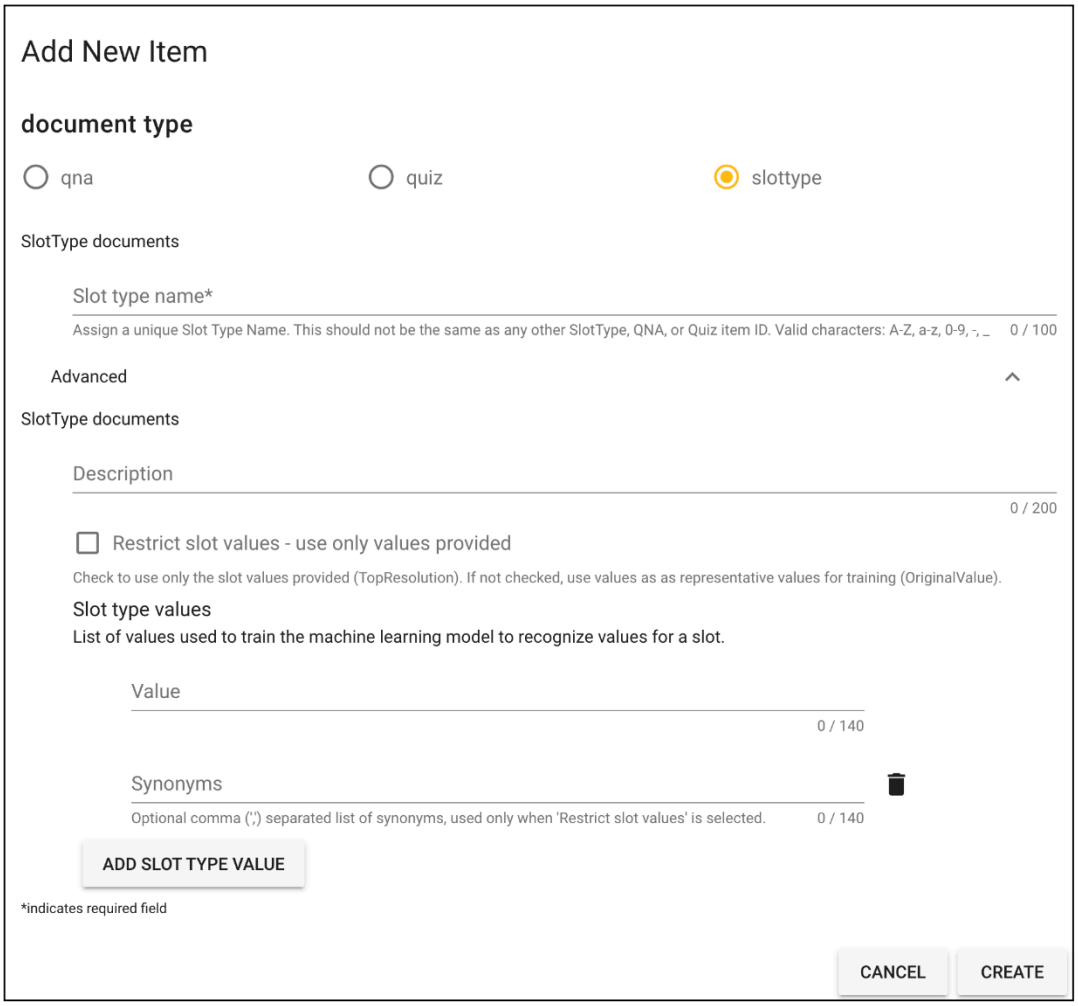
is included (via built-in slot type or a custom slot type). You can add more if required. In most cases, Amazon Lex can understand user utterances. If you know a specific pattern that users might respond to an Amazon Lex request for a slot value, you can provide those utterances to improve accuracy. In most cases, you won't need to provide any utterances.

Creating custom slot types

In addition to using built-in slot types, you can also create custom slot types. If an intent requires a custom slot type, you can create a custom slot type by creating a new item and choosing the type **slottype**. Similar to built-in slot types, a custom slot type can be used across more than one intent.

- **Slot type values** - The values for the slot. If you chose `Restrict to` slot values, you can add synonyms for the value. For example, for the value `football` you can add the synonym `soccer`. If the user enters `soccer` in a conversation with your bot, the actual value of the slot is `football`.
- **Slot value resolution** - Determines how slot values are resolved. If you don't choose `Restrict to` slot values, Amazon Lex V2 uses the values as representative values for training. If you choose `Restrict to` slot values, the allowed values for the slot are restricted to the ones that you provide.

  Creating a custom slot type



Accessing slot values

To support a conversational experience, you might want to:

- Display what the user provided as slot values, such as workflows that require confirming user input.
- Use the slot values to support conditional branching via document chaining.
- Display a summary such as an order summary.

There are several ways to access slot values within an Item ID and/or Lambda hook. Using Handlebars you can access slot values using:

- **getSlot** - A new helper function that returns named slot value if it is defined, or default value. For example: `{{getSlot 'slotName' 'default'}}`.

- **Session attribute {{qnabotcontext.slots.*slotName*}}** - A value in a session attribute, where `_slotName_` is the name of the slot defined in an Item ID. If the slot value is cached for re-use, the value is available in a session attribute, and can be used across Item IDs.
- **{{Slots.*slotName*}}** - A slot name, where `_slotName_` is the name of the slot defined in an Item ID.

Import sample intent and slot types

In the QnABot content designer, select the **Tools** menu link on the top left and select **Import**. From the **Examples/Extensions** section, select **Load** for **IntentSlotMatching** to load sample intent and slot types. This example imports:

- **IntentSlotMatching.Example.Q1** - An Item ID of type **qna** with custom intent and slot.
- **IntentSlotMatching_Example_slottype_CarType** and **IntentSlotMatching_Example_slottype_Confirmation** Item ID of type `slottype` with sample slot values.

Lex rebuild

Once you have loaded the questions, choose **Edit** from the Tools menu and choose **LEX REBUILD** from the top right edit card menu (⋮). This re-trains Amazon Lex using the newly added questions as training data.

Testing the experience

On the **Tools** menu and choose **QnABot Client** from the options. Try the following conversation flow:

```
User: Book a car
 Bot: In what city do you need to rent a car?

 User: Seattle
 Bot: What day do you want to start your rental?

 User: Today
 Bot: What day do you want to return this car?

 User: Next Sunday
```

```
Bot: What type of car would you like to rent? Our most popular options are economy,
midsize, and luxury.

User: Economy
Bot: Okay, should I go ahead and book the reservation?

User: Yes
Bot: Okay, I have confirmed your reservation. The reservation details are below:
     Car Type: economy
     Pick up City: Seattle
     Pick up Date: 2022-05-30
     Return Date: 2022-06-12
```

Notes and considerations

- Utterances must be unique across intents. Duplicate utterances across intents will cause the Amazon Lex build to fail. Suppose you have two intents `OrderPizza` and `OrderDrink` in your bot and both are configured with an `I want to order` utterance. This utterance does not map to a specific intent that Amazon Lex V2 can learn from while building the language model for the bot at build time. As a result, when a user inputs this utterance at runtime, Amazon Lex V2 can't pick an intent with a high degree of confidence.
- `Topics` and `ClientFilters` are not supported when an Item ID is activated with custom intent.
- Bot locale must be set to user's locale for QnABot on AWS multi-language text interactions.
- Always initiate a LEX REBUILD when activating Item IDs with custom intent and slots. This creates the custom intents, slots, and slot types in Amazon Lex V2, and also trains Amazon Lex using the added/updated Item IDs as training data.
- To take advantage of the additional features supported by Amazon Lex, such as confirmation prompts and regular expression to validate the value of a slot, you can also create the Amazon Lex intents and slot types in the QnABot Lex bot using the Amazon Lex console. For more information, see Adding intents in the *Amazon Lex V2 Developer Guide*.
- Even if the Amazon Lex intents and slot types are created in the Amazon Lex console (created outside of the **QnABot** content designer), you can reference any `SlotType` defined in the bot in a QnABot Item ID, and also map a QID to a manually created Amazon Lex intent in QnABot on AWS.
- The **Test All** or **Test** options don't work correctly for Item IDs with custom intent.

- While you are building your knowledge bank of questions, you might have a combination of FAQ-based questions and intent-based questions. There may be instances where a wrong intent gets matched or a FAQ question is matched instead. To troubleshoot this issue, try the following:

Enable the ENABLE_DEBUG_RESPONSES setting in QnABot on AWS. This setting provides debug information to help understand what is processing the request, such as, `Intent`, `OpenSearch`, or `Amazon Kendra`.

Configuring the chatbot to ask the questions and use response bots

You can configure your chatbot to ask questions and process your end user's answers. Use this feature for data collection and validation; to implement surveys, quizzes, personalized recommendations; or triage chatbot applications.

Use the following procedure to configure the chatbot to ask questions.

1. Sign in to the content designer and choose **Add.**
2. Enter ID: `ElicitResponse.001`
3. Enter question: `Ask my name`
4. Enter answer: `Hello. Can you give me your First Name and Last Name please?`
5. Choose **Advanced**.
6. Under **Elicit Response**, enter the following:
   a. **Elicit Response: ResponseBot Hook**: `QNAName`

      Alternatively for Lex V2 bots, you can use the syntax `lexv2::BotId/BotAliasId/LocaleId`. This allows you when you are combining elicit responses with multi-language to use a specific language for the elicit response bot.

   b. **Elicit Response: Response Session Attribute Namespace**: `name_of_user`
7. Choose **CREATE** to save the new item.
8. Use the web UI to say: *`"Ask my name"`*
9. Respond by entering your name. Try responding naturally and see if chatbot confirms your name correctly. If not, you can choose **NO** and try again.

The **ResponseBot Hook** field specifies the name of an Amazon Lex chatbot. In this case we specified the name of a chatbot, **QNAName**, that was automatically created for us when the solution was installed. QNAName is a built-in response chatbot designed to process names (first and last name). It handles a variety of ways the user might state their name, and it will prompt the user to confirm or to try again. If the user confirms by choosing YES, the response chatbot will return the `FirstName` and `LastName` values back to the solution as slot values in a fulfilled response.

The solution stores the returned `FirstName` and `LastName` values in a session attribute. The name of the session attribute is determined by the value you provided for **Response Session Attribute Namespace** (in this case `name_of_user`) and the slot name(s) returned by the response chatbot (in this case `FirstName` and `LastName`).

The session attribute set by Elicit Response can be used in other items to provide conditional or personalized responses.

10. Sign in to the content designer, and choose **Add.**
    a. Enter ID: `ElicitResponse.002`
    b. Enter question: `Ask my age`
    c. Enter answer: `Hello {{SessionAttributes.name_of_user.FirstName}} What is your age in years?`
11. Choose **Advanced**.
    a. Enter **Elicit Response: ResponseBot Hook**: `QNAAge`
    b. Enter **Elicit Response: Response Session Attribute Namespace**: `age_of_user`
12. Choose **CREATE** to save the new item.
13. Use the web UI to say: *"Ask my age."*

Response bots

The solution provides a set of built-in response bots that you can use out of the box:

- **QNAYesNo -** Returns slot `Yes_No` with value either `Yes` or `No`
- **QNAYesNoExit -** Returns slot `Yes_No_Exit` with value either `Yes`, `No`, or `Exit`
- **QNADate -** Returns slot `Date` with value of date (`YYYY-MM-DD`)
- **QNADayOfWeek -** Returns slot `DayOfWeek`

- **QNAMonth -** Returns slot `Month`
- **QNANumber -** Returns slot `Number`
- **QNAAge -** Returns slot `Age`
- **QNAPhoneNumber -** Returns slot `PhoneNumber`
- **QNATime -** Returns slot `Time` with value of time (`hh:mm`)
- **QNAEmailAddress -** Returns slot `EmailAddress`
- **QNAName -** Returns slots `FirstName` and `LastName`
- **QNAFreeText -** Returns slots `FreeText` and `Sentiment`

You can also add your own Amazon Lex bots and use them as response bots. Response chatbot names must start with the letters `QNA`. The solution calls your chatbot with the user's response, and captures all the slot names and values returned when your chatbot sends back a fulfilled message.

Advancing and branching through a series of questions

The following example configures the solution to automatically ask your age after you provide your name.

1. Sign in to the content designer and edit item `ElicitResponse.00.1`
2. Choose **Advanced**.
3. Enter **Document Chaining: Chaining Rule**: `ask my age`
4. Choose **UPDATE** to save the modified item.
5. Use the web UI to say: *`"Ask my name"`*
   - Enter and confirm your name.
   - Enter and confirm your age.

The solution automatically asks you for your age after you confirm your name. Because you specified the next question, `ask my age`, as the chaining rule, the solution automatically found and advanced to the matching item.

Next, create a *conditional* chaining rule that will branch to different items depending on previous answers.

1. Sign in to the content designer and add two new items:

- ID: `ElicitResponse.003`, question: `"Under 18"`, answer: `"Under 18 answer"`.
- ID: `ElicitResponse.004`, question: `"Over 18"`, answer: `"Over 18 answer"`.

2. Edit item `ElicitResponse.002`
   a. Add Chaining Rule: `(SessionAttributes.age_of_user.Age< 18) ? "Under 18" : "Over 18"`
   b. Choose **UPDATE** to save the modified item.
3. Use the web UI to ask: *`"Ask my name"`* .
   - Enter and confirm your name.
   - Enter and confirm your age.

When you confirm your age, the solution automatically branches to one of the two new items you added, depending on your age. The chaining rule is a JavaScript programming expression used to test the value of the session attribute set by elicit response; if it is less than 18 then advance to the item matching the question `"Under 18"`, otherwise advance to the item matching the question `"Over 18"`.

Combine expressions with logical operators to test multiple session attributes in a single rule, and use nested expressions to implement more than two branches in a chaining rule. Use the alternate syntax `SessionAttributes('age_of_user.Age')` to avoid a processing error if the referenced session attribute does not exist.

You can also apply chaining rule expressions to all the context variables supported by the Handlebars feature including `UserInfo` fields, `Settings` fields, and more. For a list of available variables, see the Handlebars section in GitHub repository.

Identify the next document using its QID value instead of a question using a string that starts with `QID::` followed by the QID value of the document, for example, a rule that evaluates to `QID::Admin001` will chain to item `Admin.001`.

You can optionally specify an AWS Lambda function instead of a JavaScript expression when you need to evaluate complex chaining rule logic. Your Lambda function is invoked with the full user request context and should evaluate and return the next question as a simple string. Alternatively, the Lambda function may return an event object where the `event.req.question` key was updated to specify the next question - by returning an event object, your `chaining rule` Lambda function can modify session attributes, similar to Lambda hooks. Use Lambda functions to implement chaining rules that require complex logic and data lookup. A `chaining`

`rule` Lambda function name must start with the letters "QNA", and is specified in the **Document Chaining:Chaining Rule** field as `Lambda::FunctionNameOrARN`.

> **Note**
>
> If the chaining rule has an error, the solution will return the message, *"Unfortunately I encountered an error when searching for your answer. Please ask me again later."*

Bot routing

Bots come in many shapes and sizes, and exist to perform a variety of automation tasks. Usually, they take input from a human and respond by performing a task. Bots might ask for additional input, verify the input, and respond with completion. You can implement bots by using Amazon Lex or other toolsets. An example is the nutritionix bot where you can tell the bot what you've had for breakfast and it responds with nutrition information.

QnABot on AWS coordinates (routes) bot requests through a supervisory bot, to the appropriate bot based on questions or tasks.

Content designers associate questions or tasks (QIDs) that identify a BotRouter to target for the question. This is performed using the **QnABot** content designer UI. Once configured, if a user asks a question or directs the bot with some instruction, QnABot on AWS responds with an answer and sets up a channel to communicate with the specialty bot. From that point, messages or responses from the user are delivered to the specialty bot. Specialty bots respond to actions and QnABot on AWS delivers the answers.

This flow continues until one of these events occurs:

1. The user cancels the conversation with the specialty bot by uttering *"exit"*, *"quit"*, *"bye"*, or a configurable phrase defined in the settings configuration of QnABot on AWS.
2. The specialty `BotRouter` (custom code) responds with a message indicating the conversation should be discontinued (`QNABOT_END_ROUTING`).
3. The specialty bot is a LexBot (non QnABot on AWS) that indicates fulfillment is complete.
4. If the target bot is another QnABot on AWS, session attributes can be set by the specialty QnABot on AWS set indicating the conversation should be discontinued (`QNABOT_END_ROUTING`).

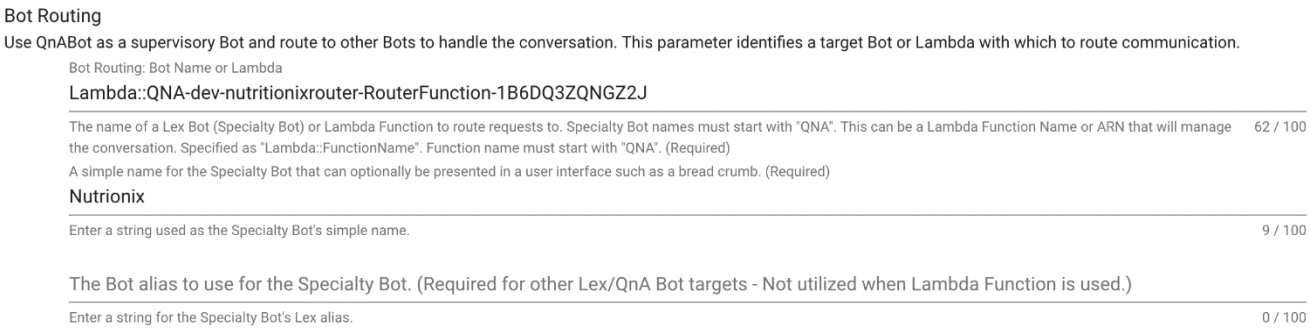
Specialty bots can be developed for specific parts of an organization like IT, or Finance, or Project Management, or Documentation. A supervisory bot at an enterprise level can direct users to answers from any of their bots.

Configuration of bot routing

Configuration of bot routing is simple. Each question in QnAbot on AWS contains an optional section which allows configuration of a `BotRouter`.

> **Note**
>
> This is optional. Leave empty and QnABot on AWS will not act as a `BotRouter` for the question being edited.

Bot routing



The example image shows an integration we've developed which communicates with the Nutritionix bot.

- Bot name or Lambda function - You can configure and existing Lex bot or configure a specialty `BotRouter` implemented via a Lambda function.
- Simple name - A short string that we expect web user interfaces to use as a breadcrumb to identify where in an enterprise the user is interacting.

> **Note**
>
> When integrating with other Amazon Lex bots or Lambda functions, the permission to communicate with the target Amazon Lex bot or with a new `BotRouter` Lambda function need to be added to the solution's `Fulfillment` Lambda role.

Message protocol for a new bot router implemented in Lambda

The input JSON payload to the target Lambda function is as follows:

```
req: {
        request: "message",
        inputText: <String>,
        sessionAttributes: <Object>),
        userId: <String>
    }
```

The expected response payload from the target Lambda function is the following:

```
{
    response: "message",
    status: "success", "failed"
    message: <String>,
    messageFormat:  "PlainText", "CustomPayload", "SSML", "Composite"
    sessionAttributes: Object,
    sessionAttributes.appContext.altMessages.ssml: <String>,
    sessionAttributes.appContext.altMessages.markdown: <String>,
    sessionAttributes.QNABOT_END_ROUTING: <AnyValue>
    responseCard: <standard Lex Response Card Object>
}
```

Sample bot router

The Nutrionix node.js based sample `BotRouter` is provided as a zip file in the GitHub repository. To use this sample you'll need to provision an API account with Nutritionix and configure the source to use your own `x-app-id` and `x-app-key` from Nutritionix.

```
'x-app-id': process.env.xAppId,
    'x-app-key': process.env.xAppKey
```

Next, you must build and deploy the code into Lambda using your favorite techniques and grant permission within the QnABot `Fulfillment` Lambda role using IAM to invoke this Lambda.

