Guide: Provision, Deploy (local + remote), Package and other Configurations for Teams App

Objective

To deploy a Teams app on Azure. This guide covers all the necessary steps to achieve it.

Prerequisites

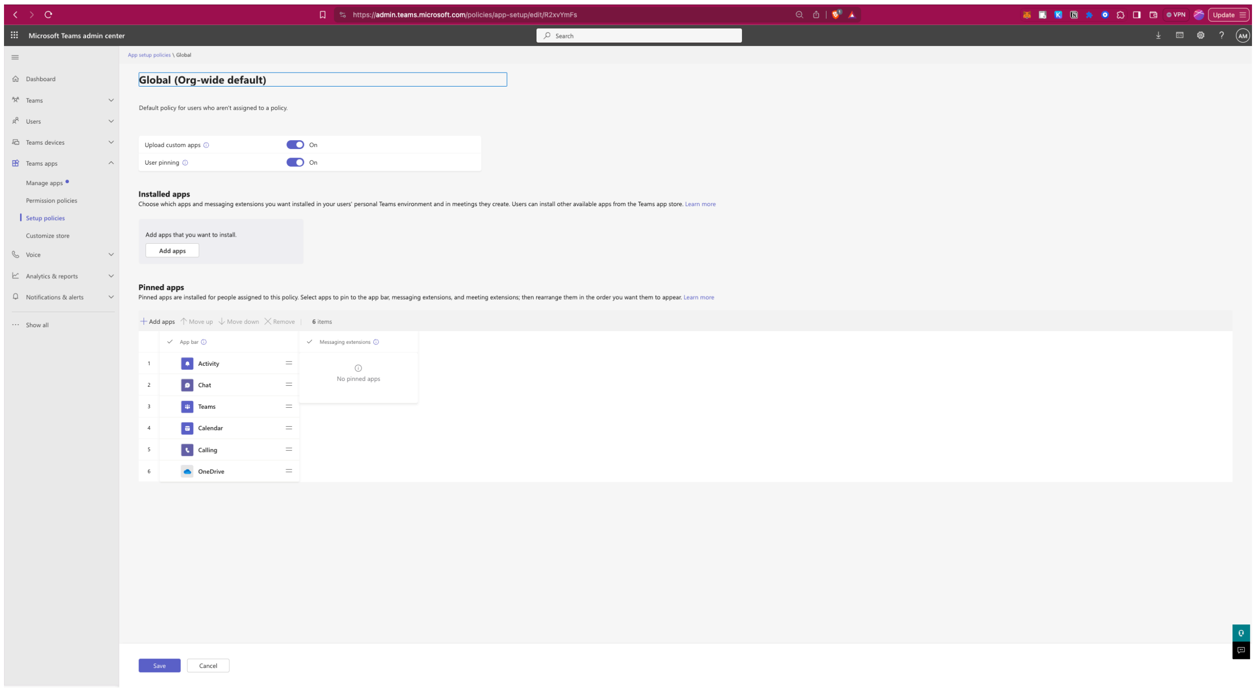
- You should have these locally on your development machine:

```
% node -v
```

```
v18.16.0
```

```
% npm install -g @microsoft/teamsapp-cli
% teamsapp -h
```

- Teamsapp cli reference: https://learn.microsoft.com/en-us/microsoftteams/platform/toolkit/teams-toolkit-cli?pivots=version-three
- Enable upload of customised app: Follow this guide: https://learn.microsoft.com/en-us/microsoftteams/teams-custom-app-policies-and-settings#allow-users-to-upload-custom-apps
- You need to make sure 'upload custom app' is enabled in Global (Org-wide default) as shown in the following screenshot



Global (Org-wide default)

Default policy for users who aren't assigned to a policy.

Upload custom apps	On
User pinning	On

Installed apps

Choose which apps and messaging extensions you want installed in your users' personal Teams environment and in meetings they create. Users can install other available apps from the Teams app store. Learn more

Add apps that you want to install.

Pinned apps

Pinned apps are installed for people assigned to this policy. Select apps to pin to the app bar, messaging extensions, and meeting extensions; then rearrange them in the order you want them to appear. Learn more

6 items

	App bar	Messaging extensions
1	Activity	No pinned apps
2	Chat	
3	Teams	
4	Calendar	
5	Calling	
6	OneDrive	

Azure resources

Here's a brief explainer on the key Azure resources that will be provisioned to deploy a Teams app:

1. Azure Subscription

- **Description**: An Azure subscription is the billing unit and the container for your Azure resources. All resources you create, including those needed for your Teams app, will be associated with a specific subscription. It allows you to manage access, resource creation, and cost tracking for all your Azure services.

- **Purpose**: Provides the overarching structure for managing resources and billing.

2. Azure Resource Group

- **Description**: A resource group is a logical container that holds related Azure resources. For your Teams app, you would create a resource group to organize and manage all the resources (e.g., bots, web apps) associated with the app.
- **Purpose**: Simplifies management and organization by grouping related resources together.

3. Azure Bot Service

- **Description**: Azure Bot Service is a managed service that allows you to create, manage, and deploy bots. For a Teams app, the bot service enables the integration of conversational bots within Teams, allowing users to interact with your app via chat.
- **Purpose**: Powers the bot functionality in your Teams app, handling bot hosting, messaging, and interactions.

