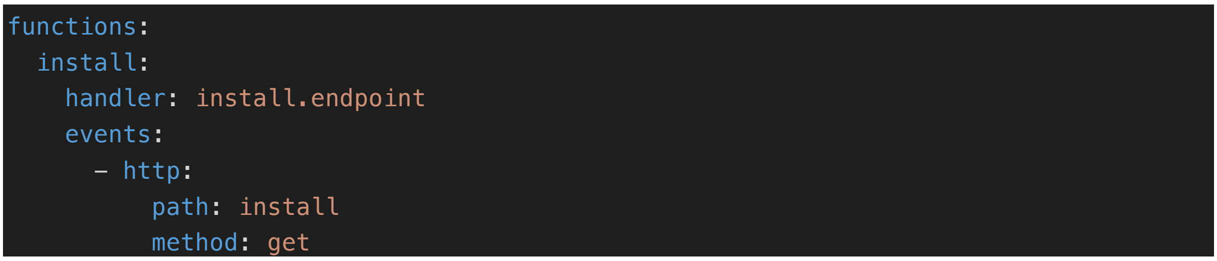
LESSON 6

In Lesson 6, you will learn how to share your app with other teams.

1. MAKING THINGS RIGHT

Your Slack Bot is coupled to your Slack team. That's OK while you've been learning; but what if you wanted to make it installable by anyone? After all, it would be fun to ship our bot and make it available for everyone to use. In this lesson, you'll create a Lambda function and a DynamoDB table that will allow users to install and register their bots. Our DynamoDB table will maintain a list of registration and various bots. And, we'll grow it to contain other information too.

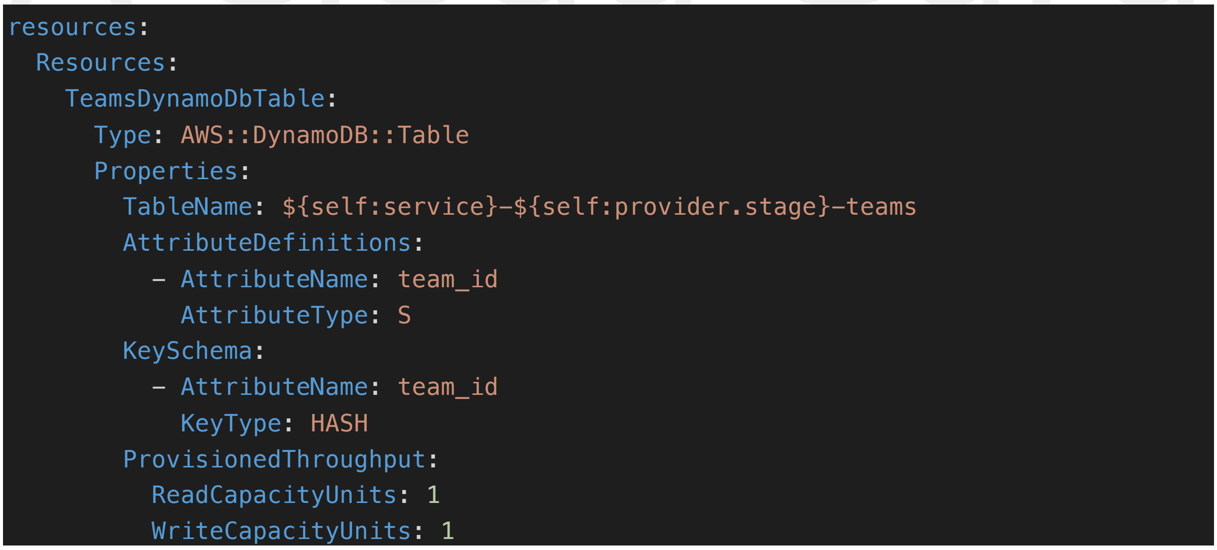
- Make a copy of the **lesson5** folder and name it **lesson6**.
- In the **lesson6** folder, create a new file called **install.js**
- Add the following to **serverless.yml**



```
functions:
  install:
    handler: install.endpoint
    events:
      - http:
          path: install
          method: get
```