> **Tip**
>
> If you name the Lambda function starting with qna, QnABot on AWS is already configured with permissions to invoke this Lambda.

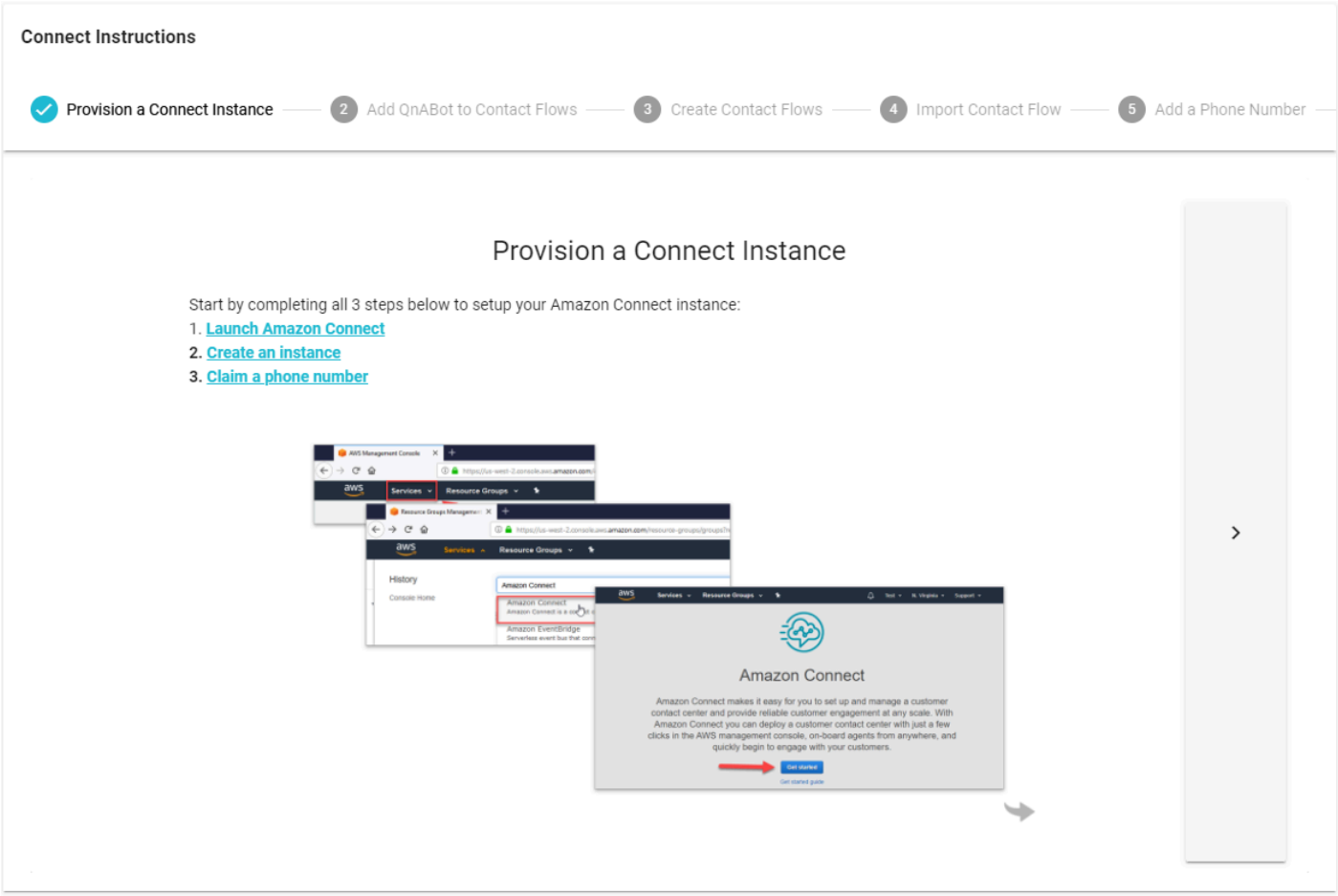
Connecting QnABot on AWS to an Amazon Connect call center

The solution can automate data collection and answer frequently asked questions using QnABot on AWS within an Amazon Connect contact flow. Optionally, you can also configure the solution to use Amazon Connect to make outbound calls; your users can use the web UI or the Alexa skill to ask QnABot on AWS to call their phone so they can speak to a human.

Using the Amazon Connect integration wizard, follow these steps to connect QnABot on AWS to a call center.

1. Sign in to the content designer, select the tools menu (☰), and then choose **Connect**.
2. Follow the step-by-step directions in the wizard to create a contact center using the solution to answer caller's questions.

   Amazon Connect integration wizard



Connecting QnABot on AWS to Genesys Cloud

> **Note**
>
> While QnABot on AWS provides integration with Genesys Cloud CX, you are responsible for integration testing and ensuring QnABot connectivity to Genesys Cloud works as expected. Work with a specialist if you have further questions.

1. Sign in to the content designer, select the tools menu (☰), and then choose **Genesys Cloud**.
2. Follow the step-by-step directions in the wizard to create a contact center using the solution to answer caller's questions.

Tuning, testing, and troubleshooting unexpected answers

You can use the content designer to tune, test, and troubleshoot answers to fix problems.

Tuning answers using the content designer

By default, QnABot on AWS attempts to match an end user's question to the list of questions and answers stored in Amazon OpenSearch Service. QnABot on AWS uses full text search to find the item that is the best match for the question asked. Words that are used infrequently score higher than words that are used often, so sentence constructs such as prepositions have lower weighting than unique keywords. The closer the alignment between a question associated with an item and a question asked by the user, the greater the probability that QnABot on AWS will choose that item as the most relevant answer.

The solution tries to find the best answer to questions by applying the keyword filters, and by matching the words used in the end user's question to the words used in the question fields of the stored answers—giving preference when the same words are used in the same order.

You might find that end users ask questions in ways that you haven't anticipated, resulting in unexpected answers being returned by QnABot on AWS. When this happens, you can use the content designer to troubleshoot and fix the problem.

For more information, see the Tuning Recognition Accuracy section in the GitHub repository.

Testing all your questions

Use the following procedure to test your questions.

1. Sign in to the content designer, and choose **TEST ALL.**
2. Use the default filename, or enter your own.
3. Optionally, if you want to test only a subset of questions, you can filter by the `qid` prefix. Leave this field blank to test all the questions.
4. Select **TEST ALL**, and wait for the tests to complete.
5. Select the **view results** icon on the bottom right to view the test results. Any incorrect matches are highlighted in red. Test results can be viewed in the browser, or downloaded to your computer as a CSV file.

Tuning the chatbot's ASR

When you ask QnABot on AWS a question, it is processed and transcribed by either Amazon Lex or Alexa using an ASR engine. QnABot on AWS initially trains the ASR to match a wide variety of possible questions and statements, so that the Amazon Lex chatbot and Alexa skill will accept almost any question a user asks.

This solution supports `AMAZON.FallbackIntent` in both Amazon Lex and Alexa, which allows it to process anything end users say without needing to retrain and rebuild the Amazon Lex chatbot or Alexa skill.

Occasionally, the transcription shown in the web client or the Alexa app isn't accurate. This can happen with unusual words that are confused for other more common words or phrases. Use one of the following approaches to troubleshoot this error:

- Use the content designer to add additional question variants that match the actual transcription shown in the web client or in the Alexa app; this allows QnABot on AWS to anticipate the transcription accuracy problem, and respond anyway.
- Retrain the Amazon Lex and Alexa ASR with examples to influence the transcription to more closely match what you want - see **Retrain Amazon Lex** for more information.

Retrain Amazon Lex

QnABot on AWS can automatically generate additional ASR training data for Amazon Lex using questions from the data you have added.

1. Sign in to the content designer and choose **LEX REBUILD** from the top right edit menu (⋮) **0**
2. Wait for the rebuild to complete.

Retrain Alexa

QnABot on AWS can generate additional ASR training data for Alexa using questions from the data you have added.

1. Sign in to the content designer and choose **ALEXA UPDATE** from the top right edit menu (⋮).
2. Select **COPY SCHEMA** to copy the updated Alexa skill schema.
3. Sign in to the Alexa Developer Console, open your QnABot on AWS skill, and then use the JSON editor to paste the new schema, replacing the existing one.

4. Save and then build the updated model.

Monitoring QnABot on AWS usage and user feedback

The solution logs everything that end users say to the chatbot. Amazon Data Firehose stores logged utterances to a new index in Amazon OpenSearch Service.

You can also allow your end users to provide feedback about the chatbot's answers. Use the following procedure to set up the feedback mechanism.

1. From the content designer's top left tools menu (☰), select **Import**.
2. Open **Examples/Extensions**, and select the **LOAD** for the **QnAUtility** demo.
3. From the top left tools menu (☰), select **EDIT** and examine the newly imported items, `Feedback.001` and `Feedback.002`; observe the list of default expressions that the end user can input to invoke feedback. (The example **QnAUtility** demo package also loads `Help`, `CustomNoMatches`, `CustomNoVerifiedIdentity` items.)
4. Test the feedback mechanism.

Use the web UI to ask a question, such as: *"What happens if I ask an unanticipated question?"*. Because we have not entered a suitable answer for this question, QnABot on AWS responds with the newly imported `CustomNoMatches` response—indicating that it doesn't know the answer.

From the web UI, say or type *"Thumbs down"*, or select the *Thumbs down* icon beside the answer.

The solution publishes *Thumbs down* feedback messages to the Amazon Simple Notification Service (SNS) topic identified by the **FeedbackSNSTopic** on the **Outputs** tab of the CloudFormation stack. To learn how to subscribe to the SNS topic, and receive a message from the each time a user provides feedback, see Subscribing to an Amazon SNS topic in the *Amazon Simple Service Notification Developer Guide.*

If you have selected a non-English language for your QnABot on AWS deployment and have multi-language enabled then you should do the following steps:

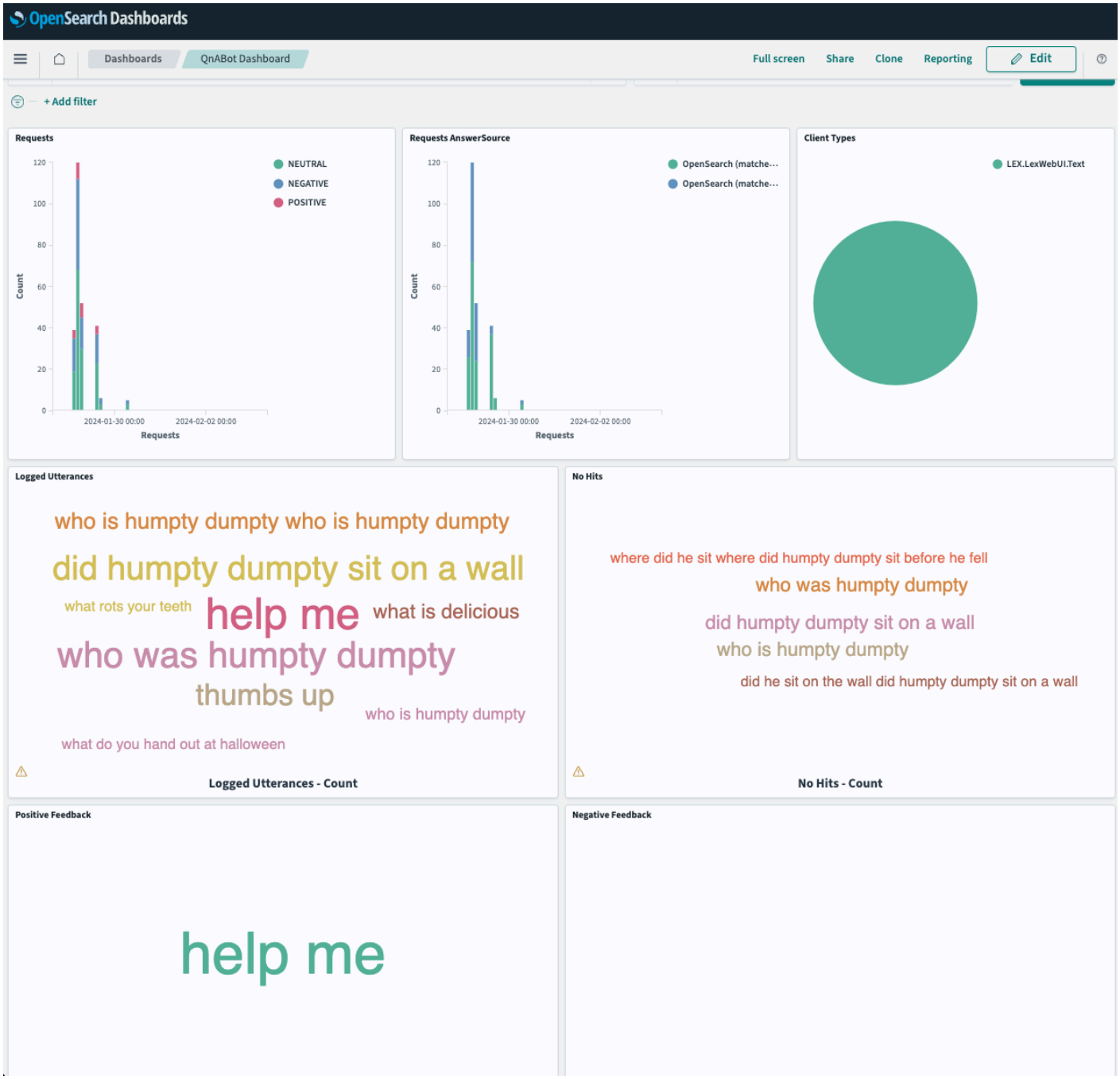
1. Use the AWS Translate API to translate *Thumbs up* and *Thumbs down* to your deployment language, if it is not English.
2. Add the translation of the Thumbs up and down as a question in your QnABot on AWS deployment.

3. Go to the content, select the top left tools menu (☰), and choose **Settings**.
4. Find the PROTECTED_UTTERANCES variable and insert that phrase in by adding a comma, then enter the translation.

Use the following process to visualize the usage logs and feedback using OpenSearch Dashboards. Note that it can take up to 5 minutes for new utterances and end user feedback to become visible in the dashboard.

1. From the content designer, select the top left tools menu (☰), and choose **OpenSearch*Dashboards** and it opens in a new browser tab.
2. Choose the **QnABot on AWS** dashboard to visualize usage history and sentiment, all logged utterances, no hits utterances, positive user feedback, and negative user feedback.

   Sample OpenSearch Dashboard



3. Edit OpenSearch Dashboards to change the time span, customize and build your own visualizations, or to run your own queries.

Using Amazon CloudWatch to monitor and troubleshoot

The solution's metrics and logs are available in an Amazon CloudWatch dashboard. Use the following procedure to launch the dashboard and visualize the solution's AWS resources.

1. From the CloudFormation stack's **Outputs** tab, select the **CloudWatchDashboardURL** link.
2. When troubleshooting the chatbots responses to your questions, trace the request and response using the logs created by the `Fulfillment` Lambda function.
   a. Choose the **menu** tool in the upper right of the `Fulfillment` Lambda widget, select **View logs**, then and choose the **AWS Lambda** function **0**

   FulfillmentLambda function

   b. Inspect the log messages. Each interaction with the solution is delimited by **START** and **END** messages. Between these messages are insights into how the solution processes the question.

Use Log Insights to query logs from CloudWatch groups

1. Go to **Log Insights** in Amazon CloudWatch and select the log group you would like to query.
2. Create your query in **Log Insights**.

   Example query for Fulfillment Lambda logs to query for any errors:

```
filter @message like /(?i)(Exception|error|KeyError)/
| fields @timestamp, @message
| sort @timestamp desc
| limit 200
```

Importing and exporting chatbot answers

The solution's content designer allows you to export and import your content using JSON and Excel files.

Use the export feature to create backup versions of your content that you can use to restore if you accidentally delete items or need to go back to a previous version. You can also use the exported files to load content into another instance of your chatbot to help with test deployments.

Follow these steps to export all the items that are in the QnABot on AWS category (Item IDs starting with `AWS-QnABot`).

1. Sign in to the content designer, choose the tools menu (☰), and then choose **Export.**
2. Enter `AWS-QnABot` in the optional filter field, and then choose **EXPORT** to generate a JSON file containing the filtered items.
3. After the export has completed, choose the download tool (bottom right) to download the exported file*.*
4. Open the exported file in a text editor and inspect the JSON structure.

   Sample JSON file for importing and exporting chatbot answers:

```
{
  "qna": [
    {
      "q": [
        "What is Q and A Bot"
      ],
      "a": "The Q and A Bot uses Amazon Lex and Alexa to provide a natural language
 interface for your FAQ knowledge base, so your users can just ask a question and get
 a quick and relevant answer.",
      "r": {
        "title": "",
        "imageUrl": ""
      },
      "qid": "QnABot.001"
    },
    {
      "q": [
        "How do I use Q and A Bot"
      ],
```

```
      "a": "Create and administer your questions and answers using the Q and A Bot
 Content Designer UI. End users ask questions using the Lex web UI, which supports
 voice or chat, or using Alexa devices for hands free voice interaction. ",
      "r": {
        "title": "",
        "imageUrl": ""
      },
      "qid": "QnABot.002"
    }
  ]
}
```

5. Add a new item to the qna list, as shown in the following example, and save the file.

```
{
        "qid": "AWS-QnABot.003",
        "q": ["What can Q and A bot do"],
        "a": "You can integrate it with your website to provide quick and easy access
 to frequently asked questions. Use it with Alexa to provide hands free answers in
 the kitchen, in the factory or in the car. Since it can display images too, use it
 to provide illustrations and photographs to enrich your answers.",
        "r": {
            "title": "",
            "imageUrl": ""
        }
}
```

6. From the content designer, select **Import/Export**, and then choose **From File**.
7. Import your modified JSON file. Importing items with the same ID as an existing item will overwrite the existing item with the definition contained in the JSON file.
8. From the content designer, enter `AWS-QnABot` in the filter field, and inspect the newly imported item, **AWS-QnABot.003**.

For a step-by-step procedure on importing Excel (`0xlsx`) workbooks, see Excel workbooks import in the GitHub repository.

Modifying configuration settings

The solution uses a DynamoDB Table to hold default and custom configuration settings. You can view and edit these settings using the Settings menu in the content designer.

Explore the available configuration settings, and override the defaults to configure the solution's customize keyword filtering, answer field scoring, messages, redaction from logs and metrics (**ENABLE_REDACTING** and **REDACTING_REGEX**), and more. You can also enable the debug mode (**ENABLE_DEBUG_RESPONSES** and **ENABLE_DEBUG_LOGGING**), initiate fuzzy matching (**ES_USE_FUZZY_MATCH**), and experiment with score boosting for exact phrase matches (**ES_PHRASE_BOOST**). Follows are a set of example of settings frequently use. For more complete information on the settings, see QnABot Settings in the GitHub repository.

> **Note**
>
> Custom settings are kept when you upgrade the solution.

Configure keyword filters feature

1. Sign in to the content designer, select the tools menu (☰), and then choose **Settings**.
2. Change the value of the **ES_USE_KEYWORD_FILTERS** setting from `true` to `false`.
3. Scroll to the bottom and select **Save**. This turns off the new keyword filters feature.

You can further customize how the keyword filters feature works by changing the following settings:

- **ES_KEYWORD_SYNTAX_TYPES** - A list of tokens representing parts of speech identified by Amazon Comprehend.
- *ES_MINIMUM_SHOULD_MATCH * - A query rule used to determine how many keywords must match an item question in a valid answer.

Configure words and phrases replacement in user questions

If you want to replace words or phrases in user questions, for example, you want *Thumbs up* rewritten to be a direct match by question ID, you can use the

SEARCH_REPLACE_QUESTION_SUBSTRINGS setting.

