
MLH localhost
Workshop
Build and Deploy a Full Stack Web App with AWS
Presented by
aws educate
MLH
MAJOR LEAGUE HACKING

1

Using your Web Browser,
Open this URL:

http://mlhlocal.host/lhd-resources

2

Click on the workshop you're attending, and find:

- Setup Instructions
- The Code Samples
- A demo project
- A Workshop FAQ
- These Workshop Slides
- More Learning Resources

Our Mission is to Empower Hackers.

65,000+
HACKERS

12,000+
PROJECTS CREATED

400+
CITIES

We hope you learn something awesome today!
Find more resources: http://mlh.io/

Table of Contents

1. Full Stack & Application Introduction
2. Setting up our Dev Environment
3. Creating an AWS Account
4. Creating an IAM User and Group
5. Building and Running the Application
6. Connecting to the Eventbrite API
7. Deploying to AWS Elastic Beanstalk
8. Review & Next Steps

What will you learn today?

1. How to build a full-stack web app with **Flask**, a Python web framework
2. How to deploy apps to **AWS Elastic Beanstalk** using the `awsebcli`
3. How to configure our **AWS Elastic Beanstalk**

What is a web application anyway?

A **web app** is a program that runs on a remote **server**.

A web app tends to establish a *two way relationship* with the **client**. This means the app will update depending on what users clicked or typed.

A **website** will return one way static data from the server.

What does full stack mean?

A **full stack** application is one that includes **models**, **views**, and **controllers**

Models - manages the data of the application

Views - the visual page that users see on your app

Controllers - the logic layer that controls user input and output

What types of files are in a full stack app?

.css - style files that design the elements
.html - structure files that shape the page
.js - controls user behaviors like clicking and typing
.py - python files that call our database and manage logic

What is AWS?

Amazon Web Services (AWS) is a collection of **cloud computing services**.

These services abstract a lot of the work you would need to do **hosting**, **provisioning**, and **managing** your code.

Both **NASA** and **Netflix** are customers of AWS!

Who uses AWS?

AWS is generally used by people called **Developer Operations (DevOps)** or **System Administrators**. These people help you set up your AWS stack, and provision servers for you, manage upgrades, etc.

Many **developers** trust AWS as well because it has **100% uptime** and works **at scale**, which means your website can have many visitors online at the same time

Working Demo

Today we'll be building an **Event Suggestion App**! In the app you'll be able to search events in your city using the Eventbrite API

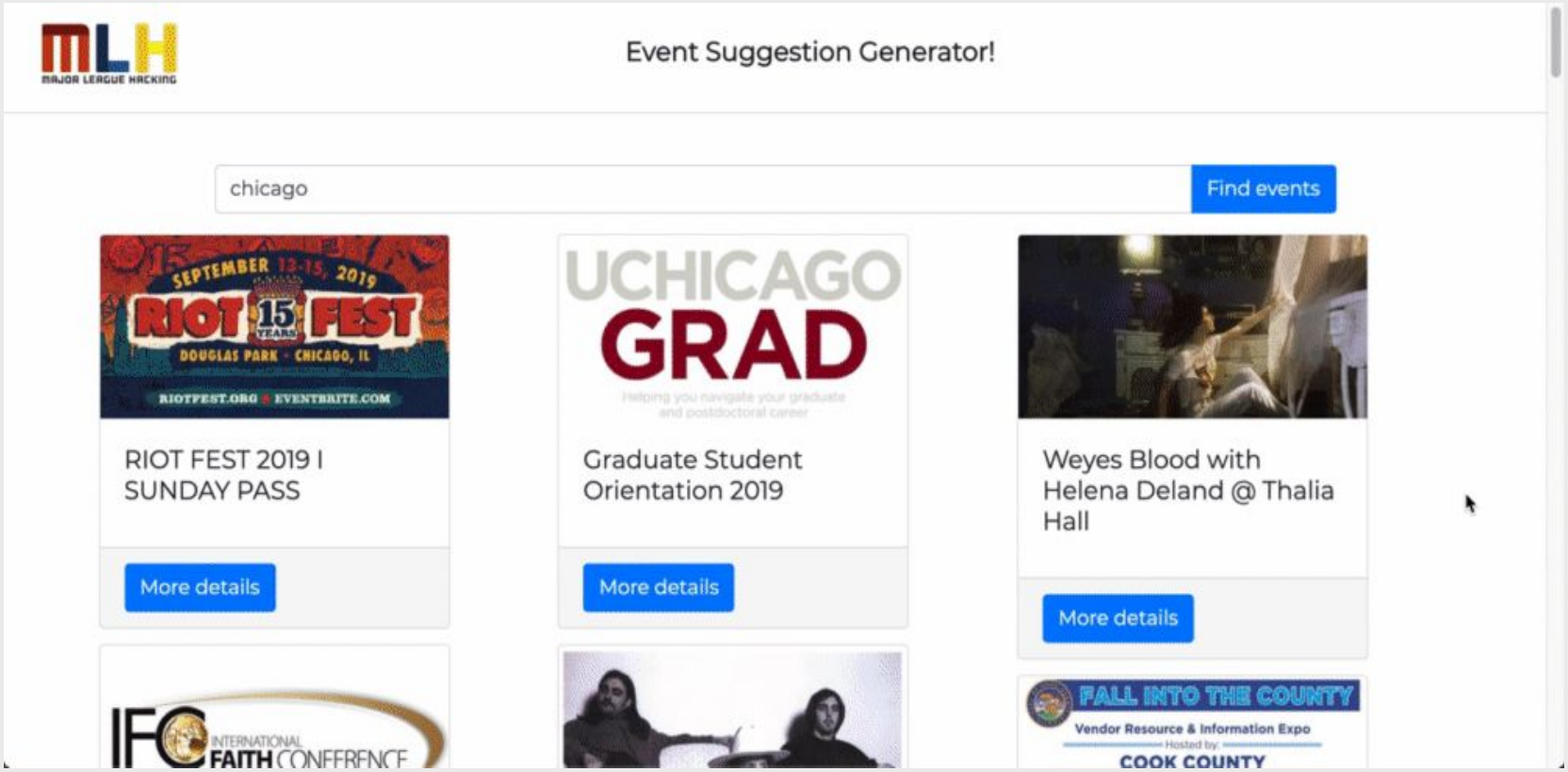


Table of Contents

1. Full Stack & Application Introduction
2. Setting up our Dev Environment
3. Creating an AWS Account
4. Creating an IAM User and Group
5. Building and Running the Application
6. Connecting to the Eventbrite API
7. Deploying to AWS Elastic Beanstalk
8. Review & Next Steps

Setting Up our Dev Environment

Before we jump into the project we need to make sure we need to have a place to code!

Development vs. Production Environment

In this worksop, we will be creating and working in a **Development** environment which is the environment that's on your computer.

Here is where you'll do all of your code updates. The development environment is usually configured differently from the environment that users work in.

It'll be connected to some local database or a dummy database so that you can write your code without messing up the real data. The development environment is also known as you **Local** environment.

Development vs. Production Environment

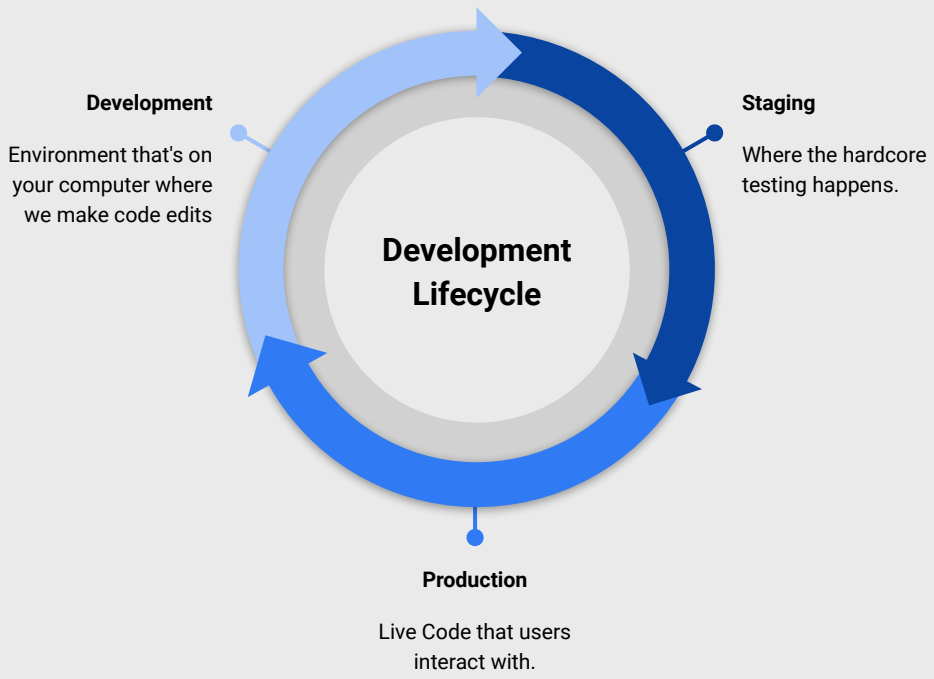
Every time we talk about making our project live, we are talking about the production environment. This is where users access the final code after all of the updates and testing. Of all the environments, this one is the most important.

Once you're in production, any bugs or errors that remain will be found by a user and you can only hope it's something minor.

Development Lifecycle

Working in multiple environments is standard for any developer. The development life cycle even includes a middle step, a **stage environment**, where the hardcore testing happens and is similar to the production environment as it can be.

We won't be creating a staging environment for this workshop since we will be pushing our code straight from **development** to **production**. But getting used to the development life cycle will be great practice.



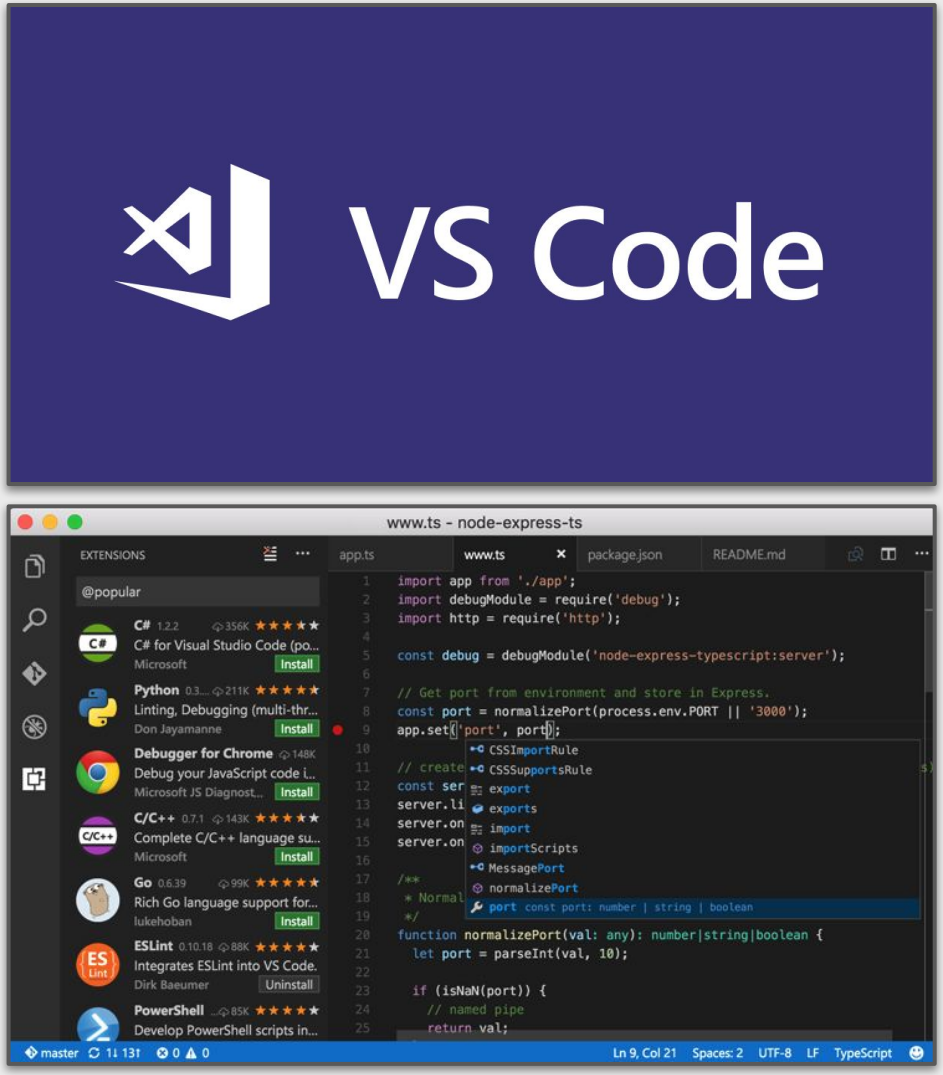
What tools do we need to get started?

For this workshop, we'll be using:

- **Visual Studio Code** - a text editor that specializes in code, also has a Terminal built in
- **Terminal** - Where we run commands for our computers to execute
- **Python** - a popular programming language
- **Pip** - a code package manager, allows us to use code we find on the internet
- **MySQL** - a database we can run on our computers (For our Local Environment)
- **AWS Account** - a reliable, scalable, and inexpensive cloud computing service.

Install Visual Studio Code

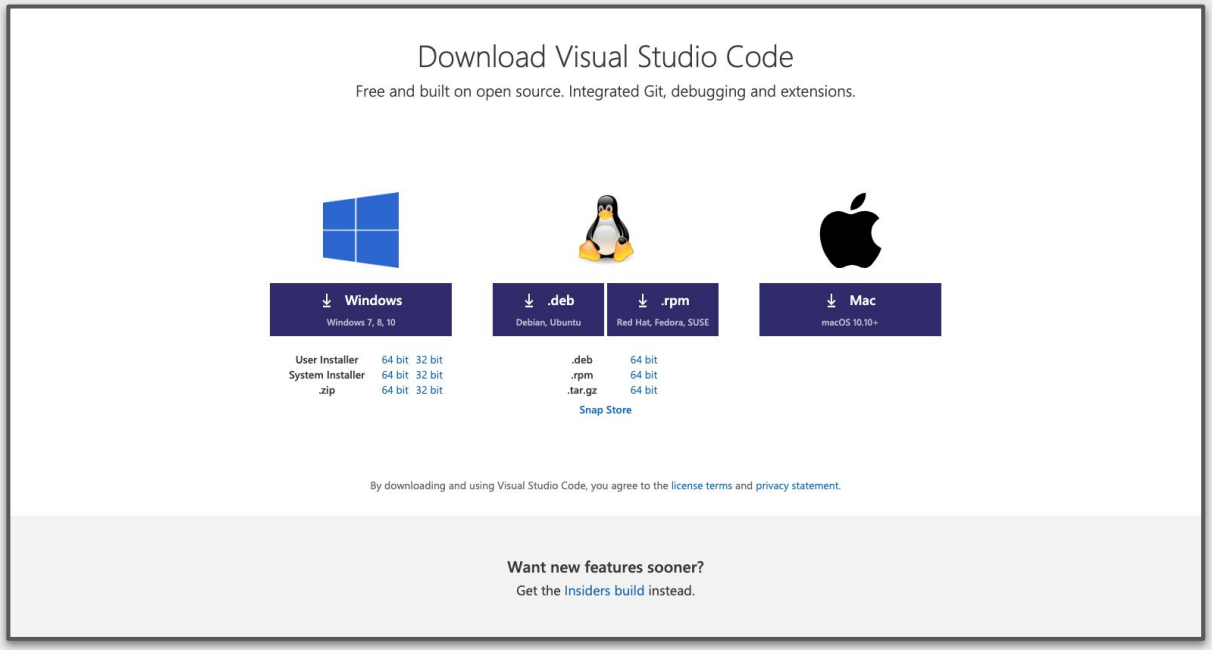
In order to edit, write and deploy the code for this workshop, we will need an editor. We will be using **Visual Studio Code, or VS Code**, a lightweight but powerful code editor which runs on your desktop and is available for Windows, macOS and Linux.