4. Azure App Service

- **Description**: Azure App Service is a fully managed platform for building, deploying, and scaling web apps and APIs. For a Teams app, it hosts your web services (e.g., bot code) and provides the backend logic and APIs that power your app's functionality.
- **Purpose**: Hosts the web components of your Teams app.

5. App Service Plan

- **Description**: The App Service Plan defines the underlying infrastructure (e.g., region, instance size, scaling) for your Azure App Service. It determines the compute resources that your app can use.
- **Purpose**: Manages the scale, performance, and cost of hosting your app by configuring the compute resources.

Creating env files for remote deployment

First, we will need to create two env files in env directory: `.env.dev` and `.env.dev.user` . Once created, you should have these content in each of them. PLEASE NOTE THIS IS ONLY REQUIRED FOR THE FIRST TIME, ONCE WE GO THROUGH THE STEPS LISTED BELOW, IT WILL HAVE SENSITIVE INFORMATION LIKE BOT ID AND BOT PASSWORD.

`.env.dev`:

```
# This file includes environment variables that will be committed to git by default.

# Built-in environment variables
TEAMSFX_ENV=dev
APP_NAME_SUFFIX=dev

# Updating AZURE_SUBSCRIPTION_ID or AZURE_RESOURCE_GROUP_NAME after provisi
AZURE_SUBSCRIPTION_ID=
AZURE_RESOURCE_GROUP_NAME=
RESOURCE_SUFFIX=

# Generated during provision, you can also add your own variables.
BOT_ID=
TEAMS_APP_ID=
BOT_AZURE_APP_SERVICE_RESOURCE_ID=
BOT_DOMAIN=
```

`.env.dev.user`:

```
# This file includes environment variables that will not be committed to git by default. You

# Secrets. Keys prefixed with `SECRET_` will be masked in Teams Toolkit logs.
SECRET_BOT_PASSWORD=
```

Provisioning step (on Azure)

We will use terminal(bash) to provision resources on Azure. We will use teamsapp cli. On Azure portal end, this guide assumes you already have an active Azure subscription and a corresponging resource group (RG) created. If not, you can go ahead and create one resource group.

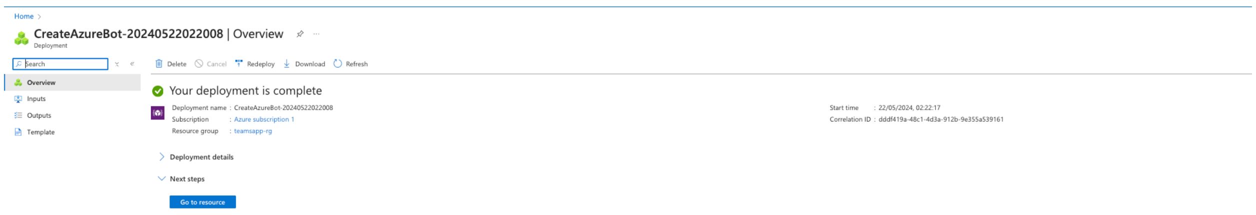
We will first provision the resources on Azure. Run `teamsapp provision`. If this is the first time, it will ask you to login into azure portal; go ahead and sign in. Following is a real example:

```
ankitshubham@Ankits-MacBook-Pro kn_chi_teams_bot_ts % teamsapp provision
? Select an environment: dev
```

```
(x) Error: [Login] Failed to retrieve token silently. If you encounter this problem multiple
times, you can delete `/Users/ankitshubham/.fx/account` and try again. invalid_grant: <
redacted>: The provided grant has expired due to it being revoked, a fresh auth token i
s needed. The user might have changed or reset their password. The grant was issued
on '2023-08-21T12:32:09.4377074Z' and the TokensValidFrom date (before which toke
ns are not valid) for this user is '2023-11-22T23:21:03.0000000Z'. Trace ID: <redacted>
Correlation ID: <redacted> Timestamp: 2024-01-16 21:46:18Z - Correlation ID: <redacte
d> - Trace ID: <redacted>
Log in to your Microsoft 365 account - opening default web browser at https://login.mi
crosoftonline.com/common/oauth2/v2.0/authorize?client_id=<redacted>

? Select a resource group: teamsapp-rg
? Azure account: user2@fake.onmicrosoft.com
Azure subscription: Azure subscription 1
Microsoft 365 account: user2@fake.onmicrosoft.com

Cost may incur according to the usage. Do you want to provision resources in dev envir
onment using accounts listed above?: Yes
(√)Done: Teams Package /Users/ankitshubham/workspace/fiverr/kn_chi/kn_chi_teams_
bot_ts/appPackage/build/appPackage.dev.zip built successfully!
Teams Toolkit has checked against all validation rules:

Summary:
1 warning, 48 passed.
(!) Warning: Short name should not contain “Microsoft” or any of the “Microsoft ...
...
...
 ████████████████████ 100% | [7/7] Provision  (✔) Done.
Successfully executed 7/7 actions in provision stage. View the provisioned resources fr
om https://portal.azure.com/#@ffake/resource/subscriptions/ffake/resourceGroups/tea
msapp-rg
```

Fetch Microsoft App ID and Generate Client Secret (App Password)

The Azure Bot will be shortly created and you will be redirected to this resource.



