Interactive Kiosk AI Chatbot

Interactive AI Chatbot / Documentation

# Overview

Build and deploy the Interactive Kiosk AI Chatbot reference implementation (RI) for a standalone kiosk, designed for a banking use case. The Interactive Kiosk AI Chatbot contains automatic speech recognition (ASR), text to speech (TTS), and natural language processing (NLP) as microservices and leverages deep learning algorithms of Intel® Distribution of OpenVINO™ toolkit. This RI provides microservices that will allow your system to listen through the mic array, understand natural language expressions, determine intent and entities, and formulate a response.

**Table 1**

|  |  |
| --- | --- |
| **Time to Complete**  **Programming Language**  **Software** | Approximately 30 to 90 minutes  Python\* 3  Docker\*  Git\*  Intel® Distribution of OpenVINO™ toolkit.  Rasa Open Source\*  Mozilla\* DeepSpeech ASR  Kaldi\* ASR  Open Bank Project (OBP)\*  OpenSSL\* |

**Other Requirements**

Some software packages require root permissions.

# Target System Requirements

* Ubuntu\* 18.04 LTS or 20.04.2 LTS
* 8th Generation (and above) Intel® Core™ Processor, 32GB RAM and 100GB of free space on HDD.
* USB mic array or microphone
* Wired speaker
* Network with speed greater than 60 mbps to support speech

**Note:** The application requires mic and speaker devices. It has been validated with Seeedstudio\* ReSpeaker**,** Plantronics\* Blackwire 3220 Series and Maono\* AU-A04 Condenser Microphone Kit (Black).

# Learning Objectives

Use the reference implementation to:

* Integrate an end-to-end speech and audio pipeline that can be expanded with a digital human interface (DHI).
* Incorporate a deep learning microservice architecture into a retail banking proof-of-concept.
* Take advantage of workload optimization as implemented in Intel® Distribution of OpenVINO™ toolkit.

Integrate the entire RI or individual microservices with an existing chatbot pipeline. Replace any of the microservices provided in the RI and extend the capability.

## Helpful Background Knowledge

It is helpful to have a working knowledge of:

* Machine learning technology and deep learning models for ASR, TTS, NLP and Rasa.
* Docker and associated commands.
* Authentication methods, such as the Open Bank Project (OBP) server.

However, it is possible to install and use the reference implementation without this background knowledge.

# How It Works

The Interactive Kiosk AI Chatbot RI contains microservice framework with deep learning algorithms. Table 2 lists the microservices and models included.

|  |  |  |
| --- | --- | --- |
| **Microservice** | **Function** | **Details** |
| ASR | Converts speech-to-text using four **pre-trained** deep learning models. | **Included in Intel® Distribution of OpenVINO™ toolkit:**  [Kaldi](https://docs.openvinotoolkit.org/latest/omz_demos_text_to_speech_demo_python.html)  [Mozilla DeepSpeech](https://docs.openvinotoolkit.org/latest/omz_demos_speech_recognition_deepspeech_demo_python.html)  [Quartznet](https://docs.openvinotoolkit.org/latest/omz_demos_speech_recognition_quartznet_demo_python.html)  **Facebook\*:**  [Huggingface Facebook’s Wav2Vec2-Base-960h](https://huggingface.co/facebook/wav2vec2-base-960h) |
| NLP | Classifies the text by stories, example conversations, and user intent and makes software calls corresponding to the speaker’s classified intent. | NLP uses Rasa OpenSource\*, a natural language framework for automated text and voice applications. For more details, refer to [Introduction to Rasa Open Source](https://rasa.com/docs/rasa/).  To understand stories and intents, see [Rasa Playground.](https://rasa.com/docs/rasa/playground) |
| TTS | Converts text to speech | **Supported Models:**   * forward tacotron * melgan   For more details, refer to [Text to Speech Python\* Demo](https://docs.openvinotoolkit.org/latest/omz_demos_text_to_speech_demo_python.html). |
| Audio Ingestion2 | Records audio from microphone and publishes it to 0MQ on a specified port and topic using respeaker sdk. | For product information, refer to [Seeed ReSpeaker Mic Array](https://wiki.seeedstudio.com/ReSpeaker_Mic_Array_v2.0/). |
| Audio Ingestion | Records audio from microphone and publishes it to 0MQ. | Refer to [Wav File Ingestion](#_Wav_File_Ingestion_3) below if using pre-recorded files in wav format. |
| AuthZ | Checks user credentials at login. | The RI uses Open Bank Project (OBP) and requires that the user obtain credentials for the OBP server, including an OBP JSON Web Token (JWT) token. The microservice supports several operations, including create-session, publish session-id, verify session-id and get OBP JSON JWT token. |

**Table 2: List of Components**

In addition to ASR, NLP, and TTS, the RI uses:

* **ZeroMQ or 0MQ:** A data bus for broker-less asynchronous messaging and exchange of data between the microservices. The RI supports the Publish–Subscribe data distribution pattern for the exchange of messages by the microservice over 0MQ.
* **Open Bank Project (OBP):** A sandbox that allows users to experiment with banking APIs outside a production environment. RI uses cloud based OBP sandbox, but users can deploy the OBP server locally.

The RI checks OBP user credentials, including an OBP JSON Web Token (JWT), at login with the microservice authz. The microservice supports several operations, including:

* create-session
* publish session-id
* verify session-id
* get OBP JSON JWT token.

## What You’ll Do

In Get Started and Tutorials, you will:

* Install and build a framework of microservices for wave and speech ingestion.
* Interact with your microphone and speaker to experience both wav file and live speech ingestion in a defined conversation scenario.
* Complete a tutorial to Replace the ASR Model, Modify the NLP, and Modify the Wake Word.

Later, check out [Learn More](#_Learn_More) to see how to expand the capabilities of the framework by replacing its microservice components and models to create a custom design.

For more details about the supported models, including word error rate (WER) and training limitations, see [Feature Support](#_Feature_Support) and [Learn More: US English ASR Models](#_US_English_ASR).

## Recommendations

Consider integrating either of these features:

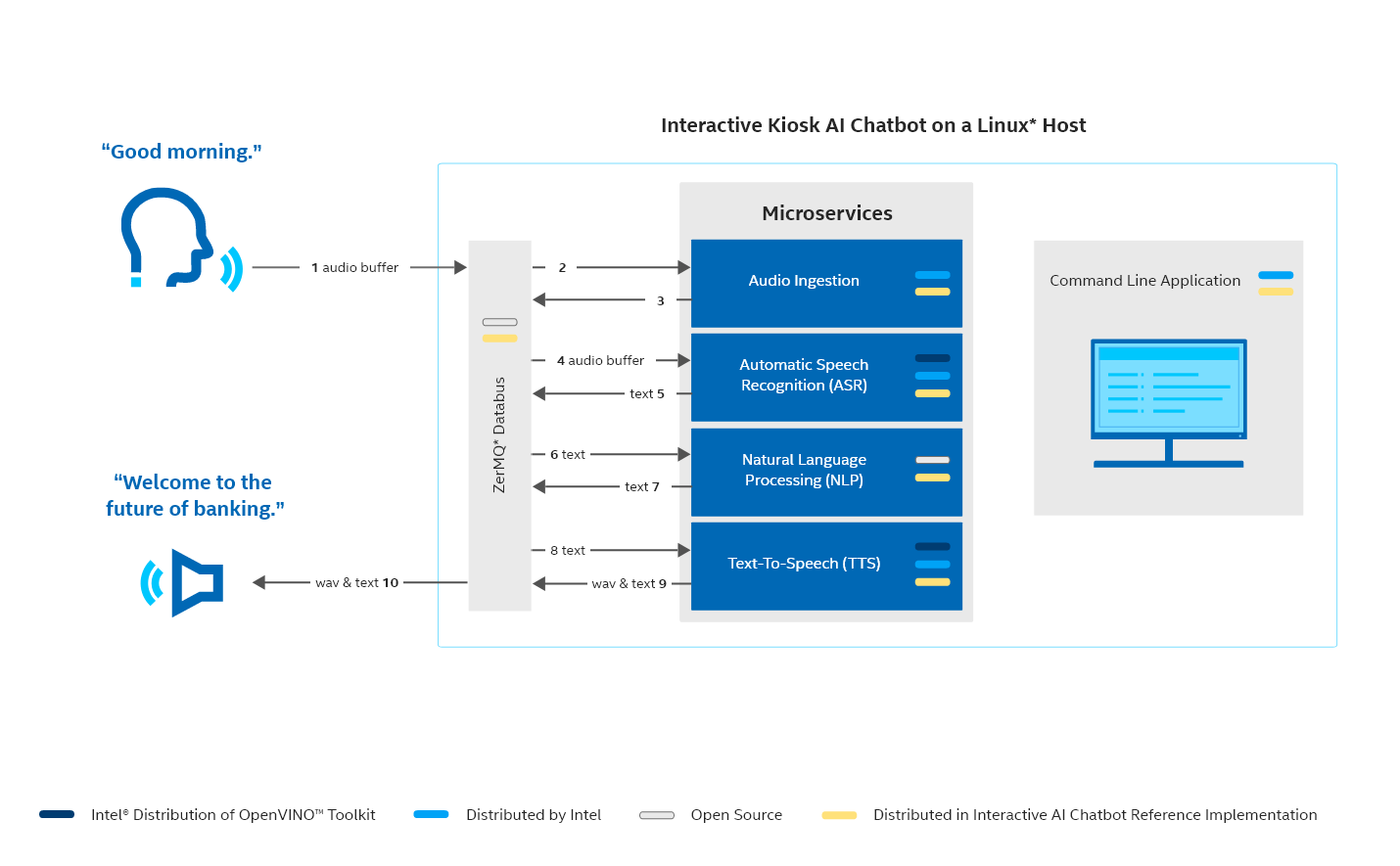
* Digital Human Interface (DHI): An avatar or character-based interface.
* Text-based instant message interface

**NOTE:** Consider using the RI with a USB mic array and custom audio ingestion microservice for basic noise cancellation. See section [Step 1: Input and Output](#_Step_1:_Input) for speaker installation details.