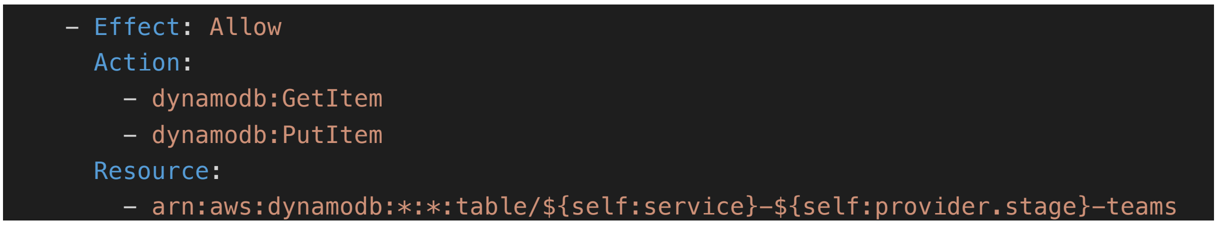
- You are going to add a DynamoDB table that will contain information about each team that has installed the bot. So, you need to specify a DynamoDB table in the **resource** section of the **serverless.yml** file and the Serverless Framework will create it for you. Add the following to the bottom of your **serverless.yml**.

A Cloud Guru



```
resources:
  Resources:
    TeamsDynamoDbTable:
      Type: AWS::DynamoDB::Table
      Properties:
        TableName: ${self:service}-${self:provider.stage}-teams
        AttributeDefinitions:
          - AttributeName: team_id
            AttributeType: S
        KeySchema:
          - AttributeName: team_id
            KeyType: HASH
        ProvisionedThroughput:
          ReadCapacityUnits: 1
          WriteCapacityUnits: 1
```

- Finally, add the following to the **iamRoleStatements** section



```
    - Effect: Allow
      Action:
        - dynamodb:GetItem
        - dynamodb:PutItem
      Resource:
        - arn:aws:dynamodb:*:*:table/${self:service}-${self:provider.stage}-teams
```

Remember to save your serverless.yml file once you are finished with it.

2. INSTALL FUNCTION

You have an install function in **install.js** so open it up and copy the following implementation to it. In a nutshell, this function will be invoked by Slack. It will then go out and get an OAuth token with the additional information about the team. Finally, it will store this information in to a DynamoDB table.

```
'use strict';

const aws = require('aws-sdk');
const qs = require('querystring');
const request = require('request');

const db = new aws.DynamoDB.DocumentClient();

const extractCode = function(event) {
    return new Promise((resolve, reject) => {
        if (event.queryStringParameters && event.queryStringParameters.code) {
            return resolve(event.queryStringParameters.code);
        }

        reject('Code not provided');
    });
};

const getOAuthToken = function(code) {
    return new Promise((resolve, reject) => {
        if (code === null) { return reject('Could not provided'); }

        const params = {
            client_id: process.env.CLIENT_ID,
            client_secret: process.env.CLIENT_SECRET,
            code
        }

        const url = process.env.SLACK_OAUTH + qs.stringify(params);

        request.get(url, (err, res, body) => {
            if (err || res.statusCode !== 200) {
                reject(err);
            } else {
                resolve(body);
            }
        })
    });
};

const saveToDynamo = function(response) {
    return new Promise((resolve, reject) => {
        const params = {
            TableName: process.env.TABLE_NAME,
            Item: JSON.parse(response)
        }

        db.put(params, (err, data) =>{
            if (err) {
                reject(err);
            } else {
                resolve();
            }
        })
    });
}

const successResponse = function() {
    return {
        statusCode: 200
    }
}

const errorResponse = function() {
    return {
        statusCode: 302
    }
}

module.exports.endpoint = (event, context, callback) => {
    extractCode(event)
        .then((code) => getOAuthToken(code))
        .then((response) => saveToDynamo(response))
        .then(() => callback(null, successResponse()))
        .catch((err) => callback(null, errorResponse()))
};
```

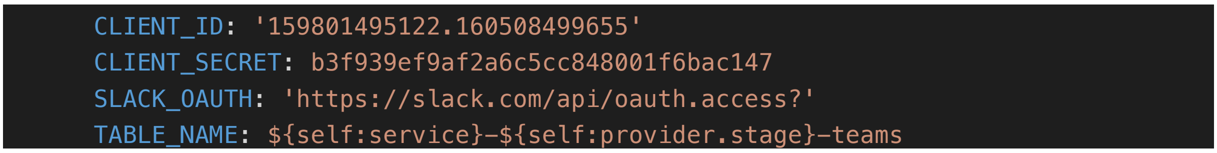
- There are a couple of changes you need to make to **serverless.yml:** we need to add environment variables to the existing **hello:** function, and we need to add a couple of environment variables to the **install:** function.

