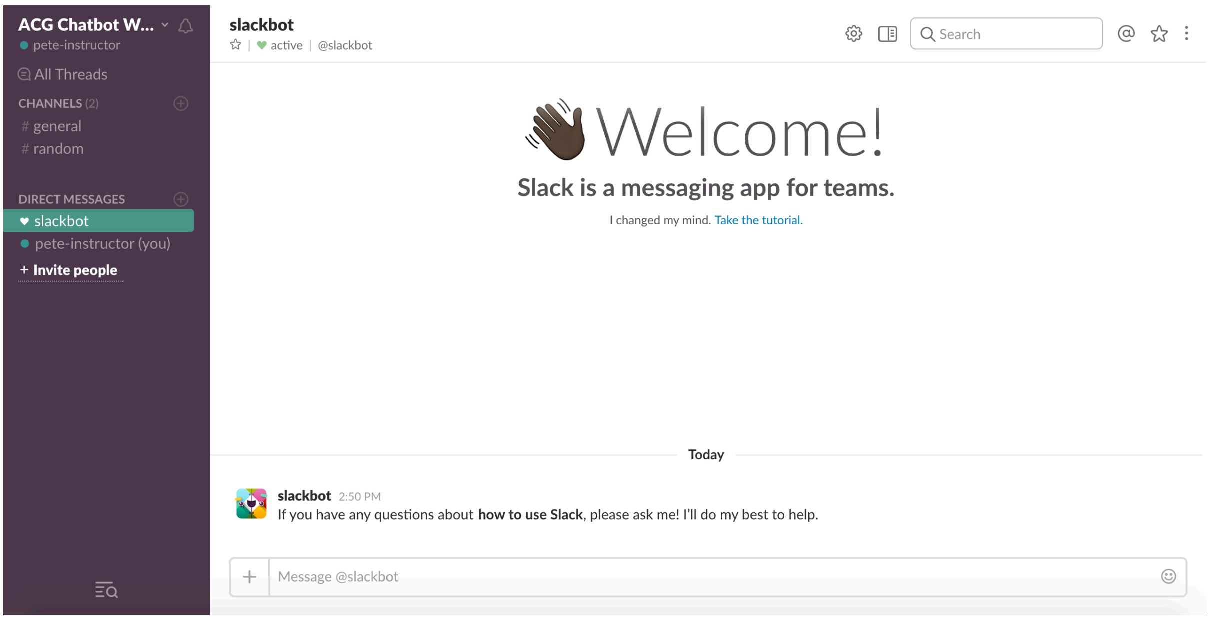
LESSON 2

In lesson 2, you are going to prepare Slack for our bot. You will create a brand new Slack account, configure an app, and create a bot.

1. SIGN UP TO SLACK

You are going to create a new slack account to test your bot.

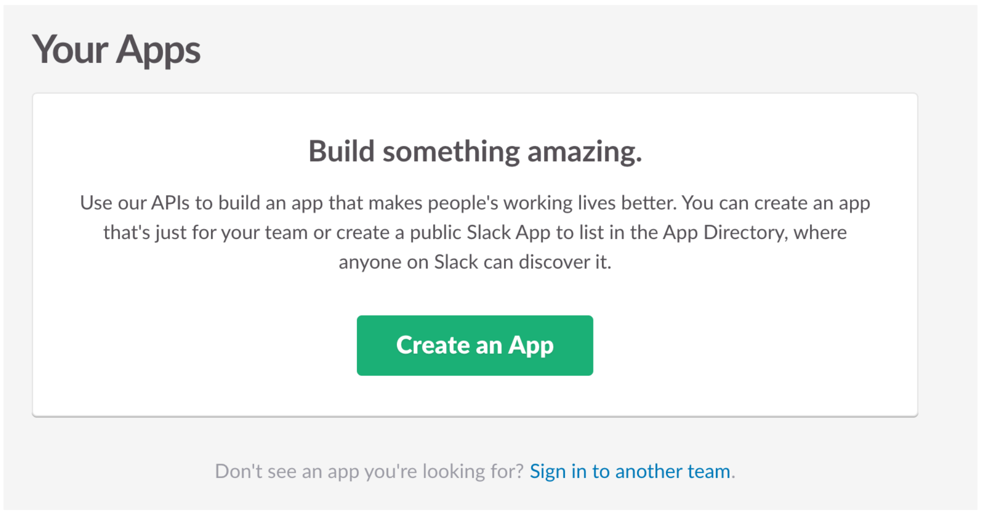
- Go to slack.com.
- Click the **Get Started** button.
- Select **Create a new team.**
- Type in your email address and click **Confirm**.
- You should receive a confirmation code, which you can enter on the page.
- Register yourself as a user, and create a new a Slack team.
- Specify a name and a URL for your new Slack account.
- At this point, you should be logged in to Slack.



slackbot 2:50 PM
If you have any questions about **how to use Slack**, please ask me! I'll do my best to help.

2. CREATE A SLACK APP

To build a bot, first you need to create an app in Slack. An app will allow you to distribute your bot to other teams. Go to https://api.slack.com/apps and click the green **Create an App** button.

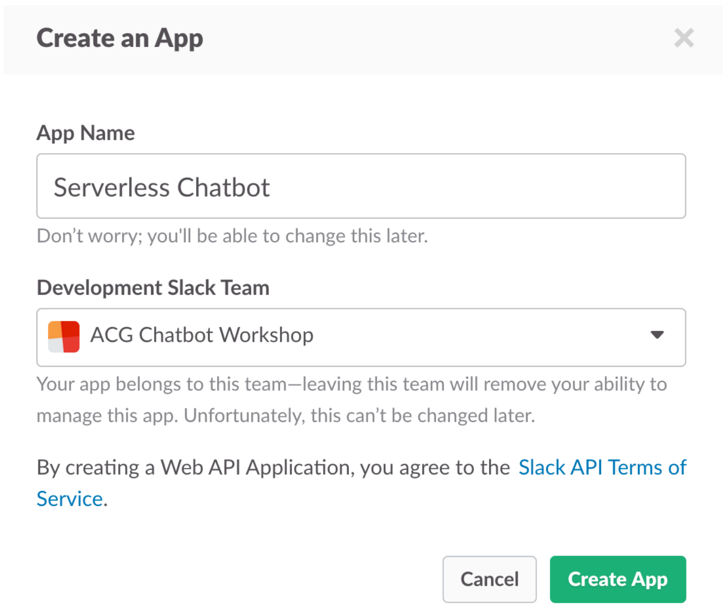

Your Apps

Build something amazing.

Use our APIs to build an app that makes people's working lives better. You can create an app that's just for your team or create a public Slack App to list in the App Directory, where anyone on Slack can discover it.