## Conversation Type: Greetings and Basics

The sample chatbot responds to a few greetings and some basic conversational speech that is not related to banking.

Figure 1 presents the software flow that occurs when an end user engages with the RI in conversation that is not related to banking.



**Figure 1: Software Flow (Greeting)**

Figure 1 illustrates the data flow of the RI. Speech described in this section is presented as an example only. For the supported phrases, see Wav File Ingestion.

Microservices and bus, A-E in Figure 1, are open source and distributed as part of the RI. The ASR and TTS use models distributed within the Intel® Distribution of OpenVINO™ toolkit.

1. User greets a mic with **Good morning**.

2. Audio input from Mic is fed to audio ingestion.

3. Audio ingestion publishes the audio buffer onto 0mq with topic ‘audio’.

4. The ASR subscribes to the topic ‘audio’ on the 0mq databus where it receives the audio buffers in chunks. ASR converts speech-to-text using pre-trained deep learning models.

5. ASR publishes the text on the 0mq with topic as 'text'.

6. NLP receives the text and classifies it by intent. It then makes software calls corresponding to those intents. It can take software actions corresponding to that intent.

7. NLP processes and publishes the action as text on the 0mq databus on topic 'nlp'.

8. TTS, subscribed for the topic ‘nlp’, converts the text to wave format.

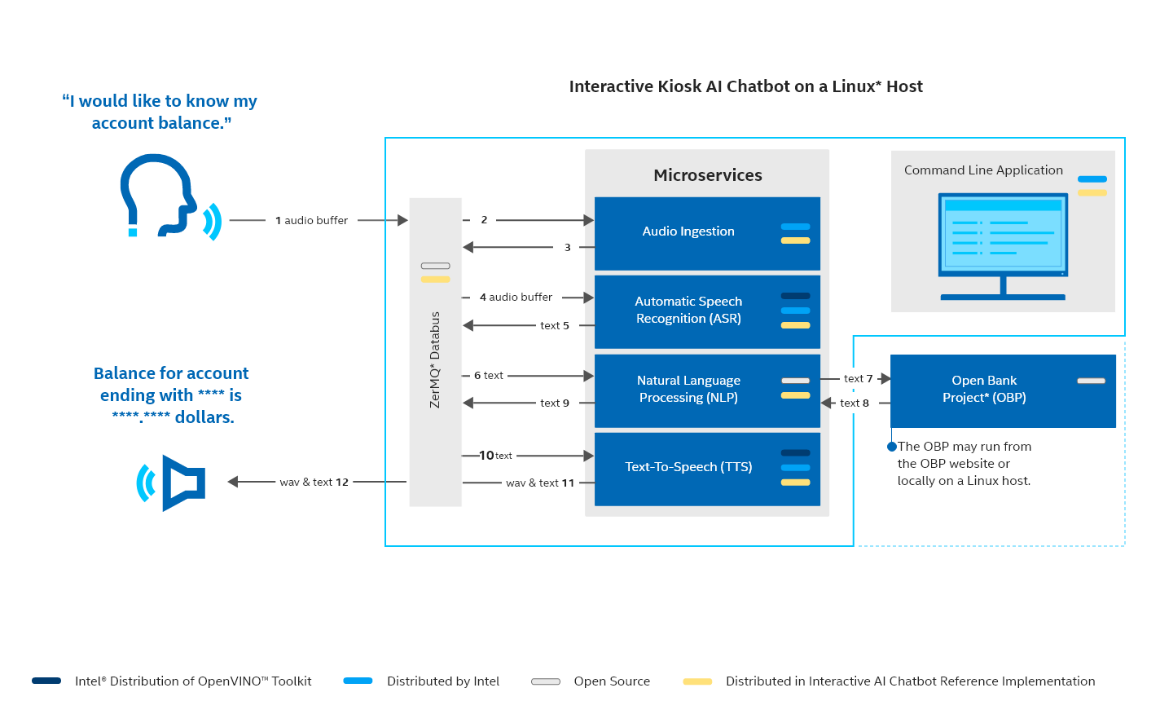
9. TTS publishes the text and wav on the 0mq with the topic 'tts'.

10. The TTS output is sent to speaker. Speaker annunciates the reply **Welcome to the future of banking.**

## Conversation Type: Banking Queries and Information Requests

The RI’s NLP microservice discerns banking-related queries and requests through keywords and intent. It then makes corresponding calls to Open Bank Project (OBP).

Figure 2 presents the software flow that occurs when an end user engages with the RI in banking-related conversation.



**Figure 2: Software Flow: Banking Query**

Figure 2 illustrates the data flow of the RI. Speech described in this section is presented as an example only. For the supported phrases, see Wav File Ingestion.

1. User greets with **I would like to know my account balance**.

2. Audio input from the mic is fed to audio ingestion.

3. Audio ingestion publishes audio buffer onto 0mq with topic ‘audio’.

4. The ASR subscribes to the topic ‘audio’ on the 0mq databus where it receives the audio buffers in chunks. ASR converts speech-to-text using pre-trained Deep learning models.

5. ASR publishes the text on the 0mq with topic as 'text'.

6. NLP receives the text and classifies it by intent. It then makes software calls corresponding to the intent. It can take software actions corresponding to that intent.

7. NLP makes a REST API (Application Program Interface) call to the OBP server to query the account balance.

8. OBP server responds to the REST API call with the account balance.

9. NLP processes and publishes the REST API call response on the 0mq databus with the topic as 'nlp'.

10. TTS, subscribed for the topic ‘nlp’, converts the text to wave format.

11. TTS publishes the text and wav on the 0mq with the topic 'tts'.

12. The TTS output is sent to the speaker and the reply is **Balance for the savings account ending with XXXX is XXXX.XXXX dollars**.

## Feature Support

Tables 3, 4, and 5 list the supported features in Interactive Kiosk AI Chatbot.

**NOTE:** We are not supporting all the APIs of a production deployment for banking. OBP provides a sandbox environment for the banking server.

|  |  |  |
| --- | --- | --- |
| **ASR Features** | **Support** | **Details** |
| Models are distributed as part of the Intel® Distribution of OpenVINO™ toolkit. | 1. Kaldi (default) 2. Mozilla DeepSpeech 3. QuartzNet | For more models, see [Learn More: US English ASR Models](#_US_English_ASR). |
| Other Models | 4. Huggingface |  |
| Languages and Accent | 1. Kaldi ASR: Generic US English 2. DeepSpeech ASR: American 3. QuartzNet ASR: LibriSpeech, Mozilla Common Voice (validated clips from en\_1488h\_2019-12-10), WSJ, Fisher, Switchboard, and NSC Singapore English. 4. Huggingface ASR: Librispeech clean | Retrain models for other languages and accents or replace them with a pre-trained model. |

**Table 3: ASR Feature Support**

For details about ASR models, see [Learn More](#_Learn_More).

|  |  |  |
| --- | --- | --- |
| **NLP Feature** | **Support** | **Details** |
| Machine learning framework | RASA Open Source version 1.10.1 | This feature is not suitable for production use. It is an open-source project, and no support is available. An enterprise version of RASA exists for production use. |
| Middleware | Open Bank Project | This middleware feature is not suitable for production use. The RI enables only the operations listed on the left for demo purposes only. |
| OBP Conversation Operations | Check account balance |
| Check bank name |
| Check for available cash machine |
| Greet the ChatBot |
| List the Accounts |
| Get bank location |

**Table 4: NLP Feature Support**

|  |  |  |
| --- | --- | --- |
| **TTS Feature** | **Support** | **Details** |
| Language | English | This feature produces .wav and text stream content users can use to develop a [DHI](#_Recommendations) proof-of-concept. Retrain if a male voice or a different accent is required. |
| Accent | American |
| Voice | Female |
| Trained Models | forward tacotron and melgan | Retrain for other models. |

**Table 5: TTS Feature Support**

The Conversational AI Chatbot generates text and wave data from spoken or typed content. Vendors can use these streams to create custom user interface implementations

# Get Started

## Run Methods

Decide how you’d like to run the application before attempting installation and build. There are two run modes:

* **Wav File Ingestion:** A quick test of the RI to illustrate how it responds to speech. Use this method to experience the RI with sample audio files. Use a speaker or headphones as the output device or monitor the log files.
* **Live Speech Ingestion:** An interactive test of the RI that involves speaking words and phrases into a microphone or mic array. Use this method with Seeed\* ReSpeaker or another device as the input mic. Use a speaker or headphones as the output device or monitor the log files.