First, add the following to the environment variables of the **hello:** function in **serverless.yml**:

```
TEAMS_TABLE: ${self:service}-${self:provider.stage}-teams
```

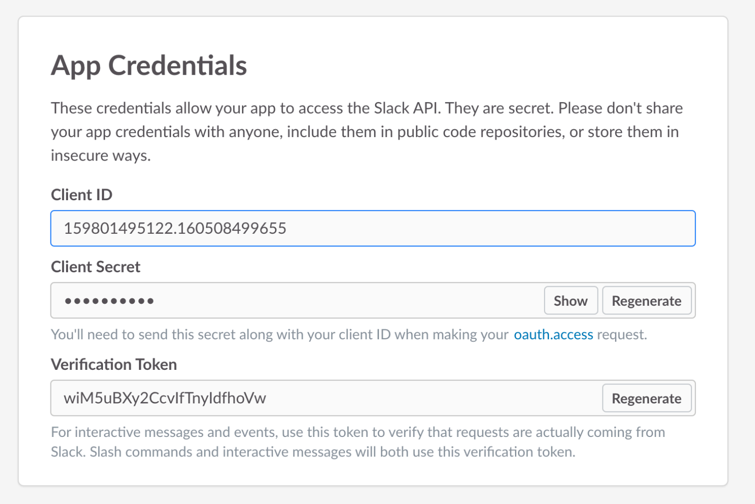
Then, add your environment variables to the **install:** function you added earlier in this lesson.

```
functions:
  install:
    handler: install.endpoint
    events:
      - http:
          path: install
          method: get
    environment:
```



```
    CLIENT_ID: '159801495122.160508499655'
    CLIENT_SECRET: b3f939ef9af2a6c5cc848001f6bac147
    SLACK_OAUTH: 'https://slack.com/api/oauth.access?'
    TABLE_NAME: ${self:service}-${self:provider.stage}-teams
```

You will need to replace the Client ID and the Client Secret with your own values. They can be found in your Slack API settings if you click **Basic Information** and then scroll down to **App Credentials**.



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

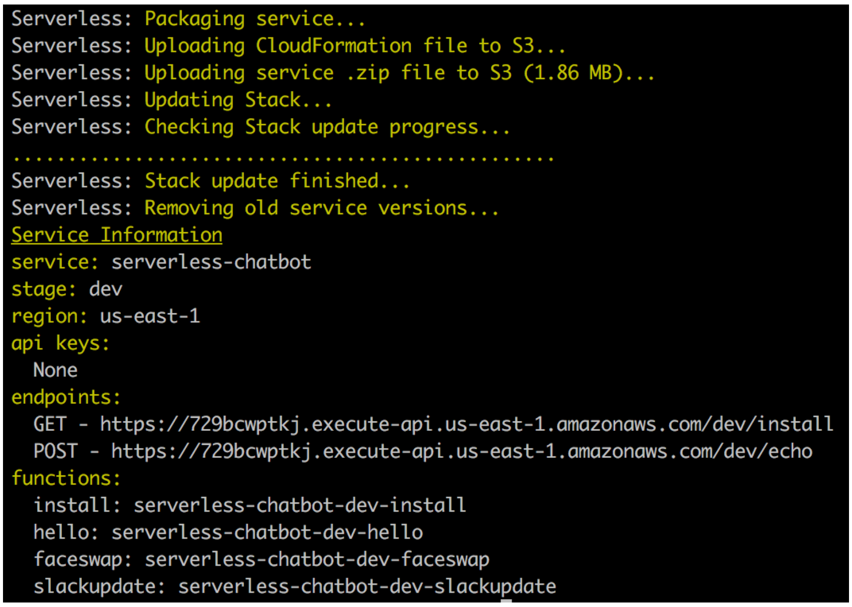
wiM5uBXy2CcvIfTnyIdfhoVw

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.

