Installation Guide

AudioCodes WebRTC Solutions for Enterprises

Agent Assist Bot

Version 1.0

audiocodes

Table of Contents

Notice iii

WEEE EU Directive iii

Customer Support iii

Stay in the Loop with AudioCodes iii

Abbreviations and Terminology iii

Document Revision Record iii

Documentation Feedback iv

1 Introduction 1

2 Agent Assist Bot Functionality 2

3 VoiceAI Connect Web Portal Bot Configuration 3

4 Bot/Phone JSON Protocol 4

5 Language Configuration 5

6 Docker Installation 7

6.1 Start Bot 8

6.2 Build Bot 9

6.3 Dockerfile 10

6.4 Configuration 11

6.4.1 Environment Variables 11

6.4.2 Secrets 11

6.4.3 Health Check 12

7 Kubernetes Installation 13

7.1 ReplicaSet and Deployments 13

7.2 Preparations 14

7.2.1 Create Secrets 14

7.2.2 Build Docker Image and Save it to k0s Local Registry 14

7.3 Single Container Instance 15

Notice

Information contained in this document is believed to be accurate and reliable at the time of printing. However, due to ongoing product improvements and revisions, AudioCodes cannot guarantee accuracy of printed material after the Date Published nor can it accept responsibility for errors or omissions. Updates to this document can be downloaded from https://www.audiocodes.com/library/technical-documents.

This document is subject to change without notice.

Date Published: June-21-2023

WEEE EU Directive

Pursuant to the WEEE EU Directive, electronic and electrical waste must not be disposed of with unsorted waste. Please contact your local recycling authority for disposal of this product.

Customer Support

Customer technical support and services are provided by AudioCodes or by an authorized AudioCodes Service Partner. For more information on how to buy technical support for AudioCodes products and for contact information, please visit our website at https://www.audiocodes.com/services-support/maintenance-and-support.

Stay in the Loop with AudioCodes

Abbreviations and Terminology

Each abbreviation, unless widely used, is spelled out in full when first used.

Document Revision Record

LTRT	Description
14220	Initial document release for Version 1.0.

Documentation Feedback

AudioCodes continually strives to produce high quality documentation. If you have any comments (suggestions or errors) regarding this document, please fill out the Documentation Feedback form on our website at https://online.audiocodes.com/documentation-feedback.

1 Introduction

This document describes Agent Assist bot functionality and its installation.

2 Agent Assist Bot Functionality

This bot was written in NodeJS and uses the AudioCodes bot API. For more information, refer to the AudioCodes bot API.

It is configured to be called from the AudioCodes Voca server during a SIP call and receives speech-to-text recognition from both sides of the call. It is assumed that one of the call participants is calling from the AudioCodes React WebRTC phone.

After the bot session has started, it receives information about the call participants including:

- SIP usernames
- Call initiator
- Which party (call initiator or answering the call) is the AudioCodes React phone

Roles (e.g., agent/customer) can be assigned to the React phone user/remote side. After the session has started, the bot sends information to the React phone (via API “sendMetaData”). It provides its own URL (e.g., wss://bot.example.com/session-key/) and is used in the session language.

The React phone opens a secure WebSocket connection to the bot. During the call, the bot and React phones send each other messages using the custom JSON protocol:

- The bot sends transcoded speech from both sides.
- The React phone can ask Open AI completion requests from the bot.
- The bot is configured to work with Azure Open AI API and has the authority token to use it. (The React phone doesn’t have such authority because it is running on the customer’s browser. Such tokens can be read using browser development tools.)
- The Agent Assist bot doesn’t close the WebSocket connection immediately after call termination, to allow the phone to send Open AI requests with the call summary.

3 VoiceAI Connect Web Portal Bot Configuration

VAIC (VoiceAI Connect) acts as a voice portal which receives the audio streams of the caller and callee, and then transfers them to an speech-to-text engine.

The returned text from the speech-to-text engine is transferred by the VoiceAI Connect to the bot, which provides it via WebSocket to the WebRTC agent.