Install Visual Studio Code

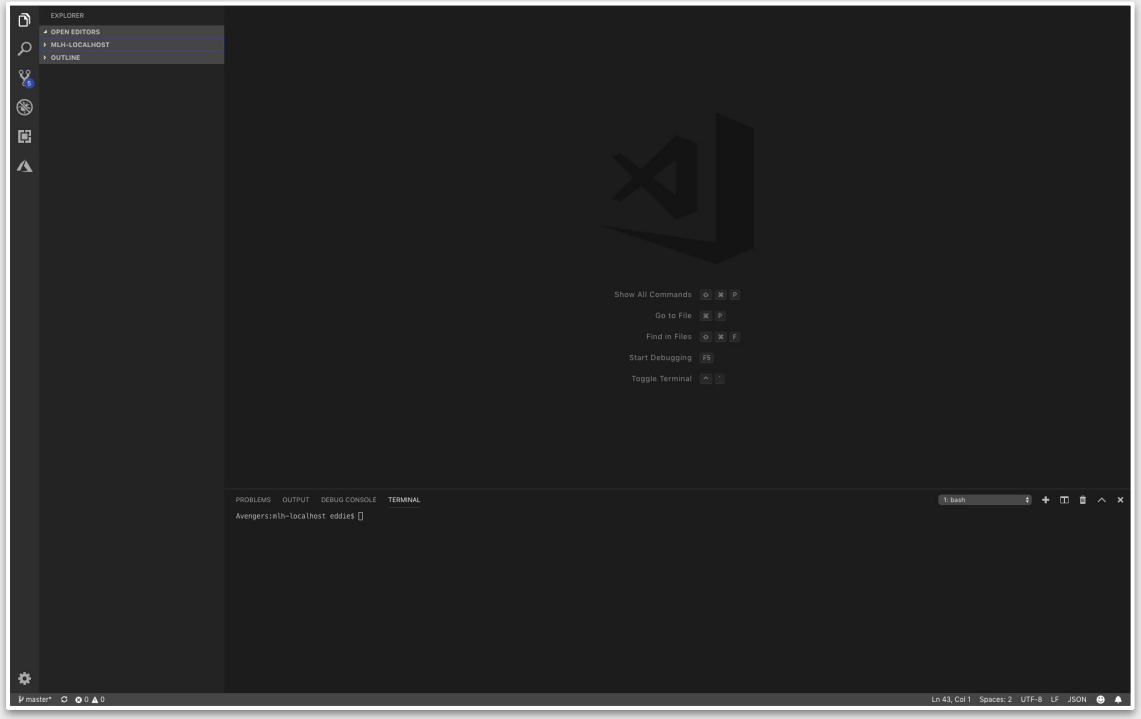
To Download VSCode, go to the following link and select the download for your operating system.

mlhlocal.host/vscode-download



Install Visual Studio Code

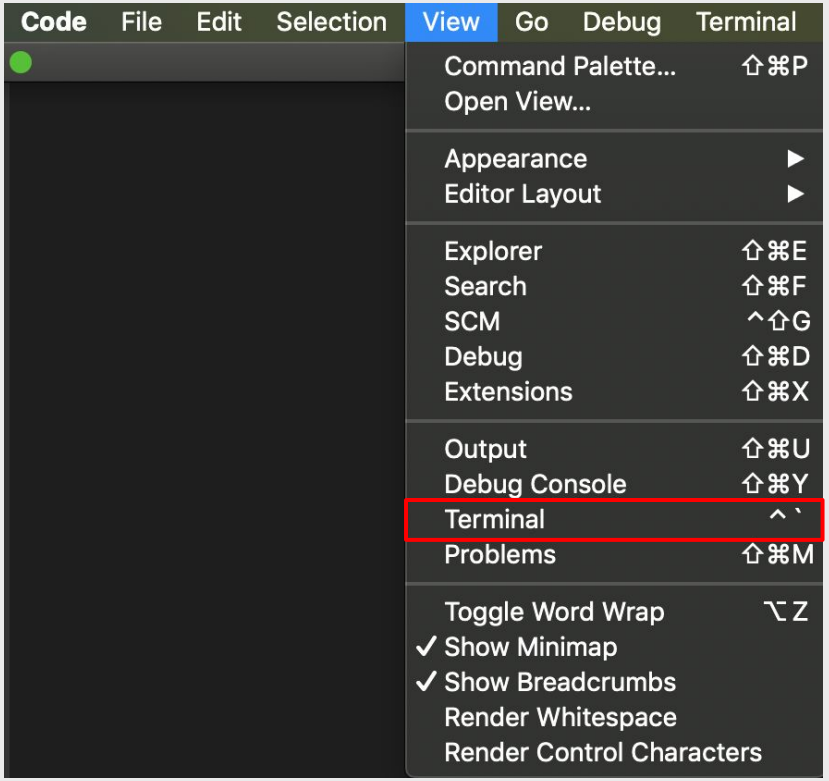
If downloaded successfully, open the application and you should see the following window. Congratulations you downloaded VS Code!



Open VS Code Terminal

We will be doing a fair bit of work using the **terminal.** Lucky for us their is a terminal built in VS Code!

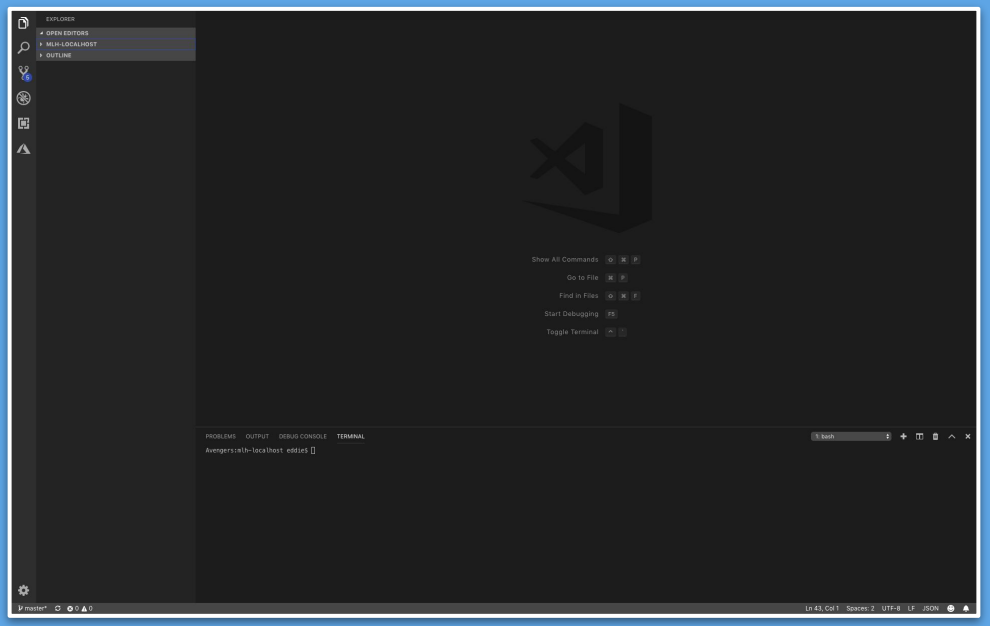
To open the terminal go into **View > Terminal** seen below:



Key Terms

Terminal: Known as the command line or a Terminal emulator, provides an efficient interface to access the computer better than any graphical interface.

Great Work!



We just setup:

- A popular code editor, Visual Studio Code
- Opened our Terminal in Visual Studio Code

Next, we'll:

- Set up Python so we can deploy our application locally

What is Python?

Python is a popular general purpose programming language used in many places

Python is a **interpreted language** which means that it gets evaluated at runtime instead of compiling

Python is popular for **Data Science** and **Machine Learning** applications

Python has 2 major versions currently in use, Python 2 and Python 3. We will be using **Python 3+** for this workshop

Check for Python 3

Let's check if we have Python installed. **Open Visual Studio Code** and open the **Terminal (View > Terminal)**

Type this command

If you have Python 3 installed, you should see a **shell** like this pop up. To exit the shell, type **exit()**

```
→  mlh-localhost-build-and-deploy-aws git:(student) python3
Python 3.7.4 (default, Sep  7 2019, 18:27:02)
[Clang 10.0.1 (clang-1001.0.46.4)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

If you get an error about Python 3 not being installed, don't worry! We'll download Python in the next slide

Install Python 3

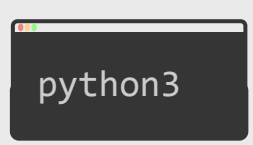
Installing Python **varies from system to system**, so it's best to follow the instructions from the Python website! Download Python using the following link

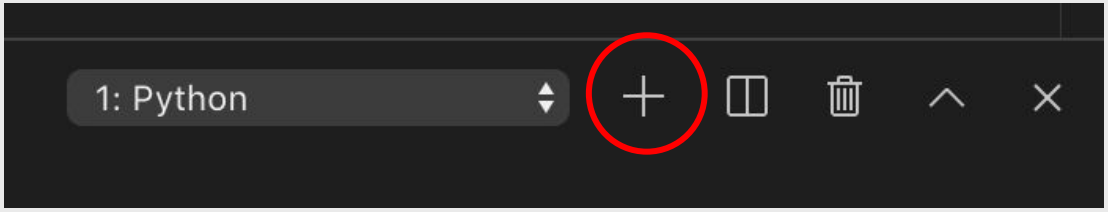
mlhlocal.host/get-python

Install Python 3

After installing **Python 3,** check for it again in VSC. You many need to create a new Terminal shell. You can do that by clicking the **+** button.

Type this command and test if Python 3 is installed successfully





Great work! Let’s recap what we did so far

- Checked for Python
- Installed Python

Next we’ll:

- Install pip, the Python package manager

What is pip?

Pip is a package installer for Python. In this workshop, we will be using a wrapper for pip called `pipenv`

Pipenv (needs to be downloaded), is a wrapper for pip. It manages our **virtual environment**, and makes a Pipfile (a requirements file of the packages the project uses)

A **virtual environment (venv)** is an environment that runs locally on a project, and won't affect your *global* installation

Check for pip

The Python we installed in the prior step should automatically include **pip**.

Check for it with this command


```
Avengers:~ eddie$ pip --version
pip 19.0.3 from /Library/Python/2.7/site-packages/pip (python 2.7)
```

Pip is installed!

Great work! Let’s recap what we did so far

- Learned about Pip, a Python package manager
- Checked for Pip

Next we’ll:

- Install MySQL, the database we’ll be using for the project.
- MySQL will be used for local development, so we can see how our data moves through the app

Setting up MySQL

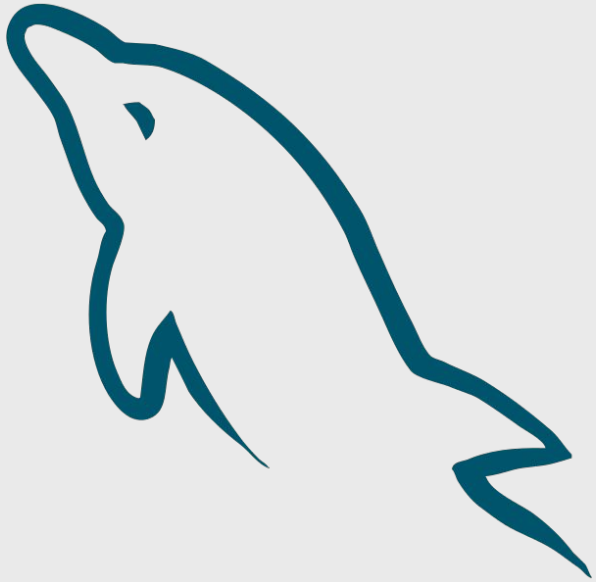
What is MySQL?

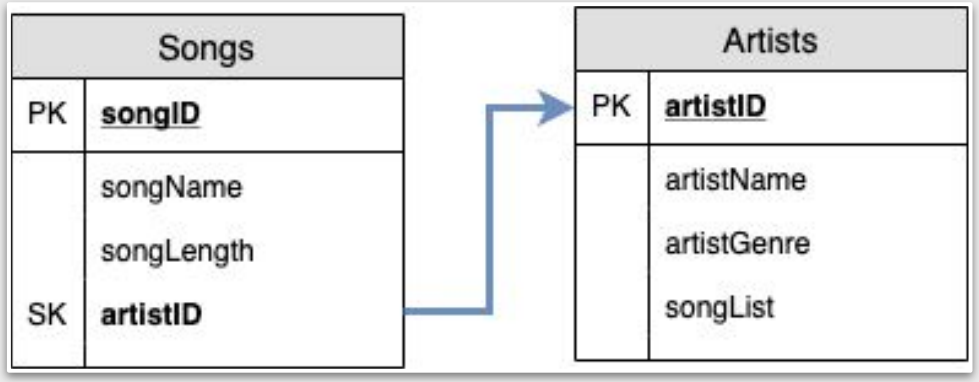
MySQL is an open source **relational database**

MySQL uses a **query language**, a specific language that makes **reading and writing tabular data easy**

Relational databases are databases that have **tables** that refer to each other

For example if I have a table for **songs** and a table for **artists** I can link them as follows:





Installing MySQL

Similar to Python, installing MySQL varies from **platform to platform**.
The installation process may be complex, so ask your facilitator for help!

MySQL can be installed for **Windows**

mlhlocal.host/mysql-windows

MySQL can be installed for **Mac**

mlhlocal.host/mysql-mac

Installing MySQL (Mac + Linux)

For Mac Users: Once we have MySQL installed, we need to add an **alias** so we can use it from the command line. **Copy and paste** the command below

Mac:

```
alias mysql=/usr/local/mysql/bin/mysql
echo "alias mysql=/usr/local/mysql/bin/mysql"
>> ~/.profile
```

Running MySQL

Mac

```
mysql -u root -p

(hit enter when password prompt pops up)
```

Windows

```
C:\> "C:\Program Files\MySQL\MySQL Server
8.0\bin\mysqld" --console
```

Generating our mysql url for Dev environment

Next, we create a database with the **CREATE DATABASE** command. You can run **SHOW DATABASES** in your mysql shell to see current databases available.

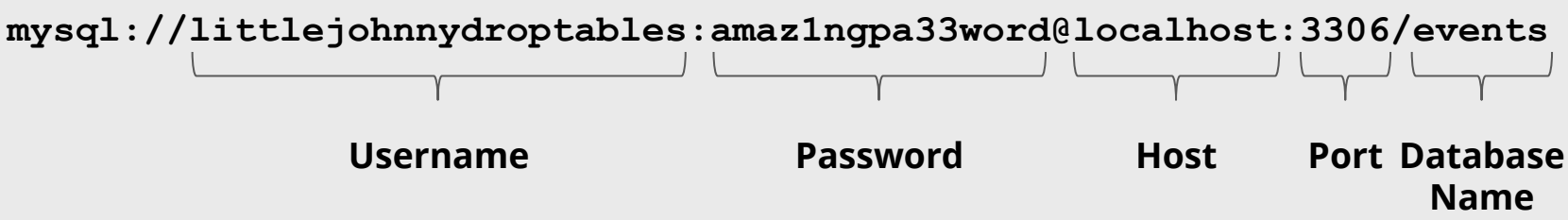
Here we are creating a database called *events*

```
mysql> CREATE DATABASE events;
```

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| event_fvorites     |
| events             |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
6 rows in set (0.00 sec)

mysql>
```