A Cloud Guru

3. DEPLOYMENT

Having done everything in the previous steps, deploy your bot by running `serverless deploy` from the terminal. ***Note the new URL (it ends with /dev/install) you will get after the deployment is finished. You will need it in the next step.***

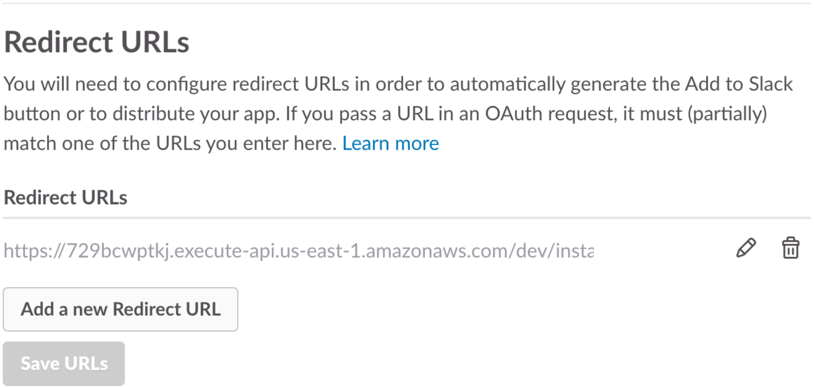


```
Serverless: Packaging service...
Serverless: Uploading CloudFormation file to S3...
Serverless: Uploading service .zip file to S3 (1.86 MB)...
Serverless: Updating Stack...
Serverless: Checking Stack update progress...
................................................
Serverless: Stack update finished...
Serverless: Removing old service versions...
Service Information
service: serverless-chatbot
stage: dev
region: us-east-1
api keys:
  None
endpoints:
  GET - https://729bcwptkj.execute-api.us-east-1.amazonaws.com/dev/install
  POST - https://729bcwptkj.execute-api.us-east-1.amazonaws.com/dev/echo
functions:
  install: serverless-chatbot-dev-install
  hello: serverless-chatbot-dev-hello
  faceswap: serverless-chatbot-dev-faceswap
  slackupdate: serverless-chatbot-dev-slackupdate
```

4. GENERATING INSTALL LINK

You can finally generate an install link and an install button to share with others.

- In the Slack API click **Oauth & Permissions**.
- Scroll to **Redirect URLs** and add a new URL. This should be your new **install** URL from the previous step. Don't forget to save!



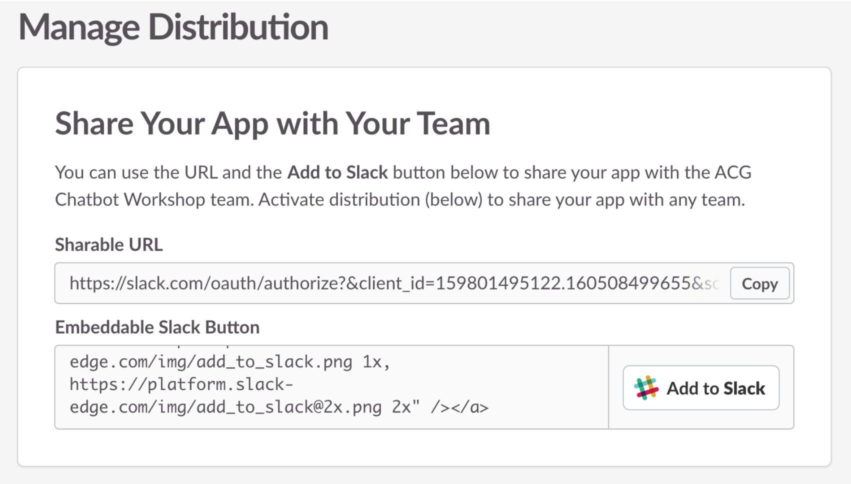
Redirect URLs

You will need to configure redirect URLs in order to automatically generate the Add to Slack button or to distribute your app. If you pass a URL in an OAuth request, it must (partially) match one of the URLs you enter here. Learn more

Redirect URLs

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

- Click **Manage Distribution** on the left.
- You should see a **sharable URL** and code for an **Embeddable Slack Button**



Manage Distribution

Share Your App with Your Team

You can use the URL and the **Add to Slack** button below to share your app with the ACG Chatbot Workshop team. Activate distribution (below) to share your app with any team.

