Overview

Agents quickstart

Building Agents

- Anatomy of an Agent
- Inside a session
- Working with tracks
- The voice assistant

Working with plugins

Agents Playground

Deployment and scaling

Conversational AI Quickstart

Build an AI-powered voice assistant that engages in realtime conversations using LiveKit, Python, and NextJS.

This quickstart tutorial walks you through the steps to build a conversational AI application using Python and NextJS. It uses LiveKit's Agents SDK and React Components Library to create an AI-powered voice assistant that can engage in realtime conversations with users. By the end, you will have a basic conversational AI application that you can run and interact with.

On this page

Prerequisites

Steps

1. Set up your environment
2. Create the server agent
3. Run the server agent
4. Set up the frontend env...
5. Create an access token...
6. Create the UI
7. Run the frontend applic...
8. Talk with the AI-power...

Overview

Agents quickstart

Anatomy of an Agent

Inside a session

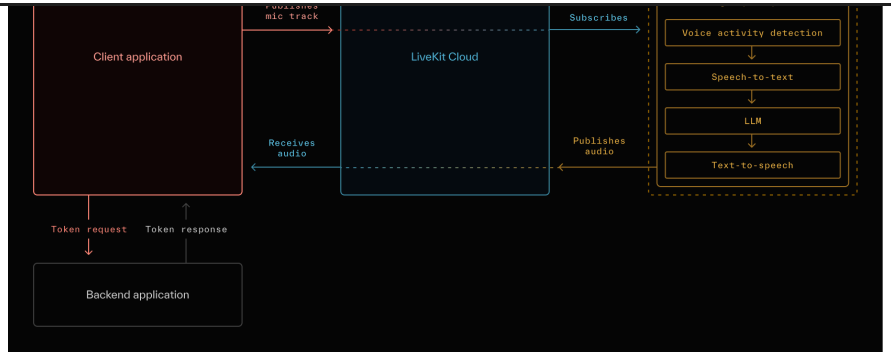
Working with tracks

The voice assistant

Working with plugins

Agents Playground

Deployment and scaling



Prerequisites

- LiveKit Cloud Project ↗ or open-source LiveKit server
- Deepgram API Key ↗
- OpenAI API Key ↗
- Python 3.10+ ↗

Steps

1. Set up your environment

Overview

Agents quickstart

Anatomy of an Agent

Inside a session

Working with tracks

The voice assistant

Working with plugins

Agents Playground

Deployment and scaling

```
export DEEPGRAM_API_KEY=<your Deepgram API key
export OPENAI_API_KEY=<your OpenAI API key>
```

Set up a Python virtual environment:

```
python -m venv venv
source venv/bin/activate
```

Install the necessary Python packages:

```
pip install \
livekit \
livekit-agents \
livekit-plugins-deepgram \
livekit-plugins-openai \
livekit-plugins-silero
```

2. Create the server agent

Create a file named `main.py` and add the following code:

```
import asyncio
```

Overview

Agents quickstart

Anatomy of an Agent

Inside a session

Working with tracks

The voice assistant

Working with plugins

Agents Playground

Deployment and scaling

```python
async def entrypoint(ctx: JobContext):
    # Create an initial chat context with a sy
    initial_ctx = llm.ChatContext().append(
        role="system",
        text=(
            "You are a voice assistant created
            "You should use short and concise
        ),
    )

    # Connect to the LiveKit room
    # indicating that the agent will only subs
    await ctx.connect(auto_subscribe=AutoSubsc

    # VoiceAssistant is a class that creates a
    # See https://github.com/livekit/agents/tr
    # for details on how it works.
    assistant = VoiceAssistant(
        vad=silero.VAD.load(),
        stt=deepgram.STT(),
        llm=openai.LLM(),
        tts=openai.TTS(),
        chat_ctx=initial_ctx,
    )

    # Start the voice assistant with the LiveK
    assistant.start(ctx.room)

    await asyncio.sleep(1)
```

Overview

Agents quickstart

Anatomy of an Agent

Inside a session

Working with tracks

The voice assistant

Working with plugins

Agents Playground

Deployment and scaling

```
    cli.run_app(WorkerOptions(entrypoint_fnc=e
```

3. Run the server agent

Run the agent worker:

```
python main.py dev
```

After running the command above, the worker will begin listening for job requests from a LiveKit server. You can run multiple workers to scale your Agent, and LiveKit's servers will load balance the requests between them.