Generating our mysql url for Dev development

From here we use our **username, password and database name** to create a **mysql url.** The URL is formatted like so:



Using the above format, create your Database URL and save it somewhere safe! We will be using it later. Exit the MySQL shell by entering `exit()`

```
DATABASE_URL=mysql://littlejohnnydroptables:amaz1ngpa33word@
localhost:3306/events
```

Great work! Let’s recap what we did so far

- Learned about MySQL, a relational database
- Installed MySQL
- Created a database
- Created a user

Next we’ll:

- Get our AWS account set up so that we can deploy our app to production

Table of Contents

1. Full Stack & Application Introduction
2. Setting up our Dev Environment
3. Creating an AWS Account
4. Creating an IAM User and Group
5. Building and Running the Application
6. Connecting to the Eventbrite API
7. Deploying to AWS Elastic Beanstalk
8. Review & Next Steps

Create an AWS Account

There are two ways to create an AWS account.

aws educate

If you're a student you can sign up for **AWS Educate**. AWS Educate is an Amazon program that provides students comprehensive resources for building skills in cloud technology. Join AWS Educate and you will receive **$100** in AWS Promotional Credits to use in this workshop!

mlhlocal.host/aws-student-signup

aws

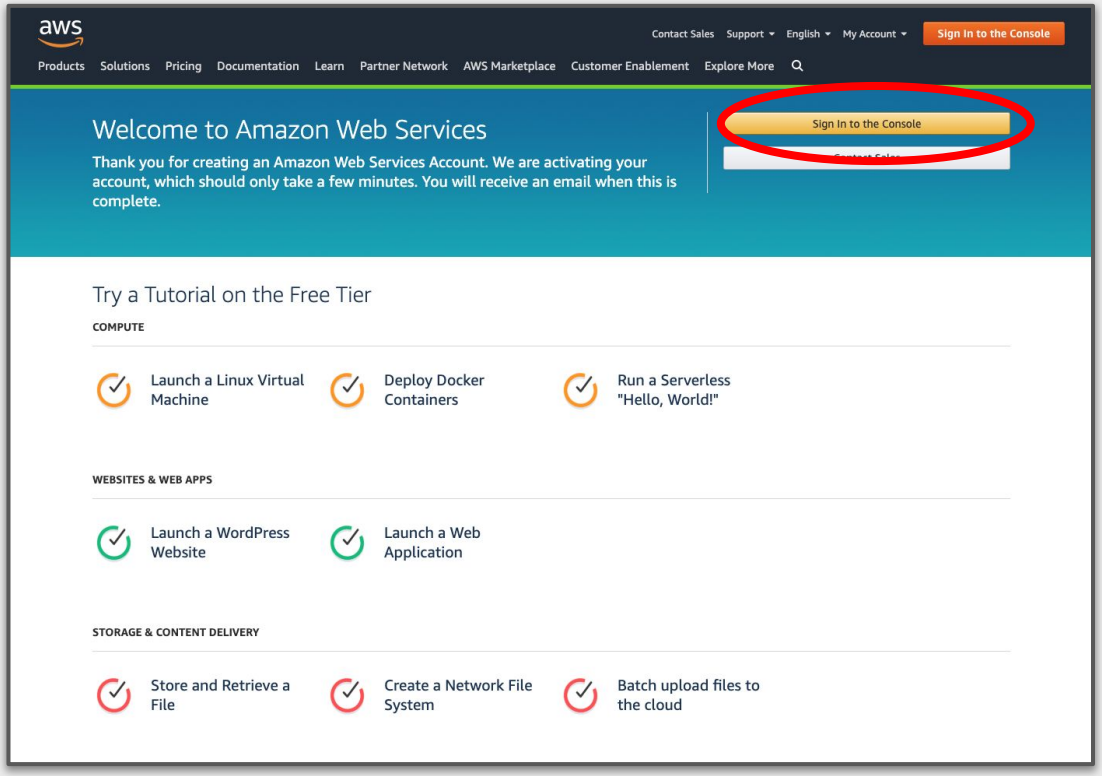
If you're a non-student you can still sign up for an **AWS** account with 12 months of free tier access. However, a Credit Card is required to complete your registration.

mlhlocal.host/aws-signup

Complete your AWS Registration!

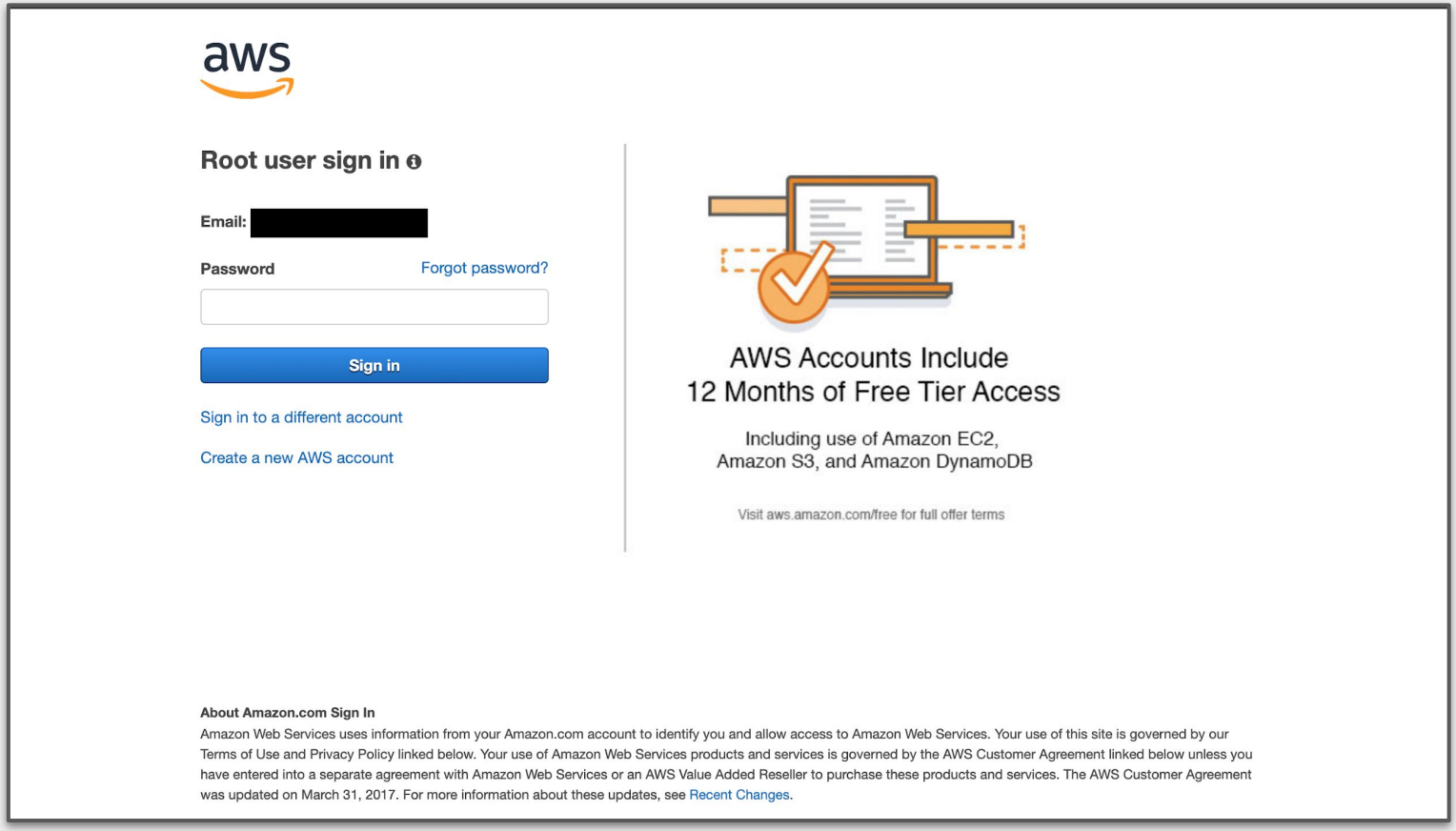
Sign in to your AWS Account

After you complete your sign up, we need to sign in to the account we created. Log in with the **Sign in to the Console** button



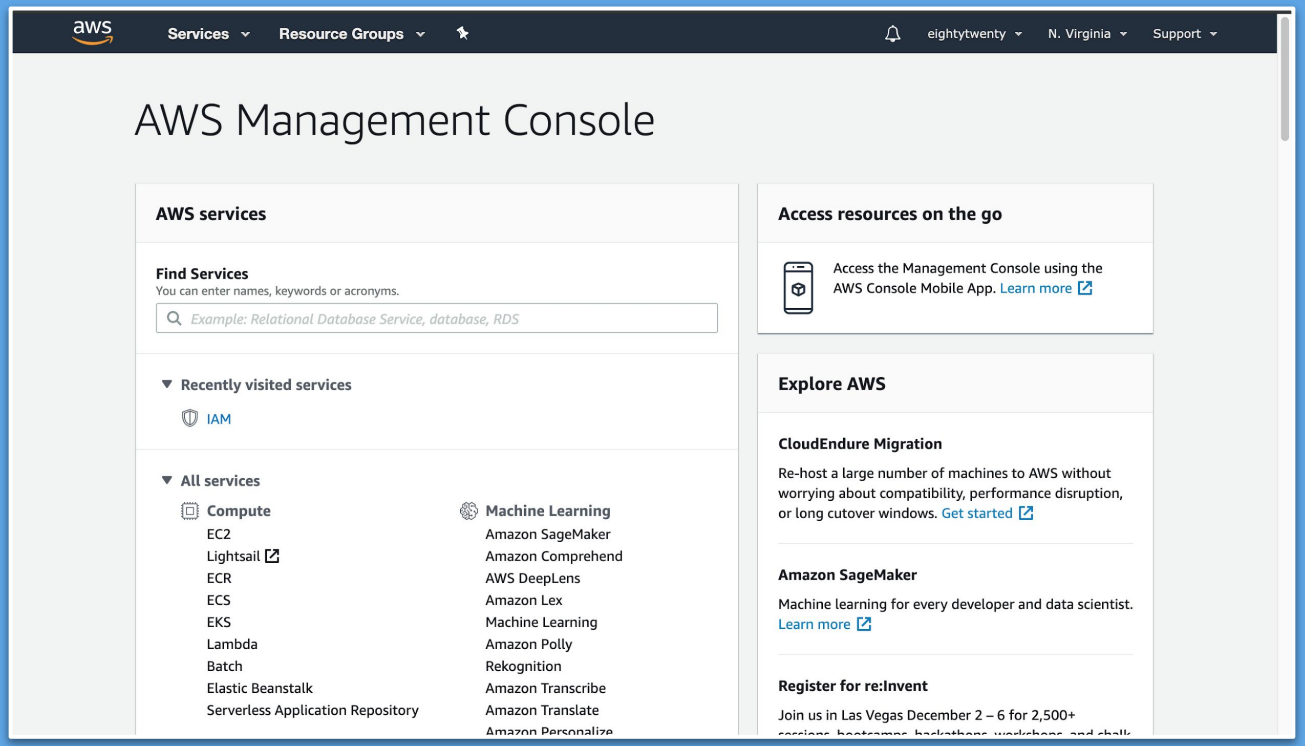
Sign in to your AWS Account

Sign in with your AWS credentials that you just made



You're In!

After you successfully log in, you should be able to see the **AWS Management Console**



Great Work!

We just setup:

- Created our AWS account
- Learned about two ways to set up an AWS account

Next up:

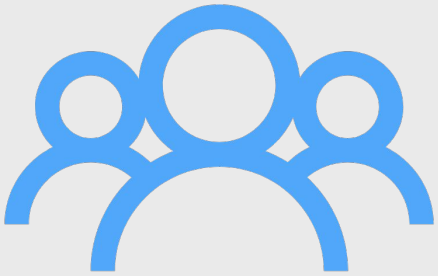
- Creating an IAM user

Table of Contents

1. Full Stack & Application Introduction
2. Setting up our Dev Environment
3. Creating an AWS Account
4. Creating an IAM User and Group
5. Building and Running the Application
6. Connecting to the Eventbrite API
7. Deploying to AWS Elastic Beanstalk
8. Review & Next Steps

Creating an IAM User

Now that we have our AWS Account, we need to create an **Identity and Access Management** (**IAM**) User.



An IAM User is an entity that you create in AWS to represent the person or application that uses it to interact with AWS.

A user in AWS consists of a **name** and **credentials**. An IAM user with **administrator** permissions is **not** the same thing as the AWS account **root** user.

Creating an IAM User

Why do I want an IAM User?

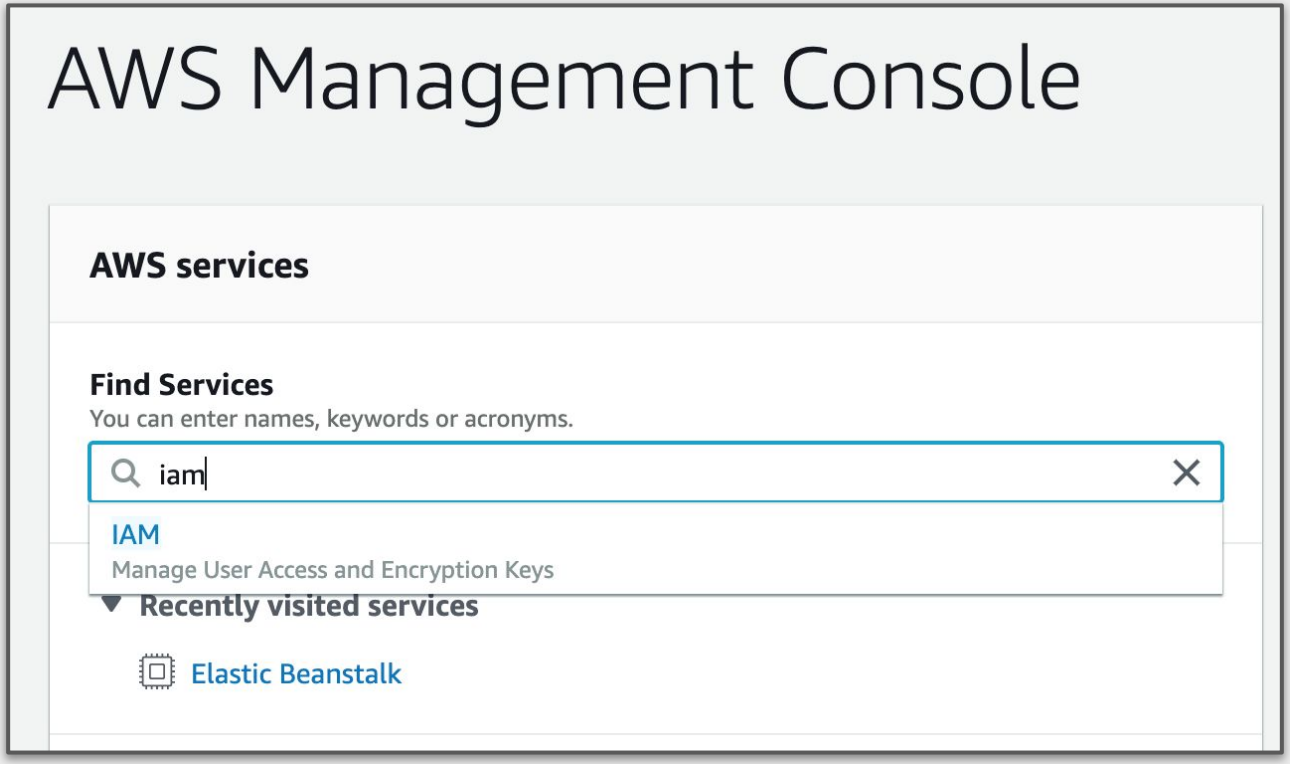
When we first created our AWS account, you are given a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the a **root user** and is accessed by signing in with the email address and password that you used to create the account.