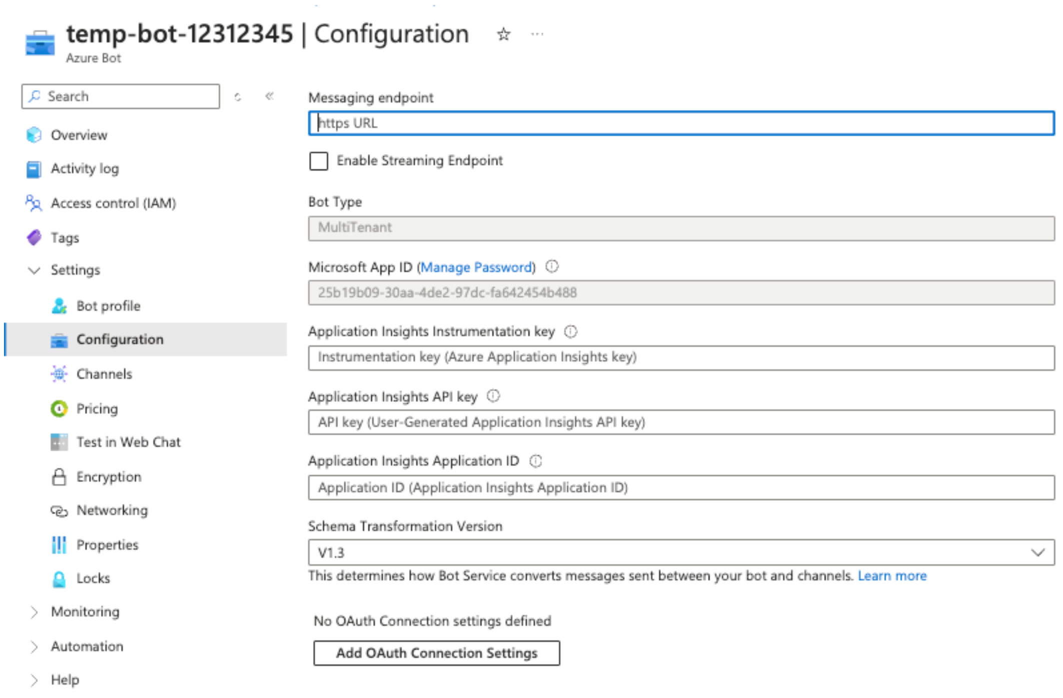
CreateAzureBot-20240522022008 | Overview
Deployment

Your deployment is complete

Deployment name : CreateAzureBot-20240522022008
Subscription : Azure subscription 1
Resource group : teamsapp-rg

Start time : 22/05/2024, 02:22:17
Correlation ID : dddf419a-48c1-4d3a-912b-9e355a539161

Under the Configuration pane, you will find 'Microsoft App Id'. Please take a note of this



temp-bot-12312345 | Configuration
Azure Bot

Messaging endpoint
https URL

Enable Streaming Endpoint

Bot Type
MultiTenant

Microsoft App ID (Manage Password)
25b19b09-30aa-4de2-97dc-fa642454b488

Application Insights Instrumentation key
Instrumentation key (Azure Application Insights key)

Application Insights API key
API key (User-Generated Application Insights API key)

Application Insights Application ID
Application ID (Application Insights Application ID)

Schema Transformation Version
V1.3
This determines how Bot Service converts messages sent between your bot and channels. Learn more

No OAuth Connection settings defined

Next, you need the bot password. We have a few ways to retrieve this:

1. If you have used VS Code, you will see encrypted password in env.dev.user file as SECRET_BOT_PASSWORD. If you have Teams Toolkit installed, you will see a small nudge 'Decrypt Secret'; on clicking it it will decrypt the encrypted secret and you can use it as the BOT_PASSWORD.

2. On Azure portal, go to the App service, then Environment Variables. You should be able to locate the env var named BOT_PASSWORD.

If you can’t do this (or don’t want to reuse the password), please follow the next para.

Under the Configuration pane, you will see a link ‘Manage Password’; click on it.

1. In the left-hand menu, click on **Certificates & secrets**.
2. Under the **Client secrets** section, click on **New client secret**. You might see there is already a secret created. You can ignore this or delete this.
3. **Description:** Enter a description for the client secret (e.g., "BotSecret").
4. **Expires:** Select the expiration period. Choose an appropriate expiration based on your needs, but note that you will need to update the password before it expires.
5. Click on **Add**.
6. **Copy the Value** of the client secret. This is the App Password. You will not be able to retrieve this value again once you leave this page, so make sure to copy it and store it securely.

Graph API

We need additional Graph permissions. To assign this to the same bot, we need to make changes to the associated Graph app.