**NOTE:** You can switch between wav file ingestion and live speech ingestion after installation by uninstalling, resetting the environment variable, and reinstalling.

Table 6 outlines the difference between ingestion approaches. The reference implementation has been tested with several different microphones and mic arrays, but it has not been tested exhaustively. See Target System Requirements.

|  |  |  |
| --- | --- | --- |
| **INGESTION TYPE** | **INPUT – OUTPUT DEVICES** | **NOTE** |
| **WAV FILE** | **INPUT EXAMPLES:**  Reference Implementation Sample Files: audio0.wav-audio3.wav.  Use provided sample audio files or replace with your own files. For details, see Create Wav Files. | * No input device required. * Environment variable: export INGESTION\_TYPE= wave\_ingestion * Listen to speaker or read log file text. |
| **OUTPUT EXAMPLES:**  Log files (text only), Headset (e.g., Jabra\* Evolve 40). |
| **LIVE SPEECH** | **INPUT EXAMPLES:**  On-system microphone, Seeed\* ReSpeaker Mic Array. | * Input device required. * Environment variable: export INGESTION\_TYPE= speech\_ingestion * Listen to speaker or read log file text. |
| **OUTPUT EXAMPLES:**  Log files (text only), Headset (e.g., Jabra\* Evolve 40). |

**Table 6: Speech Input and Output**

#### What You’ll Do

Here is a sneak preview of what you’ll do for both types of ingestion:

* Step 1: Input and Output: Install the speaker and microphone.
* Step 2: Install the Reference Implementation: Download the RI and set the environment variables. Build the source code.
* Step 3: Run the Application: Listen to the output or read log files.

## Get the Open Bank Project Credentials

The RI uses Open Bank Project (OBP) as the banking server and authentication method.

Follow the steps below for registering, obtaining a consumer key, and generating a token on the OBP server. This enables Interactive Kiosk AI Chatbot to make API calls.

**NOTE:** It is also possible to use the publicly available credentials at the [OBP GitHub Sandbox.](https://github.com/OpenBankProject/OBP-API/wiki/Sandbox)

### Register with the Open Bank Project

1. Register at [Open Bank Project](https://apisandbox.openbankproject.com/user_mgt/sign_up).
2. Fill out the registration form and choose **Sign Up.**

**NOTE:** The form requires:

* First name
* Last name
* Email address
* Username
* Password

1. Save the **username** and **password**. They will be used later.

### Generate Consumer API Key

1. To generate a consumer API key, choose **Get API key** from the top menu of [Open Bank Project](https://apisandbox.openbankproject.com/create-sandbox-account). NOTE: You may be prompted to log on.
2. From the **Application Type** pulldown, choose **Public**.
3. Fill in the remaining fields and click **Register consumer**.

**NOTE:** The form requires:

* Application type
* Application name
* Developer email
* Description of the application
* Company

1. Save the **consumer API key.**

### Create a Sandbox Account

Create a new sandbox account with the OBP with desired currency in dollars.

1. Copy the URL below to your browser:

<https://apisandbox.openbankproject.com/create-sandbox-account>

1. Create account for Bank ‘**bank-of-pune’ and the** **Desired Account Currency** enter **USD**.
2. For the **Desired Initial Balance** enter **1000.00.**

**NOTE:** The form requires:

* Bank
* Desired Account ID
* Desired Account Currency
* Desired Initial Balance
* Company

1. Fill in the form and choose **Create Account**.

Mic and Speaker: Input and Output

For live speech ingestion use a microphone or mic array and a speaker (e.g., headphones). Before running the application, attach a mic array and speaker as described in the steps below.

**NOTE:**For wave ingestion no Input mic required, setting for output speaker done as below.

*Mic or Microphone*

1. On the Linux host where the containers are deployed, connect a wired Mic array or microphone.
2. If ReSpeaker is connected, then ReSpeaker will be selected by default as Input device. If no ReSpeaker is connected then open **Settings -> Sound**. This can be reached from the Show Applications button (nine dots icon) or from the command line by typing **Ctrl-Alt-T** and then typing **gnome-control center** on the command line.
3. Choose the correct device for input.

*Speaker*

To hear output and experience the audio function of the RI, attach a wired speaker (e.g., headset).

1. On the Linux host where the containers are deployed, connect a wired Mic array or microphone.
2. Set the speaker in the RI. Select the required speaker by setting the ALSA\_CARD variable as described in [Step 2: (DXWT Method) Build the Application](bookmark://_Step_2:_(DXWT).

If you would prefer to read the output, use docker logs as described in [Wav File Ingestion](bookmark://_Wav_File_Ingestion).

Table 6 below presents speech input and output examples.

|  |  |
| --- | --- |
| **SPEECH INPUT** | **NOTES** |
| Reference Implementation Sample Files:   * audio0.wav * audio1.wav * audio2.wav * audio3.wav | Use provided sample audio files or replace with your own files. For details, see [Create Wav Files](bookmark://_Create_Wav_Files). |
| Microphone or Mic Array (e.g., Seeed\* ReSpeaker, Jabra\* Speak 410) | The reference implementation has been tested with several different microphones and mic arrays, but it has not been tested exhaustively. See [Target System Requirements](bookmark://_Target_System_Requirements). |
| Headset (e.g., Jabra\* Evolve 40) |
| On-system Microphone |
| **SPEECH OUTPUT** | **NOTES** |
| Log files | See [Log Files](bookmark://_Docker_Logs). |
| * Headphones or headset * On-system Speaker | The reference implementation has been tested with several different microphones and mic arrays, but it has not been tested exhaustively. See [Target System Requirements](bookmark://_Target_System_Requirements). |

**Table 6: Speech Input and Output**

Step 1: Install the Reference Implementation

Check Supported Speakers

Run the test\_audio\_hardware.sh script, attached below.



1. Change the permissions:

chmod a+x test\_audio\_hardware.sh

1. Run the script:

./test\_audio\_hardware.sh

===Check for Success===

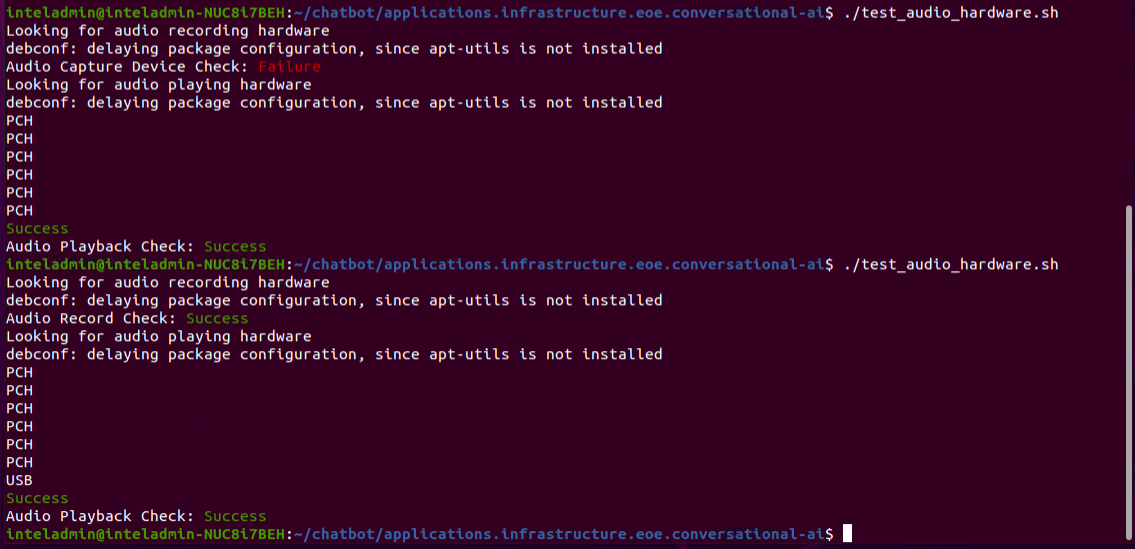
All supported devices from the Docker container will be listed as in the Figure 3 below. Table 7 describes the two script invocations.

|  |  |
| --- | --- |
| **Hardware** | **Script Output** |
| In the first script run, a set of headphones was plugged into the AUX port. No mic was available. | The script reported a **FAILURE** to find a capture device and a **SUCCESS** for finding an audio player device.\* |
| In the second, a conference speaker phone was plugged into a USB port. | The script reported a **SUCCESS**for finding the capture device and a **SUCCESS** for finding an audio player device.\*\* |

**Table 7: Script Output**

**\*NOTE:** This combination works if you are planning to use only [Wav File Ingestion](bookmark://_Wav_File_Ingestion_1) for the input.

