# Fontes de consulta para criação de Azure Bot

## ChatGPT

Beleza 🚀 Vou montar para você um **guia completo** com:

1. Um **roadmap visual (fluxo em estilo de diagrama/roadmap)** para entender a criação e integração do **Azure Bot Service**.
2. O **passo a passo detalhado** para configurar o bot no Azure e integrá-lo com backend Python.
3. Um **exemplo de código em Python (Google Colab)** mostrando como se conectar ao bot e enviar mensagens.

**📍 Roadmap: Azure Bot + Backend Python (Google Colab)**

flowchart TD

A[Criação do Recurso no Azure Portal] --> B[Configuração do Azure Bot Service]

B --> C[Registro do Bot no Azure AD]

C --> D[Configuração de Channels (ex: Direct Line)]

D --> E[Obtenção da Secret Key do Direct Line]

E --> F[Configuração do Backend em Python]

F --> G[Conexão via API do Direct Line]

G --> H[Teste no Google Colab enviando mensagens]

H --> I[Iteração com o Azure Bot Service]

**🔑 Passo a Passo Completo**

**1. Criar o recurso do Azure Bot**

* Acesse o [Azure Portal](https://portal.azure.com/).
* Clique em **Criar um recurso** → **AI + Machine Learning** → **Azure Bot**.
* Preencha os campos:
  + Nome do Bot
  + Grupo de Recursos
  + Região
  + Tipo de preço (pode começar com **F0 – Free**).

**2. Configurar o Bot Service**

* Após criar, vá até o recurso criado.
* Configure o **Bot Channels Registration**.
* Escolha o **canal Direct Line** (fundamental para comunicação com backend).

**3. Obter as credenciais**

* Dentro do canal **Direct Line**, copie a **Secret Key**.
* Essa chave será usada para autenticação nas requisições.

**4. Configurar o Backend (Python)**

* O backend Python se comunica com o bot via **REST API**.
* Endpoints principais:
  + POST https://directline.botframework.com/v3/directline/conversations → cria uma conversa
  + POST https://directline.botframework.com/v3/directline/conversations/{conversationId}/activities → envia mensagem
  + GET https://directline.botframework.com/v3/directline/conversations/{conversationId}/activities → recebe mensagens

**🐍 Código Python – Testando no Google Colab**

Este exemplo permite enviar mensagens ao Azure Bot e receber as respostas.

import requests

import time

# 🔑 Coloque aqui sua Direct Line Secret do Azure Bot

DIRECT\_LINE\_SECRET = "SUA\_DIRECT\_LINE\_SECRET\_AQUI"

DIRECT\_LINE\_URL = "https://directline.botframework.com/v3/directline"

# Headers de autenticação

headers = {

"Authorization": f"Bearer {DIRECT\_LINE\_SECRET}"

}

# 1. Criar conversa

response = requests.post(f"{DIRECT\_LINE\_URL}/conversations", headers=headers)

response\_json = response.json()

conversation\_id = response\_json["conversationId"]

print("Conversa criada:", conversation\_id)

# Função para enviar mensagem

def send\_message(message):

url = f"{DIRECT\_LINE\_URL}/conversations/{conversation\_id}/activities"

data = {

"type": "message",

"from": {"id": "user1"},

"text": message

}

r = requests.post(url, headers=headers, json=data)

return r.json()

# Função para receber mensagens

def get\_messages():

url = f"{DIRECT\_LINE\_URL}/conversations/{conversation\_id}/activities"

r = requests.get(url, headers=headers)

messages = r.json()["activities"]

return messages

# Exemplo de envio

send\_message("Olá Bot, tudo bem?")

# Aguardar um pouco e buscar respostas

time.sleep(2)

messages = get\_messages()

# Mostrar última resposta do bot

for m in messages:

if m["from"]["id"] != "user1": # filtrando mensagens do bot

print("🤖 Bot respondeu:", m["text"])

**🎯 Resultado Esperado no Colab**

1. A saída inicial será algo como:
2. Conversa criada: 7mW1abcd12345
3. Depois de enviar "Olá Bot, tudo bem?", o bot responderá com a mensagem configurada no Azure Bot (exemplo: "Olá, seja bem-vindo!").