Root User Best Practices

- We **do not** want to use this root account for day to day tasks
- It's recommend to create Individual users for anyone who needs access to your AWS account.
- By creating individual IAM users for people accessing your account, you can give each IAM user a unique set of security credentials.
- Securely lock away the root user credentials and use them to perform only a few account and service management tasks.

Creating an IAM User

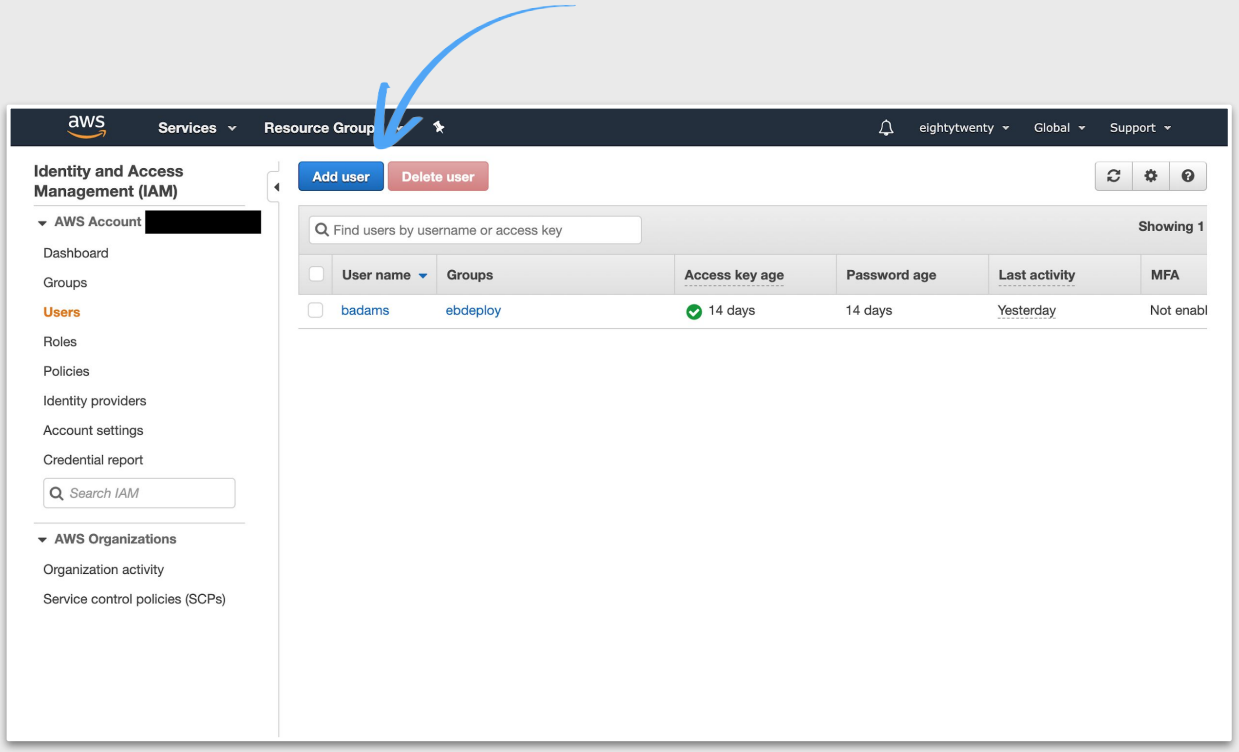
Search **IAM** in your **AWS Management Console** using the **Find Services** search box.



Creating an IAM User

In the Identity and Access Management Screen (IAM)

1. Select **Users** on the left hand side
2. Select **Add user**



Creating an IAM User

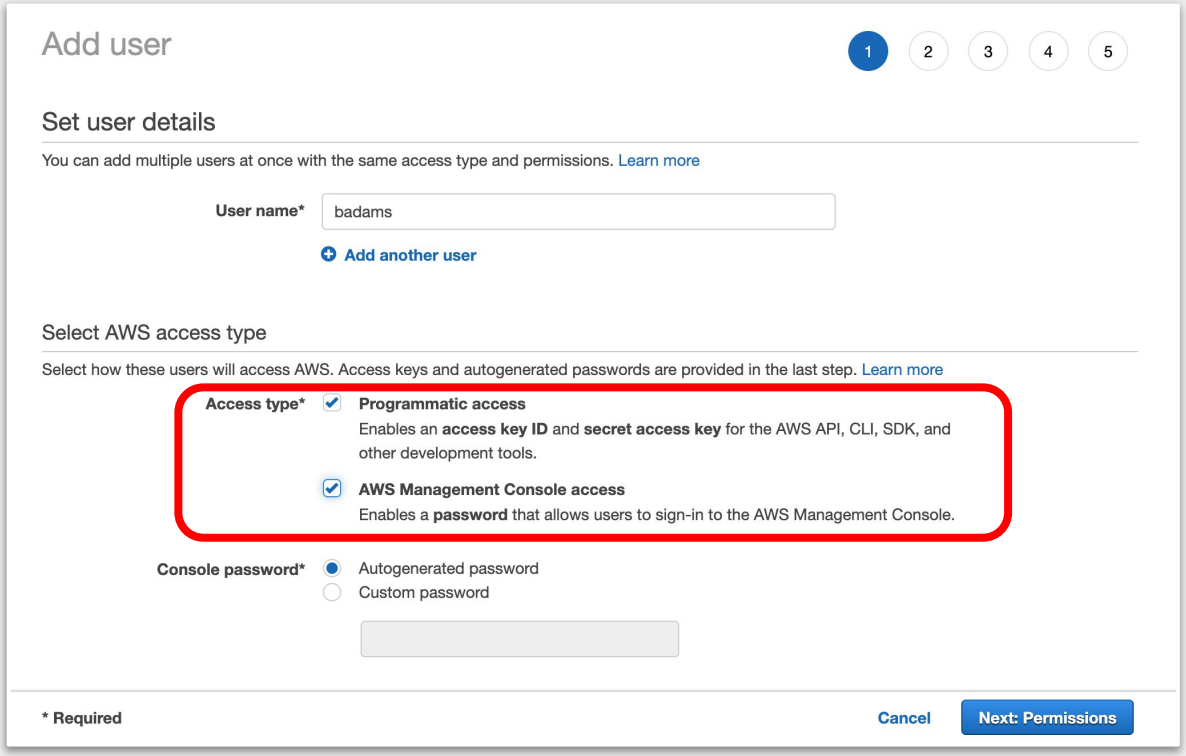
Next we will add our IAM user.

Enter a **username** of your choice.

Give the user **programmatic access** so they can log in with the **Command Line Interface**

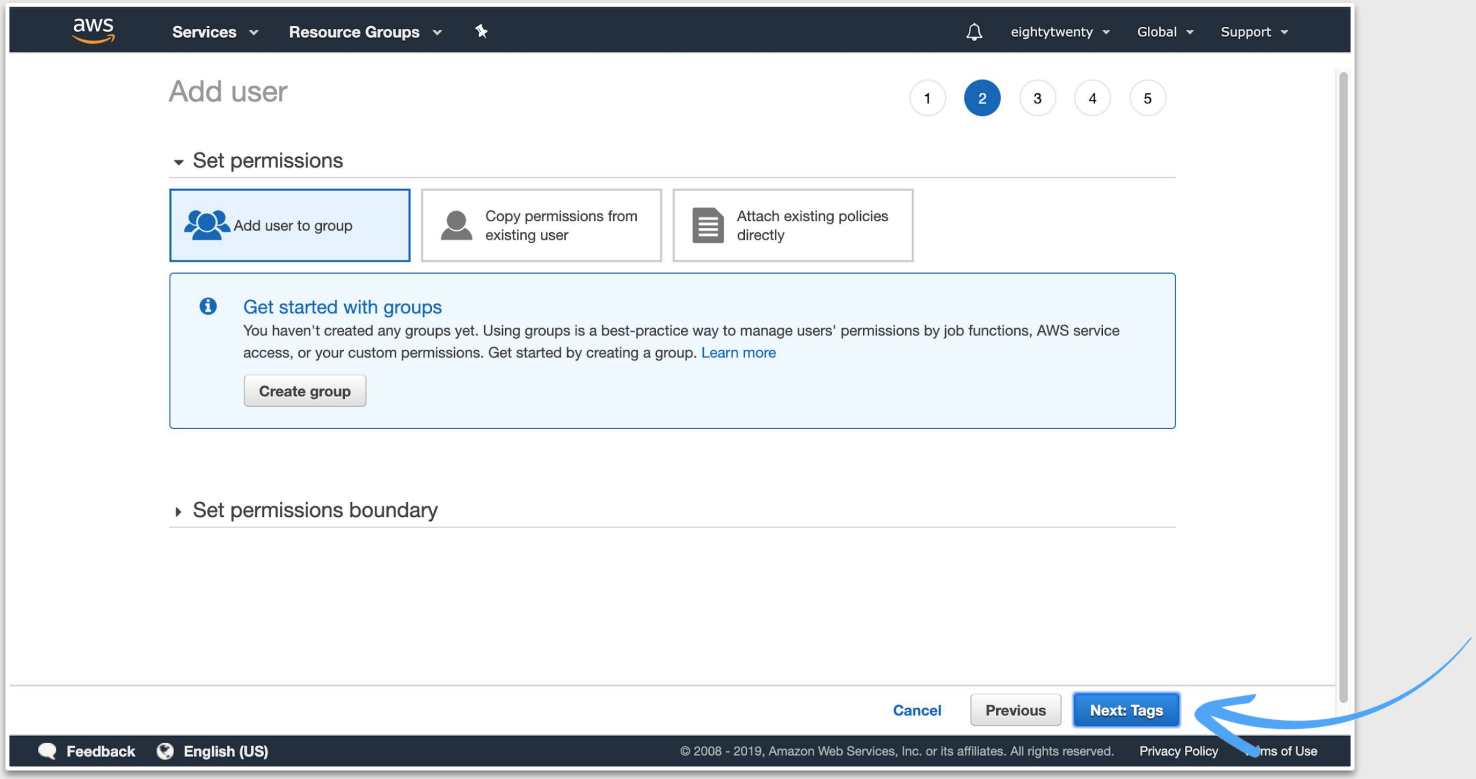
Give the user **Management Console access** so they can login the same way we created an AWS account.

You can either **auto-generate a password** for your IAM user or create your own



Creating an IAM User

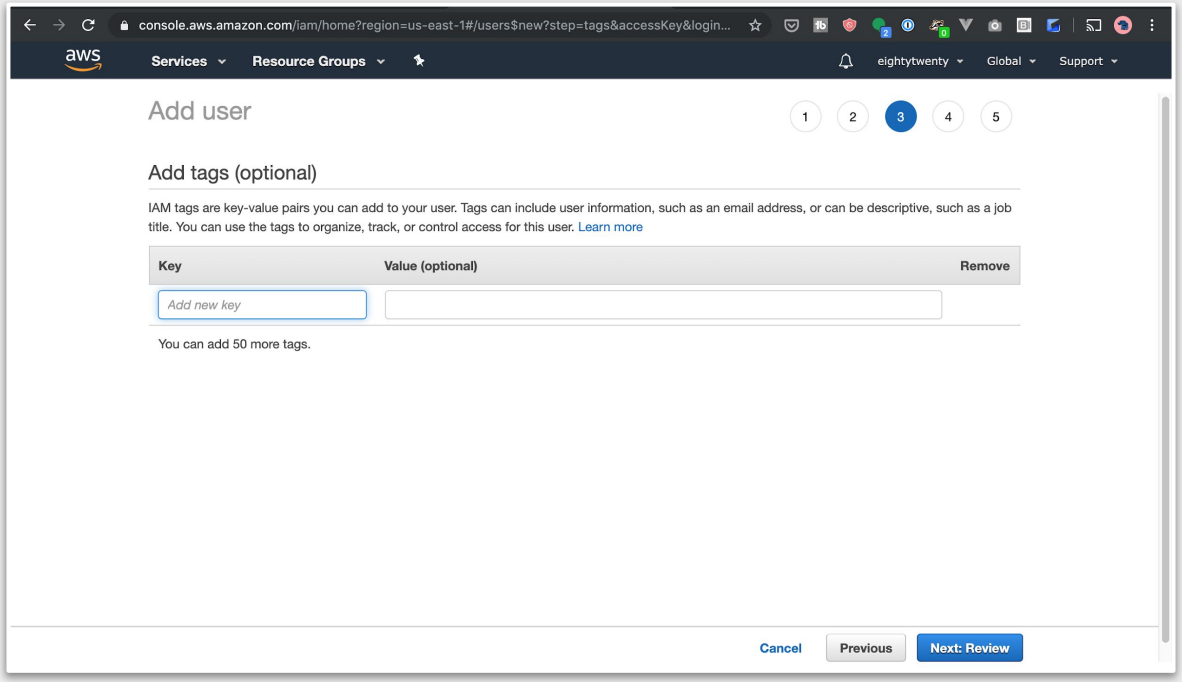
Next up we need to **set permissions** and add the user to a group.
Click **Next: Tags** *(we will be creating a group later)*



Creating an IAM User

Tags are optional ways to search users. We don't need to add a tag at this time.

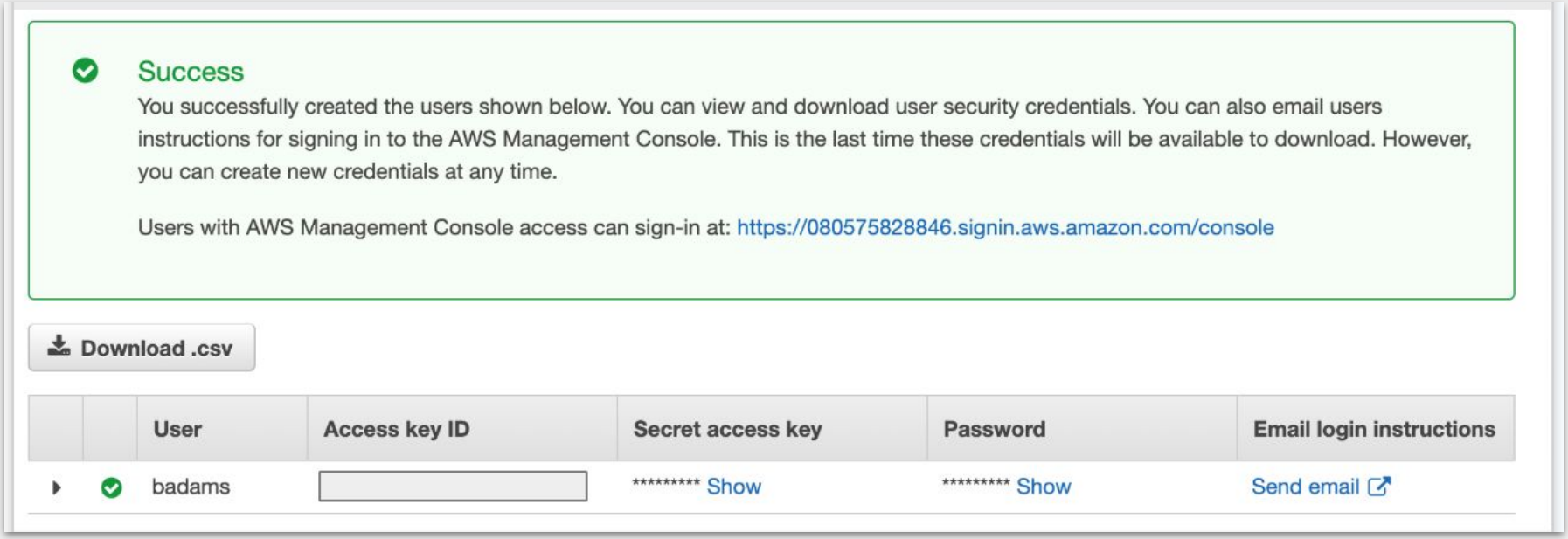
1. Hit the **Next: Review** button to review your user setup.
2. **Review your setup** on the next step. If everything looks good, hit **Next**



Creating an IAM User

After you successfully create your IAM user, you should see the following prompt. You will need to **copy your password, secret access key, and access key ID** somewhere safe and private! We will be using this information later.