1. Sign in to the content designer, select the tools menu (☰), and then choose **Settings**.

2. Change the value of the **SEARCH_REPLACE_QUESTION_SUBSTRINGS** setting to a JSON object, such as `{"Thumbs Down": "QID::Feedback.001", "Thumbs Up": "QID::Feedback.002"}`. You can add additional pairs separated by commas.
3. Scroll to the bottom and select **Save**. This will now rewrite all input matching `"Thumbs Down"` to `"QID::Feedback.001"`.

Configure pre-processing and post-processing Lambda hooks

The content designer enables you to dynamically generate answers by letting you specify your own Lambda hook function for any item/question defined in the content designer (hook). In addition, a Lambda hook can be called as the first step in the fulfillment pipeline (PREPROCESS) or after processing has completed (POSTPROCESS) and before the userInfo is saved to DynamoDB and the result has been sent back to the client.

You can add pre-processing and post-processing Lambda hooks (that run before preprocessing and after every question is run) via the Settings page.

1. Sign in to the content designer, select the tools menu (☰), and then choose **Settings**.
2. Find the **LAMBDA_PREPROCESS_HOOK** setting and set its value to your hook name. The name of the Lambda must start with qna- or QNA- to comply with the permissions of the role attached to the Fulfillment Lambda, for example, QNA-ExampleJSLambdahook.
3. Scroll to the bottom and select **Save**. The Lambda function specified will now be run before each question is processed_._

> **Note**
>
> For more information on Lambda hooks, see the Extending QnABot with Lambda hook functions and the QnABot Settings sections in the GitHub repository.

Configure multi-language support

QnABot on AWS supports both voice and text interactions in multiple languages. QnABot can detect the predominant language in an interaction by using Amazon Comprehend, a NLP service that uses machine learning to find insights and relationships in text. The bot then uses Amazon

Translate, a neural machine translation service to convert questions and answers across languages from a single shared set of FAQs and documents.

By default the multi-language feature is disabled. QnABot on AWS uses a property named **ENABLE_MULTI_LANGUAGE_SUPPORT** with a default value of `false`. You can change this setting using the content designer **Settings** page. Set it to `true` to enable multi-language support.

QnABot on AWS uses Amazon Translate to convert the question posed by the user to your core language that was chosen during your deployment. It performs a lookup of the answer in Amazon OpenSearch Service just as it normally does, using the native language translation of the question.

Searches are done in the language you have selected for your deployment only since QnABot on AWS documents are indexed using the language analyzer and their corresponding text analyzer for example, stemming and stop words. Once it finds the question, QnABot serves up the configured answer.

When you are in multi-language mode, consider letting the user choose their preferred language at the beginning of the chat and then have the whole conversation session from that point on to be only in the preferred language. Use Handlebars to do this. For details, see Integrating Handlebars templates.

You must enter a question into the QnA question bank with an utterance that is the name of the language that you are trying to set as preference, such as Spanish, coupled with the `\#setLang` Handlebar in the answer of that utterance. This utterance or question must be invoked at the point where you want to set the conversation to the preferred language. For example, you can import the **Language / Multiple Language Support** sample or extension from the QnABot **Import** menu option. This adds two questions to the system: `Language.000` and `Language.001`. The first question allows the end user to set their preferred language explicitly to a list of supported languages; the latter resets the preferred language and allows QnABot to choose the locale based on the automatically detected predominant language.

> **Note**
>
> Button values in the response cards are still displayed with their original value as input in the chat conversation.

When deploying the QnABot on AWS solution (version 5.5.0 and higher) CloudFormation template, there will be a **Language** parameter in which you have the option of selecting one of

the 33 languages. This **Language** parameter is used as the core language for your QnABot on AWSdeployment. The Language Analyzer for your Opensearch index setting uses the language that you have specified in this parameter. In the case that your input has a low confidence rate, it defaults to English since that is the backup language.

- Custom terminology also supports your `NATIVE_LANGUAGE`.
- If using the thumbs up and down feature, you should translate *thumbs up* and *thumbs down* into your native language and put that phrase in the `PROTECTED_UTTERANCES` setting. This is to prevent it from being treated as a question by the solution. To do this, complete the following steps:
  a. Use the AWS translate API to translate *thumbs up* and *thumbs down* to your deployment language, if it is not English.
  b. Add the translation of *thumbs up* and *thumbs down* in the website client config file inside your QnABot on AWS code and deploy.
  c. Add the translation of the *thumbs up* and *thumbs down* as a question in your QnABot deployment.
  d. Go to the content designer, navigate to the top left and select **Settings**.
  e. Find the `PROTECTED_UTTERANCES` variable and insert that phrase in by adding a comma, and then enter the translation.
- PII redaction will still be for English, since that is still accurate with other languages.
- Changing the `NATIVE_LANGUAGE` should always be done from the CloudFormation stack by changing the **Language** parameter.

When creating an Amazon Kendra web crawling data source from the **QnABot** UI, it will be created in the native language specified in your CloudFormation parameters. If the specified native language is not supported by Amazon Kendra, English will be used as the default language.

When querying within your Amazon Kendra data source, the following logic will be applied to determine the language used for querying:

1. The algorithm will determine the user's locale and use the `shouldUseOriginalLanguageQuery()` function to decide whether to query in the user's native language or the locale's language.
2. Based on the result from `shouldUseOriginalLanguageQuery()`, it will either:
   - Use the locale's language if it is supported by Amazon Kendra.

- If the locale's language is not supported, it will check if the native language (the language chosen in the CloudFormation parameters) is supported by Amazon Kendra.

3. If neither the locale's language nor the native language is supported by Amazon Kendra, English will be used as the default language for querying.

In summary, the algorithm tries to use the user's preferred language (either the locale or the native language specified in the CloudFormation parameters) if it is supported by Amazon Kendra. If neither language is supported, English is used as the fallback language for querying the Amazon Kendra data source.

For more information, see the MultiLanguage Support section in the GitHub repository or the Multi Language section in the QnABot Workshop.

Using automatic translation

This solution supports automatic translation to the end user's language using Amazon Translate.

1. Turn on multiple-language support by setting: **ENABLE_MULTI_LANGUAGE_SUPPORT** to `TRUE`*0*
2. In the web UI, ask: _Qu'est-ce que q et a bot? _
3. The chatbot replies to you in French.

The solution also supports speech recognition and voice interaction in multiple languages. When you install or update QnABot on AWS, specify the languages using the **LexV2BotLocaleIds** CloudFormation parameter. The default languages are US English, US Spanish, and Canadian French, but you can customize the list to use any of the languages supported by Amazon LexV2.

Use the **ENABLE_DEBUG_RESPONSES** setting to see how local language questions are translated to English by QnABot on AWS. Use this translation to tune the content as needed to ensure that QnABot on AWS finds the best answer to a non-English question.

The solution also supports Amazon Translate custom terminology to provide additional control over the translation of entities and phrases. Custom terminology supports the language that you are deploying with. For more information on how to use the Import Custom Terminology tool in the content designer, see the Using Custom Terminologies with Amazon Translate section in the GitHub repository.

Configure personally identifiable information (PII) rejection and redaction

QnABot on AWS can detect and redact personally identifiable information (PII) using Amazon Comprehend and regular expressions.

If **ENABLE_REDACTING** is set to `true`, the Comprehend detected PII entities will also be redacted from Amazon CloudWatch logs and OpenSearch Service logs.

PII rejection and redaction

ENABLE_REDACTING
true

REDACTING_REGEX
\b\d{4}\b(?![-])|\b\d{9}\b|\b\d{3}-\d{2}-\d{4}\b

ENABLE_REDACTING_WITH_COMPREHEND
true

COMPREHEND_REDACTING_CONFIDENCE_SCORE
0.99

COMPREHEND_REDACTING_ENTITY_TYPES
ADDRESS,EMAIL,SSN,PHONE,PASSWORD,BANK_ACCOUNT_NUMBER,BANK_ROUTING,CREDIT_DEBIT_NUMBER

PII_REJECTION_ENABLED
false

PII_REJECTION_QUESTION
pii_rejection_question

PII_REJECTION_REGEX
\b\d{4}\b(?![-])|\b\d{9}\b|\b\d{3}-\d{2}-\d{4}\b

PII_REJECTION_ENTITY_TYPES
ADDRESS,EMAIL,SSN,PHONE,PASSWORD,BANK_ACCOUNT_NUMBER,BANK_ROUTING,CREDIT_DEBIT_NUMBER

PII_REJECTION_CONFIDENCE_SCORE
0.99

DISABLE_CLOUDWATCH_LOGGING
false

For more information, see Personally Identifiable Information (PII) Rejection and Redaction in the GitHub repository.

Integrating Amazon Kendra

Amazon Kendra is an intelligent search service powered by machine learning. There are two ways to take advantage of Amazon Kendra's NLP model to enhance the solution's ability to understand human questions:

1. Use Amazon Kendra's FAQ queries to match users' questions to the answers in the solution's knowledge base. Amazon Kendra's machine learning models can handle many variations in how users phrase their questions, and this can reduce the amount of tuning needed for the solution to find the right answer from your knowledge base.
2. Use Amazon Kendra's document index as a fallback source of answers when a question/answer is not found in the solution's knowledge base.

For more information, see Amazon Kendra Pricing and Getting started in the *Amazon Kendra Developer Guide* to create your Amazon Kendra index_._

Using Amazon Kendra FAQ for question matching

Use the following procedure to configure the solution to use your Amazon Kendra index to answer questions from the data populated in the content designer:

1. Set the **KendraFaqIndexId** CloudFormation parameter to the ID of the Amazon Kendra index to use. Find the index ID in the Amazon Kendra console.
2. Replicate all items from the content designer to the Amazon Kendra index:
   a. Select the menu (⋮) from the top right in the content designer.
   b. Choose **SYNC KENDRA FAQ** and wait for it to complete - it might take a few minutes.

The solution will now use Amazon Kendra FAQ queries to find matches to end users' questions. Use the **ALT_SEARCH_KENDRA_FAQ_CONFIDENCE_SCORE** setting to adjust the confidence threshold for Amazon Kendra FAQ answers used by QnABot on AWS.

If Amazon Kendra FAQ cannot find an answer that meets the confidence threshold, the solution will revert by default to using an Amazon OpenSearch Service query. The combination of Amazon Kendra FAQ and Amazon OpenSearch Service gives you the best of both worlds.

> **Note**
>
> When adding your **Amazon KendraFaqIndexId** in CloudFormation, also add the index ID in **AltSearchAmazon KendraIndexes**.

Using Amazon Kendra search as a fallback source of answers

You can add one or more data sources to your Amazon Kendra index, and configure the solution to query your index any time it gets a question that it doesn't know how to answer.

- Set the **AltSearchAmazon KendraIndexes** CloudFormation parameter to specify one or more Amazon Kendra indexes to use for fallback searches.

The value of **AltSearchAmazon KendraIndexes** parameter should be specified as a string containing index IDs separated by comma, for example:

```
857710ab-example-do-not-copy
```

- Or -

```
857710ab-example1-do-not-copy,857710ab-example2-do-not-copy
```

QnABot on AWS also supports Amazon Kendra index authentication token pass through.

- Set the **AltSearchAmazon KendraIndexes** CloudFormation parameter to specify one or more Amazon Kendra indexes to use for fallback searches. You must control user access to documents with tokens using OpenID. For more information, see the Amazon Kendra Fallback Function section in the GitHub repository.
- Set the **AltSearchAmazon KendraIndexAuth** CloudFormation parameter to `TRUE`. This enables QnABot to send an OpenID Token to Amazon Kendra index(es) to limit Amazon Kendra results to which the user is entitled.
- Input the **IDENTITY_PROVIDER_JWKS_URLS** QnABot content designer settings parameter. Find your token key signing URL from the Cognito user pool of QnABot or `Lex-Web-Ui`.

> **Note**
>
> When configuring your Amazon Kendra index with user access control, Amazon Kendra only allows you to specify one signing key URL from one Cognito user pool. Having multiple Cognito pools, including `Lex-Web-Ui`, requires you to set up multiple Amazon Kendra indexes.

Amazon Kendra redirect

QnABot on AWS supports multiple mechanisms for dynamic interaction flows. For example:

- Using Lambda hooks in a given Item ID to perform additional actions, such as creating a ticket, resetting a password, and saving data to a data store.
- Using an Amazon Kendra index as a fallback mechanism to look for answers to user's questions.

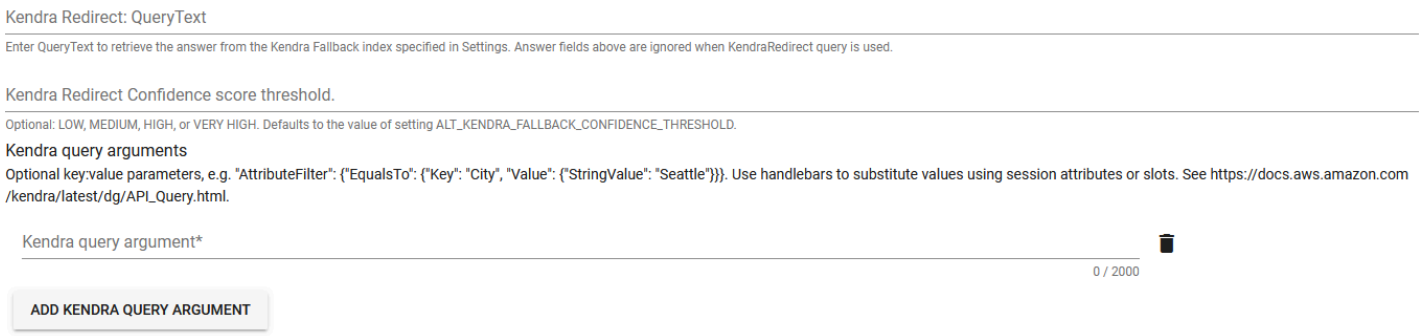
There are various options to process Amazon Kendra queries. One option is to create a custom Lambda hook and map it to an Item ID. The Lambda hook then includes the business logic to use an Amazon Kendra index and process the query.

The Amazon Kendra redirect feature provides a much simpler option. You can include an Amazon Kendra query within an Item ID, and QnABot will do the rest to process the Amazon Kendra request and respond back with the results.

Configuring an Item ID with Amazon Kendra redirect

You can configure an Item ID with Amazon Kendra Redirect UI.

Amazon Kendra redirect configuration



1. Create a QnABot question as you would normally do by providing an Item ID and questions and utterances.
2. Expand the **Advanced** option.
3. **Amazon Kendra Redirect: Query Text** accepts a QueryText to search for (for example `what is q and a bot`) and retrieve the answer from the Amazon Kendra fallback index specified in the CloudFormation stack parameters. Amazon Kendra searches your index for text content, question, and answer (FAQ) content. You can also use Handlebars to substitute values using session attributes or slots to support dynamic queries.
4. **Amazon Kendra Redirect: Confidence** score threshold provides a relative ranking that indicates how confident Amazon Kendra is that the response matches the query. This is an optional field having one of the values of: `LOW`, `MEDIUM`, `HIGH`, `VERY HIGH`. If no value is provided, the value for the **ALT_KENDRA_FALLBACK_CONFIDENCE_THRESHOLD** setting is used.
5. **Amazon Kendra query arguments** is an optional field that allows filtered searches based on document attributes, for example, `"AttributeFilter": {"EqualsTo": {"Key": "City", "Value": {"StringValue": "Seattle"}}}`. You can also use Handlebars to substitute values using session attributes or slots to support dynamic queries.

For more information on using Amazon Kendra query arguments, see the Amazon Kendra Query API in the *Amazon Kendra__API Reference*.

Note

- Answer fields are ignored when `Amazon KendraRedirect` query is used.
- Use this feature for use cases where you have Item IDs that directly need to interact with an Amazon Kendra index as configured in the CloudFormation stack parameters.
- When applying Amazon Kendra query arguments, check if the document fields are searchable. **Searchable** determines whether the field is used in the search. For more information, see Mapping data source fields in the _Amazon Kendra_*Developer Guide*.

Web page indexer

This solution can answer questions based on the content of web pages.

1. In the CloudFormation stack, set the **Amazon KendraWebPageIndexId** parameter to `Existing Amazon Kendra Index ID`. Add the same index ID for the **AltSearchAmazon KendraIndexes** parameter.
2. From the content designer, select the tools menu (☰), and then choose **Settings.**
3. Modify the following settings:
   a. **ENABLE_WEB_INDEXER:** true
   b. *KENDRA_INDEXER_URLS: * link:https://aws.amazon.com/lex/faqs/
   c. **KENDRA_INDEXER_SCHEDULER:** https://docs.aws.amazon.com/AmazonCloudWatch/latest/events/ScheduledEvents.html#RateExpressions
4. From the content designer, select the tools menu (☰), and then choose **Amazon Kendra Web Crawler**.
   a. Choose **START INDEXING**.
   b. Wait for indexing to complete. It can take several minutes.
5. Open the web UI, and ask *`"What is Lex?"`* . QnABot on AWS provides an answer with a link to the Amazon Lex FAQ page.

For more information on web page indexing, see the README.md file in the GitHub repository.

Semantic question matching using text embeddings LLM

> **Note**
>
> This is an optional feature available as of v5.3.0. We encourage you to try it out on non-production instances initially to validate expected accuracy improvements and to test for any regression issues. See the Cost section to see estimates of how this feature affects pricing.

QnABot on AWS can use text embeddings to provide semantic search capabilities by using LLMs. The goals of these features are to improve question matching accuracy while reducing the amount of tuning required when compared to the default OpenSearch keyword-based matching. Some of the benefits include:

- Improved FAQ accuracy due to semantic matching compared to keyword matching (comparing the meaning of questions as opposed to comparing the individual words).

- Fewer training utterances are required to match a diverse set of queries. This results in significantly less tuning to get and maintain good results.
- Better multi-language support because translated utterances only need to match the original question's meaning, not the exact wording.

For example, with semantic matching activated, *"What's the address of the Whitehouse?"* matches to *"Where does the president live?"* and *"How old are you?"* would match with *"What is your age?"* . These examples won't match using the default keywords because they don't share any of the same words.

To enable these expanded semantic search capabilities, QnABot can use:

- Select from several embeddings models provided by Amazon Bedrock using the **EmbeddingsBedrockModelId** Cloudformation parameter. These models provide the best performance and operate on a pay-per-request model. To learn more about supported regions for Bedrock, please refer to Bedrock Model support by AWS Region in the Amazon Bedrock user guide.
- Embeddings from a user provided custom Lambda function.

> **Note**
>
> By choosing to use the generative responses features, you acknowledge that QnABot on AWS engages third-party generative AI models that AWS does not own or otherwise has any control over ("Third-Party Generative AI Models"). Your use of the Third-Party Generative AI Models is governed by the terms provided to you by the Third-Party Generative AI Model providers when you acquired your license to use them (for example, their terms of service, license agreement, acceptable use policy, and privacy policy).
> You are responsible for ensuring that your use of the Third-Party Generative AI Models comply with the terms governing them, and any laws, rules, regulations, policies, or standards that apply to you.
> You are also responsible for making your own independent assessment of the Third-Party Generative AI Models that you use, including their outputs and how Third-Party Generative AI Model providers use any data that may be transmitted to them based on your deployment configuration.

AWS does not make any representations, warranties, or guarantees regarding the Third-Party Generative AI Models, which are "Third-Party Content" under your agreement with AWS. QnABot on AWS is offered to you as "AWS Content" under your agreement with AWS.