**\*NOTE:** You must have both a speaker and mic for [Live Speech Ingestion](bookmark://_Live_Speech_Ingestion_1).



**Figure 3: test\_audio\_hardware.sh script Run Examples**

If your speaker is not shown in the device list, the speaker is not supported from inside the docker container. Try restarting docker: sudo systemctl restart docker.

Download the Reference Implementation

Deploy the containers using docker compose file. This deployment method supports developers who want to replace any of the microservices, or the Banking server provided in the Refence Implementation. RI supports two methods of deployment, live speech ingestion or wav file ingestion.

**NOTE:** Refer to the Troubleshooting section for the docker commands to clean up the environment.

1. Clone repo on the Linux host with command:

git clone <https://github.com/intel/conversational-ai-chatbot>

The repo will be cloned under the directory ‘conversational-ai-chatbot/’.

1. Run command to change the directory to cloned:

cd conversational-ai-chatbot/

Step 2: Build the Application

1. Run the command to set environmental variables

source ./setenvvars.sh

**NOTE:** The environment variables are not optional.

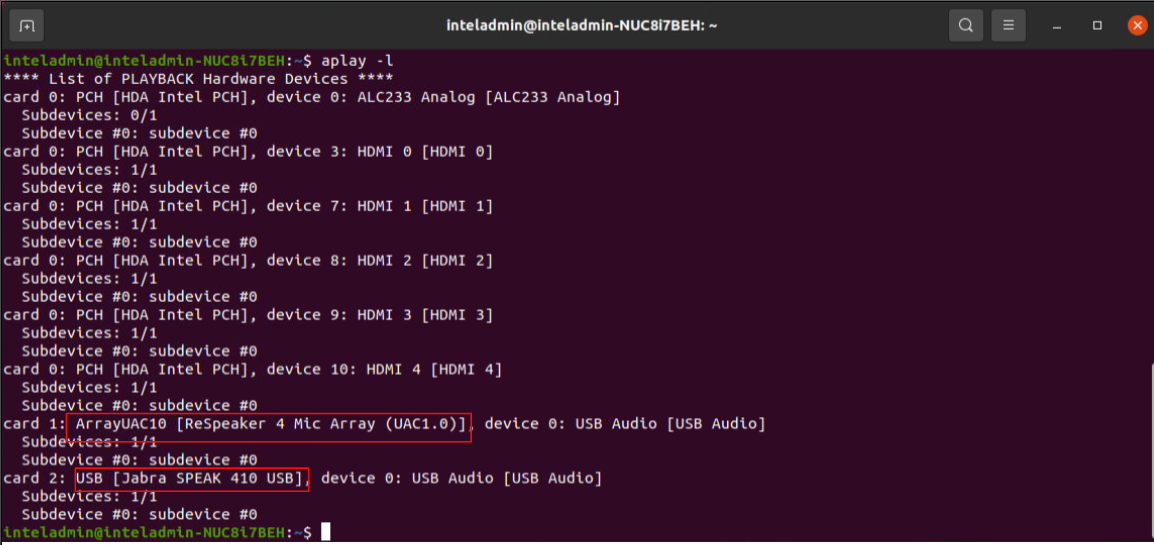
1. Enter OBP creds. List available sound cards with the aplay command.  The command will generate an output like that in the Figure 3 below.

aplay -l

1. Choose an output (i.e., the speaker which will play the application’s output)

**NOTE:** ALSA\_CARD variable is only for selecting the output device.

**NOTE**: As shown in Figure 3, ‘ArrayUAC10’ is name of reSpeaker Mic array name and ‘Seri’ is Plantronics Blackwire 3220 sound card name.



**Figure 3: List Sound Devices**

**NOTE:** For input Mic, if ReSpeaker is plugged into a USB port, the ReSpeaker device is always taken as the input mic. Else the Mic selected for Input in Sound settings will be considered as input device.

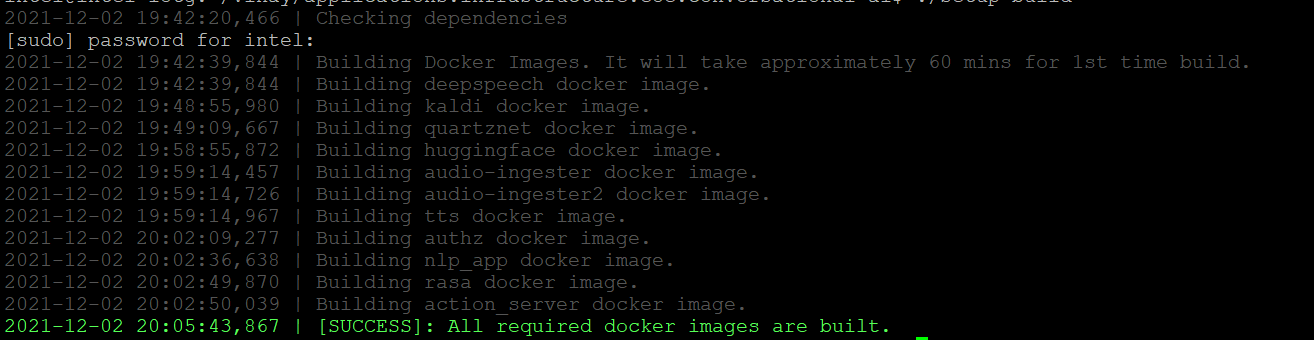
1. Build all the microservices with the command:

     ./setup build

The build may take up to **40 minutes**. After the build process is complete verify that image is created for all the services.

===Check for Success===

Look for a series of messages as shown in Figure 4.



**Figure 4: Successful Build**

Verify all images are built with command:

docker images

1. action\_server
2. rasa
3. nlp\_app
4. deepspeech\_asr
5. kaldi\_asr
6. audio-ingester2
7. audio-ingester
8. tts
9. authzap
10. huggingface\_asr
11. quartznet\_asr

Step 3: Run the Application

Run Methods

There are two ways to run the application:

1. [**Wav File Ingestion**](bookmark://_Wav_File_Ingestion_1)**:** A quick test of the RI to illustrate how it responds to speech. Use this method to see the RI run with sample audio files. Use a speaker or headphones as the output device or monitor the log files.
2. [**Live Speech Ingestion**](bookmark://_Live_Speech_Ingestion)**:** An interactive test of the RI that involves speaking words and phrases into a microphone or mic array. Use this method with Seeed ReSpeaker or another device as the input mic. Use a speaker or headphones as the output device or monitor the log files.

**NOTE:** To switch between wave file ingestion and live speech ingestion, stop the stack. See [Clean Up Stack](bookmark://_Clean_Up_Stack).

Wav File Ingestion

For wav file ingestion with sample audio files, follow the instructions below to start the microservices.

1. Run wav file ingestion:

**NOTE:** Environment variables should be set per the instructions in Build the Application. The ALSA\_CARD variable must be set before running wave ingestion if you want to listen to output. [Step 2: (DXWT Method) Build the Application](bookmark://_Step_2:_(DXWT).

./setup run wave\_ingestion

===Check for Success===

After the deployment, run:

  docker ps

Make sure all the containers are started:

1. chat\_backend\_authz
2. chat\_frontend\_audio\_ingestion\_2
3. chat\_backend\_nlp\_app
4. chat\_backend\_asr\_speech
5. chat\_backend\_rasa\_action
6. chat\_frontend\_tts\_1

1. To see input and output speech in the log files, open two terminal windows and follow the instructions in [Docker Logs](bookmark://_Docker_Logs_1).  The log files of interest are NLP and ASR, which will show the speech input and output.
2. It is also possible to listen to the audio if you are using the speaker and set the ASR\_IMAGE variable during the build. See [Step 2: (DXWT Method) Build the Application](bookmark://_Step_2:_(DXWT). The audio “Welcome to the Future of Banking” is the first greeting followed by other Banking queries.
3. To stop wav file ingestion, run:

./setup stop wave\_ingestion

===Check for Success===

You will see the last message of the wav file ingestion stop script:

[INFO] Removing Docker Secrets

However, the stop script may take one or two minutes to complete cleanup tasks. Wait for the terminal prompt before moving on to other instructions.

**NOTE:** Stop wav file ingestion before starting live speech ingestion, integrating your own audio files, or modifying the ASR model.

Live Speech Ingestion

**NOTE:**Modify the wake word (or wake-up word) in the file **docker-compose-frontend-respeaker.yml**.  In the section **services:environment**, modify the wake word by setting **WAKE\_UP\_WORD** under the audio-ingestion2 service. The wake word alerts the RI to start listening.