NOTE: Make sure to copy your your secret access key since this will be your only opportunity to do so!



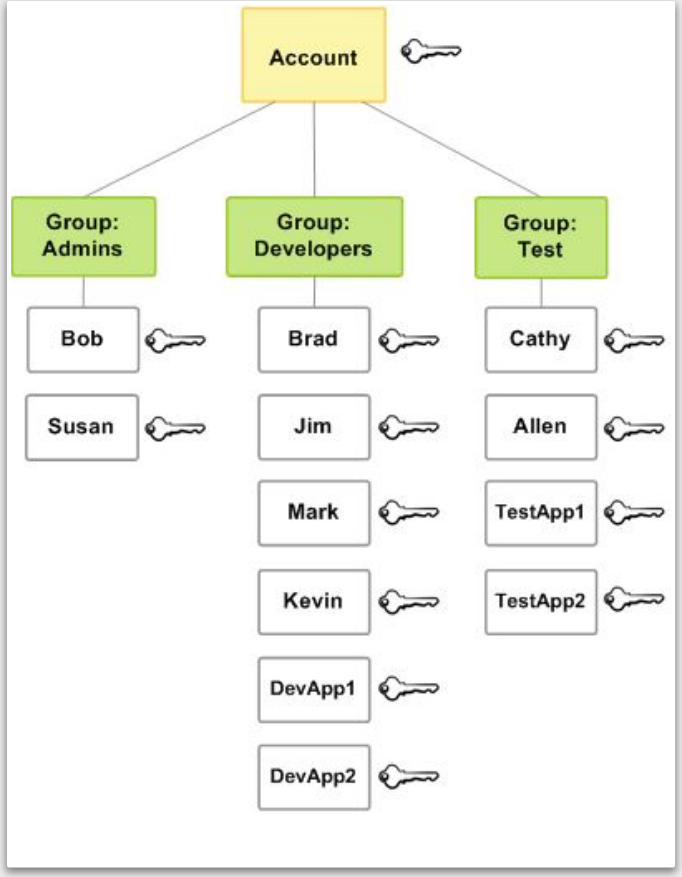
Creating a New IAM Group

The next thing we need to do is create an **IAM Group** for our user to live in.

An IAM group is a collection of IAM users. Groups allow us to assign permissions to different groups of people within an organization, like an *engineer group* or a *designer group*.

For example, you could have a group called *Admins* and give that group the types of permissions that administrators typically need.

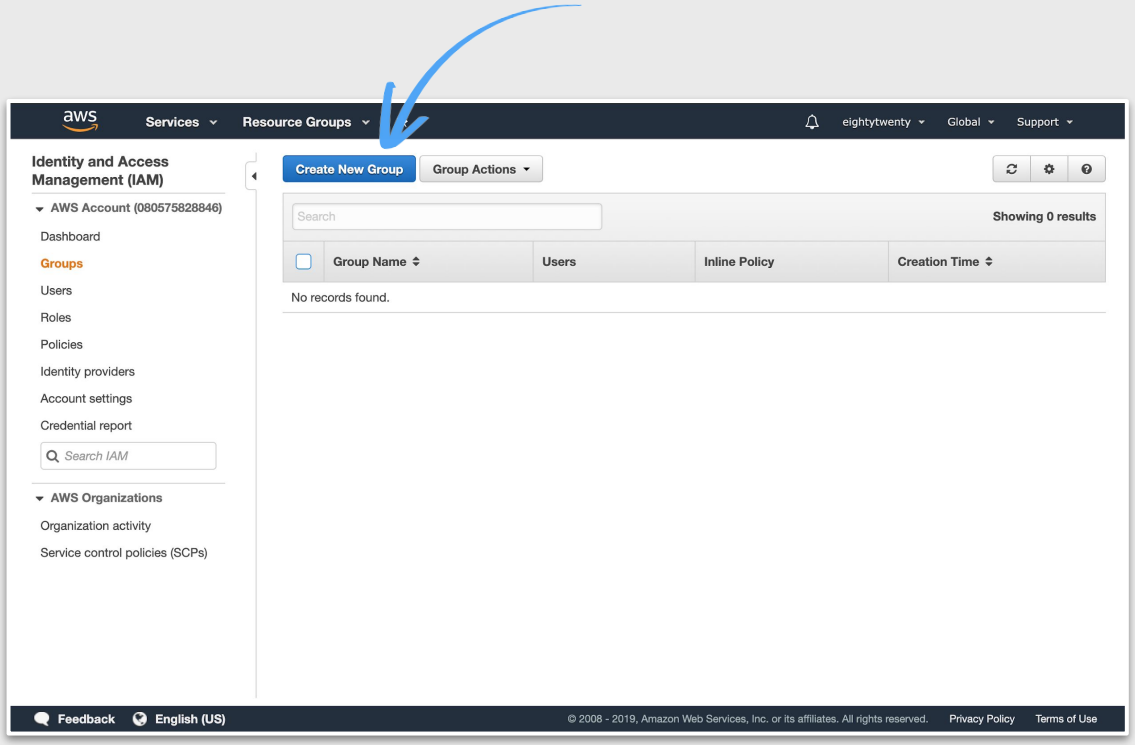
Let's add a **group** for our AWS users to join that allows them to use the **Command Line Interface** (**CLI**) with **Elastic Beanstalk** (**EB**) permissions.



Creating a New IAM Group

In the Identity and Access Management Screen (IAM)

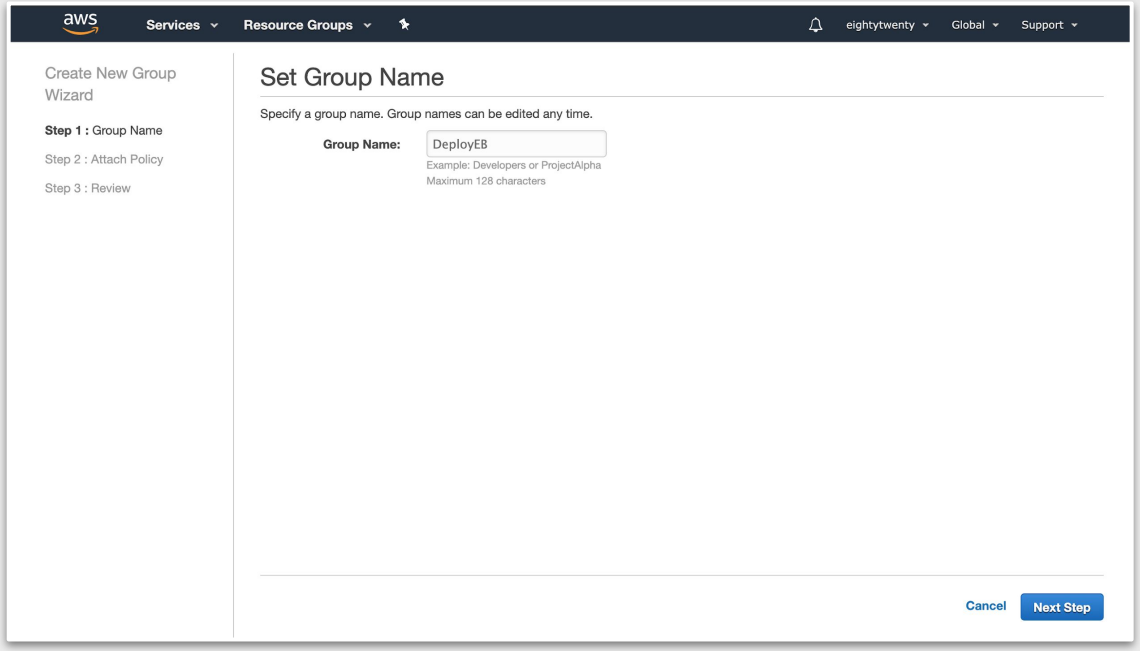
1. Select **Groups** on the left hand side.
2. Select **Create New Group**



Creating a New IAM Group

1. Set a **Group Name** this could be anything you like!

2. Hit **Next Step**



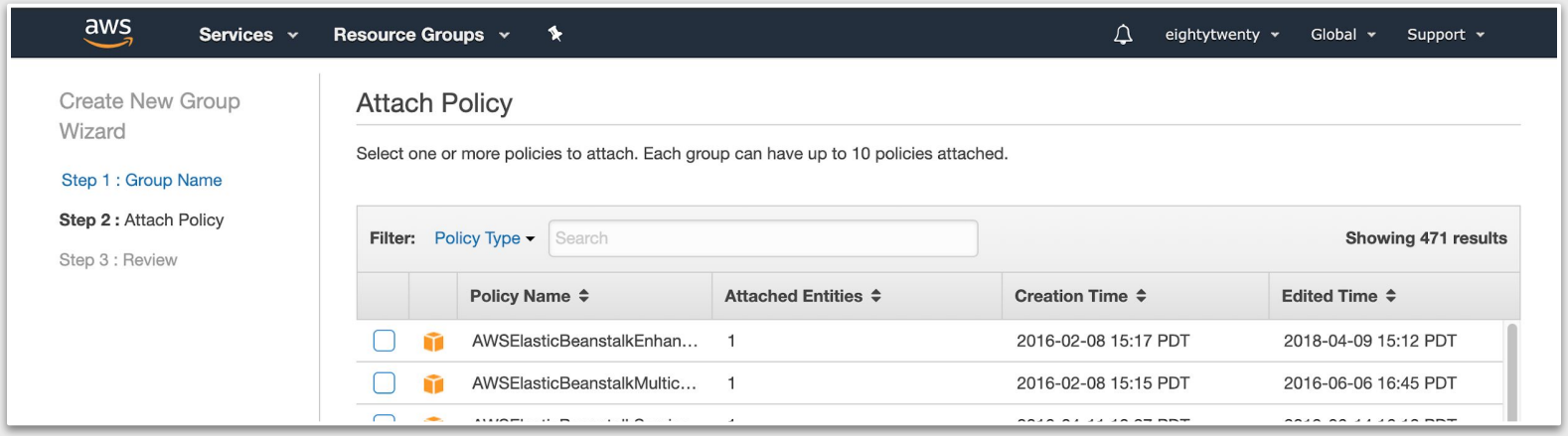
Creating a New IAM Group

Next, we'll attach **Policies** to our group.

Using the Search Bar, search for the two policies listed below:

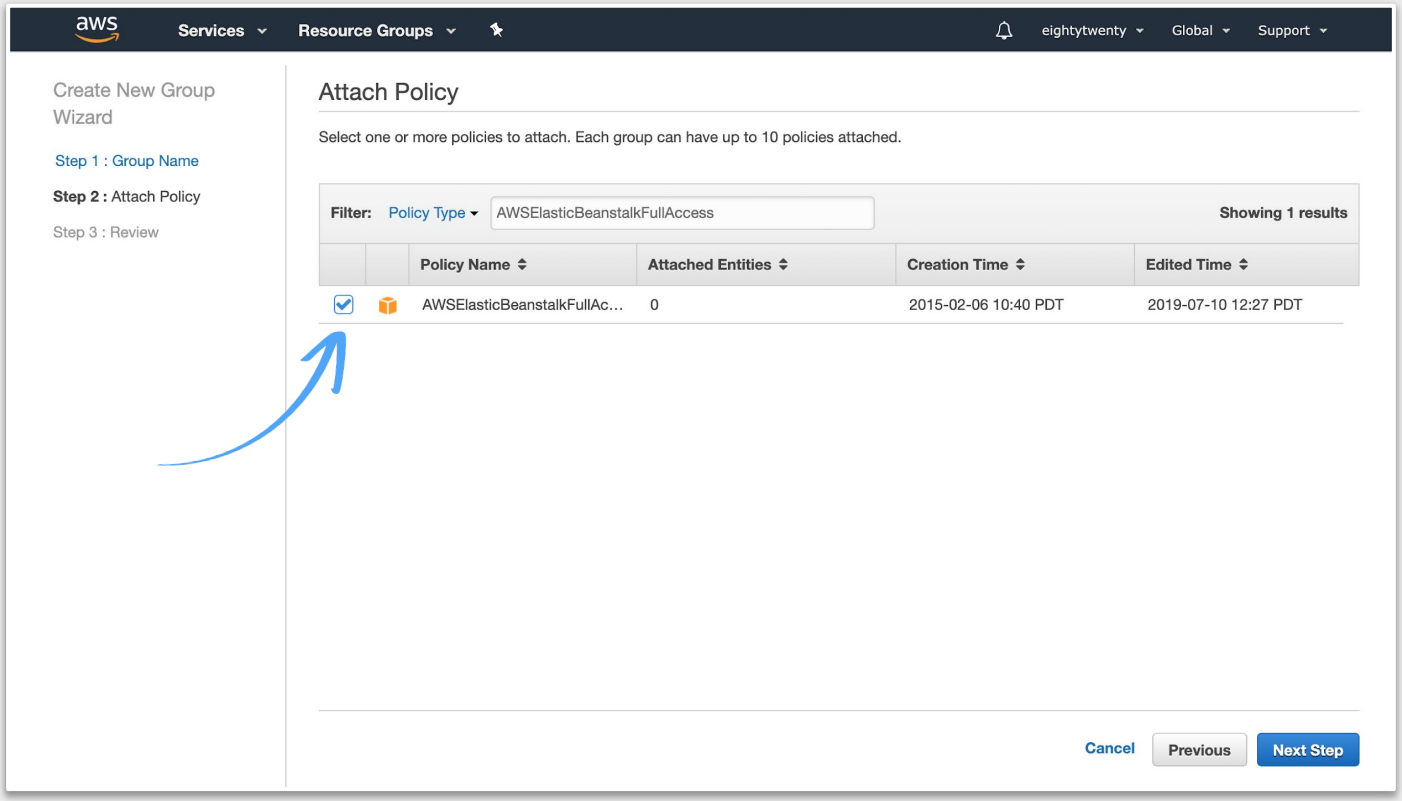
1. **AWSElasticBeanstalkFullAccess** - Allows the user to create, modify, and delete Elastic Beanstalk applications

2. **AWSElasticBeanstalkService** - Grants permissions for Elastic Beanstalk to update environments on your behalf to perform managed platform updates