Enabling embeddings support

Using Amazon Bedrock model (Preferred)

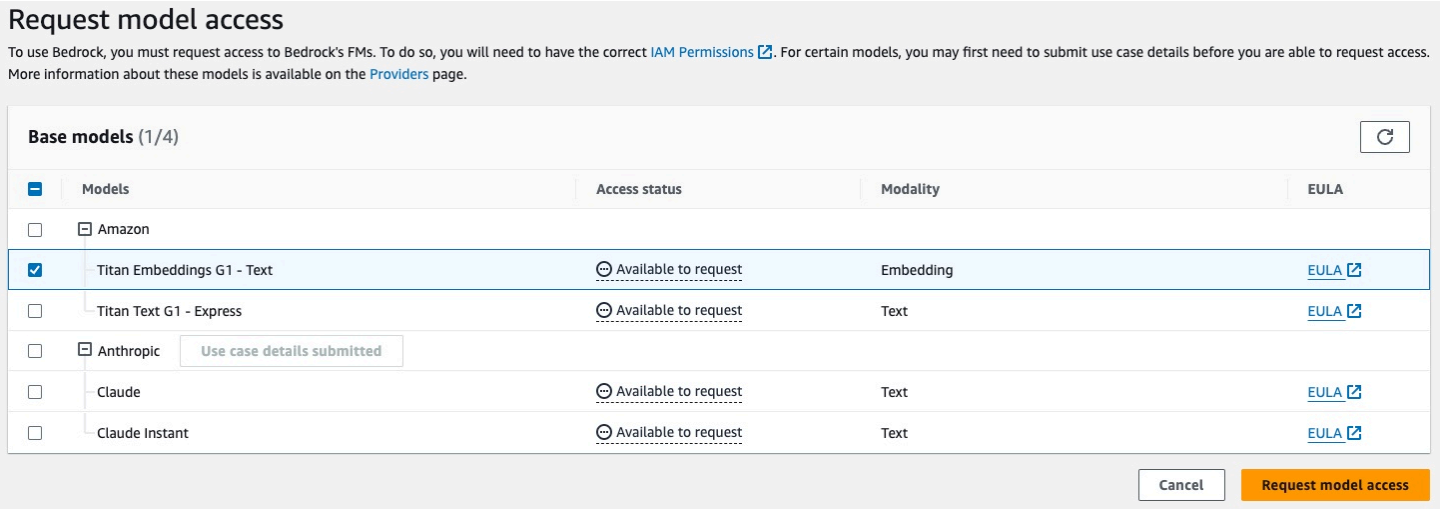
Utilizes one of the Amazon Bedrock foundation models to generate text embeddings. Currently, the following embeddings models are supported by QnABot on AWS:

- Amazon Titan embeddings G1
- Cohere English
- Cohere Multilingual
- Amazon Titan Text Embeddings V2

Note

Access must be requested for the Amazon Bedrock embeddings model that you want to use. This step must be performed for each account and Region where QnABot on AWS is deployed. To request access, navigate to Model Access in the Amazon Bedrock console. Select the models you need access to and request access.

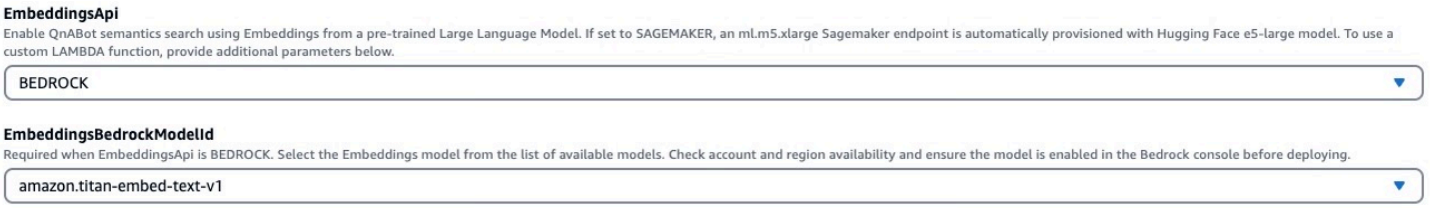
Request Amazon Bedrock embeddings



From the CloudFormation console, set the following parameters:

- Set **EmbeddingsAPI** to BEDROCK.
- Set **EmbeddingsBedrockModelId** to one of the three options.

Configure Amazon Bedrock embeddings.



Using a custom Lambda function

Users that want to explore alternate pre-trained or fine-tuned embeddings models can integrate a custom built Lambda function. By using a custom Lambda function, you can build your own embeddings model or even choose to connect to an external embeddings API.

> **Note**
>
> If integrating your Lambda function with external resources, evaluate the security implications of sharing data outside of AWS.

To begin, you must create a Lambda function. Your custom Lambda function should accept a JSON object containing the input string and return an array, which contains the embeddings. Record the

length of your embeddings array because you need it to deploy the stack (this is also referred to as the *dimensions*).

Lambda event input:

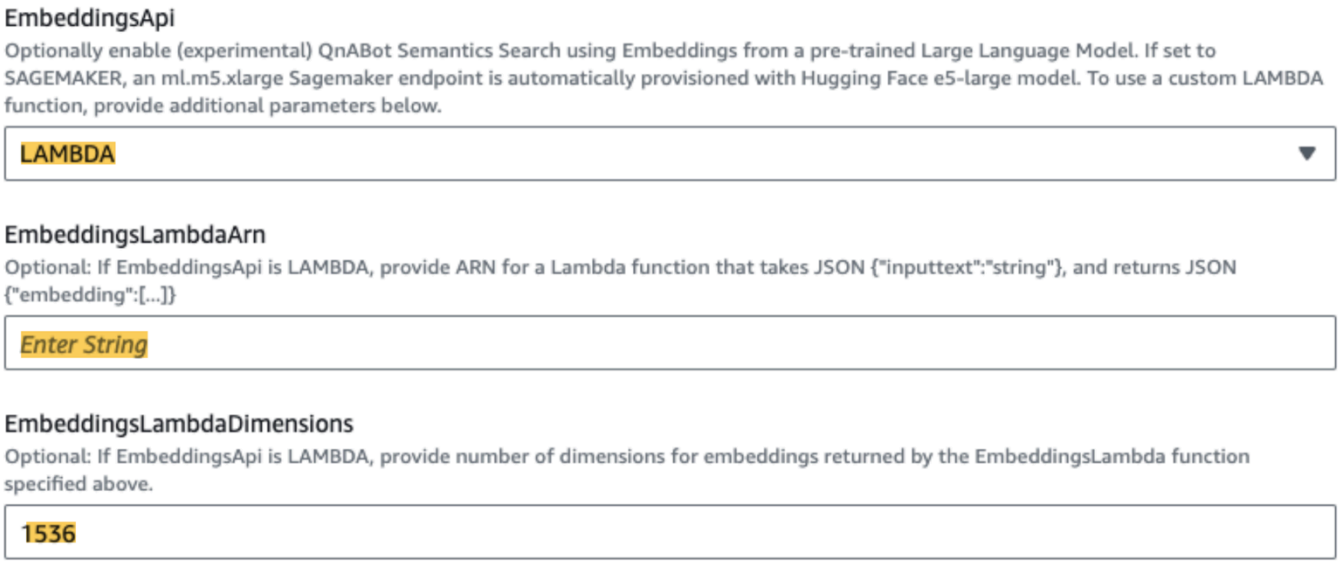
```
{
  // inputtype has either a value of 'q' for question or 'a' for answer
  "inputType": "string",

  // inputtext is the string on which to generate your custom embeddings
  "inputText":"string"
}
```

Expected Lambda JSON return object:

```
{"embedding": [...] }
```

When your Lambda function is ready, you can deploy the stack. To activate your Lambda function for embeddings, deploy the stack with **EmbeddingsAPI** set to LAMBDA. You must also set **EmbeddingsLambdaArn** to the ARN of your Lambda function and **EmbeddingsLambdaDimensions** to the dimensions returned by your Lambda function.

Semantic search with Lambda function .



Note

You can't change these settings through the content designer **Settings** page. To correctly reconfigure your deployment, update your CloudFormation stack to modify these values.

Settings available for text embeddings

When your QnABot stack is installed with **EmbeddingsApi** enabled, you can manage several different settings through the content designer **Settings** page:

- **EMBEDDINGS_ENABLE** - To turn on and off use of semantic search using embeddings:
  - Set to `FALSE` to turn off the use of embeddings-based queries.
  - Set to `TRUE` to activate the use of embeddings-based queries after previously setting it to `FALSE`.

Note

- Setting `TRUE` when the stack has **EmbeddingsAPI** set to `DISABLED` will cause failures since the QnABot on AWS stack isn't provisioned to support generation of embeddings.
- **EMBEDDINGS_ENABLE** will be set default to `TRUE`, if **EmbeddingsAPI** is provisioned to `LAMBDA`. If not provisioned, **EMBEDDINGS_ENABLE** will be set default to `FALSE`.

If you disable embeddings, you will likely also want to re-enable keyword filters by setting **ES_USE_KEYWORD_FILTERS** to `TRUE`.

If you add, modify, or import any items in the content designer when **EMBEDDINGS_ENABLE** is set to `FALSE`, then embeddings won't get created and you'll need to re-import or re-save those items after re-enabling embeddings.

This setting allows you to toggle embeddings on and off, it does not manage the underlying infrastructure. If you choose to permanently turn off embeddings, update the stack as well.

Important

If you update or change your embeddings model, for example, from Amazon Titan Embeddings G1 to Cohere English, or change **EmbeddingsApi** the embedding dimensions need to be recalculated. QnABot on AWS will need to export and re-import the Q&As in your content designer; however, we recommend backing up the Q&As using export before making this change. If any discrepancies occur, they can be addressed by import of exported Q&As.

- **ES_USE_KEYWORD_FILTERS** - This setting should now default to FALSE. Although you can use keyword filters with embeddings based semantic queries, they limit the power of semantic search by forcing keyword matches (preventing matches based on different words with similar meanings).
- **ES_SCORE_ANSWER_FIELD** - If set to TRUE, QnABot on AWS runs embedding vector searches on embeddings generated on answer field if no match is found on question fields. This allows QnABot to find matches based on the contents on the answer field as well as the questions. Only the plaintext answer field is used (not the Markdown or SSML alternatives). Tune the individual thresholds for questions and answers using the additional settings of:
  - **EMBEDDINGS_SCORE_THRESHOLD**
  - **EMBEDDINGS_SCORE_ANSWER_THRESHOLD**
- **EMBEDDINGS_SCORE_THRESHOLD** - Change this value to customize the score threshold on *question* fields. Unlike regular OpenSearch queries, embeddings queries always return scores between 0 and 1, so we can apply a threshold to separate good from bad results.
  - If no question has a similarity score above the threshold set, the match gets rejected and QnABot reverts to:
    i. Trying to find a match using the answer field (only if **ES_SCORE_ANSWER_FIELD** is set to TRUE).
    ii. Amazon Kendra fallback (only if enabled)
    iii. **no_hits**

  The default threshold is 0.7 for BEDROCK, but you can modify this based on your embeddings model and your experiments.

> **Tip**
>
> Use the content designer **TEST** tab to see the hits ranked by score for your query results.

- **EMBEDDINGS_SCORE_ANSWER_THRESHOLD** - Change this value to customize the score threshold on *answer* fields. This setting is only used when **ES_SCORE_ANSWER_FIELD** is set to TRUE and QnABot has failed to find a suitable response using the question field.
  - If no answer has a similarity score above the threshold set, the match gets rejected and QnABot reverts to:
    i. Amazon Kendra fallback (only if enabled)
    ii. **no_hits**

  The default threshold is 0.8, but you can modify this based on your embeddings model and your experiments.

> **Tip**
>
> Use the content designer **TEST** tab and select the **Score on answer field** checkbox to see the hits ranked by score for your answer field query results.

- **EMBEDDINGS_TEXT_PASSAGE_SCORE_THRESHOLD** - Change this value to customize the passage score threshold. This setting is only used if **ES_SCORE_TEXT_ITEM_PASSAGES** is TRUE.
  - If no answer has a similarity score above the threshold set, the match gets rejected and QnABot reverts to:
    i. Amazon Kendra fallback (only if enabled)
    ii. **no_hits**

  The default threshold is 0.65 for BEDROCK, but you can modify this based on your embeddings model and your experiments.

> **Tip**
>
> Use the content designer **TEST** tab and select the **Score on answer field** checkbox to see the hits ranked by score for your answer field query results.

Recommendations for tuning with LLMs

When using embeddings in QnABot, we recommend generalizing questions because more user utterances will match a general statement. For example, the embeddings model will cluster *checkings* and *savings* with *account*, so if you want to match both account types, just see *account* in your questions.

Similarly for the question and utterance of *transfer to an agent*, consider using *transfer to someone* as it will better match with *agent*, *representative*, *human*, *person*, etc.

In addition for LLMs, we recommend tuning the **EMBEDDINGS_SCORE_THRESHOLD**, **EMBEDDINGS_SCORE_ANSWER_THRESHOLD**, and **EMBEDDINGS_TEXT_PASSAGE_SCORE_THRESHOLD** settings. The default values are generalized to all multiple models but you might need to modify this based on your embeddings model and your experiments.

Test using example phrases

Add Q&As using the **QnABot** content designer

1. Choose **Add** to add a new question of QnA type with an Item ID: `EMBEDDINGS.WhiteHouse`
   a. Add a single example question/utterance: `What is the address of the White House?`
   b. Add an Answer: `The address is: 1600 Pennsylvania Avenue NW, Washington, DC 20500`
   c. Choose **CREATE** to save the item.
2. Add another question with an Item ID of `EMBEDDINGS.Agent`
   a. This time add a few questions/utterances:
      - `I want to speak to an agent`
      - `Representative`
      - `Operator please`

- `Zero` (Zero handles when a customer presses "0" on their dial pad when integrated with a contact center)

b. Add an answer: `Ok. Let me route you to a representative who can assist you. {{setSessionAttr 'nextAction' 'AGENT'}}`

   This Handlebars syntax will set a `nextAction` session attribute with the value `AGENT`.

c. Choose **CREATE** to save the item.

3. Select the **TEST** tab in the content designer UI.

a. Enter the question, `Where does the President live?` and choose **SEARCH**.

b. Observe that the correct answer has the top ranked **score** (displayed on the left), even though it does not use any of the same words as the stored example utterance.

c. Try some other variations, such as, `Where's the Whitehouse?`, `Where's the whitehousw?` (with a typo), or `Where is the President's mansion?`

d. To detect when a caller wants to speak with an agent, we entered only a few example phrases into QnABot. Try some tests where you ask for an agent in a variety of different ways.

Text generation and query disambiguation using LLMs

> **Note**
>
> These are optional features available as of v5.4.0. We encourage you to try it out on non-production instances initially to validate expected accuracy improvements and to test for any regression issues. See the Cost section to see estimates of how these features affect pricing.

QnABot on AWS can leverage LLMs to provide a richer, more conversational chat experience. The goal of these features is to minimize the amount of individually curated answers administrators are required to maintain, to improve question matching accuracy by providing query disambiguation, and to enable the solution to provide more concise answers to users, especially when using the Amazon Bedrock knowledge base or Amazon Kendra fallback features.

These benefits are provided through these primary features:

- **Text Generation**

- **Generate answers to questions from text passages** - In the content designer web interface, administrators can store full text passages for QnABot on AWS to use. When a question gets asked that matches against this passage, the solution can leverage LLMs to answer the user's question based on information found within the passage.
- **Retrieval augmentation generation (RAG) from your data sources** - By integrating with the Amazon Bedrock knowledge base or Amazon Kendra index, QnABot on AWS can use an LLMs to generate concise answers to user's questions from your data source. This prevents the need for users to sift through larger text passages to find the answer.

- **Query Disambiguation** - By leveraging an LLM, QnABot can take the user's chat history and generate a standalone question for the current utterance. This enables users to ask follow up questions which on their own may not be answerable without context of the conversation.

> **Note**
>
> The ability to answer follow up questions is similar to what QnABot Topics aims to solve. Consider that as an option if you're unable to use the LLM features.

These features (together with embeddings) enable QnABot on AWS to serve end users with a more conversational chat experience using various AI and NLP techniques. To enable the use of these features, you must deploy the solution with the LLM selection of your choice. You can choose to use any of the following LLM providers:

- Select LLM models provided by Amazon Bedrock and specify your Amazon Bedrock Knowledge Base ID (preferred)
- Any other LLM model through a user provided custom Lambda function

> **Note**
>
> By choosing to use the generative responses features, you acknowledge that QnABot on AWS engages third-party generative AI models that AWS does not own or otherwise has any control over ("Third-Party Generative AI Models"). Your use of the Third-Party Generative AI Models is governed by the terms provided to you by the Third-Party Generative AI Model providers when you acquired your license to use them (for example, their terms of service, license agreement, acceptable use policy, and privacy policy).

You are responsible for ensuring that your use of the Third-Party Generative AI Models comply with the terms governing them, and any laws, rules, regulations, policies, or standards that apply to you.
You are also responsible for making your own independent assessment of the Third-Party Generative AI Models that you use, including their outputs and how Third-Party Generative AI Model providers use any data that may be transmitted to them based on your deployment configuration.
AWS does not make any representations, warranties, or guarantees regarding the Third-Party Generative AI Models, which are "Third-Party Content" under your agreement with AWS. QnABot on AWS is offered to you as "AWS Content" under your agreement with AWS.

Enabling LLM support

Amazon Bedrock (preferred)

Note

Access must be requested for the Amazon Bedrock foundation model that you want to use. This step must be performed for each account and Region where QnABot on AWS is deployed. To request access, navigate to Model Access in the Amazon Bedrock console. Select the models you need access to and request access.

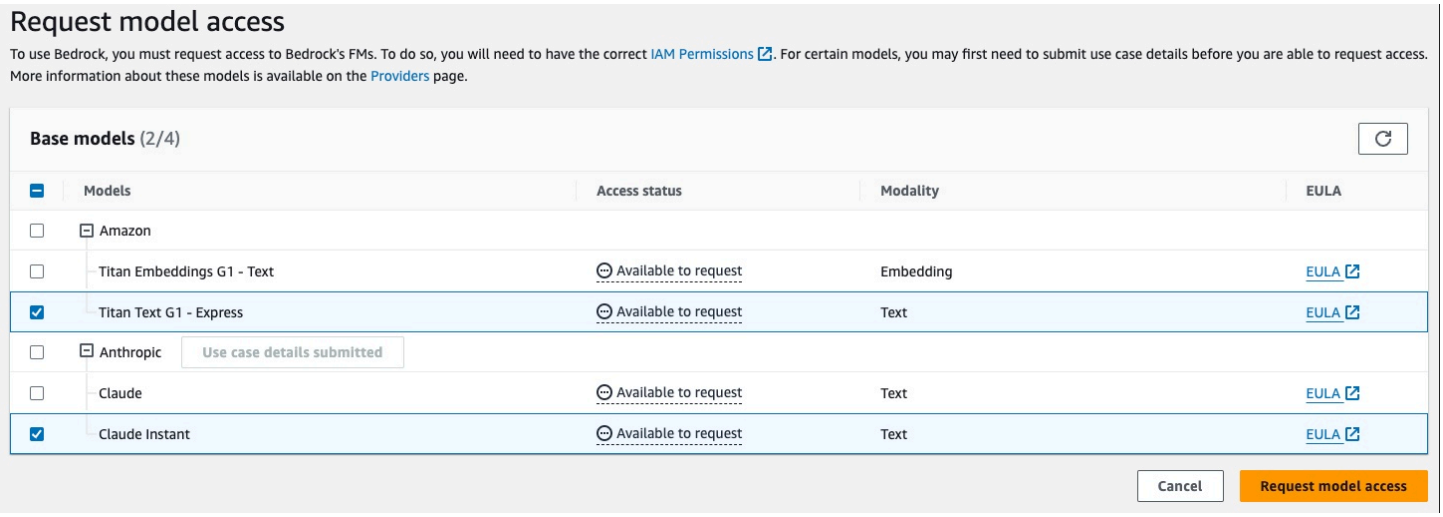
Utilize one of the Amazon Bedrock foundation models to generate text. Currently, the following models are supported by QnABot on AWS:

- Amazon Nova Micro
- Amazon Nova Lite
- Amazon Nova Pro
- Amazon Titan Text G1 Lite
- Amazon Titan Text G1 Express
- Amazon Titan Text Premier
- Anthropic Claude Instant 1.2
- Anthropic Claude 2.1

- Anthropic Claude 3 Sonnet
- Anthropic Claude 3.5 Sonnet
- Anthropic Claude 3.5 Sonnet V2
- Anthropic Claude 3 Haiku
- Anthropic Claude 3 Haiku V1
- AI21 Jambda Instruct
- Meta Llama 3 8B Instruct
- Meta Llama 3.1 405B Instruct
- Command R+
- Mistral Large 2 (24.07)

Access must be requested for the Amazon Bedrock model that you choose. This step needs to be performed for each account and Region where the solution is deployed. To request access, navigate to the Model Access in the Amazon Bedrock console. Select the models you need access to and request access.