4. Set up the frontend environment

Tip:

In the following steps we'll cover how to build a simple frontend application to communicate with your agent. If you want to test your agent without creating a frontend application, you can use Agents Playground ↗, a versatile frontend tool for testing agents.

Overview

Agents quickstart

Anatomy of an Agent

Inside a session

Working with tracks

The voice assistant

Working with plugins

Agents Playground

Deployment and scaling

By default, an agent job request is created when a `Room` is created. We'll create a NextJS application for the human participant to join a new Room talk to the AI agent.

Start by scaffolding a NextJS project:

```
npx create-next-app@latest
```

Install LiveKit dependencies:

```
npm install @livekit/components-react @livek
```

Set the following environment variables in your .env.local file:

```
export LIVEKIT_URL=<your LiveKit server URL>
export LIVEKIT_API_KEY=<your API Key>
export LIVEKIT_API_SECRET=<your API Secret>
```

5. Create an access token endpoint

Overview

Agents quickstart

Anatomy of an Agent

Inside a session

Working with tracks

The voice assistant

Working with plugins

Agents Playground

Deployment and scaling

```
export async function GET(request: Request) {
  const roomName = Math.random().toString(36).
  const apiKey = process.env.LIVEKIT_API_KEY;
  const apiSecret = process.env.LIVEKIT_API_SE
  const at = new AccessToken(apiKey, apiSecret
  at.addGrant({
    room: roomName,
    roomJoin: true,
    canPublish: true,
    canPublishData: true,
    canSubscribe: true,
  });
  return Response.json({ accessToken: await at
}
```

This sets up an API endpoint to generate LiveKit access tokens.

6. Create the UI

Create a file `src/app/page.tsx` and add the following code:

```
'use client';
import {
  LiveKitRoom,
  RoomAudioRenderer,
```

Overview

Agents quickstart

Anatomy of an Agent

Inside a session

Working with tracks

The voice assistant

Working with plugins

Agents Playground

Deployment and scaling

```
  const [token, setToken] = useState<string |
  const [url, setUrl] = useState<string | null

  return (
    <>
      <main>
        {token === null ? (<button onClick={as
          const {accessToken, url} = await fet
          setToken(accessToken);
          setUrl(url);
        }}>Connect</button>) : (
          <LiveKitRoom
            token={token}
            serverUrl={url}
            connectOptions={{autoSubscribe: tr
          >
            <ActiveRoom />
          </LiveKitRoom>
        )}
      </main>
    </>
  );
};

const ActiveRoom = () => {
  const { localParticipant, isMicrophoneEnable
  return (
    <>
      <RoomAudioRenderer />
      <button onClick={() => {
```

Overview

Agents quickstart

Anatomy of an Agent

Inside a session

Working with tracks

The voice assistant

Working with plugins

Agents Playground

Deployment and scaling

This sets up the main page component that renders the VideoConference UI.

7. Run the frontend application

In your terminal, run:

```
npm run dev
```

This starts the app on localhost:3000.

8. Talk with the AI-powered agent

Once you have the server agent and frontend application up and running, you can start talking to the agent, asking questions, or discussing any topic you'd like. When you join a room through the web UI, an agent job request is automatically sent to your worker. The worker accepts the job, and an AI-powered agent joins the room, ready to engage in conversation. The agent will listen for your voice, process your speech using Deepgram's speech-to-text (STT) technology, generate

Overview

Agents quickstart

Anatomy of an Agent

Inside a session

Working with tracks

The voice assistant

Working with plugins

Agents Playground

Deployment and scaling

PREVIOUS

Overview

UP NEXT

Building Agents: Anatomy of an Agent