**NOTE:** The default wake word is **‘respeaker’** (pronounced ree-spee-kr). The word **‘chatbot’** is another wake word that has been validated. Choose wake words carefully as background noise can distort the sound of the word. Wake words that sound like other words perform poorly.

For live speech ingestion with a microphone or mic array, follow the instructions below to start the microservices.

**NOTE:** The ALSA\_CARD variable must be set before running wave ingestion if you want to listen to output. [Step 2: (DXWT Method) Build the Application](bookmark://_Step_2:_(DXWT).

1. Run live speech ingestion:

   ./setup run speech\_ingestion

 ===Check for Success===

After the deployment, run:

  docker ps

Make sure all the containers have started:

1. chat\_backend\_authz
2. chat\_frontend\_audio\_ingestion\_2
3. chat\_backend\_nlp\_app
4. chat\_backend\_asr\_speech
5. chat\_backend\_rasa\_action
6. chat\_frontend\_tts\_1

1. To see input and output speech in the log files, open two terminal windows and follow the instructions in [Docker Logs](bookmark://_Docker_Logs).  The log files of interest are NLP and ASR.
2. Provide voice input through the mic with a wake word, a word that alerts the RI to start listening for speech. The wake word is **ReSpeaker** (ree-spee-kr).

**NOTE:** Provide a delay of at least one second between the wake word and input. Speak into a speaker at very close range—as close as possible.

Audio output will be played on the connected speaker.

**Greeting Use Case**

**Input:** ReSpeaker, [pause one second] good morning or ReSpeaker, hello.

**Expected Log or Audio Output:** Welcome to the future of banking.

**Request Use Case**

**Input:**ReSpeaker, [pause one second] I would like to know my account details.

**Expected Log or Audio Output:** TBD

Try the voice input expressions in the table below. Check the output in the log files or listen to it through a speaker.

Table 8 lists sample speech input and corresponding output; however, the framework supports more varied input. See [NLP Modification](bookmark://_NLP_Modification)

|  |  |
| --- | --- |
| **Voice Input\*** | **Example Output from TTS\*\*** |
| Good morning | Welcome to the future of banking. |
| I would like to know my account details | You have \*\*\*\* savings account with our bank. |
| I would like to know my account balance | The balance for the savings account ending with xxxx is xxxx. |
| Find me the nearest cash machine | You will find one near MG Road. |
| Goodbye. | Goodbye. Thank you for banking with us. See you soon. |

**Table 8: Possible Speech Inputs and Outputs for Testing**

\* The user input is greeting and banking related queries.

\*\* As configured with NLP and account details registered with OBP server.

**NOTE:** If there are multiple inputs not separated by the wake word, the RI will respond with only one output. The output it chooses will be for the input for which it has the most confidence.

1. To stop live speech ingestion, run:

./setup stop speech\_ingestion

===Check for Success===

You will see the last message of the wav file ingestion stop script:

[INFO] Removing Docker Secrets

However, the stop script may take one or two minutes to complete cleanup tasks. Wait for the terminal prompt before moving on to other instructions.

## Tutorials

The RI provides a microservices framework that you can customize by replacing components and models. The following sections outline how to make small, comparatively easy changes to the architecture. For resources on model training, a more complex process, see [Learn More: Train the Models](#_Train_the_Models).

### Replace the ASR model

To change the model in the RI:

1. Stop the reference implementation if it’s running:

./setup stop speech\_ingestion

or

./setup stop wave\_ingestion

1. Change the ASR model by setting the ASR\_IMAGE variable to the correct string:

The RI supports these models:

|  |  |
| --- | --- |
| **Model** | **String** |
| Quartznet | quartznet\_asr |
| Kaldi | kaldi\_asr |
| Deep Speech | deepspeech\_asr |
| Huggingface | huggingface\_asr |

**Table 9: Supported Models**

export ASR\_IMAGE=<ASR Image name string>

1. Deploy again:

./setup run speech\_ingestion

or

./setup run wave\_ingestion

### Modify NLP

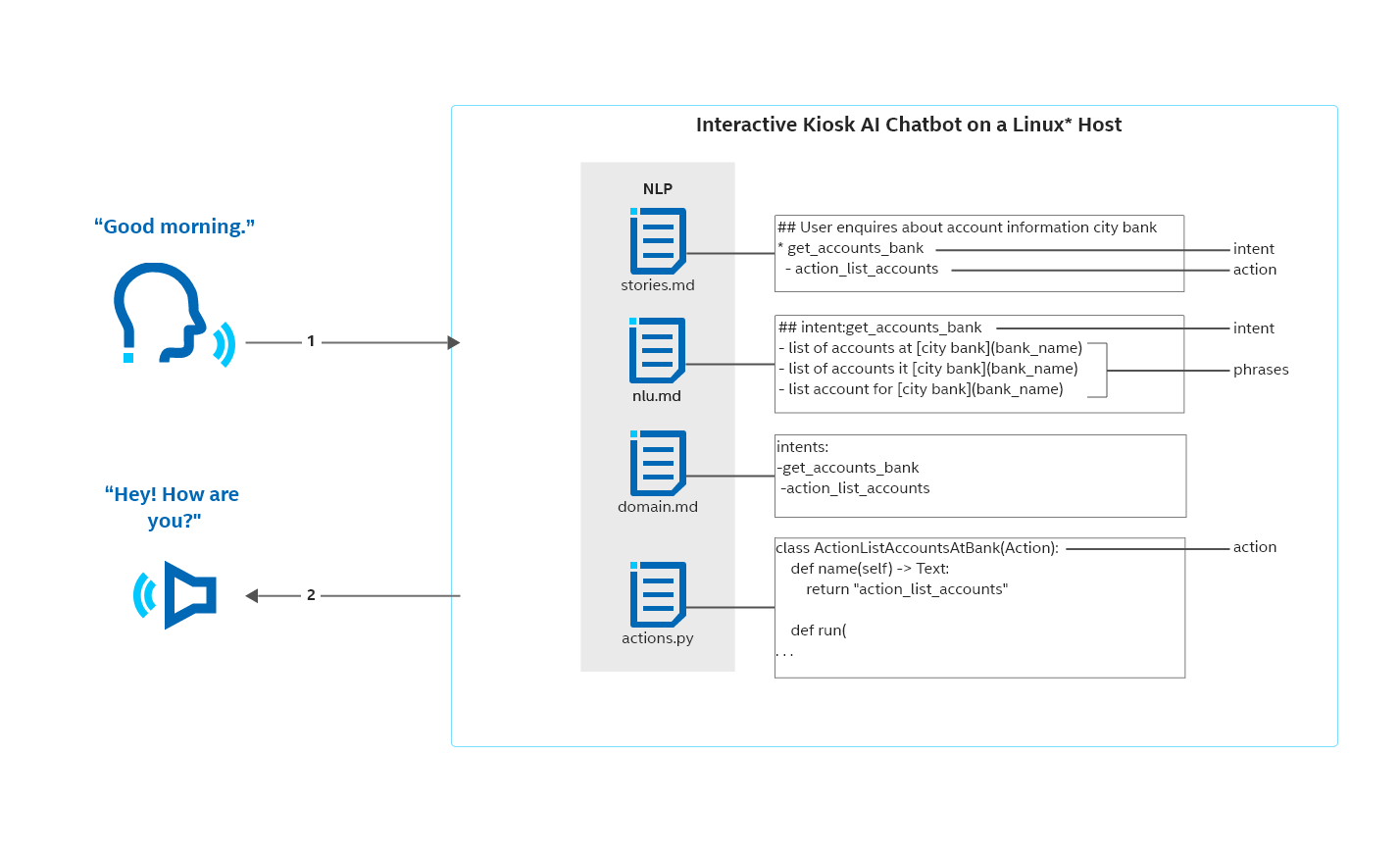
A basic understanding of Rasa is necessary to add more banking APIs or replace the default banking server, OBP, with a custom banking server solution. The following tutorial demonstrates a simple modification to the NLP.

Below is a table of terminology used in the instructions below.

|  |  |
| --- | --- |
| **RASA Terminology** | **Definition** |
| Keywords | Words or phrases that are significant in a verbal or text interaction. Example: “Hello there!” “List account for \_\_\_\_\_” |
| Intents | Groups of keywords.  Example:  #intent:greet   * hello there * hey * hiya |
| Actions | Source code that executes to support banking interactions and play TTS responses. |
| Stories | Relationships between intents and actions. |

**Table 10: RASA Terminology**

Figure 6 illustrates the relationships of files in the NLP.



**Figure 6: Files in the NLP**

See Rasa tutorials to learn more about [Rasa](https://rasa.com/). Refer to [Rasa docs](https://legacy-docs.rasa.com/docs/) for detailed instructions for extending and adding keywords, intents, and responses.