Sharable URL

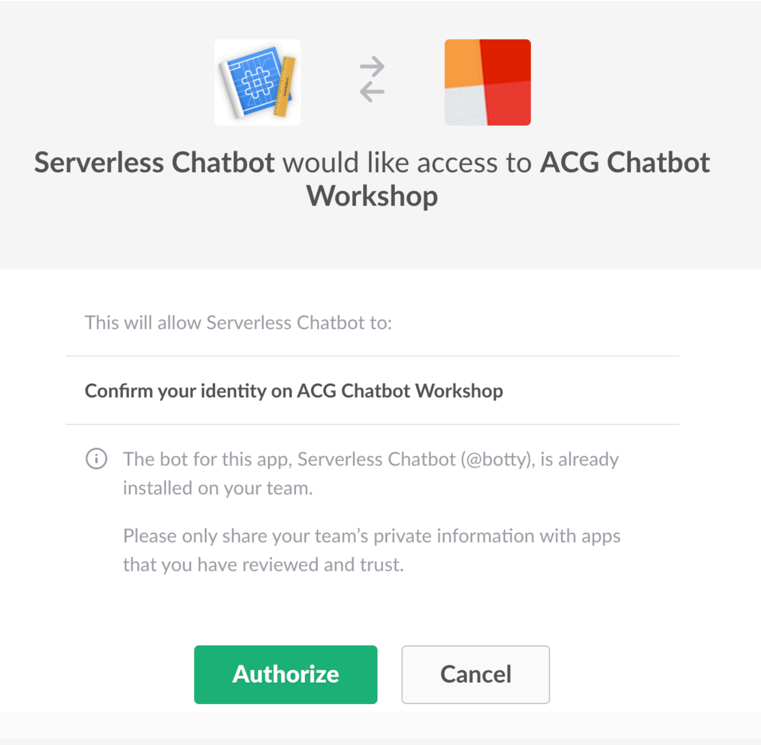
https://slack.com/oauth/authorize?&client_id=159801495122.160508499655&sc

Embeddable Slack Button

```
edge.com/img/add_to_slack.png 1x,
https://platform.slack-
edge.com/img/add_to_slack@2x.png 2x" /></a>
```

5. TESTING

Let's test the installation. Grab the **Sharable URL,** paste it in to your browser's address bar, and hit enter. You should see something like this.



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

This will allow Serverless Chatbot to:

Confirm your identity on ACG Chatbot Workshop

The bot for this app, Serverless Chatbot (@botty), is already installed on your team.

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

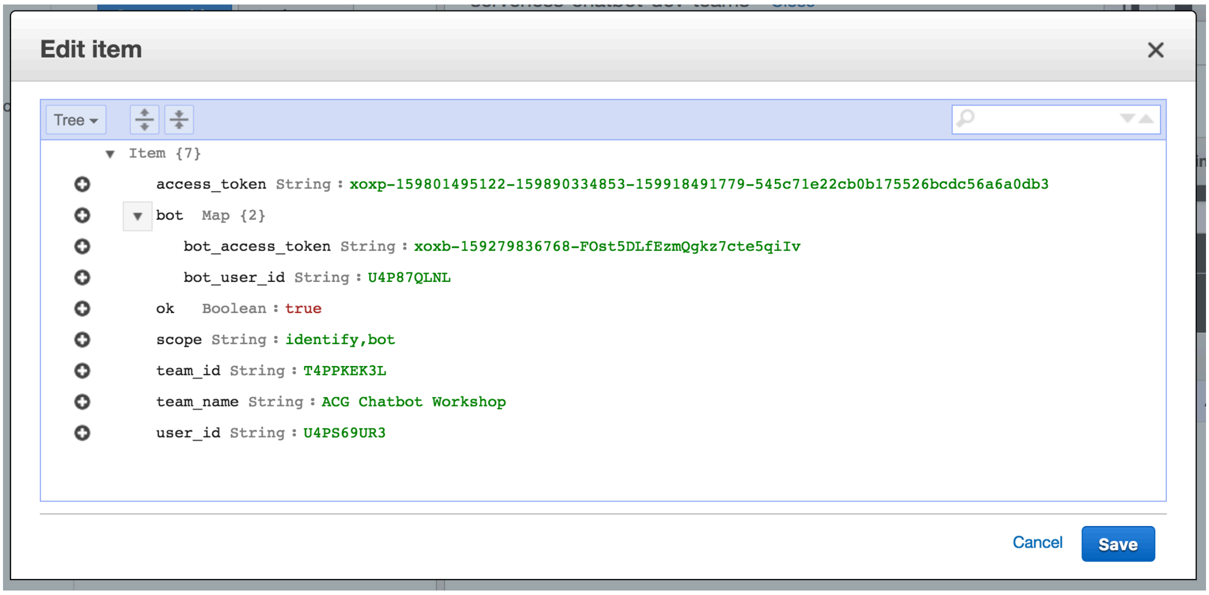
- You already have the bot installed but click **Authorize** anyway.
- You will see a blank screen because we haven't set up a redirect URL but if you inspect the network tab of your browser you should see a 200 response.