1. Sign in to the Microsoft Entra admin center as at least a Cloud Application Administrator.
2. Browse to **Identity** > **Applications** > **App registrations**
3. Try to locate the correct app. The bot id of the bot should match the Application (client) ID of the app in the list. Once located, click it.
4. Go to Manage → API Permissions
5. We need the following permission to be added and granted:
   a. Users.Read.All (Application permission)
   b. Users.ReadBasic.All (Application permission)

Deploying step (on Azure)

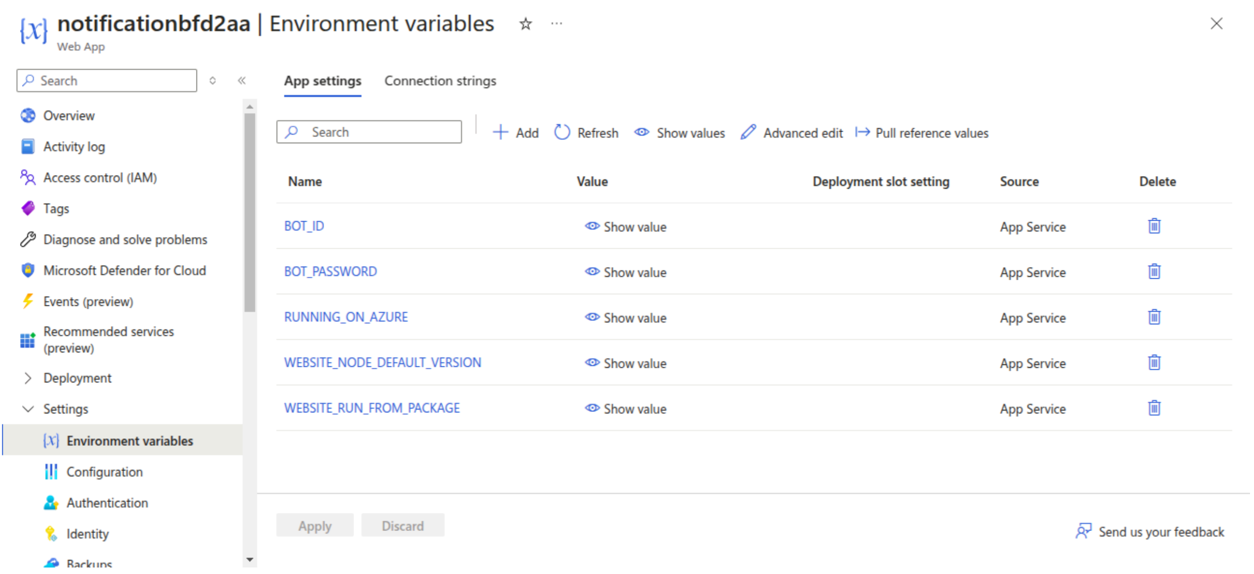
We will now deploy the code to the provisioned resources. Run `teamsapp deploy`. Following is a real example:

```
ankitshubham@Ankits-MacBook-Pro kn_chi_teams_bot_ts % teamsapp deploy
? Select an environment: dev
? Do you want to deploy resources in dev environment?: Yes
Successfully executed 1/1 actions in deploy stage.
████████████████████ 100% | [1/1] Deploy  (✔) Done.
```

Note:

1. You need to deploy whenever there is any change in the code (Express app). You do not need to provision again because provisioning only needs to be done once and initially.
2. When you first deploy, as mentioned earlier, an app service gets created. You would need to set the 1 env vars `INOVE_SIGNATURE_SECRET` in its environment variable and do deploy once more so that the environment values get picked up. You can set these as follows:
    a. Sign in to Azure Portal: Go to Azure Portal and log in.
    b. Go to the App Service where you want to set environment variables.
    c. In the left-hand menu, select "Settings" → "Environment Variables"
    d. Go to the 'App settings' tab and add the variable.

Attaching a screenshot of the screen where these environment variables need to be set:



Name	Value	Deployment slot setting	Source	Delete
BOT_ID	Show value		App Service	
BOT_PASSWORD	Show value		App Service	
RUNNING_ON_AZURE	Show value		App Service	
WEBSITE_NODE_DEFAULT_VERSION	Show value		App Service	
WEBSITE_RUN_FROM_PACKAGE	Show value		App Service	

Packaging step

We will now create the package(installer) which can be used to distribute the Teams bot.