#### Add (or Delete) a Keyword in the NLP

To add a new keyword:

1. Open ~ nlp/rasa\_api\_server/data/nlu.md.
2. Add a new keyword or multiple keywords to the greet intent by adding the keywords to the bottom of the intent, each on a new line starting with a “- “:

Examples:

* how do you do
* howdy
* greetings chatbot

**NOTE:** No sentence punctuation or capitalization is necessary.

1. Stop the chatbot stack if already deployed with the command:

./setup stop speech\_ingestion

or

./setup stop wave\_ingestion

1. Delete the action server image if it is running with the command:

docker image rm <action\_server Image ID>

1. Rebuild the rasa action server:

docker build -f dockerfiles/rasa\_action\_server.dockerfile -t action\_server:$TAG .

1. Deploy the stack with command:

./setup run speech\_ingestion

or

./setup run wave\_ingestion

1. Test the new keyword as Input by speaking into the speaker or with wave file.

### Modify the Wake Word

The wake word (or wake-up word) alerts the RI to start listening. Preface each interaction by reciting the wake word followed by a pause and then the greeting or request.

Example:

Respeaker [pause], good afternoon.

Currently the default wake word is **respeaker** (pronounced ree-spee-kr). The word **chatbot** is another wake word that has been validated. Choose wake words carefully as background noise can distort the sound of the word. Wake words that sound like other words perform poorly.

To modify:

* 1. Stop the reference implementation if it’s running:

./setup stop speech\_ingestion

or

./setup stop wave\_ingestion

* 1. Open the file ./compose/docker-compose-frontend-respeaker.yml.
  2. In the section **services:environment**, modify the wake word by setting **WAKE\_UP\_WORD** under the **audio-ingestion2 service**.
  3. Deploy the reference implementation:

./setup run speech\_ingestion

or

./setup run wave\_ingestion

# Summary and Next Steps

In Get Started and Tutorials, you learned to:

* Install and build a framework of microservices for wave and speech ingestion.
* Interact with your microphone and speaker to experience both wav file and live speech ingestion in a defined conversation scenario.
* Complete a tutorial to Replace the ASR Model, Modify the NLP, and Modify the Wake Word.

For more about the model details, model training, and other reference topics, see [Learn More](#_Learn_More).

# Learn More

### US English ASR Models

|  |  |  |  |
| --- | --- | --- | --- |
| **MODEL** | **Word Error Rate (WER)** | **Accuracy Factors** | **To Learn More** |
| Kaldi ASR | 10.5% for input as in trained set of the Librispeech Corpus | * accent and intonation of speaker * age of speaker (under representation of children’s speech), * noise level * microphone used * distance of a speaker to microphone   It cannot be used in production due to limited training dataset and domain. | [Librispeech](https://download.01.org/opencv/2020/openvinotoolkit/2020.1/models_contrib/speech/kaldi/librispeech_s5/) |
| DeepSpeech ASR | 5.9% | * best in low-noise environments * bias toward US male accents. | [DeepSpeech](https://github.com/mozilla/DeepSpeech/releases/tag/v0.8.2) |
| QuartzNet ASR | LibriSpeech: 3.79%  Dev-other: 10.05% | Trained on Six Datasets   * LibriSpeech * Mozilla Common Voice (validated clips from en\_1488h\_2019-12-10) * WSJ * Fisher * Switchboard * NSC Singapore English |  |
| Huggingface | 3.4% | Hours of Training:  16kHz sampled speech audio 960 hours on Librispeech |  |

**Table 11: Model Details**

### Create Wav Files

The RI accepts recorded audio files for ingestion. The files must meet these requirements:

* **Format: .**wav
* **Sampling rate:** 16 KHz
* **Recording channel:** Mono

Use audio software, such as [Audacity](https://www.audacityteam.org/), to create the files.

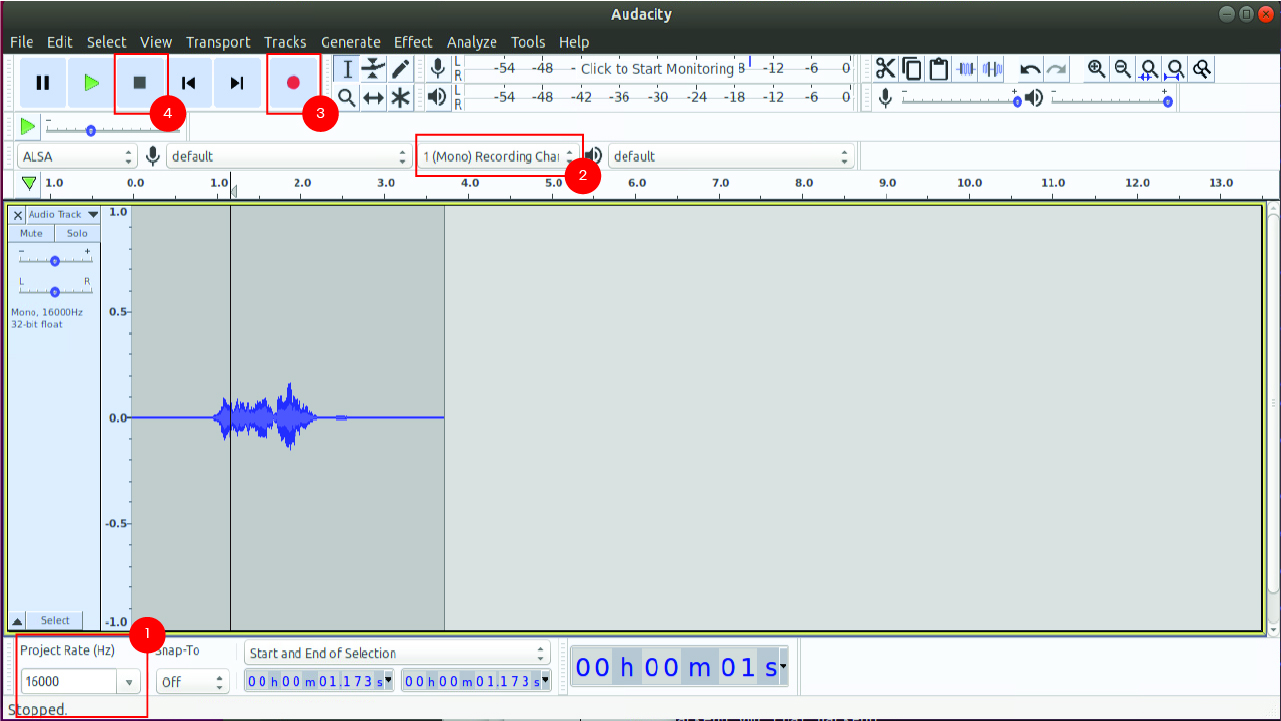
For the section [Wav File Ingestion](#_Wav_File_Ingestion_3), create separate audio files for each entry described in the table below.

|  |  |
| --- | --- |
| **Name the File** | **Speak and Record** |
| query1.wav | “Good morning” |
| query2.wav | “Where is the nearest cash machine?” |
| query3.wav | “List my accounts.” |
| query4.wav | “What is my account balance?” |
| query7.wav | “Goodbye” |

**Table 12: List of wav Files**

To create a wav file:

1. Set the sampling rate by choosing **16 KHz** in the **Project Rate** dropdown.
2. Set the recording channel to Mono in the **Recording channel** dropdown.
3. To record, click the **record button**. Speak into the mic.
4. Click the **stop button**.
5. Save file in wav file format, by choosing **File > Save As.**



**Figure 7: Audacity**

### Train the Models

#### DeepSpeech

To retrain the ASR’s DeepSpeech model:

1. Refer to [Training Your Own Model](https://deepspeech.readthedocs.io/en/v0.9.3/TRAINING.html) to understand how to re-train DeepSpeech’s model.
2. Refer to [Overview of OpenVINO™ Toolkit Intel's Pre-Trained Models](https://docs.openvinotoolkit.org/latest/omz_models_group_intel.html) for Intel pre-trained models. These are open source pre-trained models for demo purposes.
3. To convert the model generated by Step 1 to Intermediate Representation, use the tutorial [Convert TensorFlow\* DeepSpeech Model to the Intermediate Representation.](https://docs.openvinotoolkit.org/latest/openvino_docs_MO_DG_prepare_model_convert_model_tf_specific_Convert_DeepSpeech_From_Tensorflow.html)
4. Update the path for the IR generated in Step 2 in ‘inference:model\_xml’, ‘inference:model\_bin’, and ‘~asr\_deepspeech/src/model/deepspeech.cfg’.
5. Build and restart the DeepSpeech ASR container.

#### Kaldi

To retrain the pre-trained Kaldi model:

1. Download the appropriate [model](https://kaldi-asr.org/models.html).
2. Extract the zip file.
3. Refer to the steps in the Readme file to retrain model or fine-tune the model.
4. Convert the model generated by Step 3 to intermediate representation (IR). Use the tutorial [Converting a Kaldi\* Model](https://docs.openvinotoolkit.org/2021.2/openvino_docs_MO_DG_prepare_model_convert_model_Convert_Model_From_Kaldi.html.).
5. Update the path for the IR generated in Step 4 in acousticModelFName , outSymsFName, fsmFName, featureTransform in ‘~ asr\_kaldi/src/model/speech\_lib.cfg’.
6. Build and restart the kaldi ASR container.