Don't see an app you're looking for? Sign in to another team.

- In the popup, set an App Name (like **Serverless Chatbot**)
- From the Team dropdown, select the account/team you just created.
- Click **Create App**.


Create an App

App Name

Serverless Chatbot

Don't worry; you'll be able to change this later.

Development Slack Team

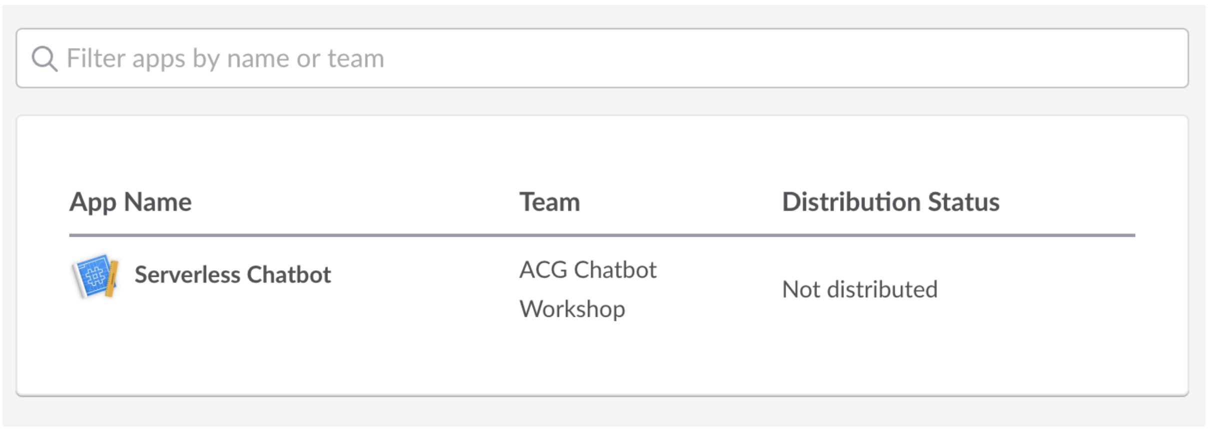
ACG Chatbot Workshop

Your app belongs to this team—leaving this team will remove your ability to manage this app. Unfortunately, this can't be changed later.

By creating a Web API Application, you agree to the Slack API Terms of Service.

You will end up on the Serverless Chatbot app configuration page. Don't close it: we will need it in the next step.

Pro tip: From now on, we will refer to this page as the ***Slack API configuration page***. You can always get back to it by going to https://api.slack.com/apps and then clicking on the Serverless Chatbot app at the bottom.

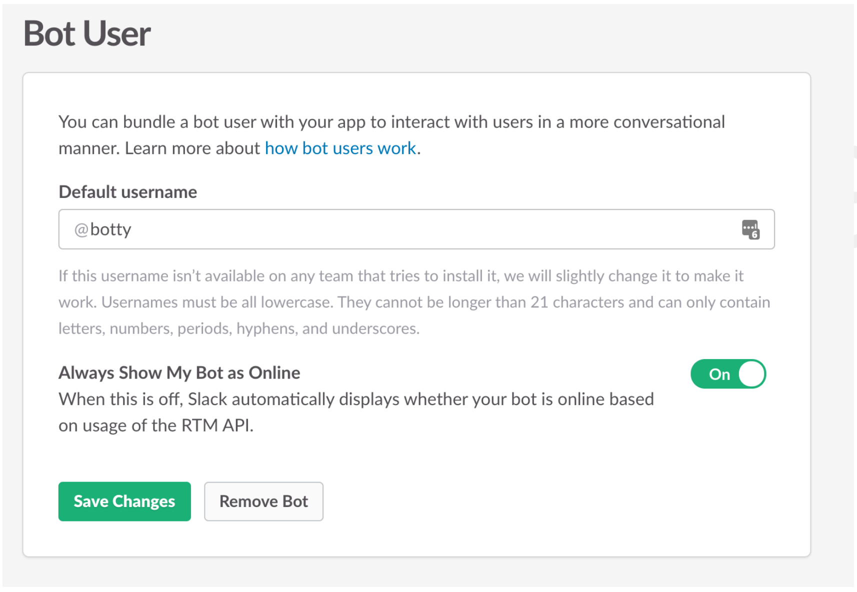

Filter apps by name or team

App Name	Team	Distribution Status
Serverless Chatbot	ACG Chatbot Workshop	Not distributed

3. CREATE A NEW BOT INTEGRATION

Let's create a bot user that will go in to your app.

- Click **Bot users** on the left hand side (under Features).
- Give your new boat a name (we personally like *botty*).
- Set **Always Show My Bot as Online** to on.
- Click **Save Changes**.


Bot User

You can bundle a bot user with your app to interact with users in a more conversational manner. Learn more about how bot users work.

Default username

@botty

If this username isn't available on any team that tries to install it, we will slightly change it to make it work. Usernames must be all lowercase. They cannot be longer than 21 characters and can only contain letters, numbers, periods, hyphens, and underscores.

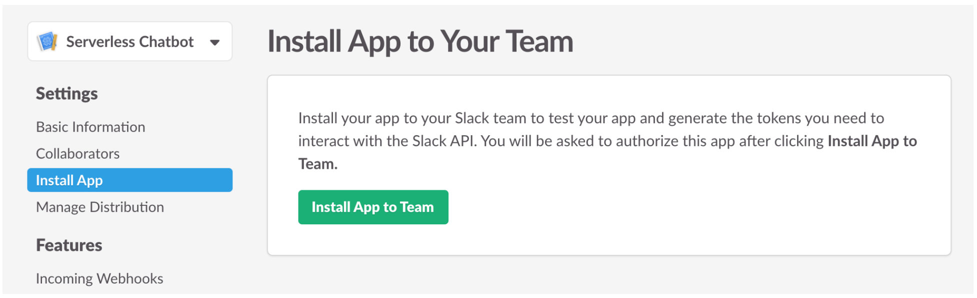
Always Show My Bot as Online: On

When this is off, Slack automatically displays whether your bot is online based on usage of the RTM API.

4. INSTALL BOT TO YOUR TEAM

Let's install your app (and your bot) to your team.

- In the left hand menu, click **Install App**.
- Click the **Install App to Team** button.