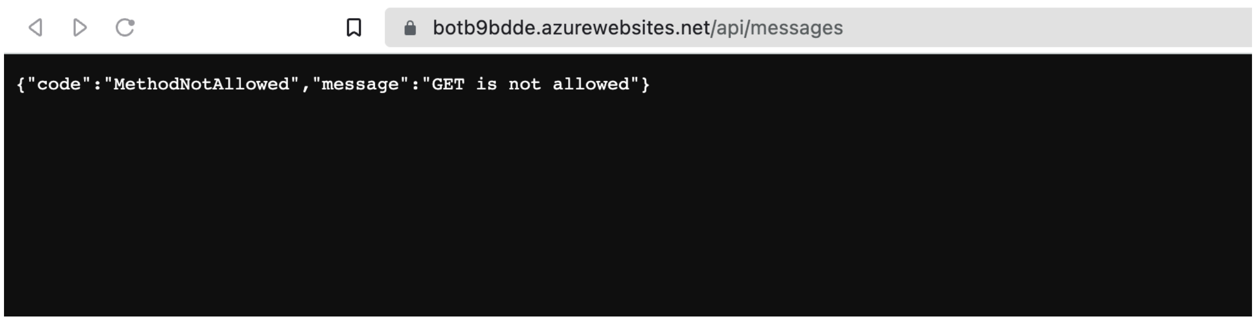
You can make cosmetic changes like renaming the Teams app, changing icons, updating URLs in this step. Locate the file appPackage/manifest.json. It would look like following:

```
{
    "$schema": "https://developer.microsoft.com/en-us/json-schemas/teams/v1.19/Microso
    "manifestVersion": "1.19",
    "version": "1.0.0",
    "id": "${{TEAMS_APP_ID}}",
    "developer": {
        "name": "Teams App, Inc.",
        "websiteUrl": "https://www.example.com",
        "privacyUrl": "https://www.example.com/privacy",
        "termsOfUseUrl": "https://www.example.com/termofuse"
    },
    "icons": {
        "color": "color.png",
        "outline": "outline.png"
    },
    "name": {
        "short": "th-test${{APP_NAME_SUFFIX}}",
        "full": "full name for th-test"
    },
    "description": {
        "short": "short description for th-test",
        "full": "full description for th-test"
    },
    "accentColor": "#FFFFFF",
    "bots": [
        {
            "botId": "${{BOT_ID}}",
            "scopes": [
                "personal",
                "team",
                "groupChat"
            ],
            "supportsFiles": false,
            "isNotificationOnly": false,
            "commandLists": [
          {
            "scopes": ["personal", "team", "groupChat"],
            "commands": [
              {
                "title": "Hi",
                "description": "Say hi to the bot."
              }
            ]
          }
        ]
      }
    ],
    "composeExtensions": [],
    "configurableTabs": [],
    "staticTabs": [],
    "permissions": [
      "identity",
      "messageTeamMembers"
    ],
    "validDomains": []
}
```

We will now build the package. Run `teamsapp package`. Following is a real example:

```
ankitshubham@Ankits-MacBook-Pro kn_chi_teams_bot_ts % teamsapp package
? Select Teams manifest.json file: /Users/ankitshubham/workspace/fiverr/kn_chi/kn_chi_teams_bot_ts/appPackage/manifest.json
? Select an environment: dev
(√)Done: Teams Package /Users/ankitshubham/workspace/fiverr/kn_chi/kn_chi_teams_bot_ts/appPackage/build/appPackage.dev.zip built successfully!
```

Go to portal azure and see the endpoint. In my case, it was:
https://botb9bdde.azurewebsites.net/api/messages and we got something like this:



```
{"code":"MethodNotAllowed","message":"GET is not allowed"}
```

Consider yourself successful if you get 405.

Note: You need to create a new package only when there is a change in the manifest.json or the icon files.

Using ngrok to create http tunnel to the locally running backend

Up until now, we have been dealing directly on Azure and provisioned resources, deployed the code and prepared the package.

However, we also need a convenient way to test the changes while developing. The ideal way would be to run the code locally and somehow let the Microsoft Teams know that they need to connect with this locally running code (and not the one deployed on Azure).

To achieve this, we need to make the locally running code 'visible' to the internet (so that Microsoft Teams can also reach out). We can use HTTP tunneling for this. In short, it assigns a publicly reachable URL to a locally running process. Next, we will need to 'register' this new URL by replacing the one Azure created (We saw that after provisioning step, we got https://botb9bdde.azurewebsites.net/ as the Azure provisioned URL).

1. Set up the local environment

Before you start local development, make sure your project is set up to run locally.

1. .localConfigs file:
   a. Get the BOT_ID and BOT_PASSWORD (using the previous sections)
   b. create 1 additional env var `INOVE_SIGNATURE_SECRET` and put the respective value.

Here is a sample .localConfigs file:

```
# A gitignored place holder file for local runtime configurations
BOT_ID=5af48070-e13d-4f8c-ae00-6bb90f6f8f45
```

```
BOT_PASSWORD=UGJ8Q~nM~ZzcjNqvr6fU4TItPnVEINLvPFfVjct3
INOVE_SIGNATURE_SECRET=qwerty1234
```

2. Install ngrok

Ngrok provides a public URL for your local development environment. If you haven't installed it yet, follow these steps:

1. Download ngrok from ngrok's website.
2. Unzip it and add the binary to your system's PATH, or navigate to the folder where you downloaded it.

You can confirm the installation by running:

```
ngrok version
```

3. Start your bot locally

Run your bot on the local machine:

```
npm run dev:teamsfx
```

Make sure your bot service is running and listening on the appropriate local port (e.g., `http://localhost:3978`).

4. Expose your local bot with ngrok

To expose your local bot to the internet, you need to use ngrok. Start ngrok on the port where your bot is running (e.g., 3978):

```
ngrok http 3978
```

This command will generate an HTTPS URL. Use the HTTPS URL, as Microsoft Teams requires a secure endpoint.

You'll see something like:

```
Forwarding                <https://abcd1234.ngrok.io> → <http://localhost:3978>
```

Test the liveliness using `/checks` . So, if the ngrok url is `https://abcd1234.ngrok.io`, making a GET request to `https://abcd1234.ngrok.io/checks` should respond with a 200 status.

5. Update the messaging endpoint

You need to update the messaging endpoint in the bot registration to point to the ngrok URL.