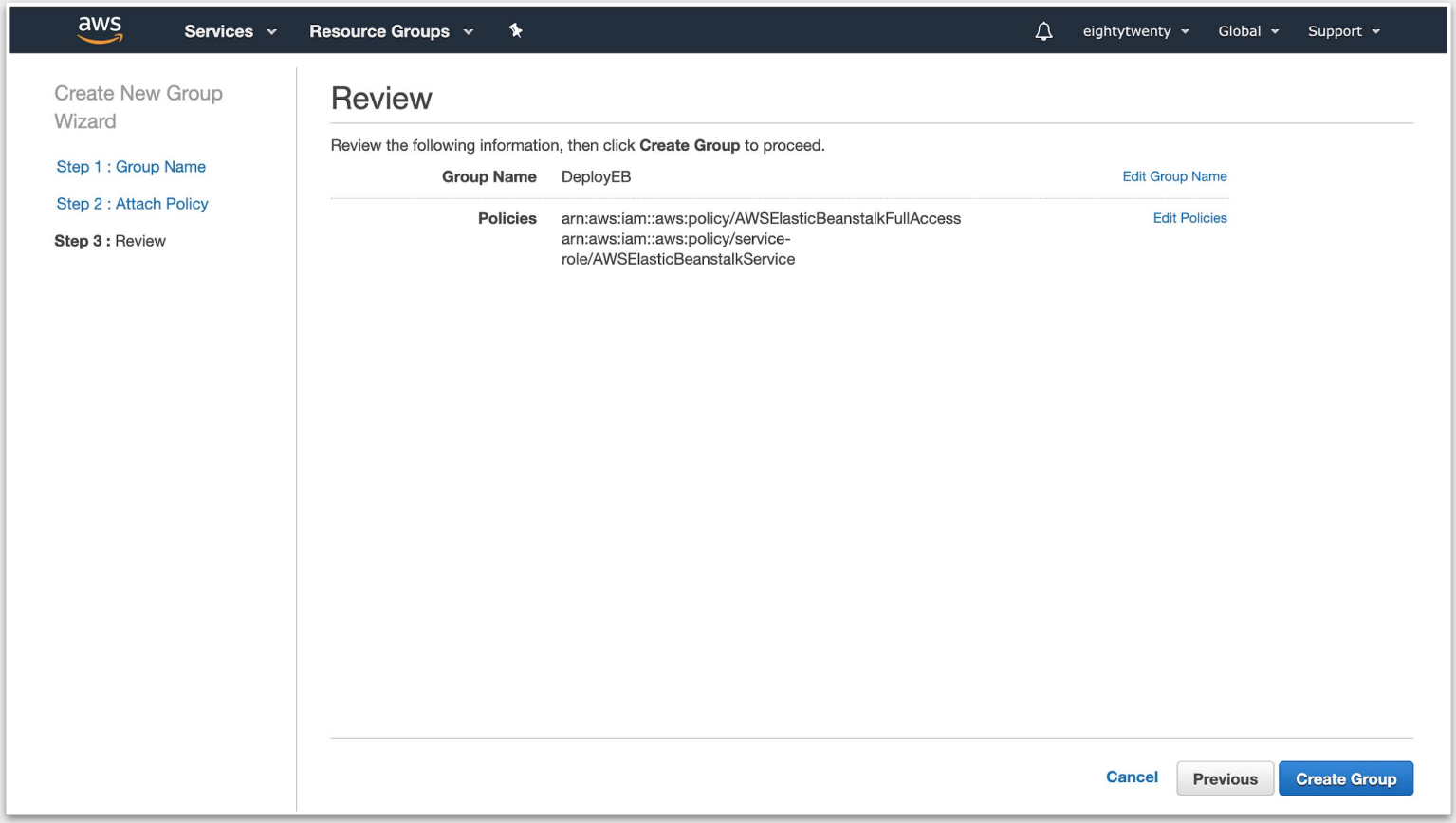
Creating a New IAM Group

Add the policy by selecting the checkbox. Hit **Next Step** once you have added both policies.



Creating a New IAM Group

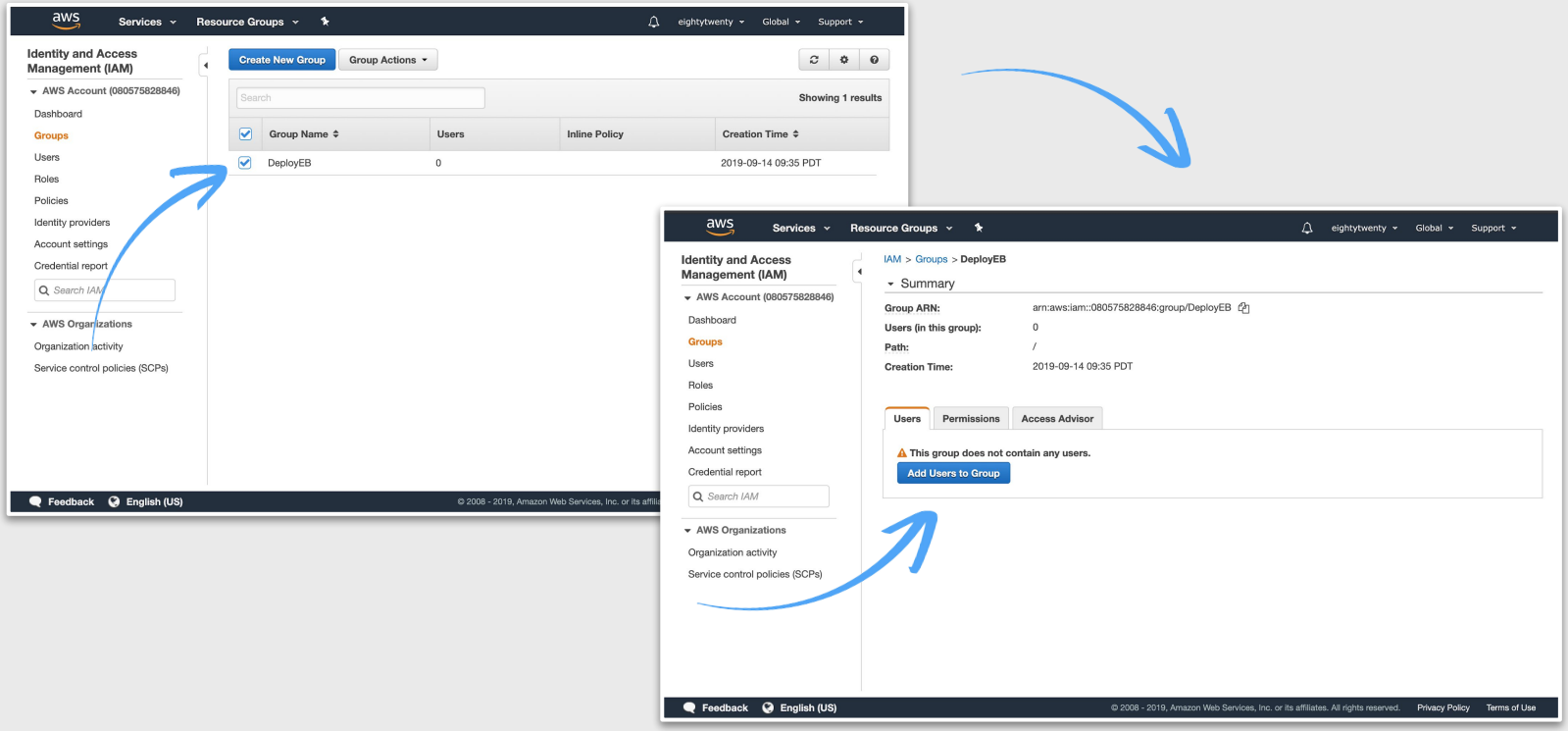
1. **Review** your group
2. Select **Create Group**



Add IAM User to Group

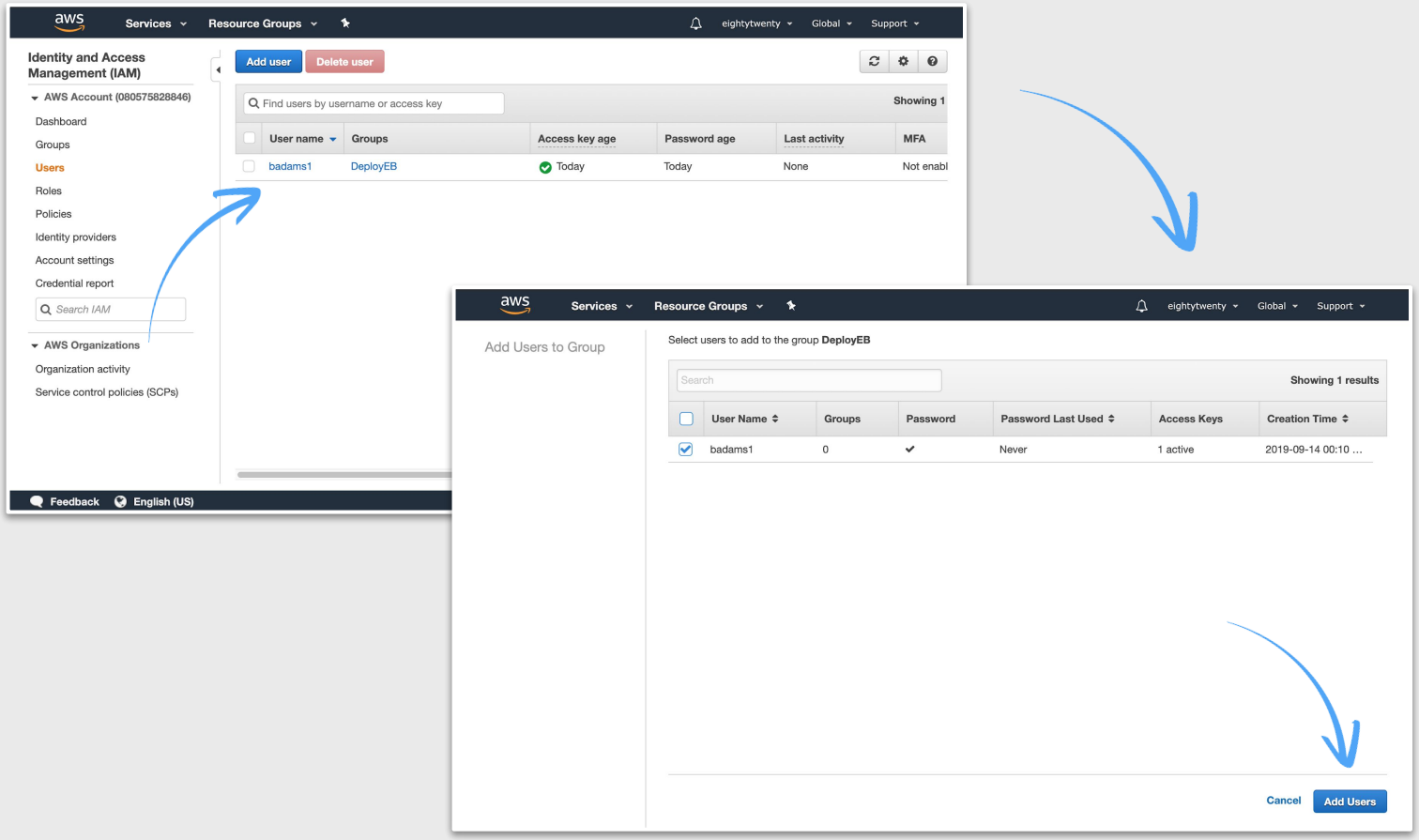
Let's add our **IAM user** to the **Group** we just created.

1. Click the **Group Name** from the list of groups in the IAM management screen. Then select **Add Users to Group** on the users tab.



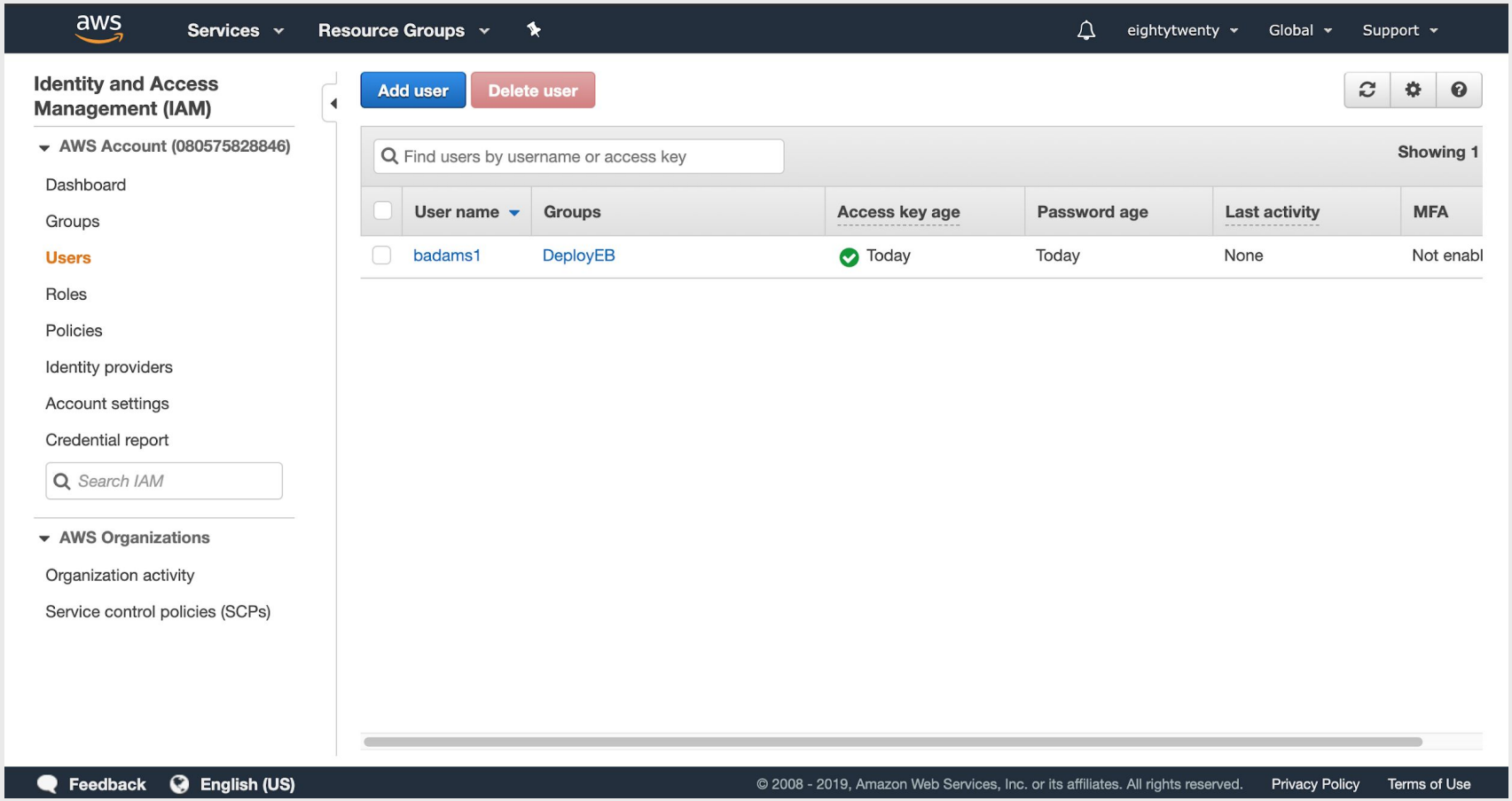
Add IAM User to Group

1. **Select** the user we just created
2. Click the **Add Users** button



Add IAM User to Group

If everything worked well, you should see this on your IAM dashboard on your **Users** tab



Great work! Let's recap what we did so far

- Created an AWS Account
- Create an IAM User for our AWS Account
- We created a Group with Elastic Beanstalk policies
- We added the user we created to the group

Let's get to building our app!