A Cloud Guru

6. DYNAMODB

Let's check out DynamoDB to see if the response from Slack was saved correctly.

- In the AWS console click on **DynamoDB**
- Click **Tables**
- Select **serverless-chatbot-dev-teams**
- Select **Items**
- Click on the item in the grid to see it



```
▼ Item {7}
    access_token String : xoxp-159801495122-159890334853-159918491779-545c71e22cb0b175526bcdc56a6a0db3
  ▼ bot  Map {2}
      bot_access_token String : xoxb-159279836768-FOst5DLfEzmQgkz7cte5qiIv
      bot_user_id String : U4P87QLNL
    ok   Boolean : true
    scope String : identify,bot
    team_id String : T4PPKEK3L
    team_name String : ACG Chatbot Workshop
    user_id String : U4PS69UR3
```

7. HARDCODED VALUES

We'd love to ship your bot now but we cannot. You have hardcoded values, including the bot_access_token, in your **serverless.yml**. Your challenge is to modify the other functions you have to read from the database to get the relevant information, rather than relying on hardcoded environment variables.

If you look at **handler.js**, it refers to the BOT_ACCESS_TOKEN environment variable. But, you can now change it to look up the bot access token in DynamoDB based on the team. You can get the team out of the request that Slack sends to our Lambda function. Let's try it now.