1. Go to the **Azure Portal** and open the resource for your bot.
2. Navigate to the **Settings** blade and locate the "Messaging endpoint."
3. Update the endpoint with the ngrok HTTPS URL. For example:

   ```
   https://abcd1234.ngrok.io/api/messages
   ```

4. Make sure the slug is `/api/messages`
5. Save the changes.

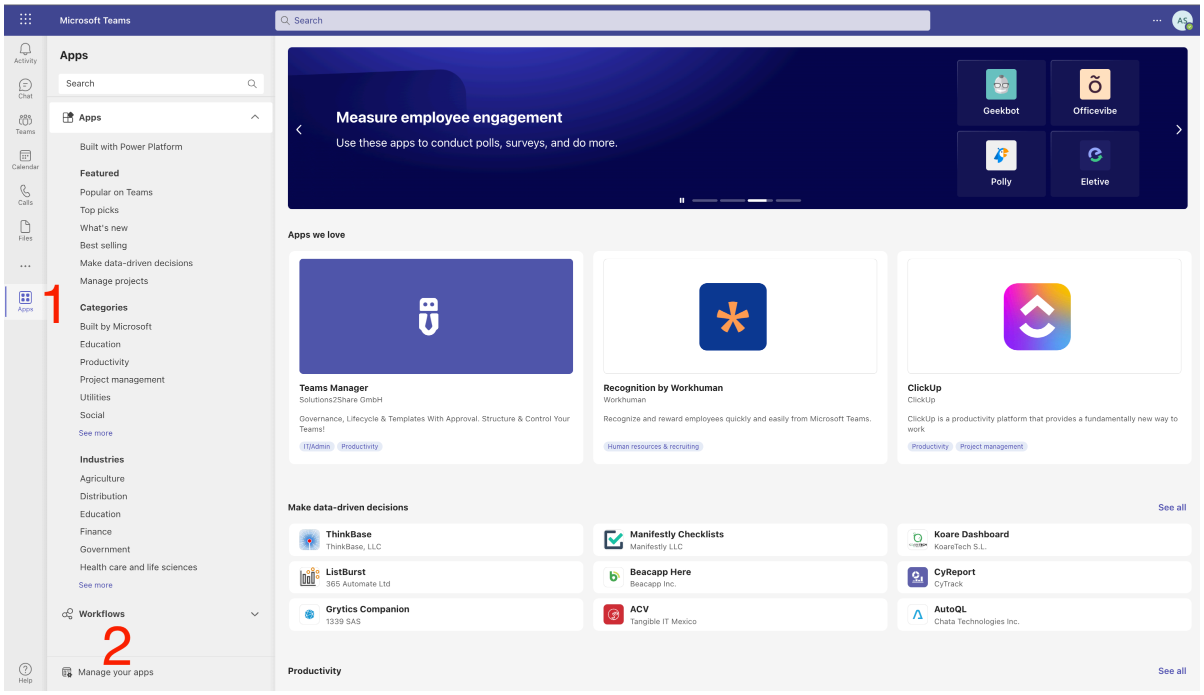
6. Test the bot in Microsoft Teams

After updating the messaging endpoint, we are ready to upload the teams package file (see next section) and test its functionality.

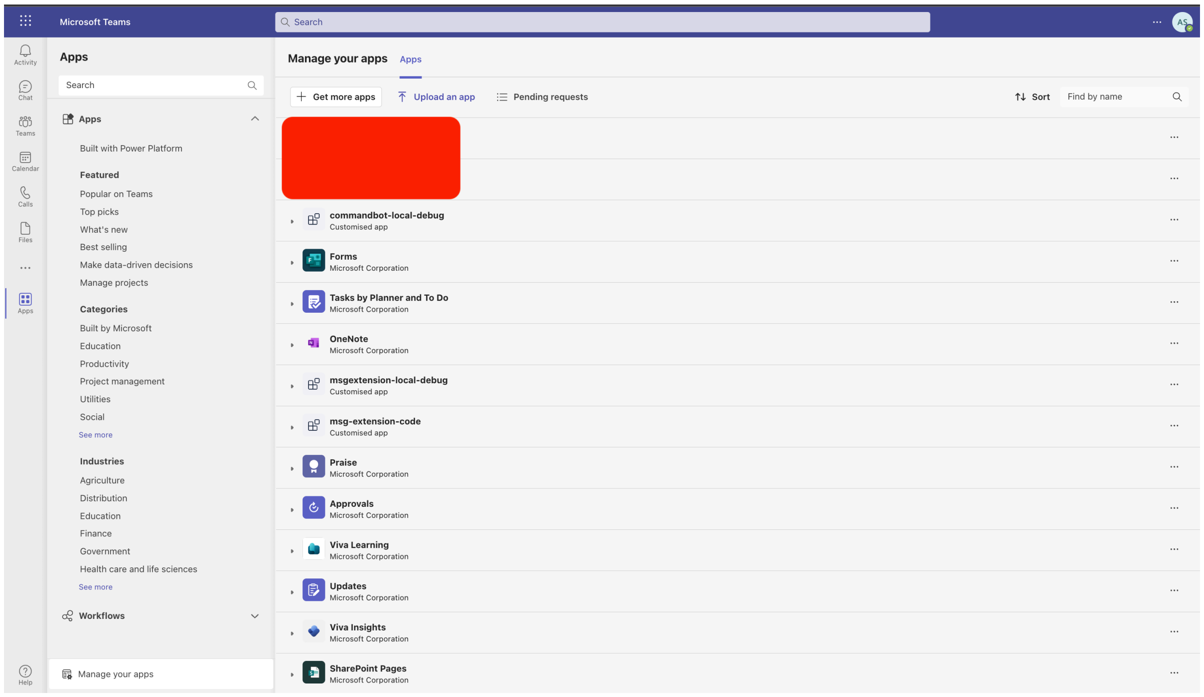
Important note: Every time you stop and restart ngrok, a new URL is generated. You'll need to update the Azure bot registration or manifest file with the new ngrok URL each time. You can also consider other services than ngrok or reserve a dedicated ngrok domain.