Serverless Chatbot

Settings

Basic Information

Collaborators

Install App

Manage Distribution

Features

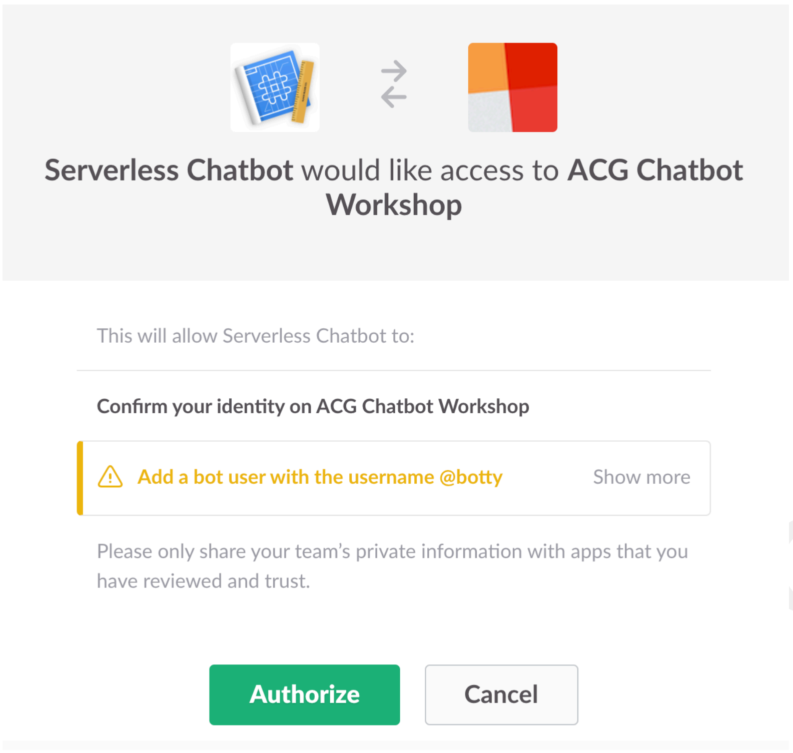
Incoming Webhooks

Install App to Your Team

Install your app to your Slack team to test your app and generate the tokens you need to interact with the Slack API. You will be asked to authorize this app after clicking **Install App to Team.**

Install App to Team

- Click **Authorize** to install the bot


Serverless Chatbot would like access to **ACG Chatbot Workshop**

This will allow Serverless Chatbot to:

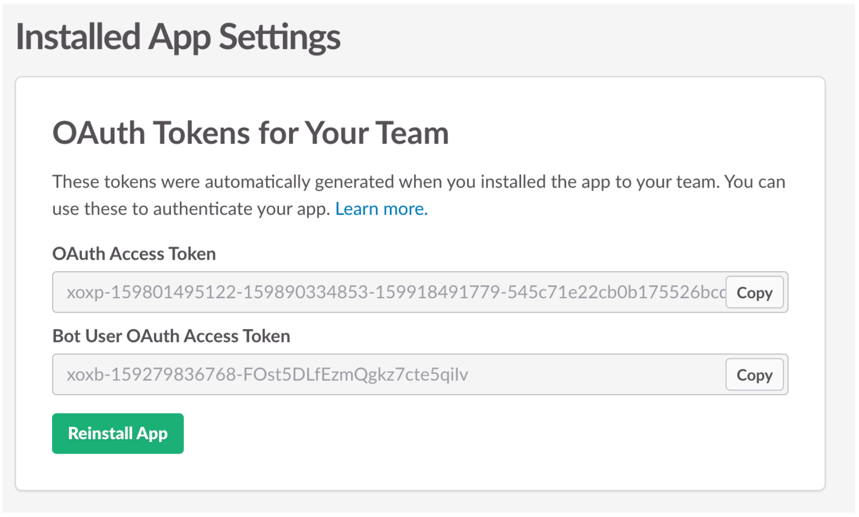
Confirm your identity on ACG Chatbot Workshop

Add a bot user with the username @botty — Show more

Please only share your team's private information with apps that you have reviewed and trust.

Authorize | Cancel

You should immediately get an email telling you that your app was installed. You will also see two tokens that will come in handy later.


Installed App Settings

OAuth Tokens for Your Team

These tokens were automatically generated when you installed the app to your team. You can use these to authenticate your app. Learn more.

OAuth Access Token

```
xoxp-159801495122-159890334853-159918491779-545c71e22cb0b175526bcc
```

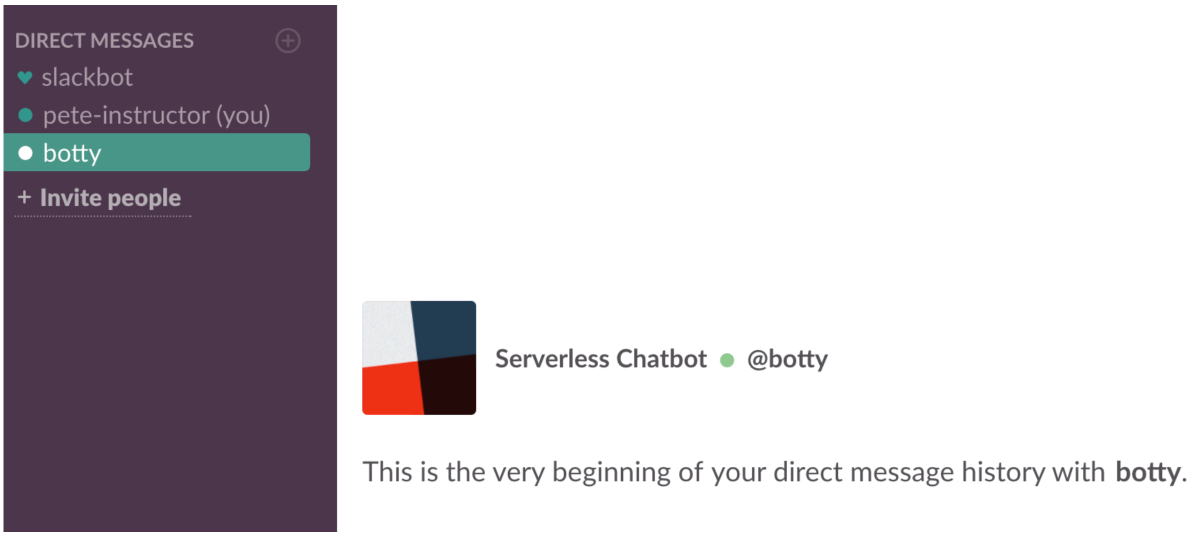
Bot User OAuth Access Token

```
xoxb-159279836768-FOst5DLfEzmQgkz7cte5qilv
```

Reinstall App

5. CHECK SLACK

Switch back to your Slack team and see if you can find your bot there. It should come up as a user *botty* (unless you've given it a different name).



This is the very beginning of your direct message history with **botty**.

Your bot isn't going to do anything just yet, but it's cool to see it there!