#### Huggingface

Find the fine-tune and training new model procedure for the Huggingface ASR at [wav2 sec](https://github.com/pytorch/fairseq/blob/main/examples/wav2vec/README.md).

#### Quartznet

Refer to for procedure to train the QuartzNet ASR model, refer to [Building Speech Recognition Models for Global Languages with the Mozilla Common Voice Dataset and NVIDIA NeMo](https://foundation.mozilla.org/es/blog/building-speech-recognition-models-for-global-languages-with-the-mozilla-common-voice-dataset-and-nvidia-nemo/).

#### TTS Models

To retrain the tacotron model, refer to [forward-tacotron (composite)](https://docs.openvinotoolkit.org/latest/omz_models_model_forward_tacotron.html).

### Deploy OBP Server Locally

OBP server can be built and deployed locally. Refer to open source code and build instructions available at [Open Bank Project](https://github.com/OpenBankProject).

### Deploy Custom Containers

Developers can leverage this microservice-based architecture and replace any component to create a customized solution. Follow the instructions below, which describe the required modifications.

To replace a microservice container:

1. Import the ‘integration\_library/zmq\_integration\_lib.py’ file for communication over 0mg.
2. Use Class ‘InputPort’ or Input data over 0mq.
3. Use Class ‘OutputPort’ for output data over 0mq.
4. Use Class ‘InputPortWithEvents’ to receive Input data over 0mq on occurrence of an event.

**NOTE:** Refer to the comments at the start of zmq\_integration\_lib.py for details about how to use the classes InputPort, OutputPort and InputPortWithEvents.

### Use a Custom NLP Model

To use a pre-trained custom model in the NLP, refer to the [Enhancing Rasa NLU models with Custom Components](https://blog.rasa.com/enhancing-rasa-nlu-with-custom-components/).

### Support Additional Banking APIs

1. Declare the intents and actions in ~nlp/rasa\_api\_server/domain.yml.
2. Add the keywords for the intents in ~nlp/rasa\_api\_server/data/nlu.md.
3. Map the action with the corresponding function call handled by the API server (requiring call to be made with OBP server) in nlp/rasa\_api\_server/data/stories.md.
4. NLP custom actions are defined in ‘actions.py’ under directory ‘nlp/rasa\_actions\_server/src/’. Replace/Add the classes for the corresponding action [for example to replace list Banks API, replace the class ‘ActionListBanks’].
5. The API call wrappers with OBP are defined in the functions inside the python files of the directory ‘nlp/obp\_api’. The developer needs to have corresponding wrappers for the new Banking server defined.
6. Stop the chatbot stack if already deployed with command:

./setup stop speech\_ingestion

or

./setup stop wave\_ingestion

1. Delete action server image if already exists with command:

docker image rm <action\_server Image ID>

1. Rebuild rasa action server with command: docker build -f dockerfiles/rasa\_action\_server.dockerfile -t action\_server:$1.0 .
2. Deploy the stack with command:

./setup run speech\_ingestion

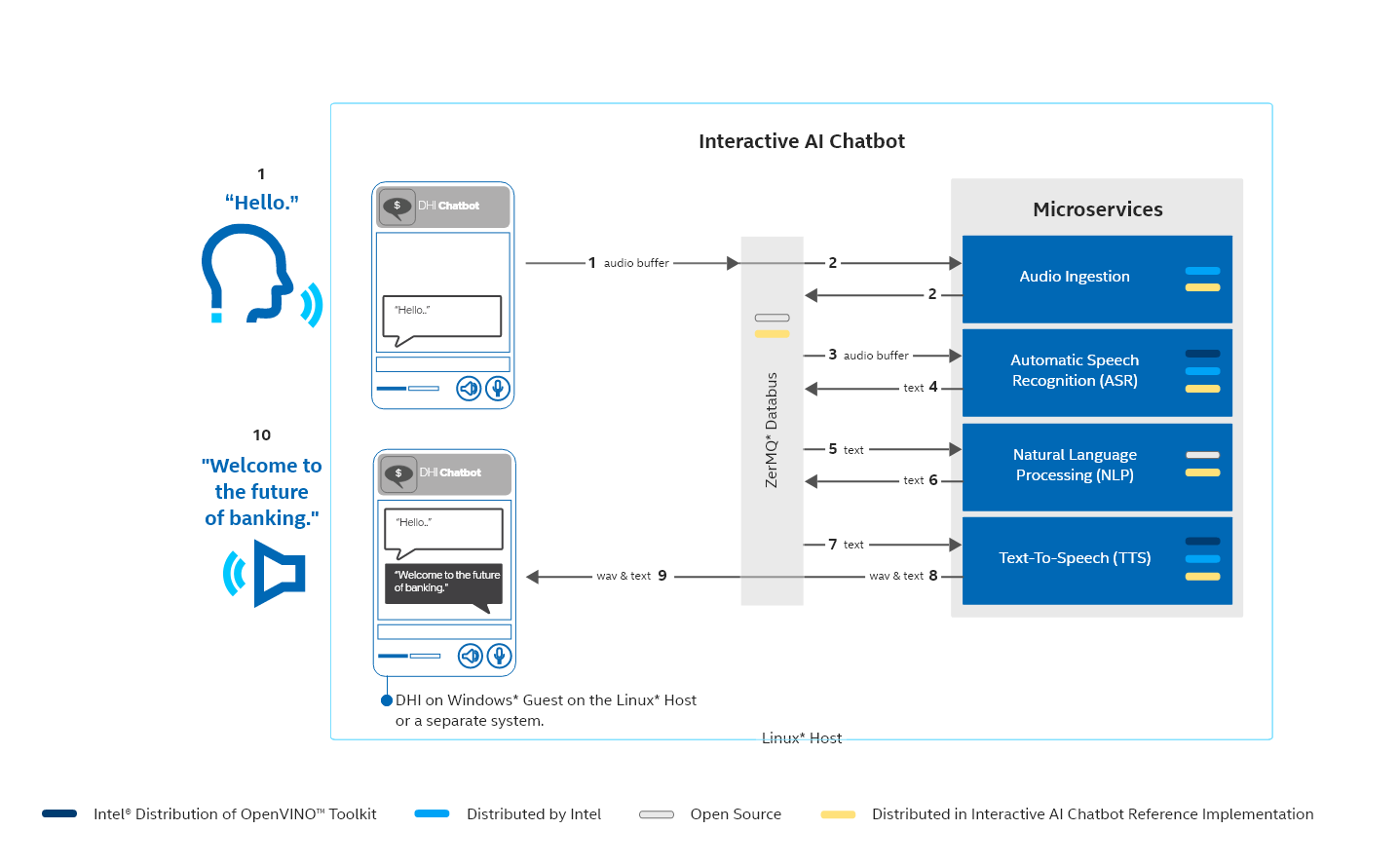
or

./setup run wave\_ingestion

### Integrate a DHI or Chat Window

The RI is optimized for Intel® platforms. It can be extended to run on a single host, with the DHI and the microservices running on the same system. Alternatively, it can be run on two different hosts, with a DHI and voice interface running on Windows.

The audio ingestion supports only voice activity detection (VAD) which can be replaced with a microservice supporting VAD, direction of arrival, beamforming, noise suppression, de-reverberation, and acoustic echo cancellation (AEC). Figure 6 shows the DHI hosted on a Linux\* system with a Windows\* guest.