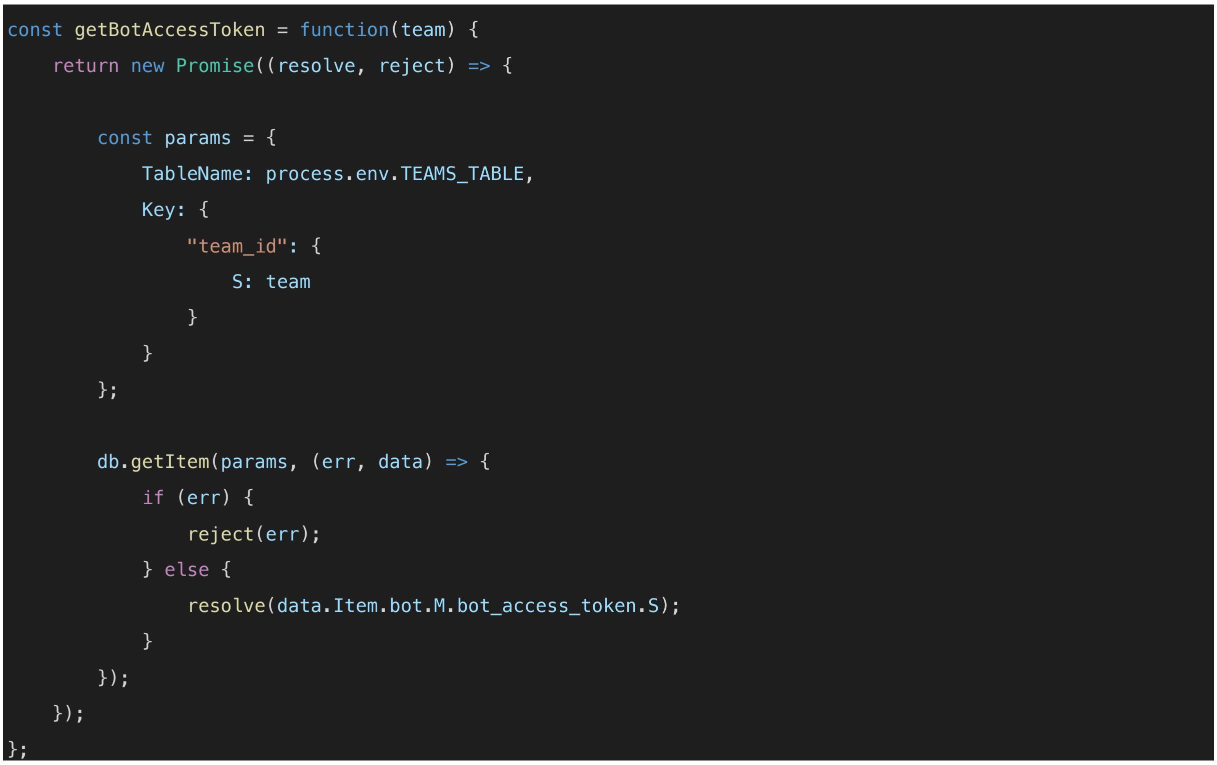
A Cloud Guru

- Open **handler.js** and replace the top section (just below **'use strict';**) with the following:

```
const https = require('https');
const fs = require('fs');
const aws = require('aws-sdk');
const qs = require('querystring');
const exec = require('child_process').exec;

const s3 = new aws.S3();
const db = new aws.DynamoDB();
```

Then, add this new function:



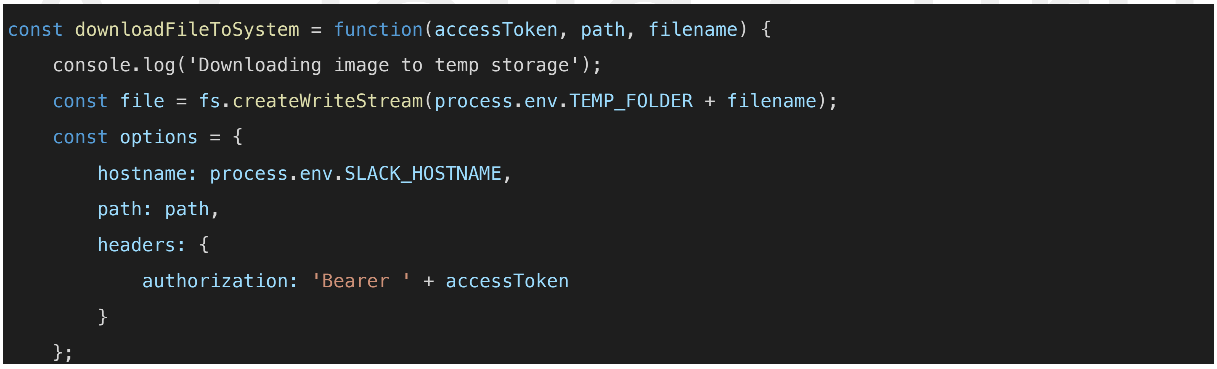
```
const getBotAccessToken = function(team) {
    return new Promise((resolve, reject) => {

        const params = {
            TableName: process.env.TEAMS_TABLE,
            Key: {
                "team_id": {
                    S: team
                }
            }
        };

        db.getItem(params, (err, data) => {
            if (err) {
                reject(err);
            } else {
                resolve(data.Item.bot.M.bot_access_token.S);
            }
        });
    });
};
```