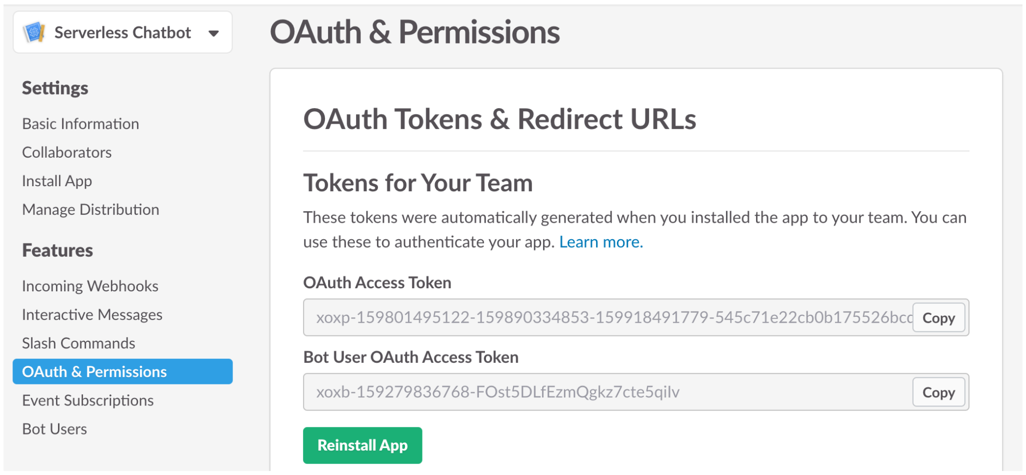
6. GET THE BOT ID

In Slack every user has an ID. A bot is also a type of a user, so it will have an ID as well. Later on, you'll see a way to get a bot ID whenever an app (like our serverless chatbot) is added to a Slack team. For now, however, we are going to write a small function to get the Bot ID out of Slack.

- In the same folder as **serverless.yml** (your main project folder), create a new file called **bot.js**.
- Copy the implementation given below into this file.

```
'use strict';

const https = require('https');
const qs = require('querystring');

module.exports.endpoint = (event, context, callback) => {
    const request = {
        token: 'ReplaceThisWithYourBotUserOAuthAccessToken’
    }

    const url = 'https://slack.com/api/users.list?' + qs.stringify(request);

    https.get(url, (res) => {
        res.setEncoding('utf8');
        let rawData = '';
        res.on('data', (chunk) => rawData += chunk);
        res.on('end', () => {
            try {
                console.log(JSON.parse(rawData));
            } catch (e) {
                console.log(e.message);
            }
        });
    });
};
```

- The one part of the implementation that you need to replace is the token. You must use your **Bot User OAuth Access Token**. Go to the Slack API and click **OAuth & Permissions**.
- Copy the **Bot User OAuth Access Token** and replace the value for **token:** in the above code.



Serverless Chatbot

Settings

Basic Information
Collaborators
Install App
Manage Distribution

Features

Incoming Webhooks
Interactive Messages
Slash Commands
OAuth & Permissions
Event Subscriptions
Bot Users

OAuth & Permissions

OAuth Tokens & Redirect URLs

Tokens for Your Team

These tokens were automatically generated when you installed the app to your team. You can use these to authenticate your app. Learn more.

OAuth Access Token

xoxp-159801495122-159890334853-159918491779-545c71e22cb0b175526bcc Copy

Bot User OAuth Access Token

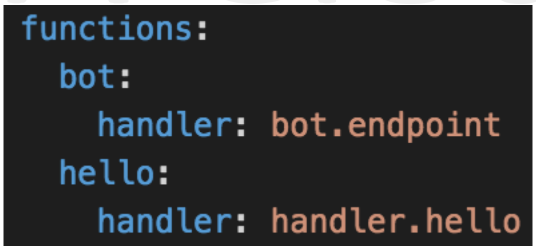
xoxb-159279836768-FOst5DLfEzmQgkz7cte5qilv Copy

Reinstall App

- Open **serverless.yml** and add the following two lines just below **`functions:`**

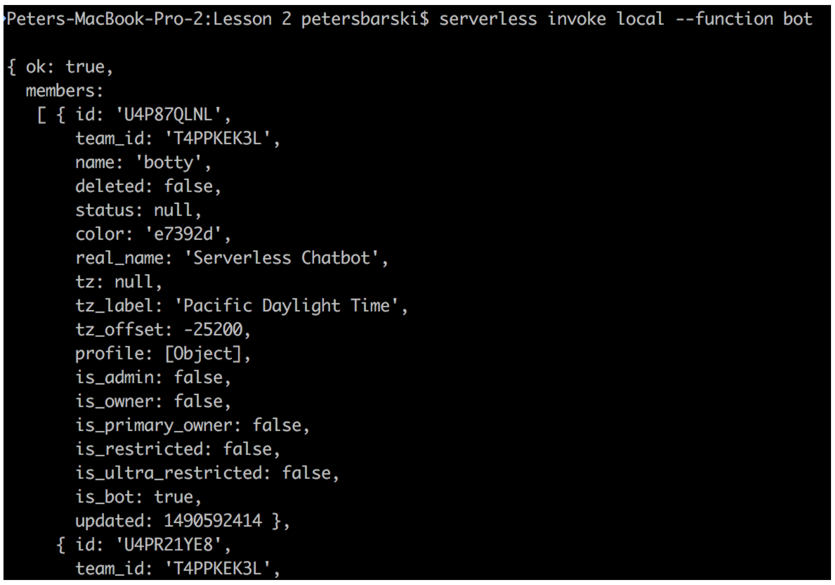
```
bot:
    handler: bot.endpoint
```

The **serverless.yml** file should look like this:



```
functions:
  bot:
    handler: bot.endpoint
  hello:
    handler: handler.hello
```

- Run **`serverless invoke local --function bot`** from the terminal. This command will run the bot function locally and get a list of all users and their IDs.



```
Peters-MacBook-Pro-2:Lesson 2 petersbarski$ serverless invoke local --function bot

{ ok: true,
  members:
   [ { id: 'U4P87QLNL',
       team_id: 'T4PPKEK3L',
       name: 'botty',
       deleted: false,
       status: null,
       color: 'e7392d',
       real_name: 'Serverless Chatbot',
       tz: null,
       tz_label: 'Pacific Daylight Time',
       tz_offset: -25200,
       profile: [Object],
       is_admin: false,
       is_owner: false,
       is_primary_owner: false,
       is_restricted: false,
       is_ultra_restricted: false,
       is_bot: true,
       updated: 1490592414 },
     { id: 'U4PR21YE8',
       team_id: 'T4PPKEK3L',
```

- Scroll up until you find your bot name (ours is 'botty.') Record your bot's user ID (2 lines above 'name'.) In the above screenshot, it's **U4P87QLNL**; you will have something similar.