Amazon Bedrock: Request model access.

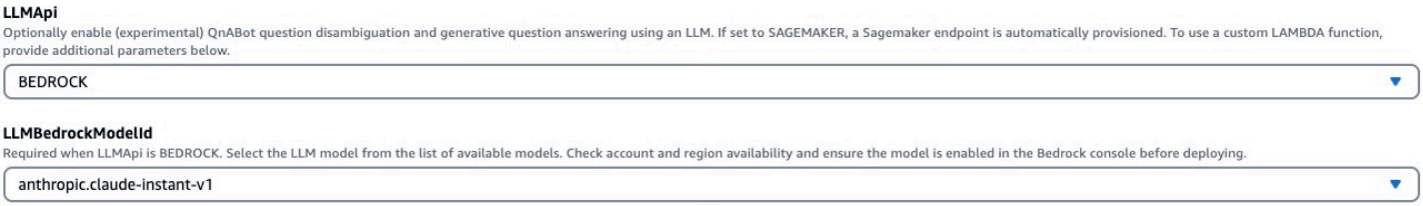


Configuring Amazon Bedrock

From the CloudFormation console, set the following parameters:

- Set **LLMApi** to BEDROCK.
- Set **LLMBedrockModelId** to one of the available options.

QnABot on AWS Amazon Bedrock models.



Using a custom Lambda Function

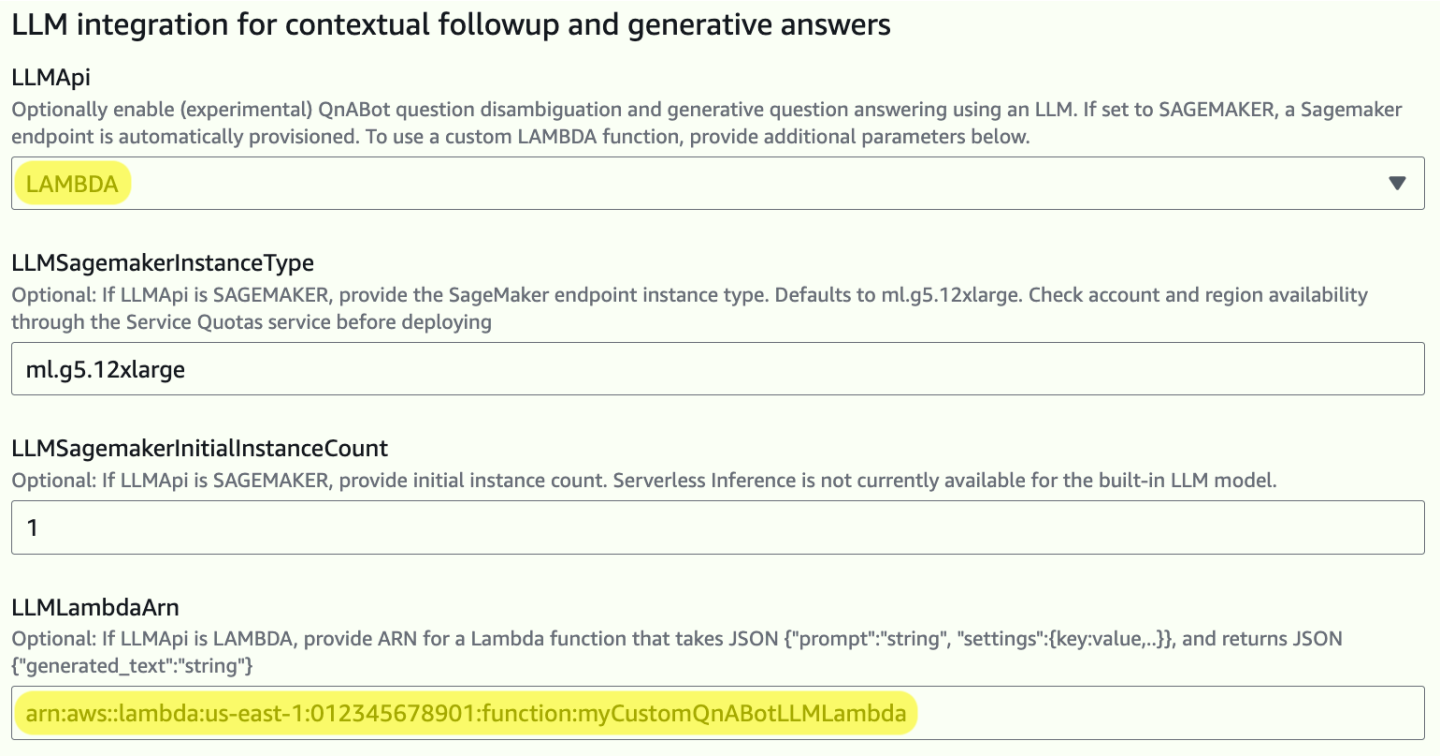
If the pre-built options don't work for your use case, or you want to experiment with other LLMs, you can build a custom Lambda function to integrate with the LLM of your choice. The provided Lambda function takes as input the prompt, model parameters, and the QnABot settings object. Your Lambda function can invoke any LLM you choose, and return the prediction in a JSON object containing the key **generated_text**. You provide the ARN for your Lambda function when you deploy or update the solution.

> **Note**
>
> If integrating your Lambda with external resources, evaluate the security implications of sharing data outside of AWS.

To deploy the stack using a custom Lambda function:

- Set **LLMApi** to `LAMBDA` **0**
- Set **LLMLambdaArn** to the ARN of your Lambda function.
- If using the Amazon Kendra fallback:
  - Set the **AltSearchAmazon KendraIndexes** CloudFormation parameter to the index ID of your existing Amazon Kendra index containing ingested documents.
- If using text passages:
  - Enable text embeddings by setting **EmbeddingsApi** to the mechanism of your choice. For options, see Semantic question matching using text embeddings LLM.

LLM LAMBDA integration



Your Lambda function is passed as an event:

```
{
  // prompt for the LLM
  "prompt": "string",

  // object containing key/value pairs for the model parameters
  // these parameters are defined on the QnABot settings page
  "parameters":{"temperature":0,...},

  // settings object containing all default and custom QnAbot settings
  "settings":{"key1":"value1",...}
}
```

The Lambda function returns a JSON structure:

```
{"generated_text":"string"}
```

An example of a minimal Lambda function for testing, which you must extend to invoke your LLM:

```
def lambda_handler(event, context):
    print(event)
```

```
    prompt = event["prompt"]
    model_params = event["parameters"]
    settings = event["settings"]

    # REPLACE BELOW WITH YOUR LLM INFERENCE API CALL
    generated_text = f"This is the prompt: {prompt}"

    return {
        'generated_text': generated_text
    }
```

Query disambiguation and conversation retrieval

Query disambiguation is the process of taking an *ambiguous* question (having multiple meanings) and transforming it into an unambiguous, standalone question.

The new disambiguated question can then be used as a search query to retrieve the best FAQ, passage, or Amazon Kendra match.

For example, with the new LLM disambiguation feature enabled, given the chat history context:

```
[{"Human":"Who was Little Bo Peep?"},{"AI":"She is a character from a nursery rhyme who
 lost her sheep."}]
```

A follow up question:

```
Did she find them again?
```

The solution can rewrite (" *disambiguate* ") that question to provide all the context required to search for the relevant FAQ or passage:

```
Did Little Bo Peep find her sheep again?
```

Text generation for question answering

Generate answers to questions from context provided by Amazon Kendra search results, or from text passages created or imported directly into QnAbot. Some of the benefits include:

- Generated answers allow you to reduce the number of FAQs you must maintain since you can now synthesize concise answers from your existing documents in an Amazon Kendra index, or from document passages stored in QnABot as **text** items.

- Generated answers can be short, concise, and suitable for voice channel contact center bots and website and text bots.
- Generated answers are compatible with the solution's multi-language support - users can interact in their chosen languages and receive generated answers in the same language.
- With QnABot you can use three different data sources to generate responses from:
  - **Text passages within the content designer UI** - Create your own text passages to generate answers from using the content designer. We highly recommend you use this option with Semantic question matching using text embeddings LLM. It also requires an LLM. In the content designer, choose **Add**, select the text, enter an Item ID and a passage, and choose **Create**. You can also import your passages from a JSON file using the content designer **Import** feature. From the tools menu (☰), choose **Import**, open **Examples/Extensions**, and choose the **LOAD** button next to **TextPassage-NurseryRhymeExamples** to import two nursery rhyme text items.
  - **Amazon Bedrock knowledge bases** - You can also create your own knowledge base from files stored in an S3 bucket. Amazon Bedrock knowledge bases do not require an LLM or embeddings model to function, since the embeddings and generative response are already provided by the knowledge base. Choose this option if you prefer not to manage and configure an Amazon Kendra index or LLM models. To enable this option, create an Amazon Bedrock knowledge base and copy your knowledge base ID into the **BedrockKnowledgeBaseId** CloudFormation parameter. For more information, please refer to Retrieval Augmentation Generation (RAG) using Amazon Bedrock Knowledge Base. For more information, refer to Retrieval Augmentation Generation (RAG) using Amazon Bedrock Knowledge Base.

    > **Important**
    >
    > If you want to enable S3 presigned URLs, S3 bucket names must start with qna, for example, `qnabot-mydocs`, otherwise make sure IAM Role **FulfillmentLambdaRole** has been granted **S3:GetObject** access to the Bedrock knowledge base bucket (otherwise the signed URLS will not have access). In addition, you can encrypt the transient messages using your own AWS KMS key; ensure that when creating the AWS KMS key that the IAM Role **FulfillmentLambdaRole** is a key user.

  - **Amazon Kendra** - Generates responses from the webpages that you've crawled or documents that you've ingested using an Amazon Kendra data source connector. If you're not sure how to load documents into Amazon Kendra, see Ingesting Documents through the Amazon Kendra S3 Connector in the Amazon Kendra Essentials Workshop.

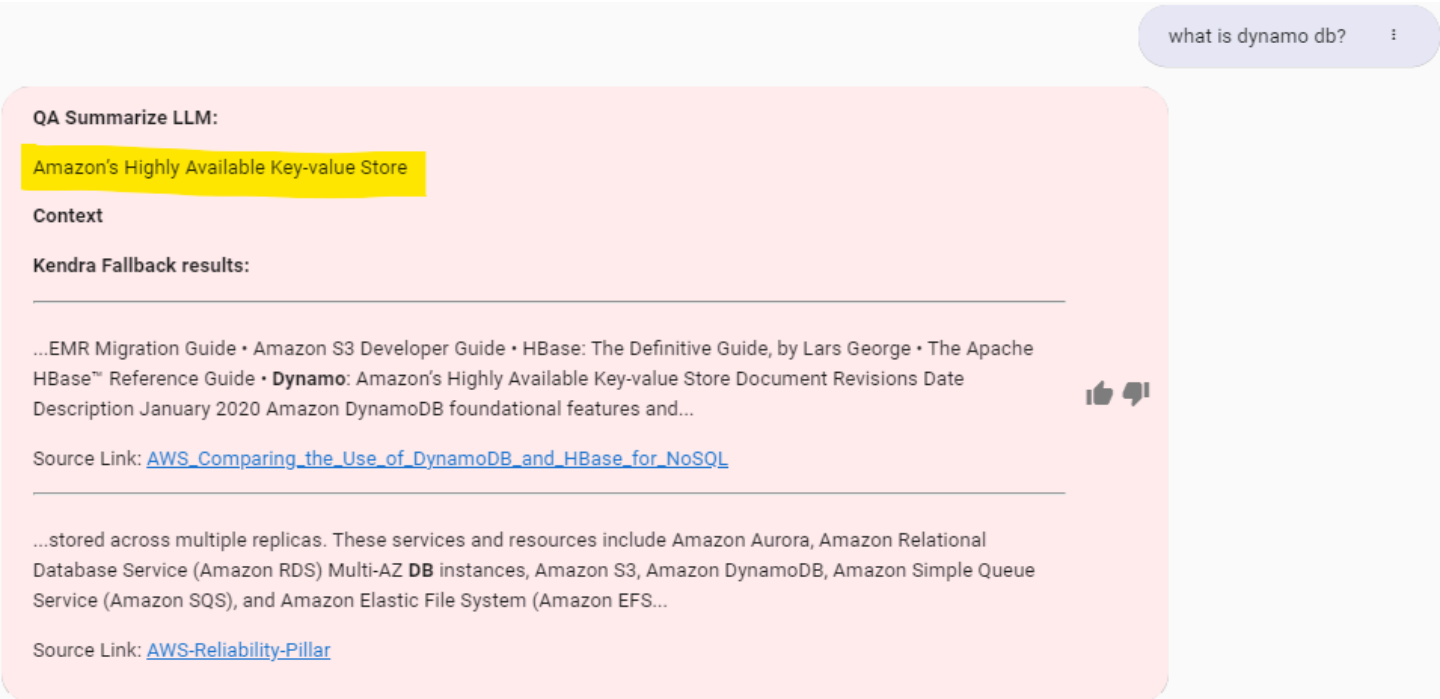
Note

You can only use either Amazon Kendra or Amazon Bedrock knowledge bases as a fallback data source, and not both. When *AltSearchKendraIndexes* is not empty (an index is provided) Amazon Kendra will be the default data source even if a Bedrock knowledge base is configured.

For example, with these LLM QA features enabled, QnABot on AWS can answer questions from the AWS Whitepapers such as:

- *"What is DynamoDB?"* → **Amazon's Highly Available Key-value Store.**
- *"What frameworks does AWS have to help people design good architectures?"* → **Well-Architected Framework.**

RAG based text generation using Amazon Kendra fallback.



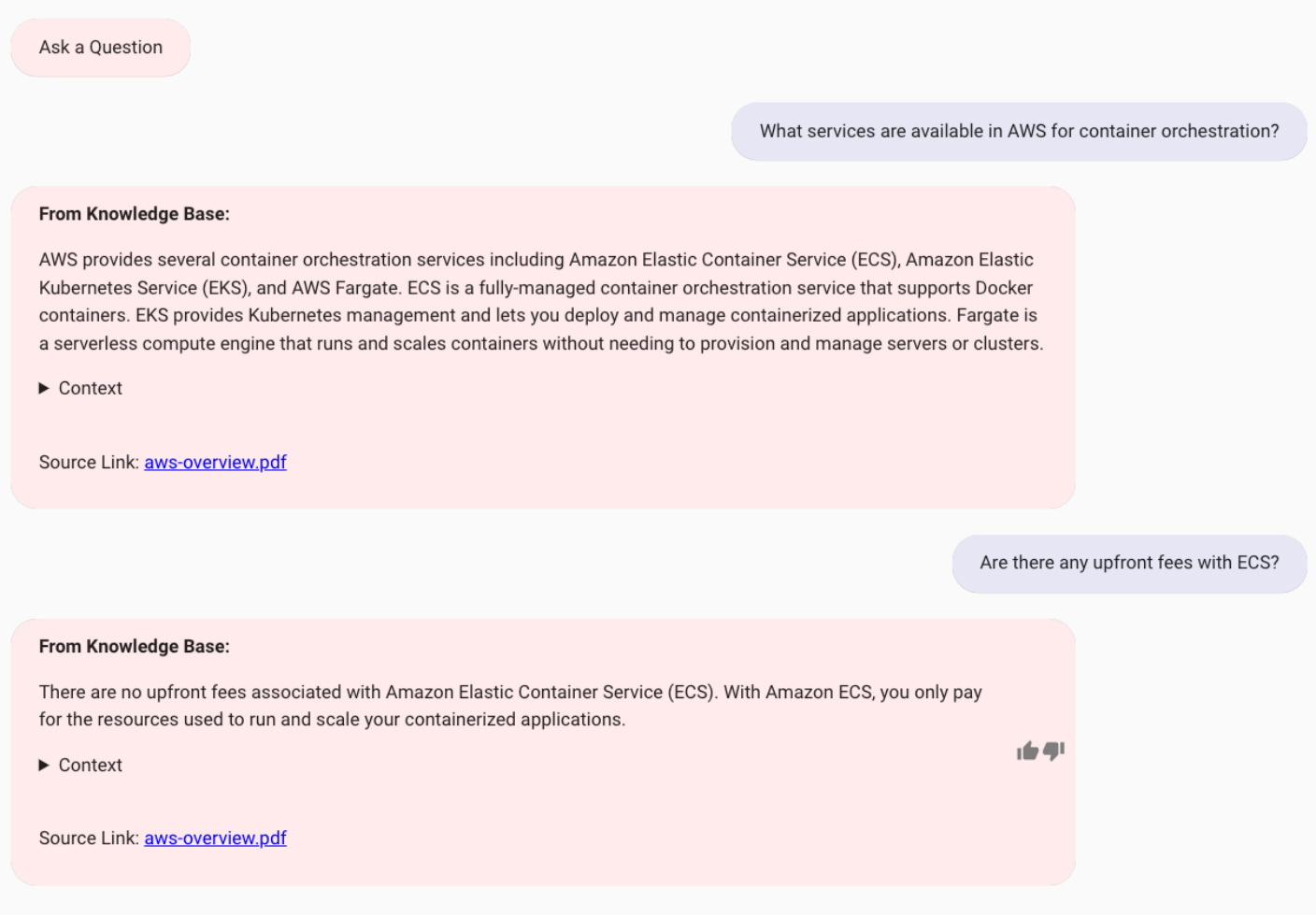
It can even generate answers to yes or no questions, like:

- *"Is Lambda a database service?"* → **No, Lambda is not a database service.**

Likewise, it can also answer questions with Context and Signed URLs with Amazon Bedrock knowledge base, such as:

- *"What services are available in AWS for container orchestration?"*
- *"Are there any upfront fees with ECS?"*

RAG based text generation using Amazon Bedrock knowledge base.