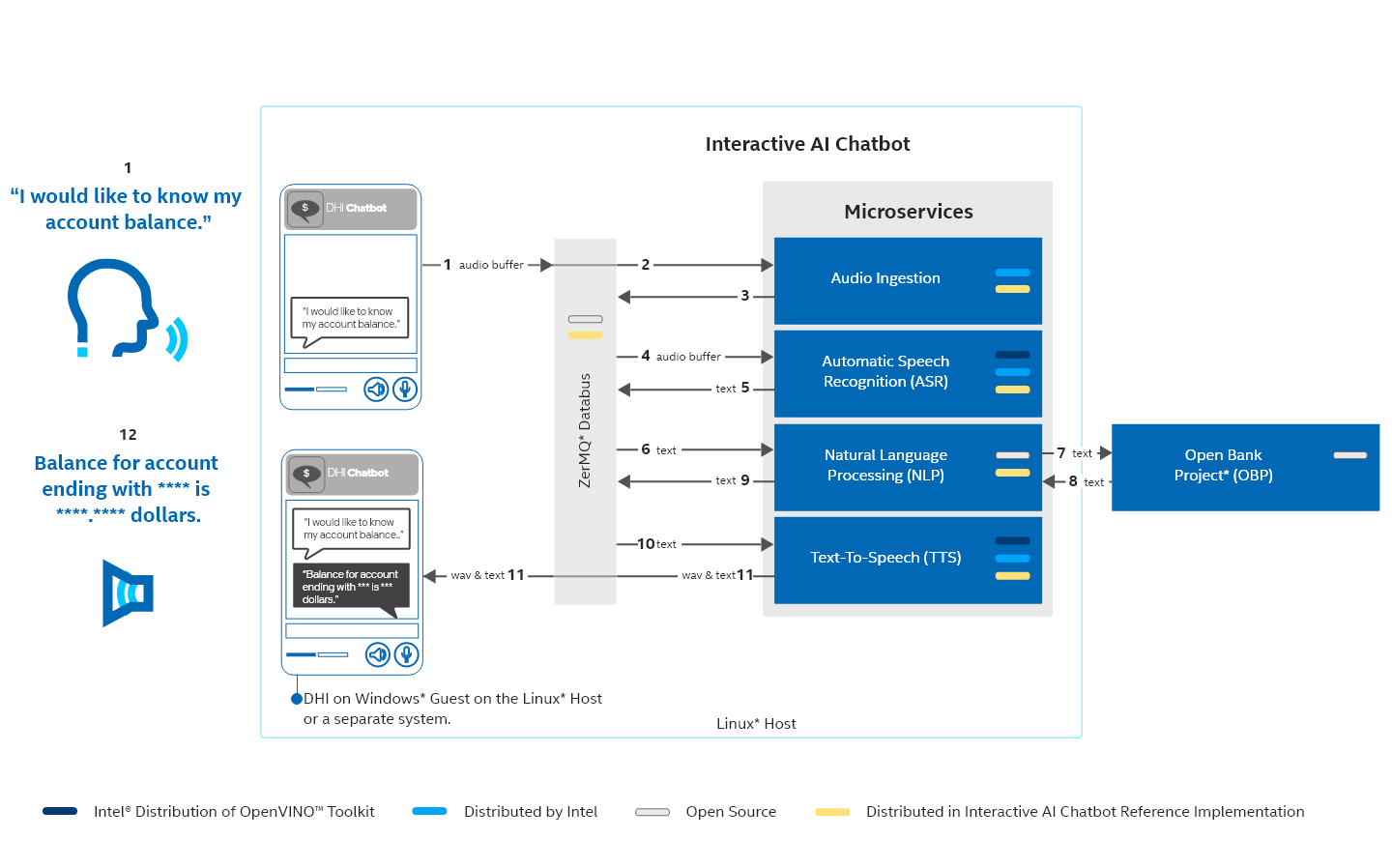


**Figure 8: Software Flow: Greeting**

Figure 8 presents the software flow that occurs when an end user engages with the chatbot in conversation that is **not** banking-related:

1. A banking customer approaches the kiosk mic and greets it with “Hello”, “**Good morning**” or a similar greeting.
2. The ReSpeaker Voice Interface performs pre-processing, including noise-cancelling, and publishes the input on the 0mq databus with topic ‘audio’ in the form of an audio buffer.
3. The ASR subscribes to the topic ‘audio’ on the 0mq databus where it receives the audio buffers in chunks. ASR converts speech-to-text using pre-trained Deep learning models.
4. ASR publish the text on the 0mq with topic as 'text'.
5. NLP receives the text and classifies it by intent. It then makes software calls corresponding to those intents.
6. NLP processes and publishes the action as text on the 0mq databus on topic 'nlp'.
7. TTS subscribed for the topic ‘nlp’, converts the text to wave format.
8. TTS publishes the text and wave on the 0mq with the topic 'tts'.
9. A DHI subscribes to the 'tts' topic on the 0mq databus.
10. The subscribed DHI outputs the wav to the speaker and text in the chat window. The DHI responds with a “Hello! How can I help you?”.

**NOTE:** Step 10 is implementation dependent. The DHI can present an avatar lip-syncing a wave file or present text that can be incorporated in a chat window.



**Figure 9: Software Flow: Banking Query**

Figure 9 presents a software flow that occurs when an end user engages with the chatbot in conversation that is banking related. The NLP in steps 5 and 6 exchanges data with the Open Banking Project module.

1. A banking customer approaches the kiosk mic with a request, “I would like to know my account balance”.
2. The ReSpeaker Voice Interface performs pre-processing, including noise-canceling, and publishes the input on the 0mq databus in the form of an audio buffer.
3. The ASR subscribes to the topic ‘audio’ on the 0mq databus where it receives the audio buffers in chunks. ASR converts speech-to-text using pre-trained Deep learning models.
4. ASR publishes the text on the 0mq with topic as 'text'.
5. NLP receives the text classifies it by intent. It then makes software call corresponding to those intents.
6. NLP makes a REST API call to the OBP server to query the account balance.
7. OBP server responds to the REST API call with the account balance.
8. NLP processes and publishes the REST API call response on the 0mq databus with the topic as 'nlp'.
9. TTS is subscribed for the topic ‘nlp’ and converts the text to wave format.
10. TTS publishes the text and wave on the 0mq with topic 'tts'.
11. A DHI subscribed to the 'tts' topic on the 0mq databus.
12. The DHI subscribed, outputs the wav to the speaker and text on chat window. The DHI responds with a playing the wave “Balance for the account is 100 dollars” on speaker and the text on chat window [example use case where in the account balance as configured with OBP].

Step 12 is implementation dependent. The DHI can present an avatar lip-syncing a wave file or present text that must be read.

#### Output from TTS to DHI

The microservices communicate over interprocess communication (IPC).

To include the DHI with the RI, create a microservice on the ubuntu host to subscribe with the output port of the TTS on the 0mq. Refer to docker-compose file for the output port. The newly created microservice will have a port exposed from which the DHI can receive the wave and text output.

## 

## Troubleshooting

### Modify Subgroup

To ensure that Docker has sudo permission, add your user to the docker subgroup after Docker has been installed:

sudo usermod -aG docker $USER

This enables you to run the reference implementation without the sudo command.

### Build Issues

ENVIRONMENT VARIABLES

If you receive the error below during a build, check that you have set all environment variables properly. In the example, the proxy variable was not set.

Example:

./setup build

Traceback (most recent call last):

File "./setup", line 508, in <module>

app = ConversationalAI(cwd, cwd)

File "./setup", line 64, in \_\_init\_\_

self.http\_proxy = os.environ["http\_proxy"]

File "/usr/lib/python3.8/os.py", line 675, in \_\_getitem\_\_

raise KeyError(key) from None

KeyError: 'http\_proxy'

OBP CREDENTIALS

If the package installs the Docker Engine and Docker Compose, but is unable to install interactive\_kiosk\_ai\_chatbot, check that the OBP credentials are entered properly.

### Docker Logs

1. In a terminal(s), get the container IDs:

sudo docker ps --format "table {{.Image}}\t{{.Status}}\t{{.ID}}"

1. View the log by running docker logs with the container ID:

docker logs -f <container-ID>

To see the input and output of the RI, use the container ID for NLP and ASR.

1. Return to [Wav File Ingestion](#_Wav_File_Ingestion_3) or [Live Speech Ingestion](#_Live_Speech_Ingestion) for details about running the application or see the models section to test with [other ASR models](#_Step_3b:_(DXWT).

|  |  |
| --- | --- |
| **Container Log File** | **Use it to confirm** |
| ASR | Speech the ASR has recognized |
| NLP | Recognized speech and Output |
| AUTHZ | OBP credential login success |

### Docker Pull Limit Issue

If you exceed your pull rate limit, you will receive an error:

ERROR: toomanyrequests: You have reached your pull rate limit. You may increase the limit by authenticating and upgrading with [Increase Rate Limit](https://www.docker.com/increase-rate-limit).

If you experience this error, login with your Docker premium account.

Example:

docker login

### Rebuilds

If an image is listed as **Up to date** during a Docker build**,** then rebuild the image.

Modify the docker file with the command touch dockerfiles/<name-of-dockerfile> which is available inside directory ‘./dockerfiles’.

### Container Logs

For any issue with the running containers, access logs with the docker logs command and container ID.

Example:

docker logs -f <container ID>

### Clean Up Stack

Stop the running AI Chatbot stack with the command that corresponds to the deployment method commands:

./setup stop speech\_ingestion

or

./setup stop wave\_ingestion

### Intel® Distribution of OpenVINO™ Toolkit Support

For help with Intel® Distribution of OpenVINO™ Toolkit, raise the support ticker at [Support for OpenVINO™ toolkit](https://www.intel.com/content/www/us/en/support/products/96066/software/development-software/openvino-toolkit.html) or at [github](https://github.com/openvinotoolkit/openvino/issues).

### Known Issues

The RI contains these known issues:

* Currently, the output of the RI is limited to a few phrases and sentences.

### Support Forum

If you're unable to resolve your issues, contact the [Support Forum](https://software.intel.com/en-us/forums/intel-edge-software-recipes).