👉 **material visual (roadmap estilizado)** em formato de **imagem/infográfico** para você ter pronto como guia de apresentação.

Tela de celular com aplicativo aberto

O conteúdo gerado por IA pode estar incorreto.

## Google Gemini

Configurar uma resposta simples no seu **Azure Bot** pode ser feito de diversas maneiras, dependendo da ferramenta que você está utilizando. A forma mais comum, e que funciona para um cenário de resposta fixa como o que você descreveu, é usando o **Composer** ou um código direto no Azure.

**Cenário 1: Usando Azure Bot Composer**

Se você criou seu bot usando o **Azure Bot Composer**, a interface visual facilita bastante. A ideia é criar um "gatilho" (trigger) que escuta por uma frase específica e, em seguida, define uma ação de resposta.

1. No Composer, navegue até a seção **"WelcomeGreetings"** ou crie um novo **"Dialog"**.
2. Crie um novo **"Trigger"** do tipo **"Intent recognized"**.
3. No campo de **"Trigger phrase"**, adicione as frases que o bot deve reconhecer.
   * Exemplo: Olá Bot, tudo bem?
   * Exemplo: Oi
   * Exemplo: Tudo bem?
4. Dentro desse **"Trigger"**, adicione a ação de **"Send a response"**.
5. No campo de texto, digite a resposta que o bot deve dar.
   * Exemplo: Olá, seja bem-vindo!
   * Exemplo: Olá! Como posso ajudar você hoje?

Com essa configuração, toda vez que o usuário digitar uma das frases definidas, o bot enviará a resposta que você configurou.

**Cenário 2: Usando um Código Personalizado no Azure**

Se você configurou seu bot com um código personalizado (como um projeto em **Python** ou **C#**), a lógica de resposta deve ser implementada no código.

A lógica é basicamente a seguinte:

1. O bot recebe uma mensagem do usuário.
2. O código verifica o conteúdo dessa mensagem.
3. Se a mensagem for Olá Bot, tudo bem?, o bot envia a resposta "Olá, seja bem-vindo!".
4. Se a mensagem for diferente, o bot pode dar uma resposta padrão ou encaminhar para outra função.

Um exemplo simplificado em **Python**, usando o SDK do Azure Bot Framework:

Python

from botbuilder.core import ActivityHandler, TurnContext

class MyBot(ActivityHandler):

async def on\_message\_activity(self, turn\_context: TurnContext):

# Obtém o texto da mensagem do usuário

user\_message = turn\_context.activity.text

# Verifica se a mensagem é a frase que queremos

if user\_message.lower() == "olá bot, tudo bem?":

# Envia a resposta configurada

await turn\_context.send\_activity("Olá, seja bem-vindo!")

else:

# Resposta padrão para outras mensagens

await turn\_context.send\_activity("Desculpe, não entendi. Por favor, tente novamente.")

Essa lógica deve ser implantada no seu serviço de aplicativo (App Service) do Azure para funcionar corretamente.

Sim, é possível e é a forma correta de fazer. A configuração do bot, incluindo sua lógica de conversação, é feita dentro do recurso do **Azure Bot** que você já criou. O nome "dioBot" e a região "Global" são os identificadores do seu recurso no Azure.

**Como configurar a conversa**

O método que você usará para configurar a conversa depende de como você criou o bot inicialmente. As opções mais comuns são:

1. **Bot Framework Composer**: Se você usou o Composer, você precisa abrir o aplicativo desktop, carregar o projeto do seu bot, fazer as alterações na lógica de conversação (adicionar um novo "dialog" ou "trigger" para a sua frase), e então **publicar** as alterações no recurso do Azure.
2. **Código personalizado (Python, Node.js, etc.)**: Se você escreveu o código para o seu bot, a lógica de conversação reside nesse código. Você precisa modificar o arquivo de código-fonte para incluir a nova regra de resposta e depois **implantar** esse código atualizado no seu **App Service** do Azure. O App Service é o serviço de hospedagem onde o código do seu bot está rodando.