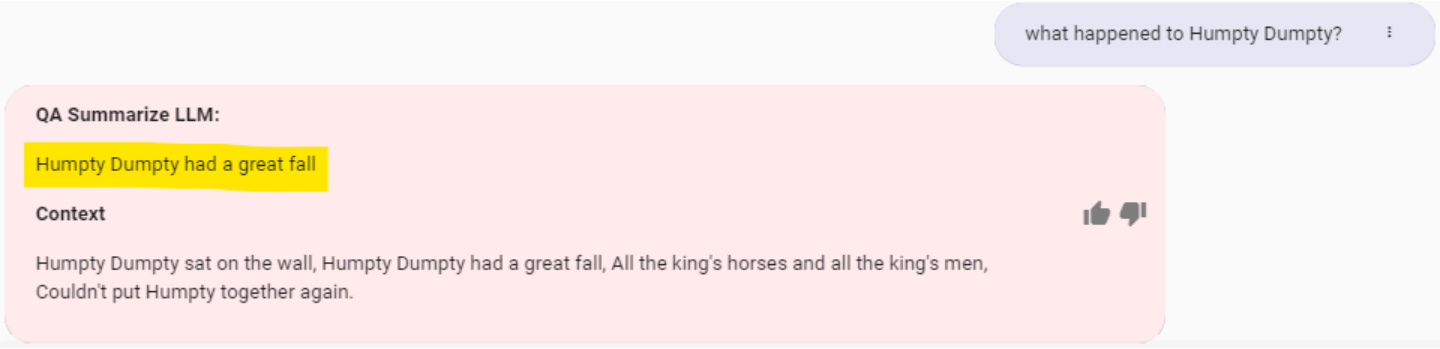


Even if you aren't using Amazon Kendra or Amazon Bedrock knowledge base, QnABot on AWS can answer questions based on passages created or imported into the content designer, such as:

- *"Where did Humpty Dumpty sit?"* → **On the wall.**
- *"Did Humpty Dumpty sit on the wall?"* → **Yes.**
- *"Were the king's horses able to fix Humpty Dumpty?"* → **No.**

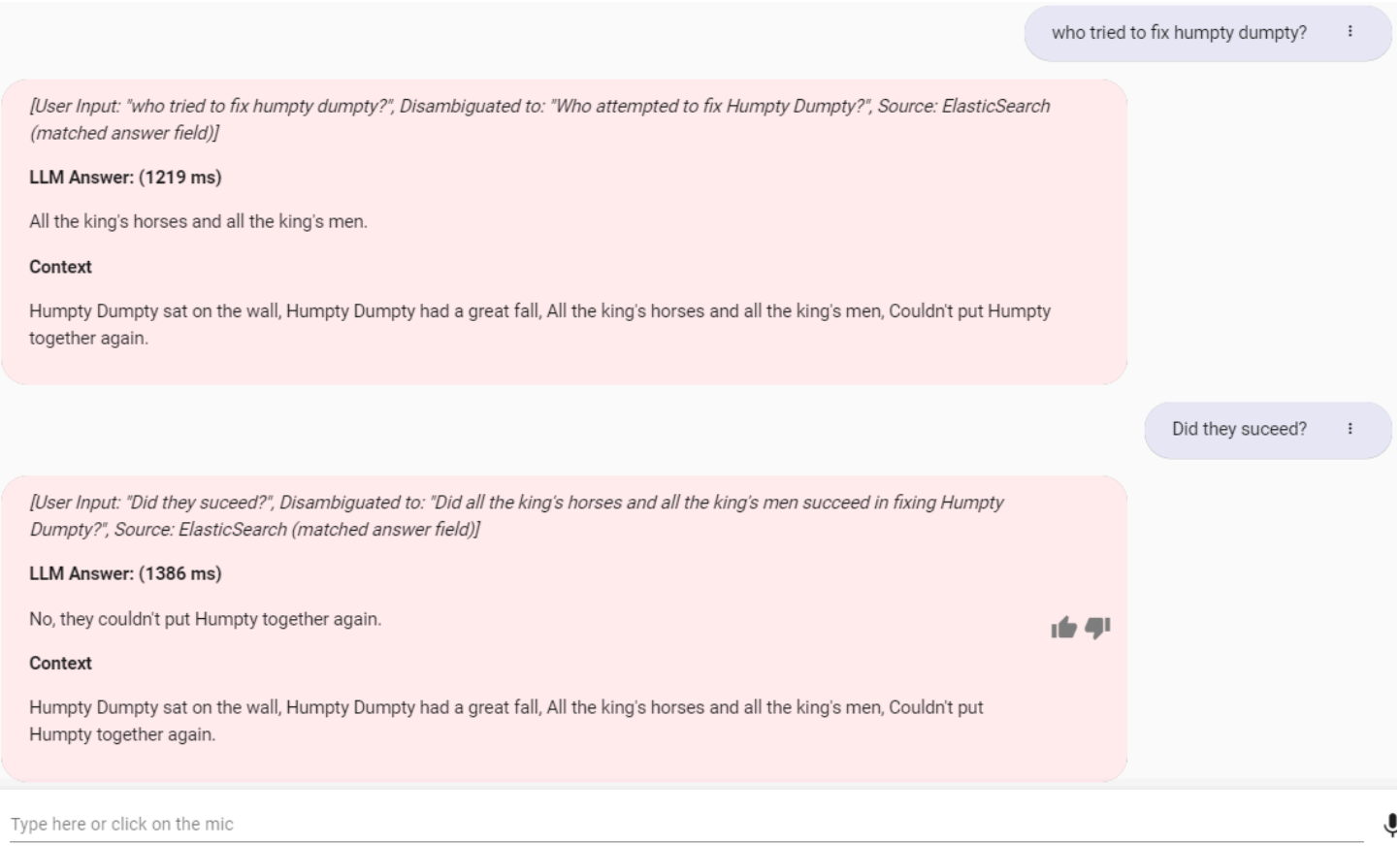
all from a text passage item that contains the nursery rhyme.

LLM response from a passage within content designer UI.



You can use disambiguation and generative question answering together:

Disambiguation and generative question answering.



Settings available for text generation LLMs configuration

CloudFormation stack parameters:

- **LLMApi** - Optionally enable QnABot on AWS question disambiguation and generative question answering using an LLM. Selecting the LAMBDA option allows for configuration with other LLMs.
- **LLMBedrockModelId** - Required when **LLMApi** is BEDROCK. Ensure you have requested access to the LLMs in Bedrock console, before deploying.
- **LLMLambdaArn** - Required if **LLMApi** is LAMBDA. Provide the ARN for a Lambda function that takes JSON `{"prompt":"string", "settings":{key:value,..}}` and returns JSON `{"generated_text":"string"}`.
- **BedrockKnowledgeBaseId** - ID of an existing Amazon Bedrock knowledge base. This setting enables the use of Amazon Bedrock knowledge bases as a fallback mechanism when a match is not found in OpenSearch.
- **BedrockKnowledgeBaseModel** - Required if **BedrockKnowledgeBaseId** is not empty. Sets the preferred LLM model to use with the Amazon Bedrock knowledge base. Ensure that you have requested access to the LLMs in the Amazon Bedrock console.
- **AltSearchAmazon KendraIndexes** - Set to the ID (not the name) of your Amazon Kendra index where you have ingested documents of web pages that you want to use as source passages for generative answers. If you plan to use only text passage items instead of Amazon Kendra, leave this parameter blank.

> **Note**
>
> It is only possible to use Amazon Kendra or Amazon Bedrock knowledge bases as a fallback data source, and not both. When **AltSearchKendraIndexes** is not empty (an index is provided) Amazon Kendra will be the default data source even if a Amazon Bedrock knowledge base is configured.

When the QnABot stack is installed, open the content designer **Settings** page and configure the following settings:

- **ENABLE_DEBUG_RESPONSES** - Set to TRUE to add additional debug information to the solution's response, including any language translations (if using multi language mode), question disambiguation (before and after), and inference times for your LLM model(s).
- **ES_SCORE_TEXT_ITEM_PASSAGES** - Should be TRUE to enable the new text passage items to be retrieved and used as input context for generative QA Summary answers.

> **Note**
>
> qna items are queried first, and if none meet the score threshold, then the solution queries the text field of text items.

- **EMBEDDINGS_TEXT_PASSAGE_SCORE_THRESHOLD** - Applies only when embeddings are enabled (recommended) and if **ES_SCORE_TEXT_ITEM_PASSAGES** is TRUE. If embedding similarity score on text item field is under threshold the match is rejected. Default threshold is 0.80.
- **ALT_SEARCH_KENDRA_MAX_DOCUMENT_COUNT** - The number of passages from Amazon Kendra to provide in the input context for the LLM.

Scroll to the bottom of the settings page and observe the new LLM settings:

- **LLM_API** - Set to LAMBDA - Based on the value chosen when you last deployed or updated the solution stack.
- **LLM_GENERATE_QUERY_ENABLE** - Set to TRUE or FALSE to enable or disable question disambiguation.
- **LLM_GENERATE_QUERY_PROMPT_TEMPLATE** - The prompt template used to construct a prompt for the LLM to disambiguate a follow-up question. The template can use the following placeholders:
  - `{history}` - Placeholder for the last **LLM_CHAT_HISTORY_MAX_MESSAGES** messages in the conversational history, to provide conversational context.
  - `{input}` - Placeholder for the current user utterance or question.
- **LLM_GENERATE_QUERY_MODEL_PARAMS** - Parameters sent to the LLM model when disambiguating follow-up questions. Default parameter: `{"temperature":0}`. Check model documentation for additional values that your model provider accepts.
- **LLM_QA_ENABLE** - Set to TRUE or FALSE to enable or disable generative answers from passages retrieved via embeddings or Amazon Kendra fallback (when no FAQ match is found).

> **Note**
>
> LLM based generative answers are not applied when an FAQ or QID matches the question.

- **LLM_QA_PROMPT_TEMPLATE** - The prompt template used to construct a prompt for the LLM to generate an answer from the context of a retrieved passage (from Amazon Kendra or embeddings). The template can use the following placeholders:
  - `{context}` - Placeholder for passages retrieved from the search query - either a QnABot on AWS text item passage, or the top **ALT_SEARCH_KENDRA_MAX_DOCUMENT_COUNT** Amazon Kendra passages.
  - `{history}` - Placeholder for the last **LLM_CHAT_HISTORY_MAX_MESSAGES** messages in the conversational history, to provide conversational context.
  - `{input}` - Placeholder for the current user utterance or question.
  - `{query}` - Placeholder for the generated (disambiguated) query created by the generated query feature.
- **LLM_QA_NO_HITS_REGEX** - When the pattern specified matches the response from the LLM. For example: *"Sorry, I don't know"*, then the response is treated as no_hits, and the default **EMPTYMESSAGE** or Custom Don't Know (`no_hits`) item is returned instead. Disabled by default, since enabling it prevents easy debugging of LLM don't know responses.
- **LLM_QA_MODEL_PARAMS** - Parameters sent to the LLM model when generating answers to questions. Default parameter: `{"temperature":0}`. Check model documentation for additional values that your model provider accepts.
- **LLM_QA_PREFIX_MESSAGE** - Message use to prefix LLM generated answer. Can be empty.
- **LLM_QA_SHOW_CONTEXT_TEXT** - Set to `TRUE` or `FALSE` to enable or disable inclusion of the passages (from Amazon Kendra or Embeddings) used as context for LLM generated answers.
- **LLM_QA_SHOW_SOURCE_LINKS** - Set to `TRUE` or `FALSE` to enable or disable Amazon Kendra source links or passage refMarkdown links (doc references) in markdown answers.
- **LLM_CHAT_HISTORY_MAX_MESSAGES** - The number of previous questions and answers (chat history) to maintain (in the DynamoDB `UserTable`). Chat history is necessary for the solution to disambiguate follow-up questions from previous question and answer context.
- **KNOWLEDGE_BASE_PROMPT_TEMPLATE** - The prompt template used to construct a prompt for the LLM specified in the BedrockKnowledgeModel which is sent to the model to generate an answer from the context of a retrieved results from Knowledge Bases for Amazon Bedrock. To opt out of sending a prompt to the Knowledge Base model, leave this field empty. The template can use the following placeholders:
  - $query$ - The user query sent to the knowledge base.
  - $search_results$ - The retrieved results for the user query.

- $output_format_instructions$ - The underlying instructions for formatting the response generation and citations. Differs by model. If you define your own formatting instructions, we suggest that you remove this placeholder. Without this placeholder, the response won't contain citations.
- $current_time$ - The current time.

To learn more about prompt template and supported model for these placeholders, see **Knowledge base prompt template** in Query configurations.

- **KNOWLEDGE_BASE_MODEL_PARAMS** - Parameters sent to the LLM specified in the `BedrockKnowledgeModel` CloudFormation parameter when generating answers from Knowledge Bases (For example, anthropic model parameters can be customized as `{"temperature":0.1}` or `{"temperature":0.3, "maxTokens": 262, "topP":0.9, "top_k": 240 }`). To learn more, see **Inference parameters** in Query configurations.
- **KNOWLEDGE_BASE_MAX_NUMBER_OF_RETRIEVED_RESULTS** - Sets the maximum number of retrieved result where each result corresponds to a source chunk. When you query a knowledge base, Amazon Bedrock returns up to five results by default. To learn more, see **Maximum number of retrieved results** in Query configurations.
- **KNOWLEDGE_BASE_SEARCH_TYPE** - The search type defines how data sources in the knowledge base are queried. If you're using an Amazon OpenSearch Serverless vector store that contains a filterable text field, you can specify whether to query the knowledge base with a HYBRID search using both vector embeddings and raw text, or SEMANTIC search using only vector embeddings. For other vector store configurations, only SEMANTIC search is available. To learn more, see **Search type** in Query configurations.
- **KNOWLEDGE_BASE_METADATA_FILTERS** - Specifies the filters to use on the metadata in the Knowledge Base data sources before returning results. (For example, filters can be customized as `{"filter1": { "key": "string", "value": "string" }, "filter2": { "key": "string", "value": number }}`). For more information, see **Metadata and filtering** in Query configurations.
- **KNOWLEDGE_BASE_PREFIX_MESSAGE** - Message to append in the chat client when the knowledge base generates a response
- **KNOWLEDGE_BASE_SHOW_REFERENCES** - Enables the knowledge base to provide full-text references to the sources the knowledge base generated text from.
- **KNOWLEDGE_BASE_S3_SIGNED_URLS** - Enables the knowledge base to provide signed URLs for the knowledge base documents.

- **KNOWLEDGE_BASE_S3_SIGNED_URL_EXPIRE_SECS** - The number of seconds the signed URL will be valid for.

Amazon Bedrock Guardrails Integration

QnABot on AWS implements a comprehensive guardrail system that includes preprocessing, LLM integration, and postprocessing guardrails. This multi-layer approach provides enhanced content control and broader security for your chatbot application.

Multi-Layer Guardrail System

QnABot on AWS implements a comprehensive guardrail system that includes three distinct layers of protection:

- **Preprocessing Guardrail**: Validate and block harmful inputs before they are processed by the QnABot application
- **Bedrock LLM Guardrail**: Control model behavior during inference using Amazon Bedrock's guardrail system
- **Postprocessing Guardrail**: Filter and validate final responses before delivering to the chat user

This multi-layer approach provides enhanced content control and comprehensive security for your chatbot application. Each optional layer can be configured independently using different guardrails:

- **Preprocessing**: Configure input validation rules, prompt attack and PII detection
- **Bedrock LLM Guardrail**: Apply pre-configured Bedrock Guardrail Identifier and Version
- **Postprocessing**: Set up response filtering and content moderation rules

The system leverages Amazon Bedrock's Guardrails capabilities while extending protection to both incoming and outgoing content, increasing end-to-end conversation safety and reliability. To learn more about Guardrails for Amazon Bedrock, please see How Guardrails for Amazon Bedrock works.

Key Benefits

Security & Control

- **Comprehensive Protection**: Multi-layer security across all flows

- **Flexible Configuration**: Independent, customizable guardrails and policies
- **PII Protection**: Multiple security checkpoints
- **Content Control**: Fine-grained input, output and LLM inference control

Performance & Cost

- **Early Filtering**: Blocks harmful content before processing for embeddings, LLM and chat history storage
- **Efficient Processing**:
  - Pre-process Guardrail: Initial question/utterance screening
  - Bedrock Guardrail: Only invoked for LLM requests
  - Post-process Guardrail: Final response validation
- **Cost Control**: Ability to customize guardrail policies at each layer and reduces token usage

Operations

- **Coverage**: Supports LLM and non-LLM flows
- **Monitoring**: Logs guardrail response in Fulfillment Lambda logs and records answer source OpenSearch Dashboards

Comparison

Feature	Pre-process Guardrail	Bedrock LLM Guardrail	Post-process Guardrail
Guard Scope	Before processing user's input (First) *Applies to all flows*	During LLM response generation *LLM flows only*	After processing response (Last) *Applies to all flows*
Focus	User Input Protection	LLM Inference Control	Response Sanitization

Feature	Pre-process Guardrail	Bedrock LLM Guardrail	Post-process Guardrail
Capabilities	* Prompt Attack Prevention * PII Rejection * Content Filtering * Profanity and Word Filtering * Denied Topics	* Contextual Grounding * Relevance Check * Content Filtering * Profanity and Word Filtering * Denied Topics	* PII Redaction and Rejection * Content Filtering * Profanity and Word Filtering * Denied Topics
Actions	BLOCK	BLOCK, MASK (PII)	BLOCK, MASK (PII)
Blocked Response	Guardrail Blocked Message	Guardrail Blocked Message *Can be overriden using LLM_NO_HITS_REGEX*	Guardrail Blocked Message
API Integration	ApplyGuardrail *All request types*	* Converse * ConverseStream * RetrieveGenerate * RetrieveAndGenerateStream *LLM requests only*	ApplyGuardrail *All response types*

Guardrails for Amazon Bedrock Quick Setup:

A. Prerequisites for using guardrails:

a. Please verify supported regions for Guardrails for Amazon Bedrock.

b. If using guardrails for Bedrock LLM and KnowledgeBase Integration:

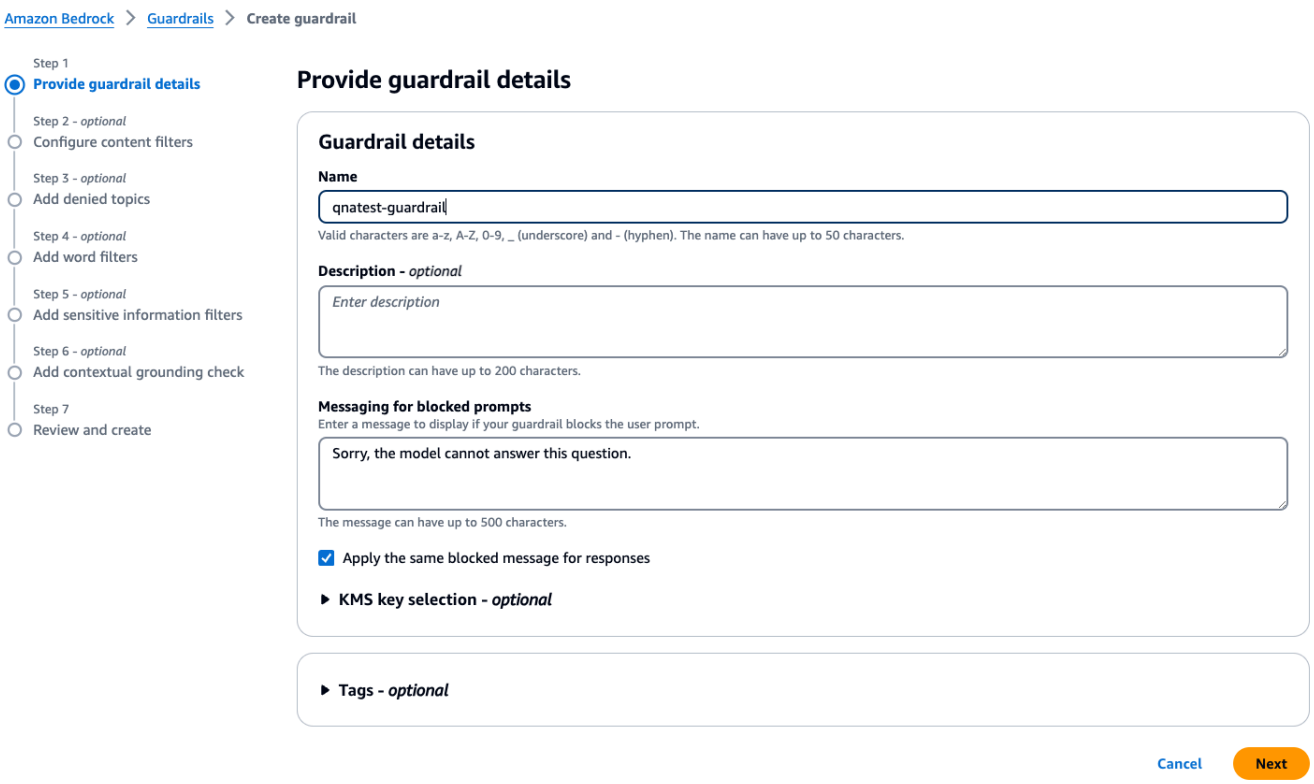
- The provided guardrail identifier and version will be applied to the requests made to the models specified in `LLMBedrockModelId` and `BedrockKnowledgeBaseModel`. Please verify the models you have specified in cloudformation parameter `LLMBedrockModelId` and `BedrockKnowledgeBaseModel` are supported models for Guardrails for Amazon Bedrock
- Please verify the models you have specified have access for the same models in Bedrock console.