VoiceAI Connect can be configured in either of two ways:

- Accessing the VoiceAI Connect Web UI and making the necessary configuration
- Editing the VoiceAI Connect configuration file which may also be accessed via the VAIC Web UI

To configure VoiceAI Connect Web portal bot:

1. Configure the providers for the Agent Assist bot (refer to the VoiceAI Connect manual for configuring providers). There should be two providers:
   - Speech-to-text provider
   - Bot program provider
2. Configure the Agent Assist bot (refer to the VoiceAI Connect manual for configuring bots).

4 Bot/Phone JSON Protocol

The following describes the bot/phone JSON protocol.

```
Request from phone

 { ”type”: ”completion”,
   ”name”: ”<name of completion>”,
  ”id”: ”some string” }

Here is name one of the completions defined in the bot local
directory for used languages.

e.g.,: “summary”, “topics”, “sentiment”.


 Response to completion request:

{ ”type": "completion-result",
 "name": “<request name>”,
 "id": “<request id>”,
 "success": true/false,
 "text": ”<open ai response>” }

When the bot receives the recognition POST from server, it saves
it to the session history and sends a message to the phone:

 { "type": "message",
"name": "recognition",
"user": <sip user name>,"language":   “<language>”,
 "text": “<recognized text>” }
```

5 Language Configuration

The following describes the language configuration. The bot and configuration are located in the cloud's virtual machine.

Currently, the he-IL.json file is located in the *locals* directory.

OpenAI prompts are configurable in the file and can be modified/improved using prompt engineering.

The current prompt values are provided as a working example.

```
Language settings are:

          {
    "params": {
        "localRole" : "סוכן",
        "remoteRole" : "לקוח"
    },
    "summary": {
        "prompt_prefix": "Please generate a summary of the
conversation in the following text and include all numbers, names,
identifiers, dates and locations mentioned there. Make this
summary in the language of the original text.\nTEXT:\n",
        "body": {
          "max_tokens": 500
        }
      },
      "topics": {
          "prompt_prefix": "Please generate a numbered list of the
topics discussed in the text below. Make this list in the language
of the original text, when each entry in the list is maximum 5
words long.\nTEXT:\n",
          "body": {
            "max_tokens": 300
          }
      },
      "sentiment": {
          "prompt_prefix": "Given the following text, detect the
language and analyze the sentiment of both the customer and agent.
Summarize how they felt throughout the conversation in the same
language as the conversation-text. Please provide a brief sentence
for each of them in the detected language, and have the entire
result as a bulleted list, with one list-item for the agent, and
one item for the customer. No need to specify the language
detected.\nTEXT:\n",
          "body": {
            "max_tokens": 300,
            "temperature": 0.5
          }
      }
}
```

This defines the local/remote roles (i.e., agent/customer) and three OpenAI completions:

- Summary
- Topics
- Sentiment

OpenAI is called with the following:

```
prompt = prompt_prefix + dialog + prompt_suffix
```

In the dialogs, SIP user names are replaced with ‘Roles’.

6 Docker Installation

The bot is provided as a Docker container. This container can run in Docker as well as in Kubernetes. The procedure below describes the Docker installation. Refer to the *docker* subdirectory to see bash scripts and the Dockerfile.

To run the bot container in Docker you must have the following:

- Cloud virtual machine with a public IP address. It can be used with Amd64 or Arm64 architecture.
  - Use a cloud provider of your choice (Azure, Amazon, Google, etc.).
- FQDN assigned to the virtual machine public IP.
- TLS certificate for the FQDN purchased from a known certificate provider.
- Installed operating system:
  - Ubuntu LTS Linux
  - Any Linux can be used. Windows is possible, but not recommended.
- Installed Docker CE (CLI tools and daemon)