**Passos para configurar a resposta**

Para a sua necessidade específica de responder à frase "Olá Bot, tudo bem?", você precisaria:

* **Identificar a lógica de resposta**: No seu código, procure a parte que lida com as mensagens recebidas do usuário. Geralmente, é um método chamado on\_message\_activity ou algo similar.
* **Adicionar a nova regra**: Dentro desse método, adicione uma condição para verificar se a mensagem é "Olá Bot, tudo bem?".
* **Definir a resposta**: Se a condição for verdadeira, o bot deve enviar a resposta desejada, como "Olá, seja bem-vindo!".

Por exemplo, no código Python que vimos anteriormente, a nova lógica ficaria assim:

Python

# Seu código que lida com a lógica do bot

async def on\_message\_activity(self, turn\_context: TurnContext):

# Obtém o texto da mensagem do usuário

user\_message = turn\_context.activity.text.lower() # Converte para minúsculas

if "olá bot, tudo bem?" in user\_message:

# Envia a resposta configurada

await turn\_context.send\_activity("Olá, seja bem-vindo!")

else:

# Resposta padrão para outras mensagens

await turn\_context.send\_activity("Desculpe, não entendi. Por favor, tente novamente.")

Depois de modificar o código, você deve **publicar** as alterações para o seu **App Service** no Azure. Se o seu bot está conectado ao Google Colab para testes, você só precisaria rodar o novo código no Colab novamente.

Se você não tiver certeza de qual método de criação foi usado, a forma mais fácil de descobrir é verificar a página de "Visão Geral" do seu recurso **dioBot** no portal do Azure e procurar por links para o "Bot Framework Composer" ou para o "App Service" associado.

## Microsoft e Azure

### Provisionar e publicar um bot

https://learn.microsoft.com/pt-br/azure/bot-service/provision-and-publish-a-bot?view=azure-bot-service-4.0&tabs=userassigned%2Cjava

### CLI instalado

C:\Users\si\_ap>az version

{

"azure-cli": "2.67.0",

"azure-cli-core": "2.67.0",

"azure-cli-telemetry": "1.1.0",

"extensions": {}

}

**Entrar com o WAM (Gerenciador de Contas da Web) no Windows**

https://learn.microsoft.com/pt-br/cli/azure/authenticate-azure-cli-interactively?view=azure-cli-latest#code-try-0

**Criar um recurso de identidade**

az identity create --resource-group "AI-900" --name "botGemini"

**Configurar recursos com modelos do ARM**

Criar o Serviço de Aplicativo e os recursos de bot do Azure para seu bot. Ambas as etapas usam um modelo do ARM e o comando da CLI do Azure az deployment group create para criar um ou mais recursos.

1. Criar um recurso do Serviço de Aplicativo para seu bot. O Serviço de Aplicativo pode estar dentro de um Plano do Serviço de Aplicativo novo ou existente.

Para obter etapas detalhadas, confira [Usar a CLI do Azure para criar um Serviço de Aplicativo](https://learn.microsoft.com/pt-br/azure/bot-service/provision-app-service?view=azure-bot-service-4.0).

az deployment group create --resource-group <resource-group> --template-file <template-file-path> --parameters "@<parameters-file-path>"

1. Criar seu recurso de bot Kudu.

Para obter etapas detalhadas, confira [Usar a CLI do Azure para criar ou atualizar um Bot do Azure](https://learn.microsoft.com/pt-br/azure/bot-service/provision-azure-bot?view=azure-bot-service-4.0).

az deployment group create --resource-group <resource-group> --template-file <template-file-path> --parameters "@<parameters-file-path>"

**Testar e depurar com o Emulator**

<https://learn.microsoft.com/pt-br/azure/bot-service/bot-service-debug-emulator?view=azure-bot-service-4.0&tabs=python>

**Instalar Emulator**

<https://github.com/Microsoft/BotFramework-Emulator/blob/main/README.md>