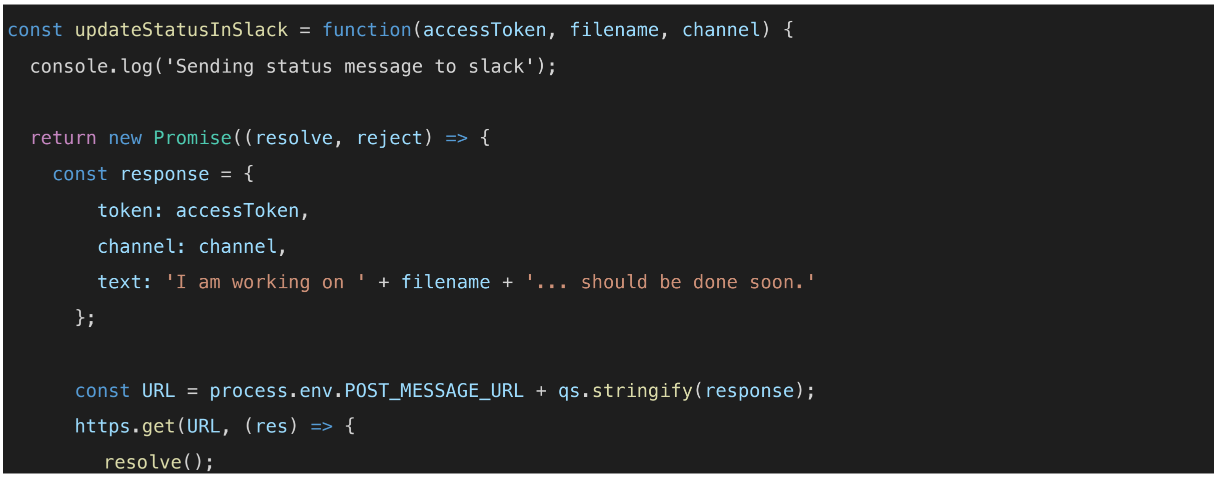
Change the start of the **downloadFileToSystem** function as follows (note the **accessToken**)

A Cloud Guru



```
const downloadFileToSystem = function(accessToken, path, filename) {
    console.log('Downloading image to temp storage');
    const file = fs.createWriteStream(process.env.TEMP_FOLDER + filename);
    const options = {
        hostname: process.env.SLACK_HOSTNAME,
        path: path,
        headers: {
            authorization: 'Bearer ' + accessToken
        }
    };
```

Replace the **updateStatusInSlack** function as follows (note the **accessToken**)



```
const updateStatusInSlack = function(accessToken, filename, channel) {
  console.log('Sending status message to slack');

  return new Promise((resolve, reject) => {
    const response = {
        token: accessToken,
        channel: channel,
        text: 'I am working on ' + filename + '... should be done soon.'
      };

      const URL = process.env.POST_MESSAGE_URL + qs.stringify(response);
      https.get(URL, (res) => {
         resolve();
      })
  });
};
```

Replace the top two lines of the **uploadToBucket** function with the following:

```
const uploadToBucket = function(filename) {
  console.log('Uploading image to S3');

  const bodystream = fs.createReadStream(process.env.TEMP_FOLDER + filename);
```

- Change the **endpoint** function to

```
module.exports.endpoint = (event, context, callback) => {
  console.log('Received event', event);

  const request = JSON.parse(event.body);

  if (request.event.type && request.event.type === 'message' &&
      request.event.subtype && request.event.subtype === 'file_share') {

      console.log('Processing uploaded file');

      const path = request.event.file.url_private_download;
      const filename = request.event.file.name;
      const channel = request.event.channel;
      let accessToken = '';

      getBotAccessToken(request.team_id)
          .then((token) => {
                     accessToken = token;
                     return downloadFileToSystem(accessToken, path, filename);
          })
          .then(() => uploadToBucket(filename))
          .then(() => updateStatusInSlack(accessToken, filename, channel))
          .then(() => {
              console.log('Returning result')
              callback(null, { statusCode: 200 })
          })
          .catch ((err) => {
              console.log('Error', err);
              callback(null, { statusCode: 500 })
          });

       return;
   }

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

- In **serverless.yml** remove **BOT_ACCESS_TOKEN** from the **hello** function.
- Redeploy your bot by typing **serverless deploy** from the terminal and upload a new image in Slack. Everything should work as before.

ADVANCED QUESTIONS

1. You still have Slack-specific variables encoded as environment variables for certain functions. Modify those functions to use DynamoDB rather than using environment variables.

A Cloud Guru