7. ECHO BOT

Let's make our bot echo anything that is said to it. When you enter *@botty hello*, your message will go up to the Lambda function and should come back downwith a timestamp as well as your original message. To do this, we need to create an endpoint (using the API Gateway) which the bot will use to send a request and receive a response. This API endpoint will invoke a Lambda function that will echo back a response with an attached timestamp to show us that it is working.

8. UPDATE SERVERLESS.YML

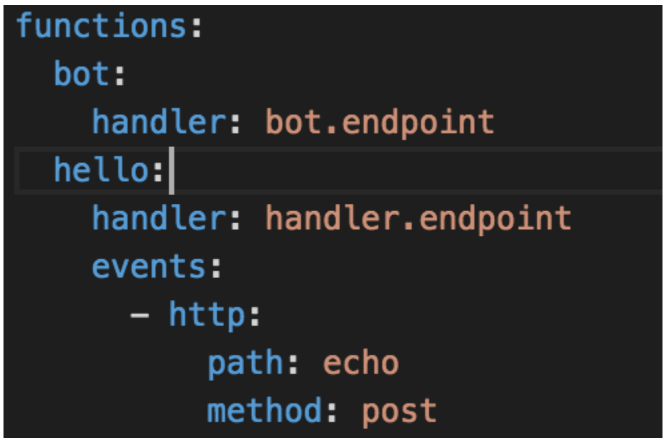
To begin adding an API endpoint, we need to modify our **serverless.yml** (we recommend making a backup of **serverless.yml** and **handler.js** before you proceed.)

- Open the **serverless.yml** you created in the previous lesson.
- Add the following under **handler: handler.hello**

```
    events:
      - http:
          path: echo
          method: post
```

- Modify **`handler.hello`** to be **`handler.endpoint`**

The whole function structure should now match the image below:



```yaml
functions:
  bot:
    handler: bot.endpoint
  hello:
    handler: handler.endpoint
    events:
      - http:
          path: echo
          method: post
```

What we did here is tell Serverless Framework to create an endpoint in the API Gateway and connect it to our function. This endpoint will be accessible via a GET method and invoke the endpoint function in our handler. Now you need to update the handler.js file to account for this.

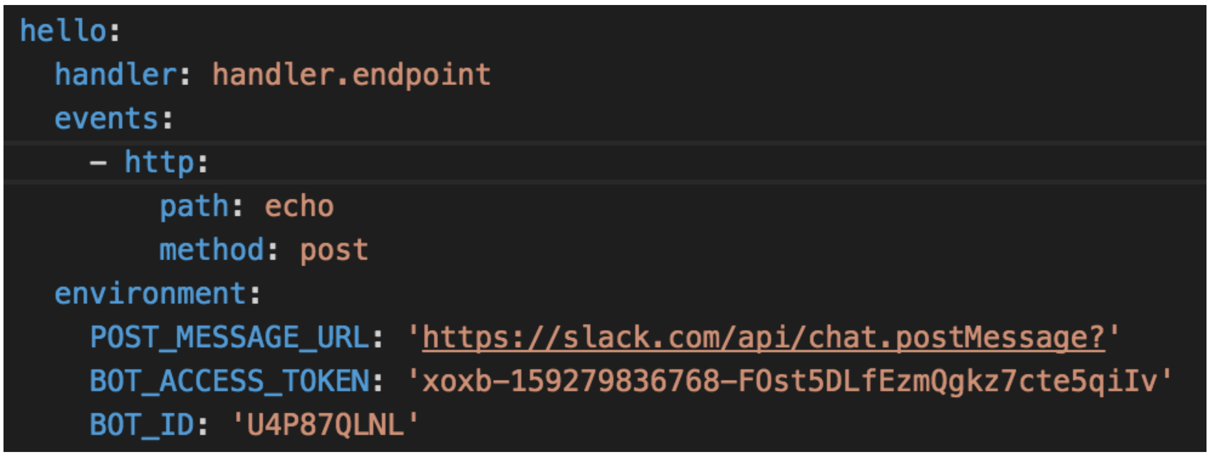
9. UPDATE HANDLER.JS

You will need to update **handler.js** to return back the message sent by the botty. Replace the contents of handler.js with the following:

```javascript
'use strict';

const https = require('https');
const qs = require('querystring');

const direct_mention = new RegExp('^\<\@' + process.env.BOT_ID + '\>', 'i');

module.exports.endpoint = (event, context, callback) => {
  const request = JSON.parse(event.body);

  if (request.event.text.match(direct_mention)) {

    const response = {
      token: process.env.BOT_ACCESS_TOKEN,
      channel: request.event.channel,
      text: request.event.text.replace(direct_mention, '') + ' [' + Date.now() + ']'
    }

    const URL = process.env.POST_MESSAGE_URL + qs.stringify(response);

    https.get(URL, (res) => {
      const statusCode = res.statusCode;

      if (statusCode !== 200) {
        console.log(res);
      }

      callback(null, {statusCode: 200});
    })
  }
};
```

If you look at the code you will see that we use 3 environment variables. These are BOT_ID, BOT_ACCESS_TOKEN, and POST_MESSAGE_URL. Environment variables are settings that the Lambda function can extract. We can set these in our **serverless.yml** and they'll be automatically added to Lambda's environment by the Framework.

- Open **serverless.yml** and add the three environment variables BOT_ID, BOT_ACCESS_TOKEN and POST_MESSAGE_URL.



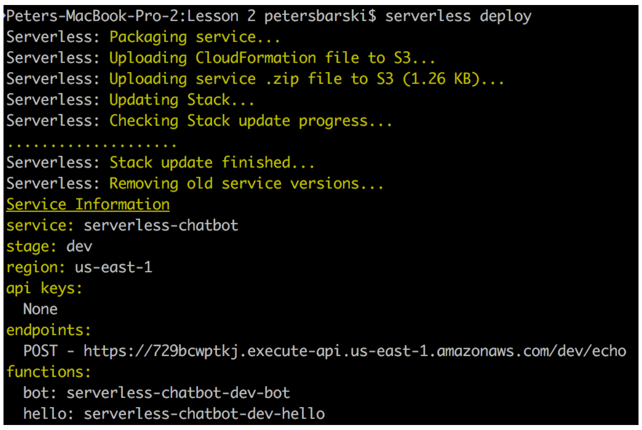
```
hello:
  handler: handler.endpoint
  events:
    - http:
        path: echo
        method: post
  environment:
    POST_MESSAGE_URL: 'https://slack.com/api/chat.postMessage?'
    BOT_ACCESS_TOKEN: 'xoxb-159279836768-FOst5DLfEzmQgkz7cte5qiIv'
    BOT_ID: 'U4P87QLNL'
```