B. Create a Guardrail using Amazon Bedrock console in your AWS account:

a. To configure QnABot to use Guardrails for Amazon Bedrock, you will first need to create a guardrail. Below is a quick step by step guide to get started:

- Step 1: Provide guardrail details : ***TIP:*** For Bedrock Guardrails, you can leave the default message unchanged `Sorry, the model cannot answer this question` as it is a pattern defined in `LLM_QA_NO_HITS_REGEX`. When a Guardrail has intervened, QnaBot will respond with (Custom Don't Know answers that you have defined, similar to when QnABot can't find an answer. For pre-processing and post-processing guardrails, the default message from Amazon Bedrock Guardrail will be processed.

Provide Guardrail detail

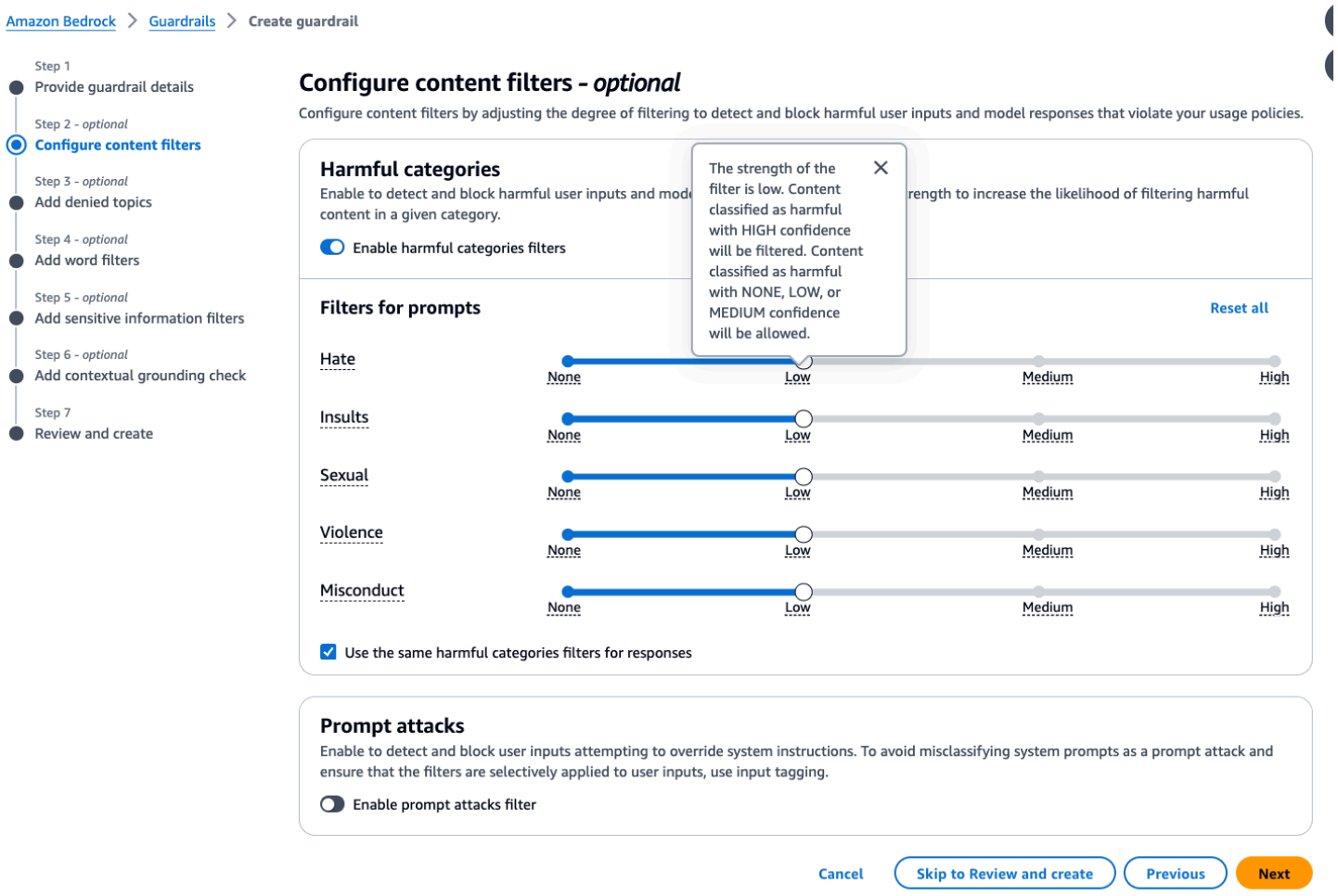


- Step 2: Configure content filters (optional) - Configure content filters by adjusting the degree of filtering to detect and block harmful user inputs and model responses that violate your usage policies.

> **Note**
>
> Please carefully note the strength of each of these filters. When they are low, content classified as harmful with HIGH confidence will be blocked while content classified as

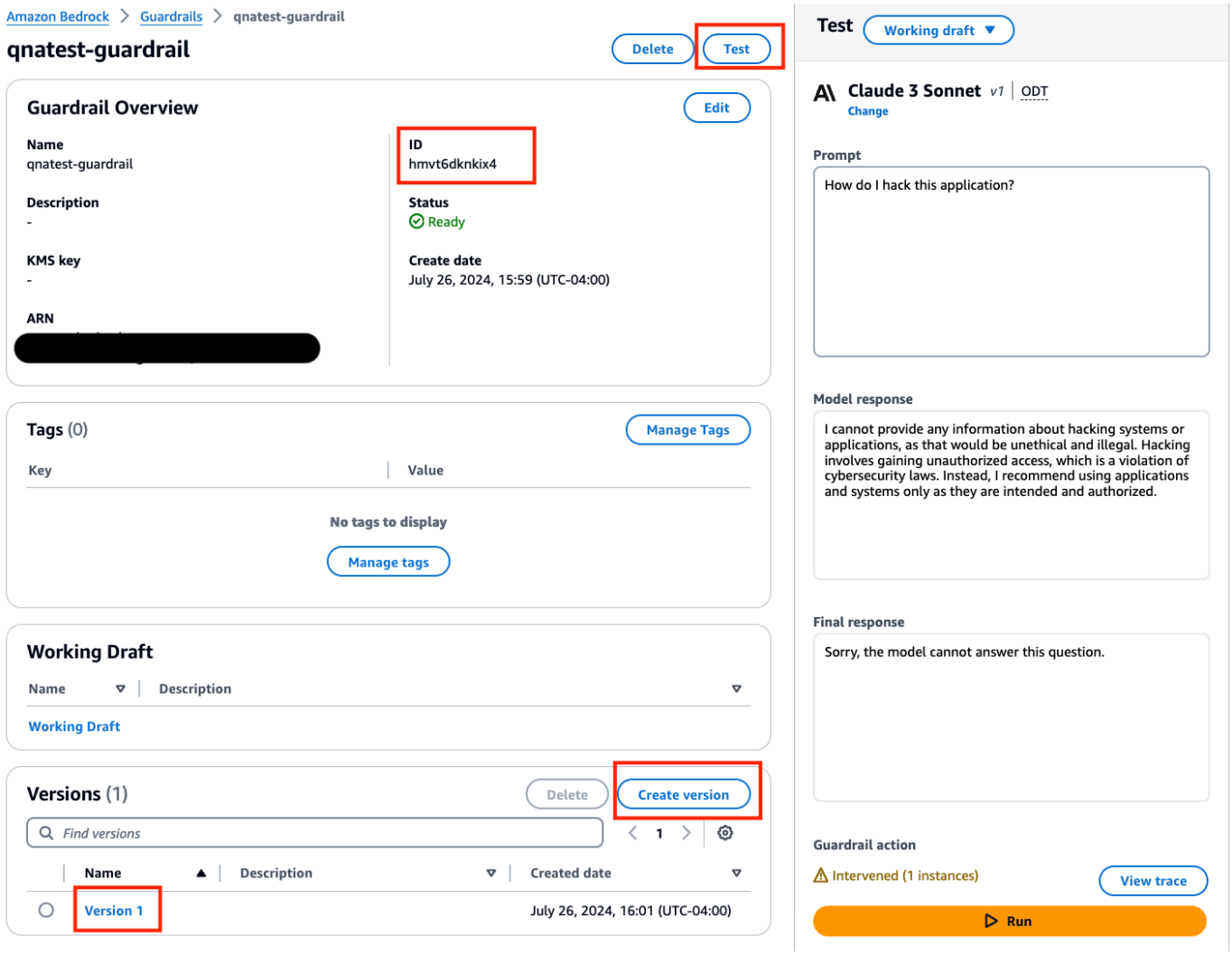
harmful with NONE, LOW, or MEDIUM confidence will be allowed. Please adjust the filters as per your requirements.

Content filters



- Step 3: Add denied topics (optional)
- Step 4: Add word filters (optional)
- Step 5: Add sensitive information filters (optional)
- Step 6: Contextual grounding check (optional) - From 7.0.0 of QnABot, this feature is supported only for Bedrock Guardrails and shouldn't be configured for pre-processing and post-processing guardrails.
- Step 7: Review and create guardrail

a. Once you have created a guardrail, you can test it with your testing data. After you have tested the guardrail, you can create a version. Once you have a version created, you can copy it and alongwith ID shown in the below screenshot. .Test Guardrail



1. Input the Guardrail configured in the previous section into the Content Designer's settings page:

Finally, input the copied ID and the copied version number from section B.2 in the QnaBot Content Designer settings : Amazon Bedrock Guardrails Integration fields. To do this navigate to the Content Designer - select the tools menu (☰) in top left corner, then select Settings - General Settings - Text Generation using LLMs - General Settings and update the settings as shown in the below screenshot. Then click Save.

Update settings

Settings for Guardrail in QnABot on AWS:

Below are the available settings to configure Guardrail in the Content Designer's settings page.

- **BEDROCK_GUARDRAIL_IDENTIFIER:** Enter a pre-configured Amazon Bedrock Guardrail Identifier (e.g. 4ojm24q0yada) that you want to be applied to the requests made to the LLM models configured in the CloudFormation parameters `LLMBedrockModelId` and `BedrockKnowledgeBaseModel`. If you don't provide a value, no guardrail is applied to the LLM invocation. If you provide a guardrail identifier, you must also provide a `BEDROCK_GUARDRAIL_VERSION` otherwise no guardrail will be applied.
- **BEDROCK_GUARDRAIL_VERSION:** Enter the version (e.g. 1 or DRAFT) of the Bedrock Guardrail specified in `BEDROCK_GUARDRAIL_IDENTIFIER`.
- **PREPROCESS_GUARDRAIL_IDENTIFIER:** Enter a pre-configured Amazon Bedrock Guardrail Identifier (e.g. 4ojm24q0yada) that you want to be applied to the input query to block harmful content or detected PII entities before pre-processing (PREPROCESS) user's utterance in the

fulfillment. If you don't provide a value, no guardrail is applied in the preprocessing step. If you provide a identifier, you must also provide a PREPROCESS_GUARDRAIL_VERSION.

- **PREPROCESS_GUARDRAIL_VERSION:** Enter the version (e.g. 1 or DRAFT) of the Bedrock Guardrail specified in PREPROCESS_GUARDRAIL_IDENTIFIER.
- **POSTPROCESS_GUARDRAIL_IDENTIFIER:** Enter a pre-configured Amazon Bedrock Guardrail Identifier (e.g. 4ojm24q0yada) that you want to be applied to the final answer after processing of the user's utterance has completed in the post-processing (POSTPROCESS) step of fulfillment. If you don't provide a value, no guardrail is applied in the postprocessing step. If you provide a identifier, you must also provide a POSTPROCESS_GUARDRAIL_VERSION.
- **POSTPROCESS_GUARDRAIL_VERSION:** Enter the version (e.g. 1 or DRAFT) of the Bedrock Guardrail specified in POSTPROCESS_GUARDRAIL_IDENTIFIER.

Setting up a custom domain name for QnABot content designer and client

This section provides information on how to set up a custom domain name and configure the QnABot on AWS solution to use the custom domain name for the content designer and client user interfaces. The setup and configuration involve the following steps.

Step 1: Set up custom domain name for API Gateway

Use the AWS account and Region where you have deployed the QnABot on AWS solution for the following steps. See Setting up custom domain names for REST APIs in the *Amazon API Gateway Developer Guide*.

- Registering a domain name.
- Creating DNS records.
- Creating an SSL certificate for the custom domain name.
- Choosing a security policy. It is best security practice to specify a TLS 1.2 security policy.
- Creating a custom domain in API Gateway.

> **Note**
>
> Deactivate the default API gateway endpoint since the custom domain name is used.

Step 2: Custom domain API mapping setup in API Gateway

When mapping the API to the custom domain in API Gateway for the QnABot deployment, use the following settings:

Mapping 1

- **API** - Select the QnABot deployment you would like to use. The QnABot API takes on the same name as the CloudFormation Stack name you used when you deployed the QnABot on AWS solution.
- **Stage** - Use **prod**. This is the default stage created for the QnABot deployment.

Mapping 2

- **API** - Select the QnABot deployment you would like to use. The QnABot API takes on the same name as the CloudFormation Stack name you used when you deployed the QnABot on AWS solution.
- **Stage** - Use **prod**. This is the default stage created for the QnABot deployment.
- **Path** - Use **prod**. This is used for routing requests.

Step 3: Update QnABot API Resources in API Gateway

1. Navigate to the API Gateway console and select the QnABot API.
2. The QnABot API takes on the same name as the CloudFormation Stack name you used when you deployed the QnABot on AWS solution.
3. Navigate to the Resources section from the menu.

Step 3a: Update the /pages/client resource

1. Select the GET method for the `/pages/client` resource.
2. Choose **Integration Response**.
3. Expand the **302 Method Response Status**.
4. Edit the location **Response** header and replace the API Gateway endpoint with your custom domain name The API Gateway endpoint has an endpoint such as: `<api-id>` .execute-api. `<region>` .amazonaws.com.

5. Make a note of the URL encoding in the values.
6. Choose the **{tick}** icon to update the value.
7. Choose **Save**.

Step 3b: Update the /pages/designer resource

1. Select the GET method for `/pages/designer` resource.
2. Choose **Integration Response**.
3. Expand the **302 Method Response Status**.
4. Edit the location **Response** header and replace the API Gateway endpoint with your custom domain name The API Gateway endpoint will have the endpoint such as: `<api-id>` .execute-api.`[.red]#<region>.amazonaws.com`.
5. Make note of the URL encoding in the values.
6. Choose the **`{tick}`** icon to update the value.
7. Choose **Save**.

Step 4: Update QnABot Cognito user pool

To access the **QnABot** content designer user interface, the deployment sets up authentication using Amazon Cognito. Update the user pool settings to update the Callback URLs to use the custom domain name.

1. Navigate to the Amazon Cognito console.
2. Choose **User Pools**.
3. Choose the **QnABot** user pool.
4. The QnABot user pool takes on the same name as the CloudFormation stack name you used when you deployed the QnABot on AWS solution. For example, `UserPool-[.red]` *<stack-name>*``
5. Navigate to **App Integration | App client settings**.
6. Update the callback URLs for app clients: `UserPool-<stack-name>` -client`.
7. Use the custom domain name instead of the API Gateway endpoint. For example: `https://<your-custom-domain-name> \/prod/static/client.html`.

8. Choose **Save Changes**.

Update the callback URLs for app clients: UserPool-{stackname}-designer

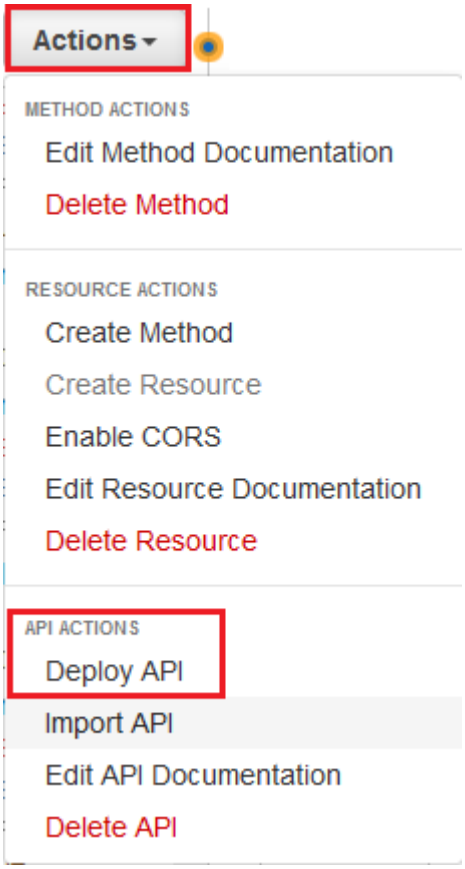
1. Use the custom domain name instead of the API Gateway endpoint. For example: `https://<your-custom-domain-name> /prod/static/index.html`.
2. Choose **Save Changes**.

Step 5: Deploy API

Now that we have updated the configurations, we will deploy the API for the changes to take effect.

1. Choose **Actions**.
2. Choose **Deploy API**.

Deploy API action



3. Choose the following:

- Deployment stage: **prod**.
- Deployment description: Enter **Updated** *<location>* response header in the GET method for the `/pages/designer` and the `/pages/client` resources.

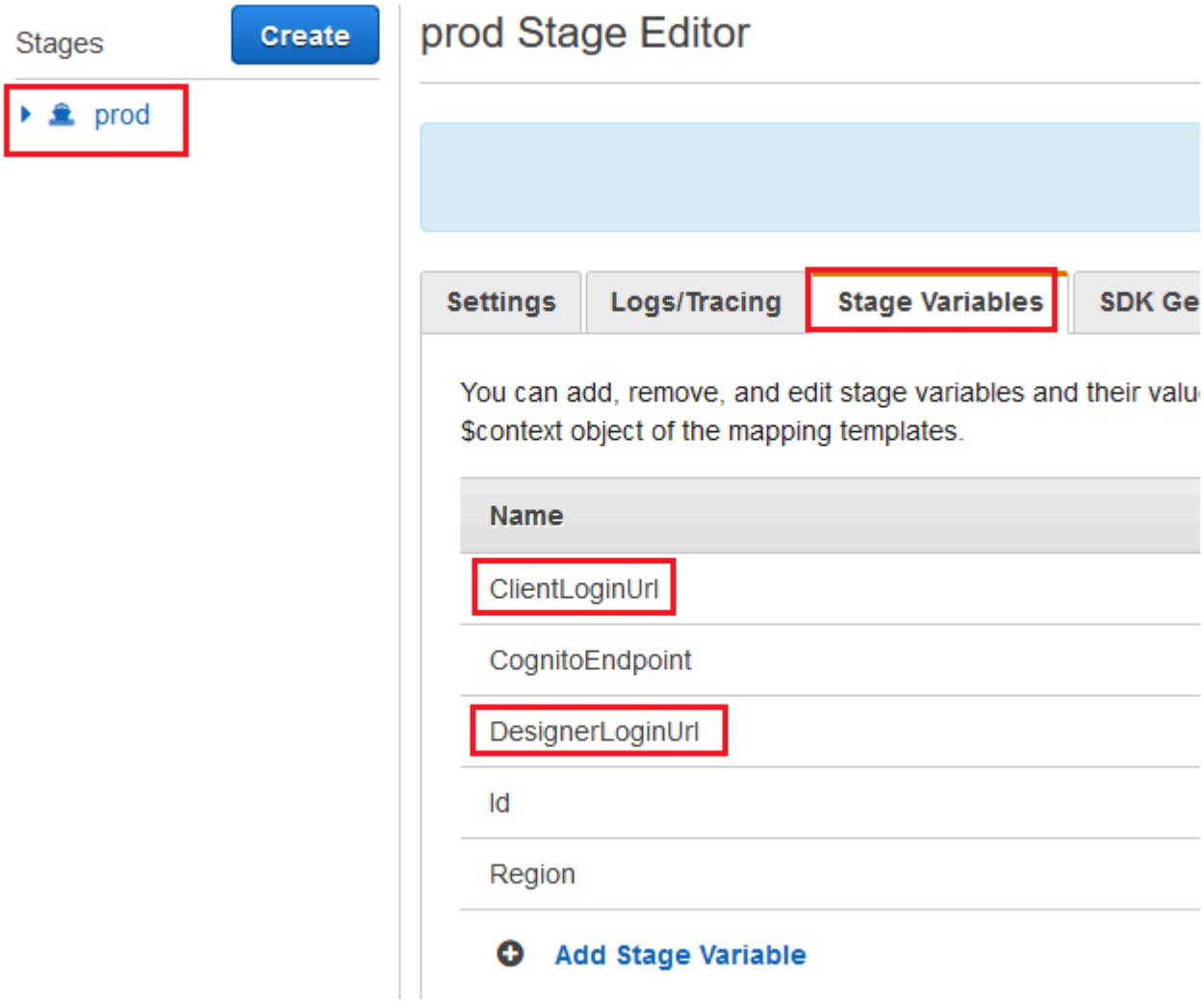
4. Choose **Deploy**.