6.1 Start Bot

If you have access to the private Azure container registry, pull a container image and start the bot:

```
  bash docker/start
Here is the script.
#! /usr/bin/bash -v
docker stop assistbot
docker rm assistbot

docker run --name assistbot \
    -d \
    --health-cmd="node healthcheck.js || bash -c 'kill -s SIGINT -
1'" \
    --health-start-period=15s \
    --health-interval=15s \
    --health-timeout=10s \
    --restart=unless-stopped \
    --log-driver=journald \
    --env-file .env \
    -v /etc/tokens/openai:/etc/tokens/openai:ro \
    -v /etc/tokens/vaic:/etc/tokens/vaic:ro \
    -v /etc/ssl/private:/etc/ssl/private:ro \
     -p 7777:7777  \
     audcreg.azurecr.io/agent_assist_bot:latest
```

6.2 Build Bot

If you have access to the bot sources, you can modify and rebuild the bot container:

If you don't have access to the **audcreg.azurecr.io** container registry, you can replace it with your registry or use the docker local registry.

- build docker image:

```
bash docker/build
```

- (Optional) Build a multi-architecture docker image (e.g., for Arm64).

```
bash docker/prepare_buildx
bash docker/buildx
```

- (Optional) Push the docker image to the private docker registry.

```
bash docker/push
```

- check logs

```
bash docker/logs
```

6.3 Dockerfile

The following describes the Dockerfile.

```
FROM node:19.8.1-bullseye-slim
RUN  apt-get update &&  apt-get install --assume-yes --no-install-recommends dumb-init && \
apt-get autoremove --assume-yes &&  apt-get clean && rm --recursive --force /var/lib/apt/lists/*
WORKDIR /usr/src/app
COPY . /usr/src/app
ENV NODE_ENV production
RUN npm ci --only=production
CMD ["dumb-init", "node", "main.js"]
```

6.4 Configuration

This section describes the following:

- Environment variables
- Secrets
- Health check

6.4.1 Environment Variables

The environment variables are defined in the *.env* file.

```
# HEALTH_CHECK_FAILURE_TEST=true
# Internal server URL
# Host name for healthcheck, internal port mapped to an external
port.
SERVER=https://lab2.webrtc.audiocodes.com:7777
# External websocket URL
WEBSOCKET=wss://lab2.webrtc.audiocodes.com:7777
```

1. Replace 'lab2.webrtc.audiocodes.com' with your site FQDN.
2. You can modify the used port '7777'.
3. Modify it in the docker *run* command and enable it in the cloud virtual machine firewall.

6.4.2 Secrets

The container mounts directories with secret files:

- File with vaic token string

  ```
  /etc/tokens/vaic/token
  ```

- Azure OpenAI subscription to enable call Azure REST API.

  Used model: DaVinci 3

- File with the JSON format:

  ```
  {
    "token": "????",
    "url": "?????"
  }
  /etc/tokens/openai/conf
  ```

- PEM certificate and private key for your FQDN

  Warning: Cannot be used with a self-signed certificate.

  ```
  /etc/ssl/private/tls.key
  /etc/ssl/private/tls.crt
  ```

Tls.crt file includes the site and intermediate certificates.

6.4.3 Health Check

The bash start script in the container runs with a health check. The *health-cmd* is sent periodically with the HTTP GET request. The Docker daemon restarts the container if the response 200 has not been received.

To simulate health-check failure, uncomment the *.env* file line:
"HEALTH_CHECK_FAILURE_TEST=true"

The first five health tests receive 'Response code 200'. The sixth test receives a 'Response code 404'.

During normal work hours, this line must be commented out.

7 Kubernetes Installation

For Kubernetes developing/testing, you can use k0s running on a single virtual machine. For more information, click here.

The Agent Assist bot container receives POST requests from the server and WSS connections from browser phone. The container does not correspond to the Kubernetes stateless model.

7.1 ReplicaSet and Deployments

If you use many replicas of the container, one replica receives the POST and creates a session. There is no guarantee that the same replicas receive WebSocket connections corresponding to the session.

Sticky sessions don’t help in this case because the browser phone IP is different from the server IP.