- The **BOT_ID** variable comes from the bot ID you discovered in step 6.
- The **BOT_ACCESS_TOKEN** can be found if you go to the Slack API (https://api.slack.com/apps), select your app and click on OAUTH & Permissions. The value that you need to copy is called ***Bot User OAuth Access Token***. You also did this in **step 6**.
- The **POST_MESSAGE_URL** needs to be **https://slack.com/api/chat.postMessage?**

Reference: the regular expression used in the code snippet before was originally created here: https://github.com/howdyai/botkit/blob/563f976a6b4c675712e5ae8a03689cf7e420149e/lib/SlackBot.js#L484

10. DEPLOY YOUR CODE

Open your terminal; from your **serverless-chatbot** folder (where your **serverless.yml** and **handler.js** are), run `serverless deploy` and wait for the operation to complete.



```
Peters-MacBook-Pro-2:Lesson 2 petersbarski$ serverless deploy
Serverless: Packaging service...
Serverless: Uploading CloudFormation file to S3...
Serverless: Uploading service .zip file to S3 (1.26 KB)...
Serverless: Updating Stack...
Serverless: Checking Stack update progress...
....................
Serverless: Stack update finished...
Serverless: Removing old service versions...
Service Information
service: serverless-chatbot
stage: dev
region: us-east-1
api keys:
  None
endpoints:
  POST - https://729bcwptkj.execute-api.us-east-1.amazonaws.com/dev/echo
functions:
  bot: serverless-chatbot-dev-bot
  hello: serverless-chatbot-dev-hello
```

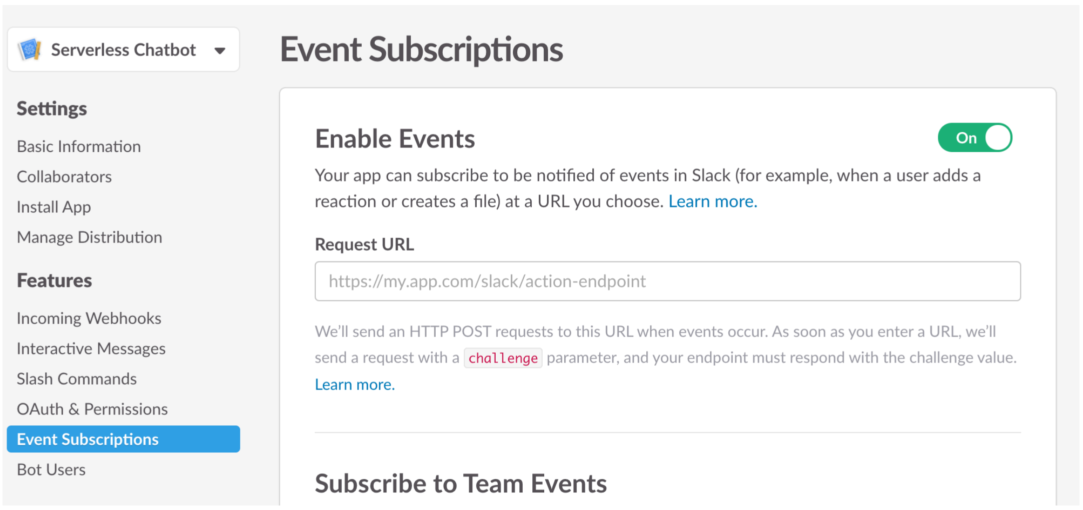
At the end of the deployment you should see an endpoint created for you by the API Gateway. Make a copy of this URL.

In ours (above), it's: `https://729bcwptjl.execute-api.us-east-1.amazonaws.com/dev/echo`

11. CONNECT ALL THE BOTS

We now need to tell your bot to invoke the URL that was created in the previous step.

- Go back to your Slack App settings.
- Under the Features menu click **Events**.
- Switch events to **On**



Event Subscriptions

Enable Events

Your app can subscribe to be notified of events in Slack (for example, when a user adds a reaction or creates a file) at a URL you choose. Learn more.

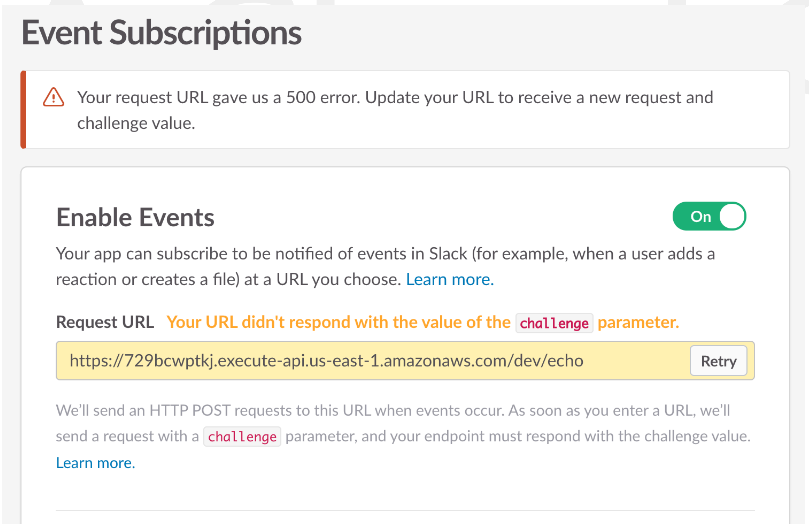
Request URL

https://my.app.com/slack/action-endpoint

We'll send an HTTP POST requests to this URL when events occur. As soon as you enter a URL, we'll send a request with a `challenge` parameter, and your endpoint must respond with the challenge value. Learn more.

Subscribe to Team Events

- Copy the endpoint URL created in the previous step and paste it in to the **Request URL** text box.
- As soon as you click out of the text box, Slack will test the URL but what's this?? Slack tells us that the URL didn't respond with the value of the **challenge** parameter.



Event Subscriptions

Your request URL gave us a 500 error. Update your URL to receive a new request and challenge value.

Enable Events

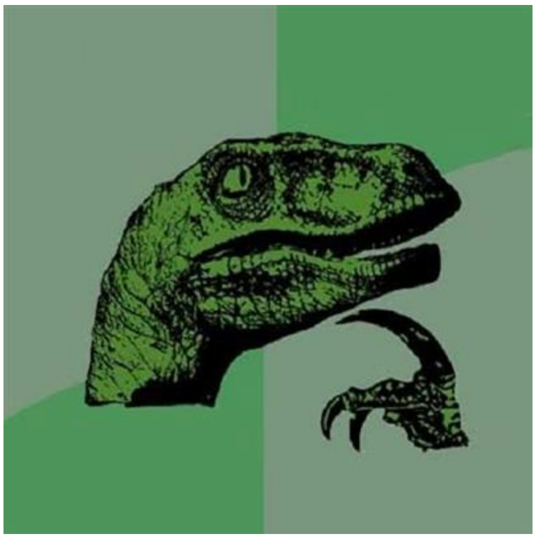
Your app can subscribe to be notified of events in Slack (for example, when a user adds a reaction or creates a file) at a URL you choose. Learn more.