Table of Contents

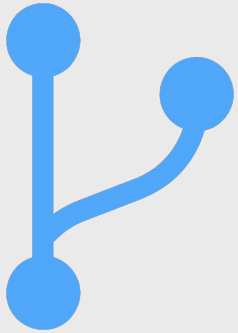
1. Full Stack & Application Introduction
2. Setting up our Dev Environment
3. Creating an AWS Account
4. Creating an IAM User and Group
5. Building and Running the Application
6. Connecting to the Eventbrite API
7. Deploying to AWS Elastic Beanstalk
8. Review & Next Steps

Getting the project code

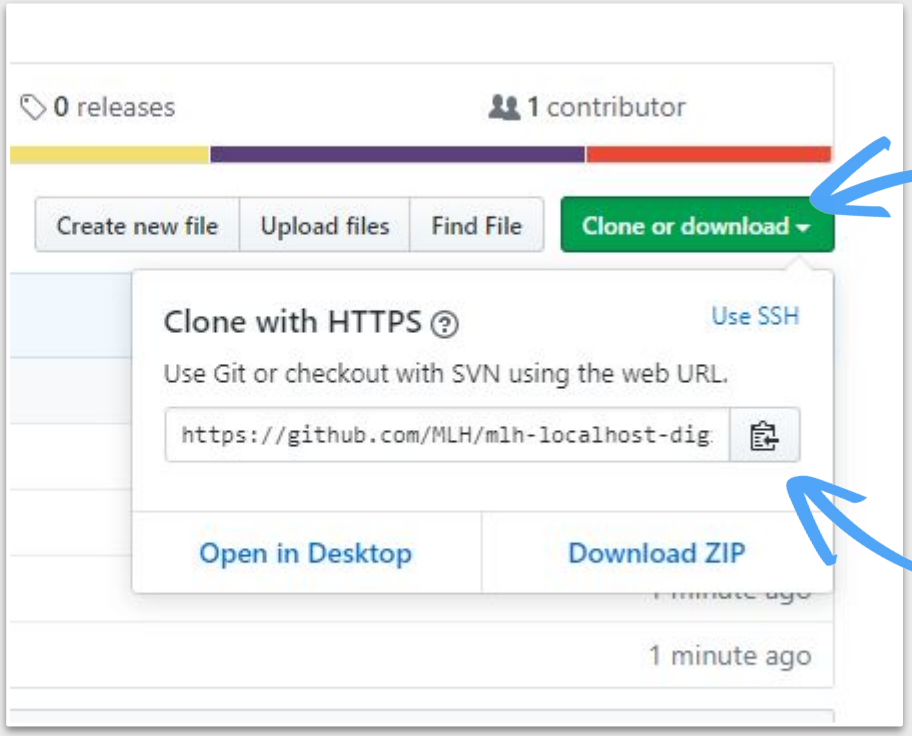
Let's get the starter code for the application.

Head to this URL!

mlhlocal.host/aws-starter-code



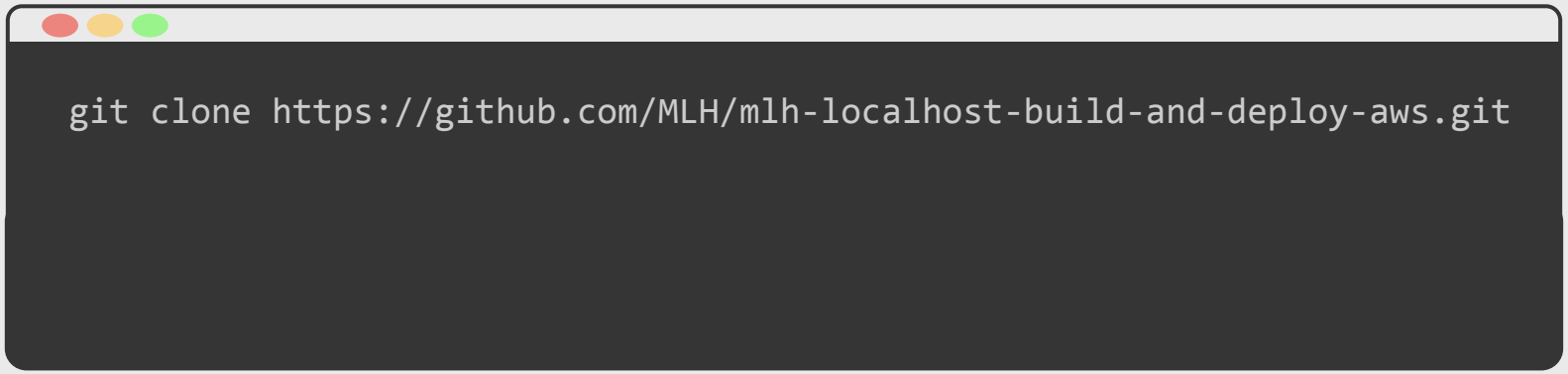
Getting the project code



Click on **Clone or download**, then **Copy the URL** by hitting the clipboard

Getting the project code

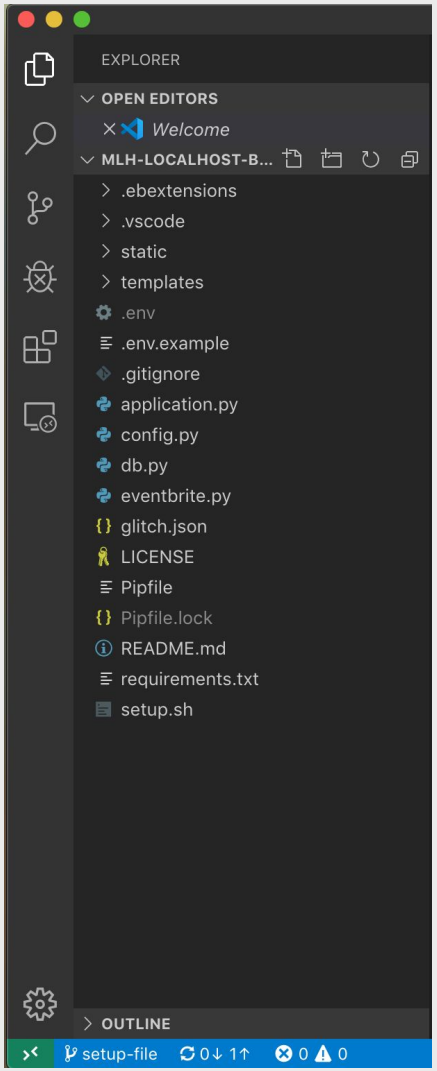
In **Visual Studio Code**, go into the terminal and type the following command, or paste it in if you copied from GitHub.



Getting the project code

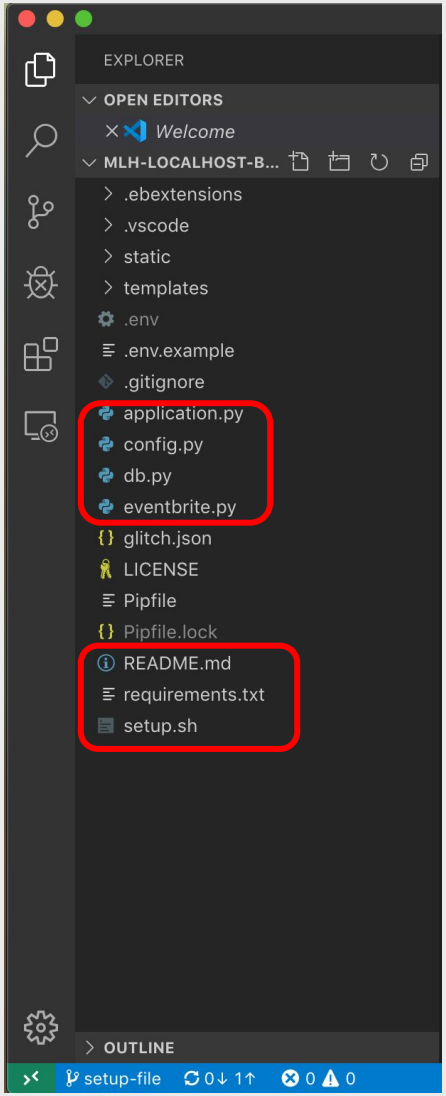
After your repo is cloned into VSC, start it with **File > Open**. Find the folder we've created and open it.

Your file structure should look **similar** to the one on the right. Don't panic if it doesn't exactly match, we'll take care of that!



Taking a look at our files

- **app.py** - where our endpoints live so users can navigate the site
- **config.py** - where our environment variables live; also sets up RDS database
- **db.py** - sets up the database table and the fields used
- **eventbrite.py** - endpoint for our eventbrite api, takes a city name
- **requirements.txt** - the packages needed to run the project
- **README.md** - the info for the project
- **setup.sh** - a setup file that installs dependencies
- **.env.example** - environment variables needed to run the project locally



Create a .env file

Let's create our first file! We need to create a **.env** file to hold our environment variables. Environment variables are usually stuff that you wouldn't want the user to see but will be used by the application, like the **database URL** we wrote down earlier.

.env

```
EVENTBRITE_AUTH_TOKEN=my-api-key
DATABASE_URL=my-db-url
```

1. Create a new file called **.env** in the root folder of the project and copy the contents of the **.env.example** file to **.env.** This file will hold our secret variables

2. Add the **database URL** we created earlier when we set up our MySQL and add it to where it says **DATABASE_URL**

Great Work!

We just setup:

- Cloned into the AWS repo
- Learned about GitHub

Next up:

- Installing pipenv
- A introduction to the files we'll be editing

Installing `pipenv`: Mac

Mac Users, We will be using **Homebrew** to install pipenv.

You can check if you have Homebrew installed using the following command in your terminal `brew --version`

If you don't have Homebrew installed, go to the following link to install it.

mlhlocal.host/get-homebrew

Once installed, run the following command in the terminal to install our virtual environment pipenv `brew install pipenv`

Key Terms

Homebrew: a free and open-source software package management system that simplifies the installation of software on macOS operating system and Linux.

Installing `pipenv`: Windows

Window Users, We will be using **pip** to install pipenv. Lucky for us you should already have pip installed when we installed Python.

Run the following command in the terminal to install our virtual environment pipenv `pip install pipenv`

Installing our libraries

Our next step will be to install the libraries we need to run our app.

A **library (or package)** is some external code that we can use in our projects.

Run the following command in your **Terminal (View > Terminal)** in **Visual Studio Code**

```
pipenv install -r requirements.txt
```

Activating the `pipenv` environment

You can activate the virtual environment using the command `pipenv shell`

You should see something like, but not exactly like:

```
→  mlh-localhost-build-and-deploy-aws git:(setup-file) pipenv shell
Loading .env environment variables…
Launching subshell in virtual environment…
 . /Users/macbookpro/.local/share/virtualenvs/mlh-localhost-build-and-deploy-aws-50U1gHep/bin/activate
→  mlh-localhost-build-and-deploy-aws git:(setup-file)  . /Users/macbookpro/.local/share/virtualenvs/mlh-localhost-build-and-deploy-a
ws-50U1gHep/bin/activate
(mlh-localhost-build-and-deploy-aws) →  mlh-localhost-build-and-deploy-aws git:(setup-file)
```

We're in!

Tip

Anytime you want to deactivate the **pipenv** shell, you can enter **exit** or hit **Ctrl - D**

Running the Project Locally

Let's **run the project**!

Our project built with Flask, a A web framework that allows Python to make and receive calls as a server.

We can run with the command below in Flask to open our project locally to see what we're working with. Run the command and see what happens!

```
(mlh-localhost-build-and-deploy-aws) bash-3.2$ FLASK_APP=application.py
FLASK_DEBUG=1 flask run
```

Running the Project locally

What just happened? When we ran the last command we ran our project locally, and our application gave us a local URL to check it out!

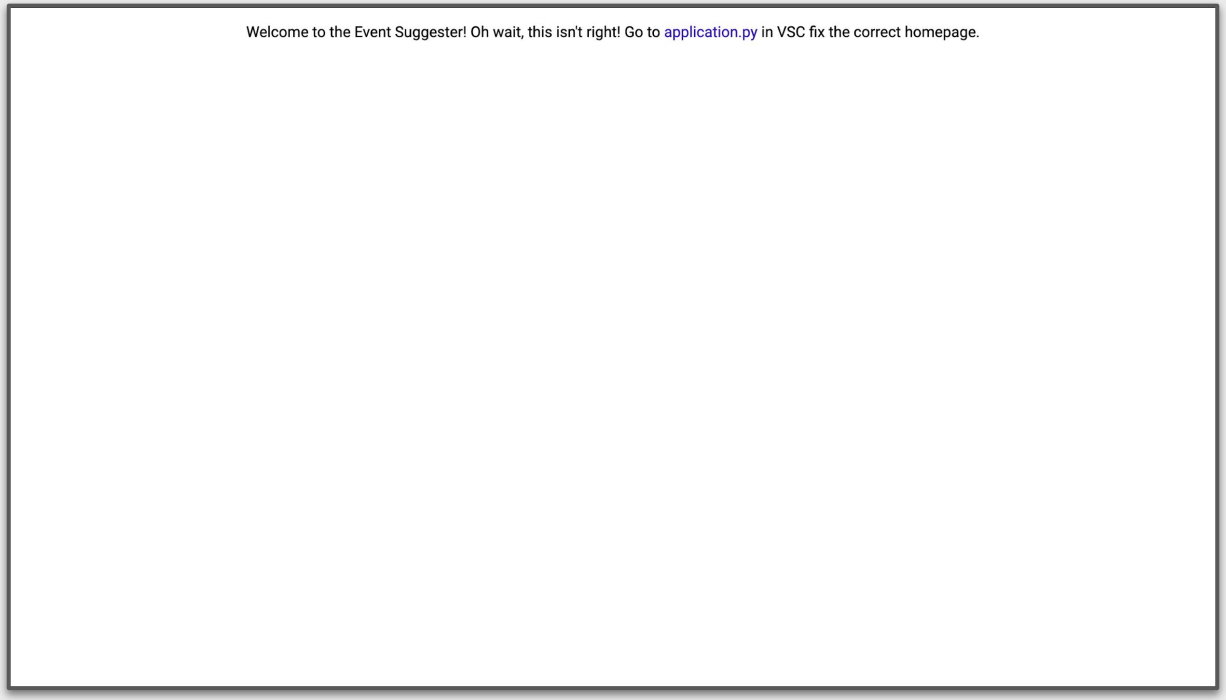
```
(mlh-localhost-build-and-deploy-aws) bash-3.2$ FLASK_APP=application.py FLASK_DEBUG=1 flask run
 * Serving Flask app "application.py" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 131-112-721
127.0.0.1 - - [23/Sep/2019 10:52:39] "GET / HTTP/1.1" 500 -
```

The local URL is http://127.0.0.1:5000/!

When we run a project *locally,* we're saying that we can open our web app on our local machine. No one else can access your web app but you. Which gives you the freedom to edit the code as you please without affecting live users!

Running the Project

If you visit the URL in your browser you should see the following web page!



Great! We have our app running locally and we can edit the code as we please. But wait, this isn't the homepage or our application. Let's fix that

Write Code - Update `render_template`

In Flask, we can use the `render_template` method in our **application.py** file to change which html file we want the app to use.