Installing the teams chat app on Teams tenant for testing

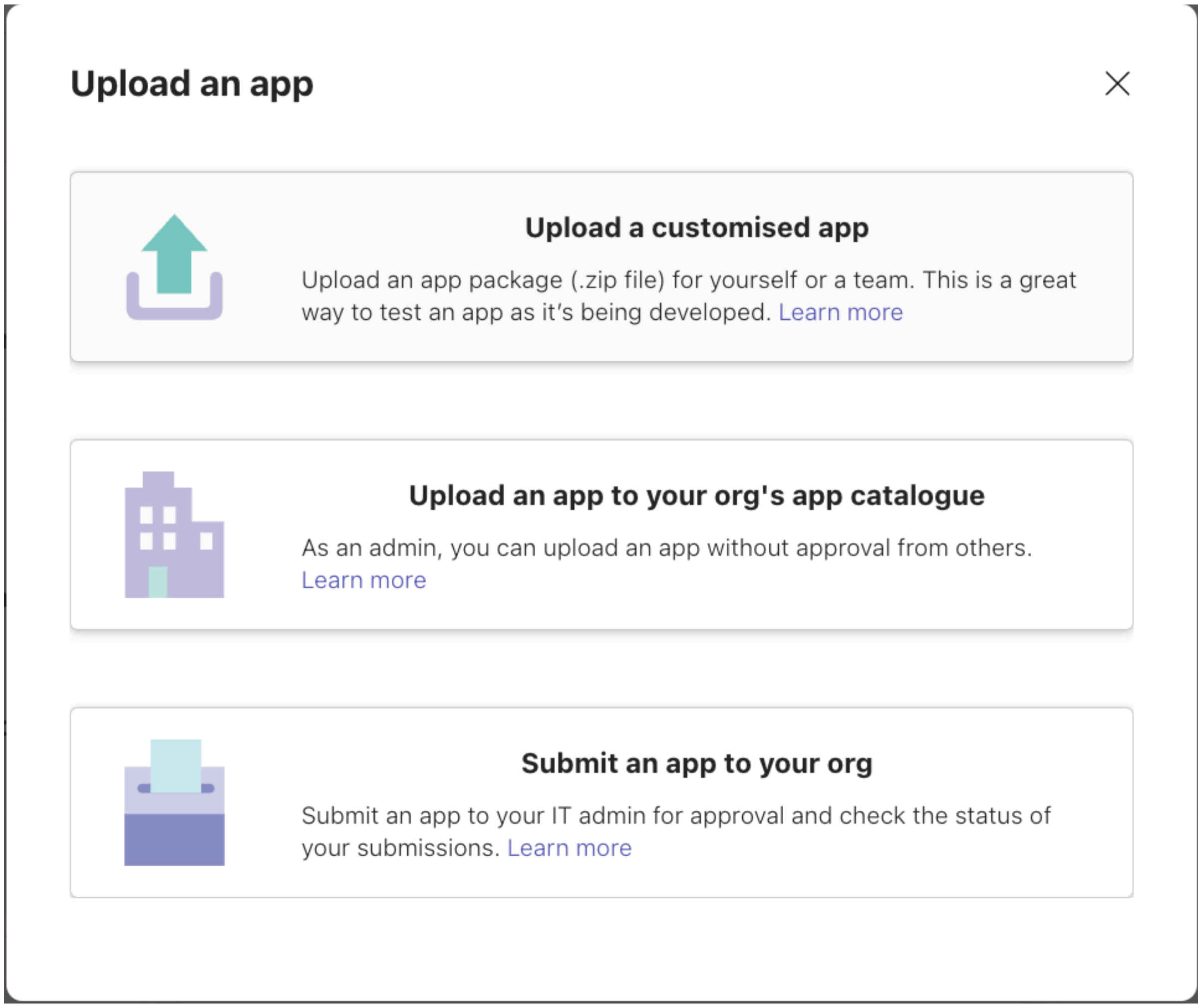
1. At the end of ‘Packaging step’, it will build the deployment zip file at build/appPackage/appPackage.dev.zip. This assumes that your environment was named ‘dev’; if it were ‘foo’, the zip file would be located at build/appPackage/appPackage.foo.zip.
2. Go to Teams and click on ‘Apps’ in the left pane. Then click on ‘Manage your apps’.