The project options include:

- **Single container instances:** Single Kubernetes service and single pod
- **Multiple container instances:** Kubernetes ingress, many services and pods

7.2 Preparations

The following preparations are necessary:

- Create secrets
- Build docker image and save it to k0s local registry

7.2.1 Create Secrets

Use the following to create secrets:

```
bash kubernetes/set_secrets
```

Use the following script:

```
sudo k0s kubectl create secret tls site-cert --
cert=/etc/ssl/private/tls.crt  --key=/etc/ssl/private/tls.key
sudo k0s kubectl create secret generic vaic-token --from-
file=/etc/tokens/vaic/token
sudo k0s kubectl create secret generic openai-conf --from-
file=/etc/tokens/openai/conf
```

7.2.2 Build Docker Image and Save it to k0s Local Registry

Use the following to build the Docker image.

```
bash kubernetes/build
```

Use the following script:

```
docker build -f docker/Dockerfile -t assistbot:local .
docker save assistbot:local > tmp.tar
sudo k0s ctr image import tmp.tar
rm tmp.tar
```

7.3 Single Container Instance

The following information describes how to use the Single Container instance.

To use the Single Container instance:

1. Review and edit start.yaml.
   - **Pod:** Replace your test virtual machine FQDN *lab2.webrtc.audiocodes.com* with your virtual machine FQDN.
   - **Edit the exposed port 9999 with your value**: (Ensure you correctly configure the virtual machine firewall)
   - **Service:** Edit externalIPs value.
2. Use the following start.yaml file:

```
#
#  Start single instance of Agent Assist Bot.
#
#  NodePort service with externalIPs (expose the bot to internet)
apiVersion: v1
kind: Service
metadata:
  name: abot
spec:
  ports:
  - port: 9999
    targetPort: 443
    protocol: TCP
  selector:
    app: abot
  type: NodePort
  externalIPs:
  - 10.0.0.4

---
# Agent assist bot configured to receive HTTPS
apiVersion: v1
kind: Pod
metadata:
  name: abot
  labels:
    app: abot
spec:
  volumes:
    - name: vol-cert
      secret:
        secretName: site-cert
    - name: vol-openai
      secret:
        secretName: openai-conf
    - name: vol-vaic
      secret:
        secretName: vaic-token
```

```
containers:
- name: assistbot
  image: assistbot:local
  volumeMounts:
    - mountPath: "/etc/ssl/private"
      name: vol-cert
      readOnly: true
    - mountPath: "/etc/tokens/openai"
      name: vol-openai
      readOnly: true
    - mountPath: "/etc/tokens/vaic"
      name: vol-vaic
      readOnly: true

  env:
    - name: SERVER
      value: "https://lab2.webrtc.audiocodes.com:443"
    - name: WEBSOCKET
      value: "wss://lab2.webrtc.audiocodes.com:9999"
  ports:
  - containerPort: 443

  livenessProbe:
    httpGet:
      scheme: 'HTTPS'
      port: 443
      httpHeaders:
        - name: 'liveness-probe'
          value: 'yes'
    failureThreshold: 2
    initialDelaySeconds: 10
    periodSeconds: 10
    timeoutSeconds: 10
```

3. Start the project:

```
kubectl apply -f kubernetes/start.yaml
```

The deployment includes pod and service. The pod contains the Agent Assist bot container configured to receive HTTPS. The NodePort service contains with an external IP, exposes the bot to the Internet.

Multiple container instances are currently not supported.

International Headquarters
1 Hayarden Street,
Airport City
Lod 7019900, Israel
Tel: +972-3-976-4000
Fax: +972-3-976-4040

AudioCodes Inc.
80 Kingsbridge Rd
Piscataway, NJ 08854, USA
Tel: +1-732-469-0880
Fax: +1-732-469-2298

Contact us: https://www.audiocodes.com/corporate/offices-worldwide
Website: https://www.audiocodes.com

Document #: LTRT-14220