Request URL Your URL didn't respond with the value of the `challenge` parameter.

https://729bcwptkj.execute-api.us-east-1.amazonaws.com/dev/echo

We'll send an HTTP POST requests to this URL when events occur. As soon as you enter a URL, we'll send a request with a `challenge` parameter, and your endpoint must respond with the challenge value. Learn more.

What is going on here? Why didn't it work?



12. VERIFYING OWNERSHIP

So, we got an error…. But why? The reason is that Slack wants us to verify that we own the endpoint that we are asking to use. It doesn't want us to abuse other endpoints, and that makes sense. We need to prove that this is our endpoint. To do this verification, Slack initially sends a request to our endpoint that looks like this:

```
{
    "token": "Jhj5dZrVaK7ZwHHjRyZWjbDl",
    "challenge": "3eZbrw1aBm2rZgRNFdxV2595E9CY3gmdALWMmHkvFXO7tYXAYM8P
    "type": "url_verification"
}
```

We need to detect when this happens and return a response with a challenge attribute that looks like this:

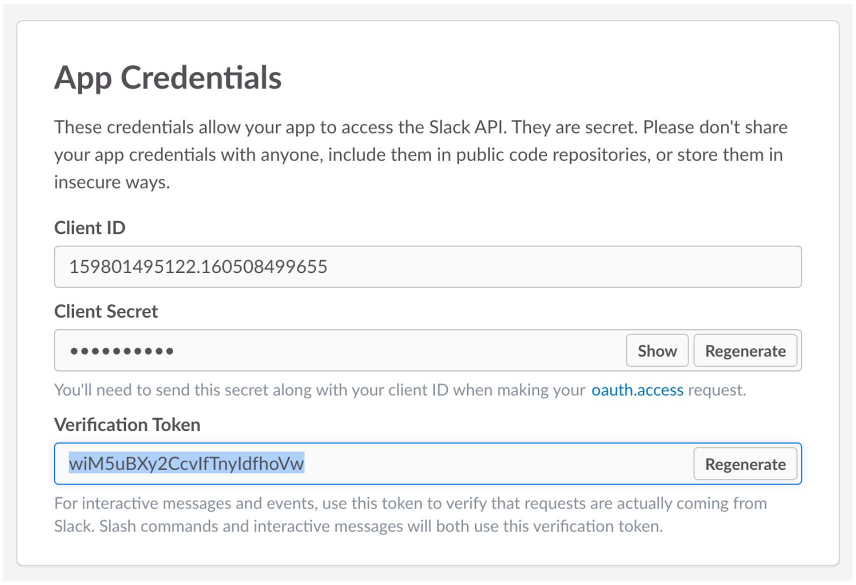
```
HTTP 200 OK
Content-type: application/x-www-form-urlencoded
{"challenge":"3eZbrw1aBm2rZgRNFdxV2595E9CY3gmdALWMmHkvFXO7tYXAYM8P"}
```

Our Lambda function can also verify that the supplied token matches our application's configured Slack token. You can read more about URL verification at https://api.slack.com/events/url_verification.

13. UPDATE SERVERLESS.YML

To do URL verification, we will need to check that the supplied token is correct. To do this we'll add the token from the Slack configuration page to our Lambda function as an environment variable. In the next step, we'll modify our Lambda function to check that the token sent in the request matches the token stored in the environment.

- Go to your Slack API page and click **Basic Information**
- Scroll down to App Credentials and find Verification Token



App Credentials

These credentials allow your app to access the Slack API. They are secret. Please don't share your app credentials with anyone, include them in public code repositories, or store them in insecure ways.

Client ID

159801495122.160508499655

Client Secret

••••••••••

You'll need to send this secret along with your client ID when making your oauth.access request.

Verification Token

wiM5uBXy2CcvIfTnyIdfhoVw

For interactive messages and events, use this token to verify that requests are actually coming from Slack. Slash commands and interactive messages will both use this verification token.

- Copy this token somewhere safe you'll need it next
- Open **serverless.yml** and add this verification token as an environment variable called VERIFICATION_TOKEN

```
environment:
  POST_MESSAGE_URL: 'https://slack.com/api/chat.postMessage?'
  BOT_ACCESS_TOKEN: 'xoxb-159279836768-FOst5DLfEzmQgkz7cte5qiIv'
  BOT_ID: 'U4P87QLNL'
  VERIFICATION_TOKEN: 'wiM5uBXy2CcvIfTnyIdfhoVw'
```

14. UPDATE HANDLER.JS

Now it's time to update **handler.js** to verify the token and return the challenge as part of the response. That should satisfy Slack's requirements. Copy the following code block into **handler.js** under the line that ends **`const request = JSON.parse(event.body);`**

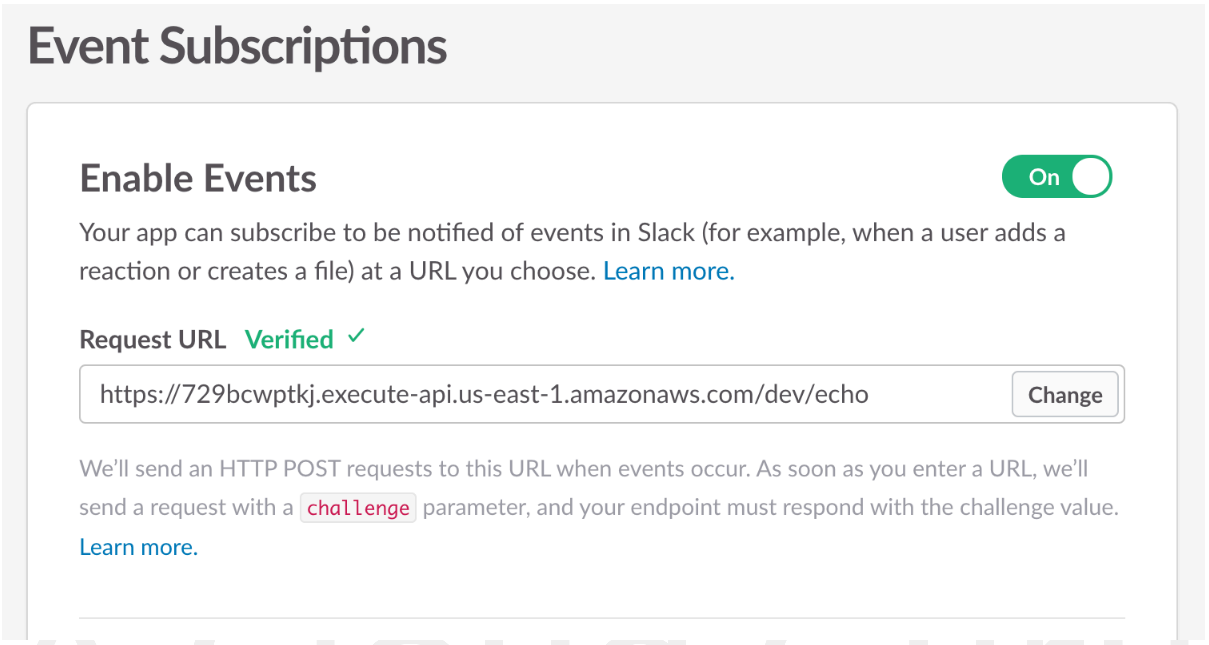
```
if (request.type === 'url_verification' &&
    request.token === process.env.VERIFICATION_TOKEN) {

      const response = {
        statusCode: 200,
        body: JSON.stringify({
          'challenge' : request.challenge
        })
      }

      return callback(null, response);
}
```