3. Click on ‘Upload an app’.



4. A popup opens. Click on ‘Upload a customized app’.



Upload an app

Upload a customised app

Upload an app package (.zip file) for yourself or a team. This is a great way to test an app as it's being developed. Learn more

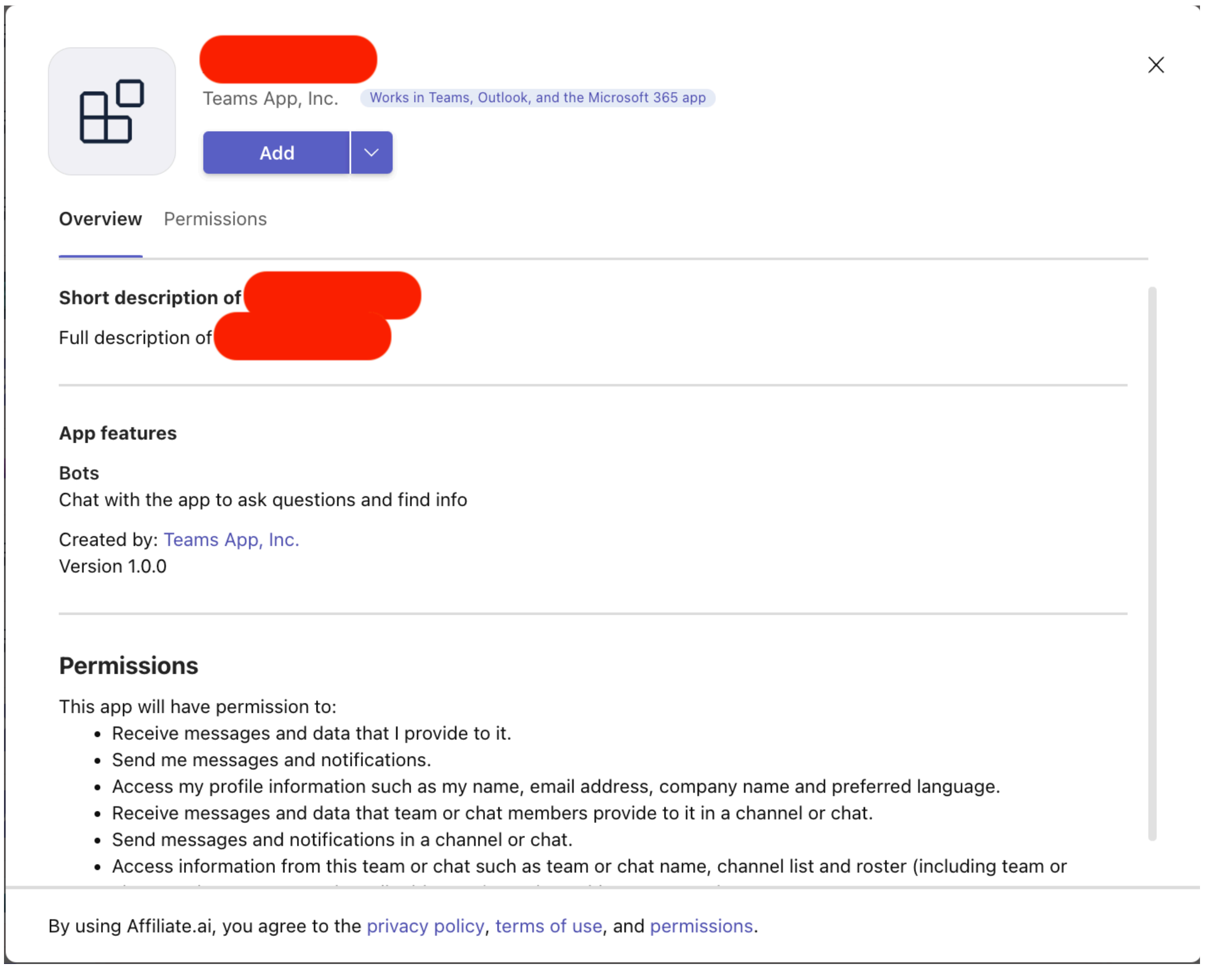
Upload an app to your org's app catalogue

As an admin, you can upload an app without approval from others. Learn more

Submit an app to your org

Submit an app to your IT admin for approval and check the status of your submissions. Learn more

5. Choose the zip file created at step 3.
6. Click on 'Add' to add the app to your Teams!


[redacted]

Teams App, Inc.

Short description of [redacted]

Full description of [redacted]

App features

Bots
Chat with the app to ask questions and find info

Created by: Teams App, Inc.
Version 1.0.0

Permissions

This app will have permission to:

- Receive messages and data that I provide to it.
- Send me messages and notifications.
- Access my profile information such as my name, email address, company name and preferred language.
- Receive messages and data that team or chat members provide to it in a channel or chat.
- Send messages and notifications in a channel or chat.
- Access information from this team or chat such as team or chat name, channel list and roster (including team or

By using Affiliate.ai, you agree to the privacy policy, terms of use, and permissions.

Congratulations! You just deployed the app!

Installing the teams chat app on Teams tenant for making it available for all members

1. In the section above, instead of selecting ‘Upload a customized app’, select ‘Submit an app to your org’.
2. Login as admin to https://admin.teams.microsoft.com/ and follow the guide: https://learn.microsoft.com/en-us/microsoftteams/manage-apps to approve this bot.
3. Once done, it would be available to all the users on Teams.