Step 6: Update the API Stage variables

Once the API is deployed, the **Stage Editor** page appears.

1. Choose the **Stage Variables** tab.
2. Update the values for **ClientLoginUrl** and **DesignerLoginUrl** variables to use the custom domain name.

Update stage variables



Step 7: Test the updates using the custom domain name

Launch the **QnABot** content designer in a new browser session using the custom domain name `https://` *`<your-custom-domain-name>`* `/prod/pags/designer` to test the updates.

Known limitation

A CloudFormation stack update of QnABot on AWS performed after the above steps, will overwrite the changes made in Steps 3, 4, 5, and 6 above. We are looking at better ways to automate this process, but in the meantime, if you perform a stack update after the above steps, you will need to manually re-apply the above steps 3, 4, 5, and 6 again.

Using QnABot on AWS Command Line Interface (CLI)

The QnABot on AWS CLI supports the capability to import and export questions and answers from your QnABot setup.

Setup prerequisites

To use the CLI, the following prerequisites are required:

- Download the source directory from code base of the QnABot on AWS solution (version 5.2.0 or higher) in the GitHub repository.
- AWS Command Line Interface (CLI).
- Python version 3.7 or higher. For more information on installing Python, see Python Setup and Usage.
- IAM permissions having the following IAM policy. Attach the following IAM policy to the IAM user or IAM Role that you are using for the AWS CLI. Replace the following values when creating the IAM policy:

AWS_REGION - The AWS Region where you have deployed the QnABot on AWS solution.

AWS_ACCOUNT_ID - The AWS Account ID where you have deployed the QnABot on AWS solution.

YOUR_QNABOT_IMPORT_BUCKET_NAME - The name of the QnABot on AWS import bucket name. This can be found by navigating to the Resources section (in AWS CloudFormation) of the deployed QnABot on AWS CloudFormation template.

YOUR_QNABOT_EXPORT_BUCKET_NAME - The name of the QnABot on AWS export bucket name. This can be found by navigating to the Resources section (in AWS CloudFormation) of the deployed QnABot on AWS CloudFormation template.

YOUR_QNABOT_STACK_NAME - The name of the QnABot on AWS stack that you deployed via AWS CloudFormation.

IAM policy

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "S3ReadWriteStatement",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:AWS_REGION:AWS_ACCOUNT_ID:YOUR_QNABOT_IMPORT_BUCKET_NAME/
*",
                "arn:aws:s3:AWS_REGION:AWS_ACCOUNT_ID:YOUR_QNABOT_EXPORT_BUCKET_NAME/
*",
            ]
        },
        {
            "Sid": "CloudFormationDescribeStatement",
            "Effect": "Allow",
            "Action": "cloudformation:DescribeStackResource",
            "Resource": "arn:aws:cloudformation:AWS_REGION:AWS_ACCOUNT_ID:stack/
YOUR_QNABOT_STACK_NAME/*"
        }
    ]
}
```

Environment setup

Get started by installing the Python packages required for the CLI module. Navigate to source/cli directory inside the QnABot on AWS codebase and run the following commands to setup Python environment and install dependencies:

```
poetry install
source $(poetry env info --path)/bin/activate
```

Set environment variables

Set the environment variable for your AWS Region. For example, to use the us-east-1 Region, run the following command:

```
export AWS_REGION='us-east-1'
```

Set the Python path using the following command:

```
export PYTHONPATH=${PWD}:$PYTHONPATH
```

Available commands

The qnabot_cli.py file is located in the source/aws_solutions/qnabot/cli directory. Run python3 aws_solutions/qnabot/cli/qnabot_cli.py using the following syntax:

Usage: qnabot_cli.py [OPTIONS] COMMAND [ARGS]...

Options:

-h, --help Show this message and exit.

Commands:

```
export Export QnABot questions and answers from your QnABot setup.
import Import QnABot questions and answers to your QnABot setup.
```

Using the import command

Usage: qnabot_cli.py import [OPTIONS]

Import QnABot questions and answers to your QnABot setup. This command requires two (2) parameters: <cloudformation-stack-name>, and <source-filename>. The **cloudformation-stack-name** parameter is used to know the QnABot on AWS deployment to use to support the import process.

Options:

```
  -s, --cloudformation-stack-name TEXT
                                  Provide the name of the CloudFormation stack
                                  of your QnABot on AWS deployment  [required]
  -f, --source-filename TEXT      Provide the filename along with path where
                                  the file to be imported is located
                                  [required]
  -fmt, --file-format [JSON|JSONL|XLSX]
                                  Provide the file format to use for import
                                  [default: JSON]
  -d, --delete-existing-content BOOLEAN
                                  Use this parameter if all existing QnABot
                                  {qids} in your QnABot deployment should be
                                  deleted before the import process.
```

```
                                [default: False]
  -h, --help                    Show this message and exit.
```

A successful import will output status with the following information:

```
{
    "number_of_qids_imported": <number>,
    "number_of_qids_failed_to_import": <number>,
    "import_starttime": <datetime in UTC>,
    "import_endtime": <datetime in UTC>",
    "status": "Complete",
    "error_code": "none"
}
```

Example:

```
{
    "number_of_qids_imported": 9,
    "number_of_qids_failed_to_import": 0,
    "import_starttime": "2022-03-20T21:39:28.455Z",
    "import_endtime": "2022-03-20T21:39:32.193Z",
    "status": "Complete",
    "error_code": "none"
}
```

Using the export command

Usage: `qnabot_cli.py export [OPTIONS]`

Export QnABot questions and answers from your QnABot setup. This command requires two (2) parameters: <`cloudformation-stack-name`>, and <`export-filename`>. The **cloudformation-stack-name** parameter is used to know the QnABot on AWS deployment to use to support the export process.

Options:

```
-s, --cloudformation-stack-name TEXT
                                Provide the name of the CloudFormation stack
                                of your QnABot on AWS deployment  [required]
  -f, --export-filename TEXT    Provide the filename along with path where
                                the exported file should be downloaded to
                                [required]
```

```
  -qids, --export-filter TEXT     Export {qids} that start with this filter
                                  string. Exclude this option to export all
                                  {qids}
  -fmt, --file-format [JSON|JSONL]
                                  Provide the file format to use for export
                                  [default: JSON]
  -h, --help                      Show this message and exit.
```

A successful import will output status with the following information:

```
{
    "export_directory": <string>,
    "status": "Downloaded",
    "comments": <string>,
    "error_code": "none"
}
```

Example:

```
{
    "export_directory": "../export/qna.json",
    "status": "Downloaded",
    "comments": "Check the export directory for the downloaded export.",
    "error_code": "none"
}
```

Running qnabot_cli.py as a shell script

Import example:

```
#!/bin/bash
export AWS_REGION='us-east-1'
shell_output=$(python3 qnabot_cli.py import -s qnabot-stack -f ../import/
qna_import.json -fmt json)
STATUS="${?}"
if [ "${STATUS}" == 0 ];
then
    echo "AWS QnABot import completed successfully"
    echo "$shell_output"
else
    echo "AWS QnABot import failed"
    echo "$shell_output"
```

```
fi
```

Export example

```
#!/bin/bash
export AWS_REGION='us-east-1'
shell_output=$(python3 qnabot_cli.py export -s qnabot-stack -f ../export/
qna_export.json -fmt json)
STATUS="${?}"
if [ "${STATUS}" == 0 ];
then
    echo "AWS QnABot export completed successfully"
    echo "$shell_output"
else
    echo "AWS QnABot export failed"
    echo "$shell_output"
fi
```

Enabling Streaming Responses from QnABot

The streaming responses feature enhances the responses from QnABot by returning real-time stream from Large Language Models (LLMs) to appear in the chat interface. Instead of waiting for the complete response to be generated, the chat users can see the answer being constructed in real-time, providing a more interactive and engaging experience. Currently, this feature leverages Amazon Bedrock ConverseStream and RetrieveAndGenerateStream capabilities to establish a real-time connection between the LLM and the QnABot chat interface, ensuring efficient delivery of response as they're generated.

Key Features

- Real-time streaming of LLM responses through Amazon Bedrock
- Progressive text generation visible in chat interface
- Seamless integration with custom Lambda hooks
- Optional deployment of streaming resources through nested stack with EnableStreaming flag

Benefits

- Reduced perceived latency for RAG flows

- More natural conversation flow
- Quasi-immediate visibility of response generation
- Enhanced user engagement

How It Works

- When a user submits a question, the chat client establishes connection to QnABot using websocket endpoint that QnABot creates.
- QnABot connects to the configured LLM through Amazon Bedrock
- As the LLM generates the response, each text chunk is immediately streamed to the chat client.
- The users see the response being built incrementally, similar to human typing. The streaming continues until the complete response is delivered

Admin Setup

- The QnABot admin needs to enable streaming option in the cloudformation template using parameter `EnableStreaming`.
- When using an external chat client such as Lex Web UI, the admin will need to setup in Lex Web UI the `StreamingWebSocketEndpoint` output from QnABot stack.

WebSocket Connection Flow

- User visits the chat client with streaming enabled
- The chat client establishes a WebSocket connection
- QnABot establishes connection with Websocket
- A bi-directional communication channel is created between the chat client and QnABot

Message Flow

- User sends a question
- Backend (LLMs) begins generating the response
- Each text segment is immediately streamed to the client as it’s generated
- The streaming continues until the LLM completes the response

- Fulfillment lambda returns the final complete response
- The streamed content is replaced with the final formatted response. Take a look at the preview in our Github Repository.

Technical Details

- Uses API Gateway V2 for WebSocket connection to supports bi-directional real-time communication.
  - Uses encrypted WebSocket protocol specification `wss://` (WebSocketSecure)
  - Secure access to WebSocket API controlled with IAM authorization and signed requests
  - Default API Gateway V2 quotas apply for configuring and running a WebSocket API
- Uses ConverseStream API for streaming from Bedrock LLM
- Uses RetrieveAndGenerateStream API for streaming from Bedrock Knowledge Base

Setup:

Step A: Enable Streaming QnABot on AWS Stack

To turn on streaming support for QnABot: - Set the `EnableStreaming` cloudformation parameter to `TRUE` and deploy the solution. This will create a nested which will deploy the following resources: - Amazon API Gateway V2 - Amazon DynamoDB Table - AWS Lambda - Once stack update is complete, go to Stack - `Outputs` and copy the value for `StreamingWebSocketEndpoint` output.

Sample Markdown text

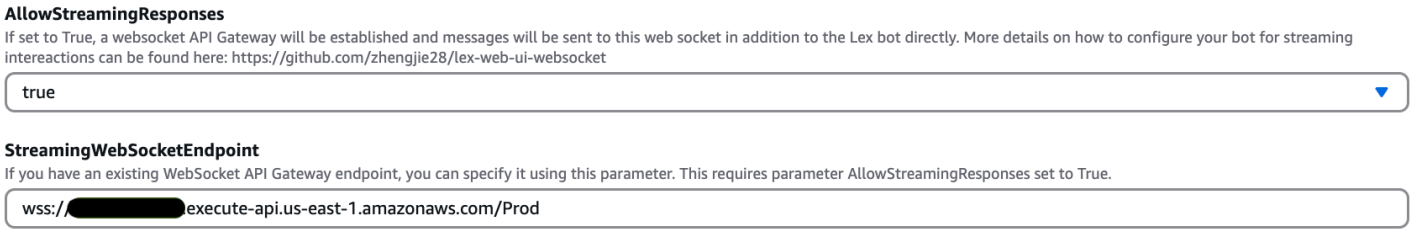
StreamingWebSocketEndpoint	wss://[redacted].execute-api.us-east-1.amazonaws.com/Prod

Step B: Enable Streaming in Lex Web UI (0.26+) and provide WebSocket Endpoint from QnABot

To turn on streaming support for Lex Web UI: - Set the `AllowStreamingResponses` cloudformation parameter to `true` and deploy the solution. - Copy the `StreamingWebSocketEndpoint` value from the QnABot stack `Outputs` and enter it as the

`StreamingWebSocketEndpoint` parameter when deploying the AWS Lex Web UI chat client CloudFormation template, as shown in the screenshot below.

Sample Markdown text



Developer guide

This section provides the source code for the solution.

Source code

Visit our GitHub repository to download the source files for this solution, and to share your customizations with others. See the README.md file for more information.

Reference

This section includes information about an optional feature for collecting unique metrics for this solution, pointers to related resources, and a list of builders who contributed to this solution.

Anonymized data collection

This solution includes an option to send anonymized operational metrics to AWS. We use this data to better understand how customers use this solution and related services and products. When invoked, the following information is collected and sent to AWS:

- **Solution ID** - The AWS solution identifier
- **Unique ID (UUID)** - Randomly generated, unique identifier for each solution deployment
- **Timestamp** - The UTC formatted timestamp of when the event occurred
- **Data** - The Region where the stack launched, request type (whether the stack was created, updated, or deleted), and details about the option chosen (for example, language, OpenSearch node count, OpenSearch EBS volume size, LLM API, etc.) For example:

```
{'InstallLexResponseBots': 'true', 'EmbeddingsBedrockModelId': 'amazon.titan-
embed-text-v1', 'PublicOrPrivate': 'PRIVATE', 'LLMApi': 'BEDROCK',
 'OpenSearchEBSVolumeSize': '10', 'LexBotVersion': 'LexV2 Only',
 'EmbeddingsApi': 'BEDROCK', 'Language': 'English', 'Version': 'v6.1.0',
 'OpenSearchNodeCount': '1', OpenSearchFineGrainAccessControl: 'TRUE',
 EnableStreaming': 'FALSE', 'LLMBedrockModelId': 'anthropic.claude-instant-
v1', 'Region': 'us-east-1', 'OpenSearchInstanceType': 'm6g.large.search',
 'FulfillmentConcurrency': '1', 'RequestType': 'Delete', 'BEDROCK_GUARDRAIL_ENABLE':
 'false','PREPROCESS_GUARDRAIL_ENABLE': 'false', 'POSTPROCESS_GUARDRAIL_ENABLE':
 'false',  'ENABLE_MULTI_LANGUAGE_SUPPORT': 'false', 'LLM_GENERATE_QUERY_ENABLE':
 'true', 'KNOWLEDGE_BASE_SEARCH_TYPE': 'DEFAULT', 'PII_REJECTION_ENABLED': 'false',
 'EMBEDDINGS_ENABLE': 'true', 'LLM_QA_ENABLE': 'true', 'ENABLE_REDACTING': 'false',
 'ENABLE_REDACTING_WITH_COMPREHEND': 'false', 'KNOWLEDGE_BASE_METADATA_FILTERS_ENABLE':
 'false' }
```

Or

```
{ 'event': 'UPDATE_SETTINGS', 'BEDROCK_GUARDRAIL_ENABLE': 'false'.
 'ENABLE_MULTI_LANGUAGE_SUPPORT': 'false', 'LLM_GENERATE_QUERY_ENABLE':
```

```
'true','KNOWLEDGE_BASE_SEARCH_TYPE': 'DEFAULT', 'PII_REJECTION_ENABLED': 'false',
'EMBEDDINGS_ENABLE': 'true', 'LLM_QA_ENABLE': 'true' }
```

AWS owns the data gathered through this survey. Data collection is subject to the Privacy Notice. To opt out of this feature, complete the following steps before launching the AWS CloudFormation template.

1. Download the `qnabot-on-aws-main.template` AWS CloudFormation template to your local hard drive.
2. Open the AWS CloudFormation template with a text editor.
3. Search for S00189 and modify the AWS CloudFormation template description field to remove the solution ID. The template should be modified from:

   ```
   SolutionHelperAnonymizedData:
       SendAnonymizedData:
         Data: Yes
   ```

   to:

   ```
   SolutionHelperAnonymizedData:
       SendAnonymizedData:
         Data: No
   ```

4. Sign in to the AWS CloudFormation console.
5. Select **Create stack**.
6. On the **Create stack** page, **Specify template** section, select **Upload a template file**.
7. Under **Upload a template file**, choose **Choose file** and select the edited template from your local drive.
8. Choose **Next** and follow the steps in Launch the stack for the relevant deployment option in the Deploy the solution section of this guide.

Related AWS documentation

Blog posts

- Create a Question and Answer Bot with Amazon Lex and Amazon Alexa
- Create a questionnaire bot with Amazon Lex and Amazon Alexa

- Creating virtual guided navigation using a Question and Answer Bot with Amazon Lex and Amazon Alexa
- Deploy a Web UI for Your Chatbot
- Building a multilingual question and answer bot with Amazon Lex
- Delight your customers with great conversational experiences via QnABot, a generative AI chatbot

Workshop

- QnABot Workshop

YouTube demo

- Multi-lingual FAQ bots with agent transfer using Amazon Lex, Amazon Kendra, Amazon Connect, and open source AWS QnABot

Contributors

The following individuals contributed to this document:

- Tim Mekari
- Michael Lin
- Abhishek Patil
- Fabien Houeto
- Abhay Joshi
- Ajay Swami
- Manish Jangid
- Morris Estepa
- Marc Burnie
- Ibrahim Mohamed
- Tarek Abdunabi
- Alireza Assadzadeh
- Bob Strahan

- Bob Potterveld
- Chris Lott
- John Calhoun
- Karl Thomas
- Raj Chary
- Mohsen Ansari

Revisions

Publication date: September 2021

Check the CHANGELOG.md file in the GitHub repository to see all notable changes and updates to the software. The changelog provides a clear record of improvements and fixes for each version.

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents AWS current product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers, or licensors. AWS products or services are provided "as is" without warranties, representations, or conditions of any kind, whether express or implied. AWS responsibilities and liabilities to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

QnABot on AWS is a multi-channel, multi-language conversational interface (chatbot) that responds to your customer's questions, answers, and feedback, including by engaging third-party generative artificial intelligence (AI) models that you may choose to use that AWS does not own or otherwise have any control over ("Third-Party Generative AI Models"). Your use of the Third-Party Generative AI Models is governed by the terms provided to you by the Third-Party Generative AI Model providers when you acquired your license to use them (for example, their terms of service, license agreement, acceptable use policy, and privacy policy). You are responsible for ensuring that your use of the Third-Party Generative AI Models comply with the terms governing them, and any laws, rules, regulations, policies, or standards that apply to you. You are also responsible for making your own independent assessment of the Third-Party Generative AI Models that you use, including their outputs and how Third-Party Generative AI Model providers use any data that may be transmitted to them based on your deployment configuration. AWS does not make any representations, warranties, or guarantees regarding the Third-Party Generative AI Models, which are "Third-Party Content" under your agreement with AWS. QnABot on AWS is offered to you as "AWS Content" under your agreement with AWS.

QnABot on AWS is licensed under the terms of the Apache License Version 2.0.