15. DEPLOY

Deploy the function again by running **`serverless deploy`** from the command line.

16. CHECK BUT VERIFY

Let's try the verification again. Jump in to the Slack API and enter the Request URL again. It should now verify and give you a nice little green tick.



Event Subscriptions

Enable Events

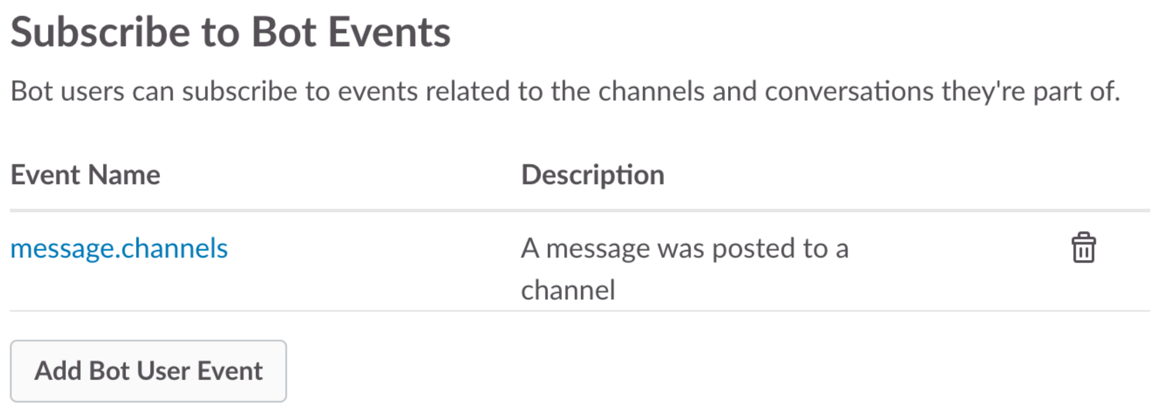
Your app can subscribe to be notified of events in Slack (for example, when a user adds a reaction or creates a file) at a URL you choose. Learn more.

Request URL Verified ✓

https://729bcwptkj.execute-api.us-east-1.amazonaws.com/dev/echo

We'll send an HTTP POST requests to this URL when events occur. As soon as you enter a URL, we'll send a request with a `challenge` parameter, and your endpoint must respond with the challenge value. Learn more.

Scroll down the page until you find a section called **Subscribe to Bot Events**. Here we must add an event that will invoke our URL. Find and select an event called **message.channels**.



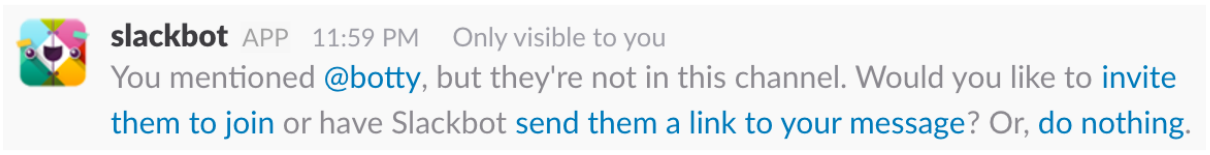
Subscribe to Bot Events

Bot users can subscribe to events related to the channels and conversations they're part of.

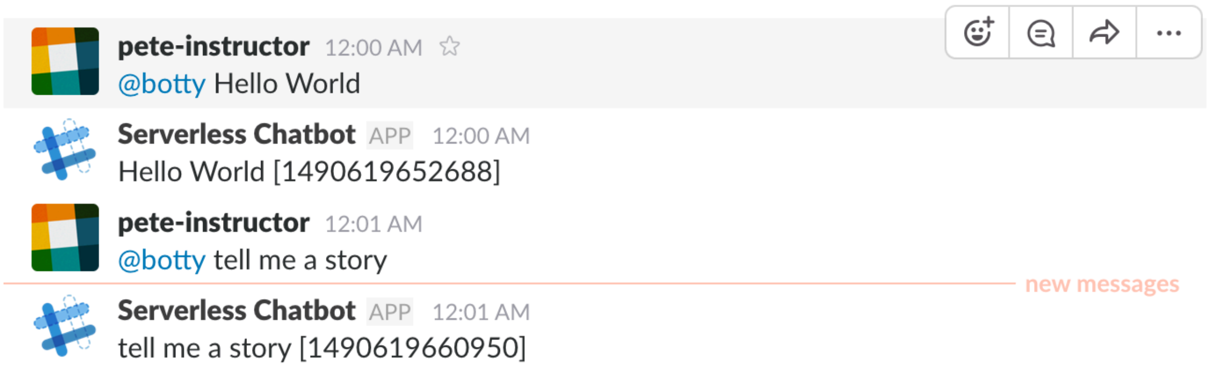
Event Name	Description
message.channels	A message was posted to a channel

17. CHECK HOW IT WORKS IN SLACK

Go back to Slack and select the general channel. You can invoke botty by typing @botty and then a message. You may get a notification telling you to invite botty to the channel first. You can do it by clicking on a link.


slackbot APP 11:59 PM Only visible to you
You mentioned @botty, but they're not in this channel. Would you like to invite them to join or have Slackbot send them a link to your message? Or, do nothing.

After that you can issue commands like @botty hello world or @botty tell me a story. Your bot should faithfully echo back everything you type and add a timestamp to the end.


pete-instructor 12:00 AM
@botty Hello World

Serverless Chatbot APP 12:00 AM
Hello World [1490619652688]

pete-instructor 12:01 AM
@botty tell me a story

Serverless Chatbot APP 12:01 AM
tell me a story [1490619660950]

This was a long lesson, but we are now set up to do really exciting things in the next one. See you there!

ADVANCED QUESTIONS

1. Modify the implementation