1. Go into `application.py`
2. On **line 16**, change `start.html` to homepage.html inside the `render_template` method. Make sure to hit Save!

application.py

```
@application.route("/")
def home():
    # 💡 renders an html page with url parameters, if any
    # 🆘 1. We need to show the correct homepage! Change start.html to homepage.html
    return render_template("start.html", city=request.args.get("city", ""))
```

Code Solution

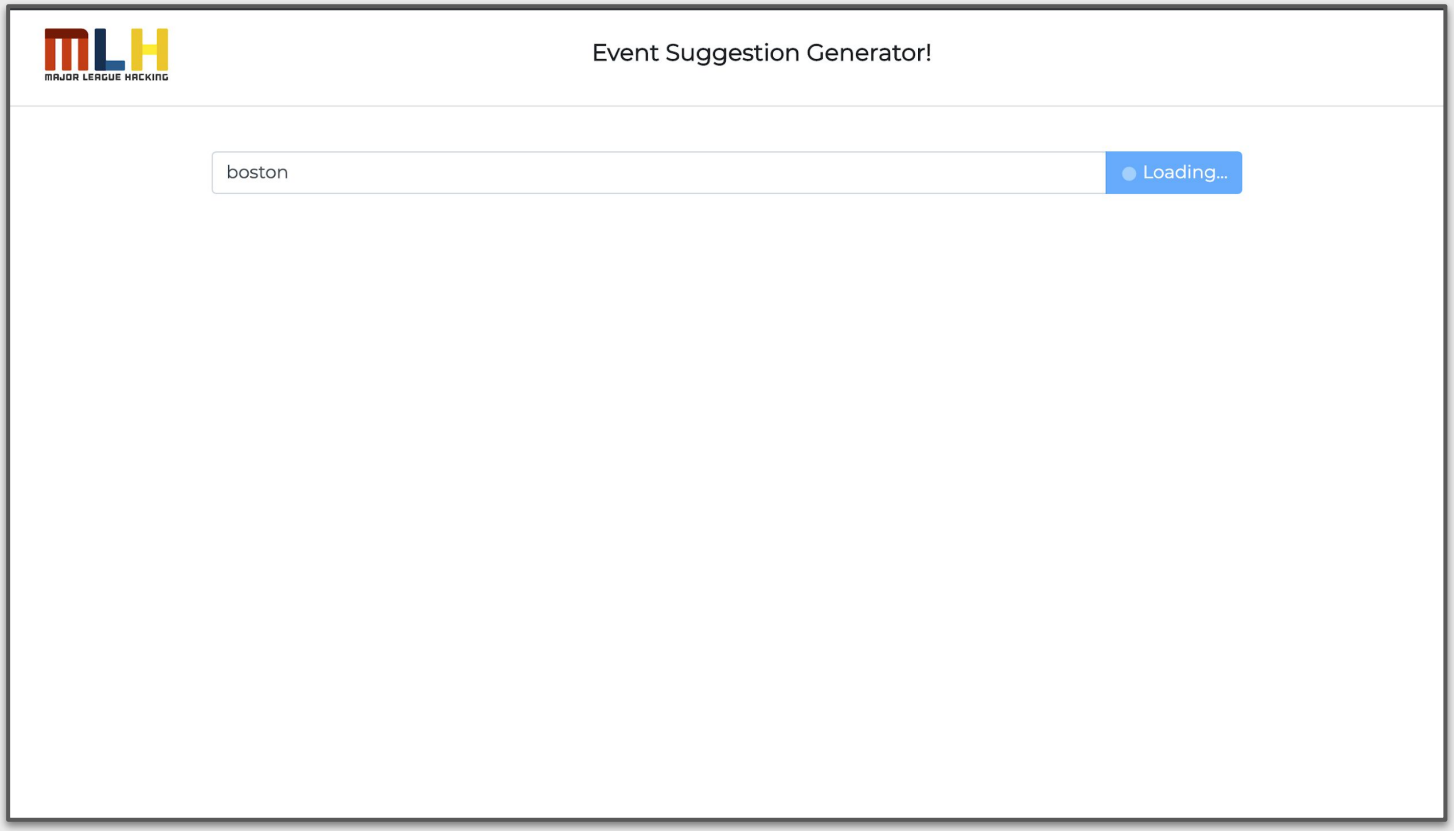
By updating the `render_template` to use homepage.html instead of start.html, we told our application what the starting point of our website will be. Flask will auto-reload the webpage once you hit save. What do you see now?

application.py

```
@application.route("/")
def home():
    # 💡 renders an html page with url parameters, if any
    # 🆘 1. We need to show the correct homepage! Change start.html to homepage.html
    return render_template("homepage.html", city=request.args.get("city", ""))
```

Local Success!

We now have our main homepage of our application! It's not much right now so let's add in some events that the user can search for in their area



Great Work!

We just setup:

- Learned about Python and Pip
- Installed packages with a BASH file
- Ran our virtual environment

Next up, we'll:

- Learn about APIs
- Get our Eventbrite API Key

Table of Contents

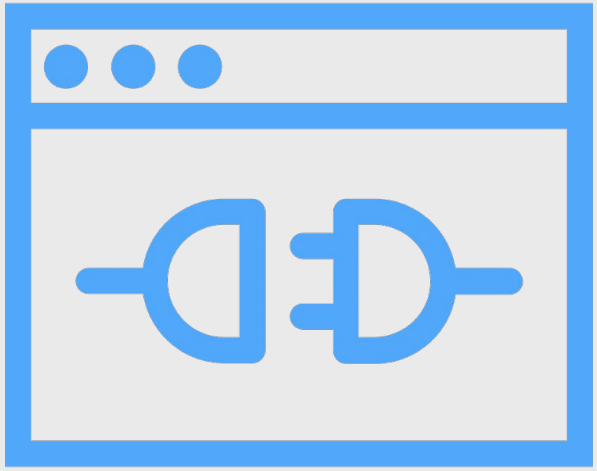
1. Full Stack & Application Introduction
2. Setting up our Dev Environment
3. Creating an AWS Account
4. Creating an IAM User and Group
5. Building and Running the Application
6. Connecting to the Eventbrite API
7. Deploying to AWS Elastic Beanstalk
8. Review & Next Steps

What is an API?

API stands for **Application Programming Interface.** APIs are used to connect a server and a client.

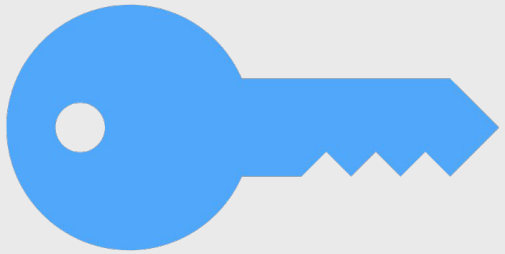
An API is a structured way for one program to offer services to other programs.

In the case of website APIs these programs are running on different machines - a program running on Facebook's servers are offering services to a program running on your computer or another web server.



What is an API?

In order to access an API from our application, we typically need an **API key** which is a unique secret code that is passed in to an API to identify the calling application or user. API keys are used to track and control how the API is being used, for example to prevent malicious use or abuse of the API.



eventbrite

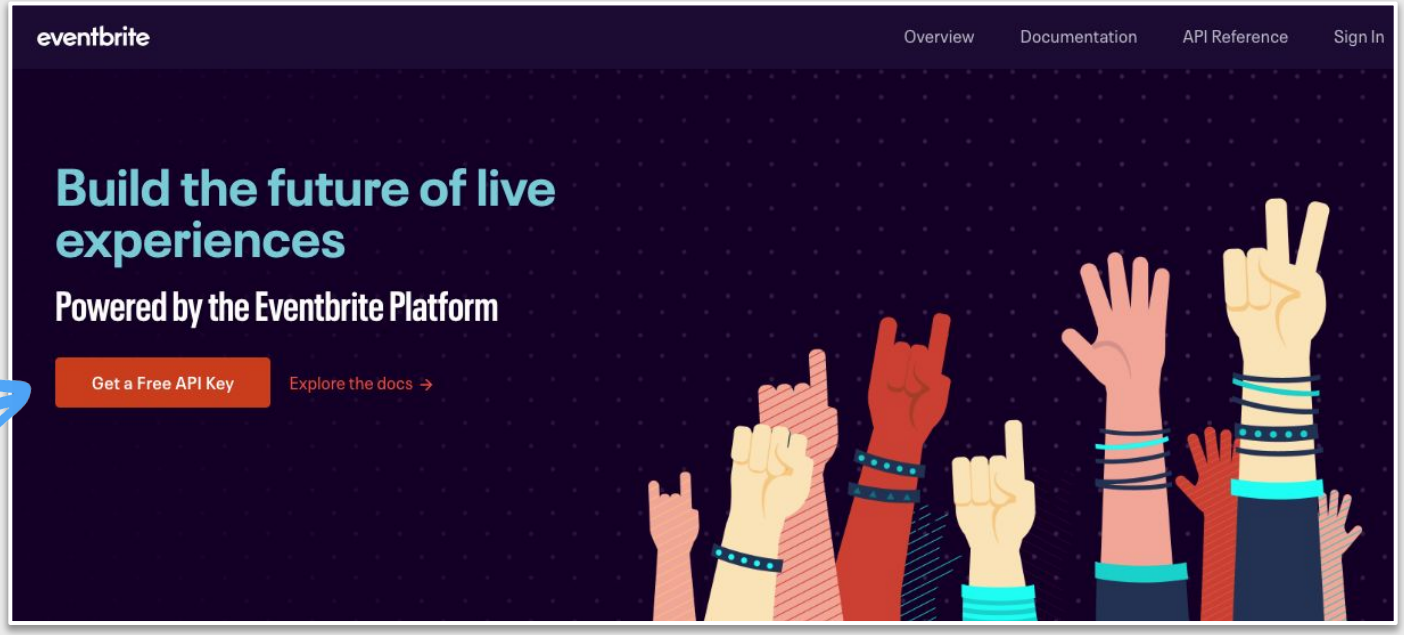
In our project we will be using the **Eventbrite API.** Eventbrite is a website that manages events in a local area. People are able to buy tickets to their favorite concerts, tours and more. The API will allow us to call the events from Eventbrite instead of us making them ourselves.

Getting an Eventbrite API Key

Getting an Eventbrite Key is easy!

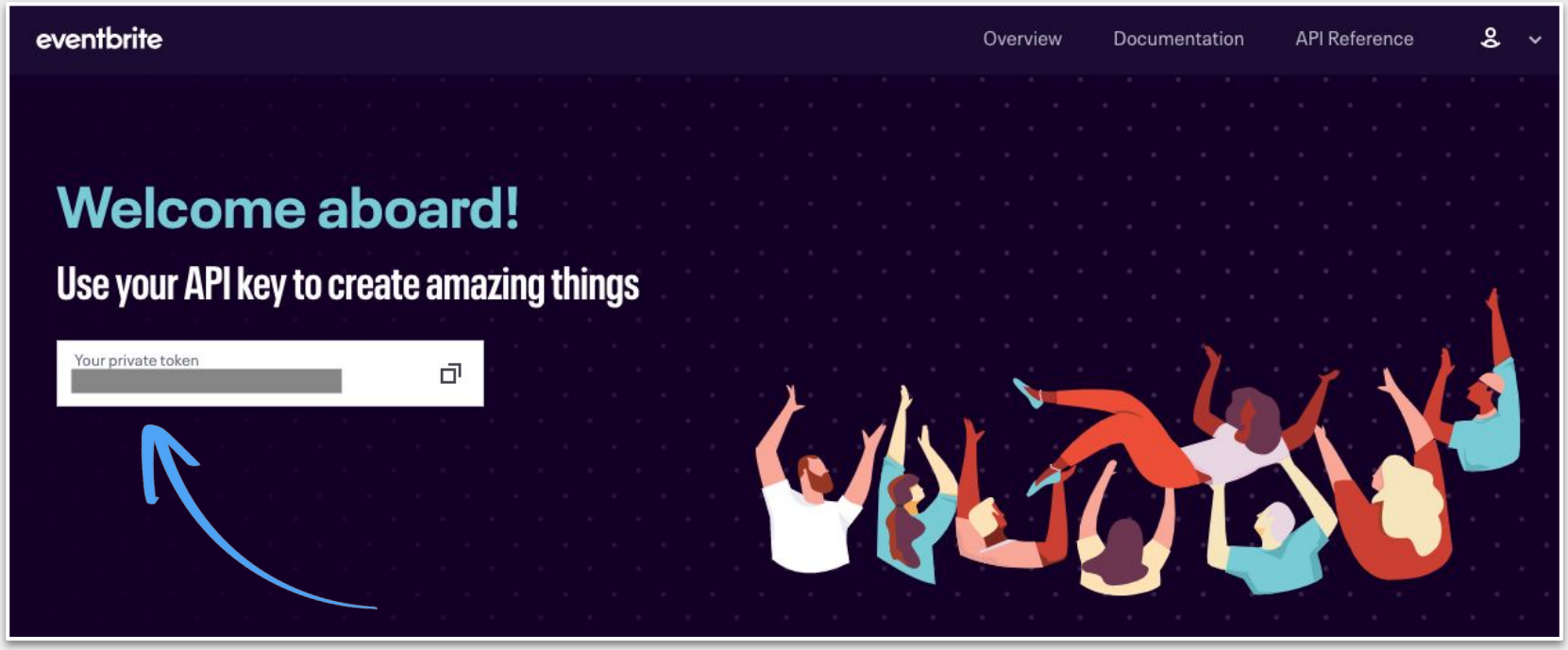
You can get a Free Eventbrite API Key by visiting this link and hitting the **Get a Free API Key** button

http://mlhlocal.host/eventbrite



Getting an Eventbrite API Key

Once you created your Eventbrite account, you'll be granted your API Key, Easy Peasy! Copy your API Key and save it somewhere private



Add your Eventbrite API key to .env

.env

```
EVENTBRITE_AUTH_TOKEN=my-api-key
DATABASE_URL=mysql://username:password@localhost:3306/events
```

1. Open the **.env** file we created earlier.
2. Add your Eventbrite API key to the **.env** file where it says **EVENTBRITE_AUTH_TOKEN**

Let's run our project and see what changed

Running the Project

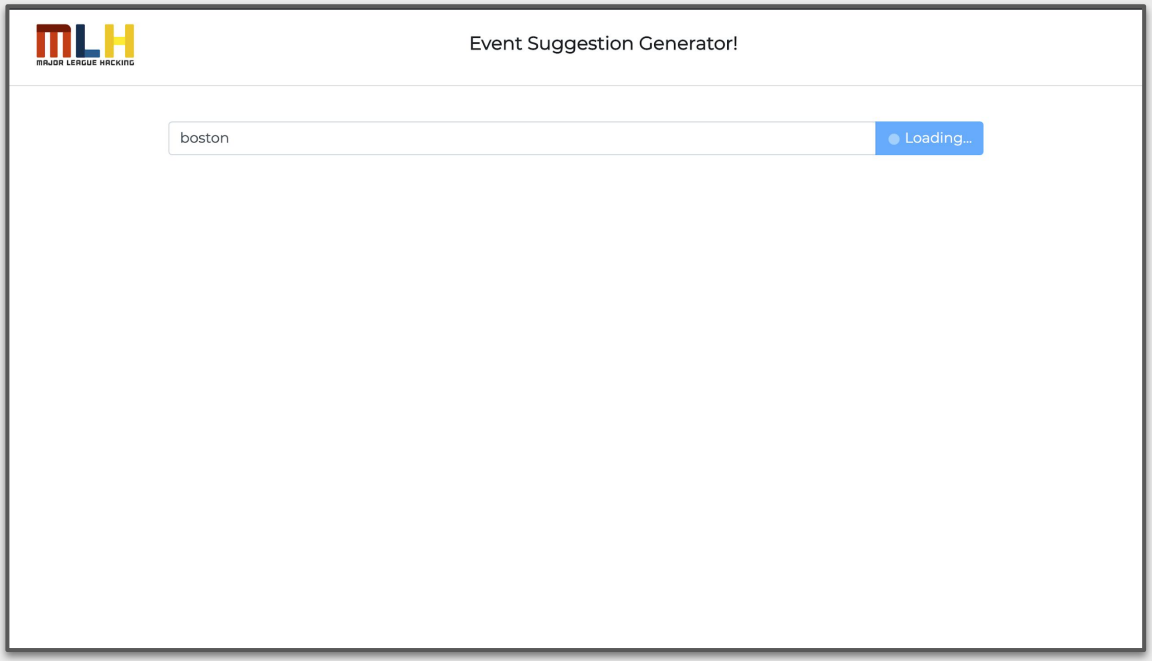
If it's not activated already, Activate your virtual environment by entering `pipenv shell` in the terminal. Once inside your shell, enter the following command to run your app.

```
(mlh-localhost-build-and-deploy-aws) bash-3.2$ FLASK_APP=application.py
FLASK_DEBUG=1 flask run
```

Enter http://127.0.0.1:5000/ in your browser to see your application in action!

Running the Project

Our page is up! but when we type in a city, the app gets stuck loading!

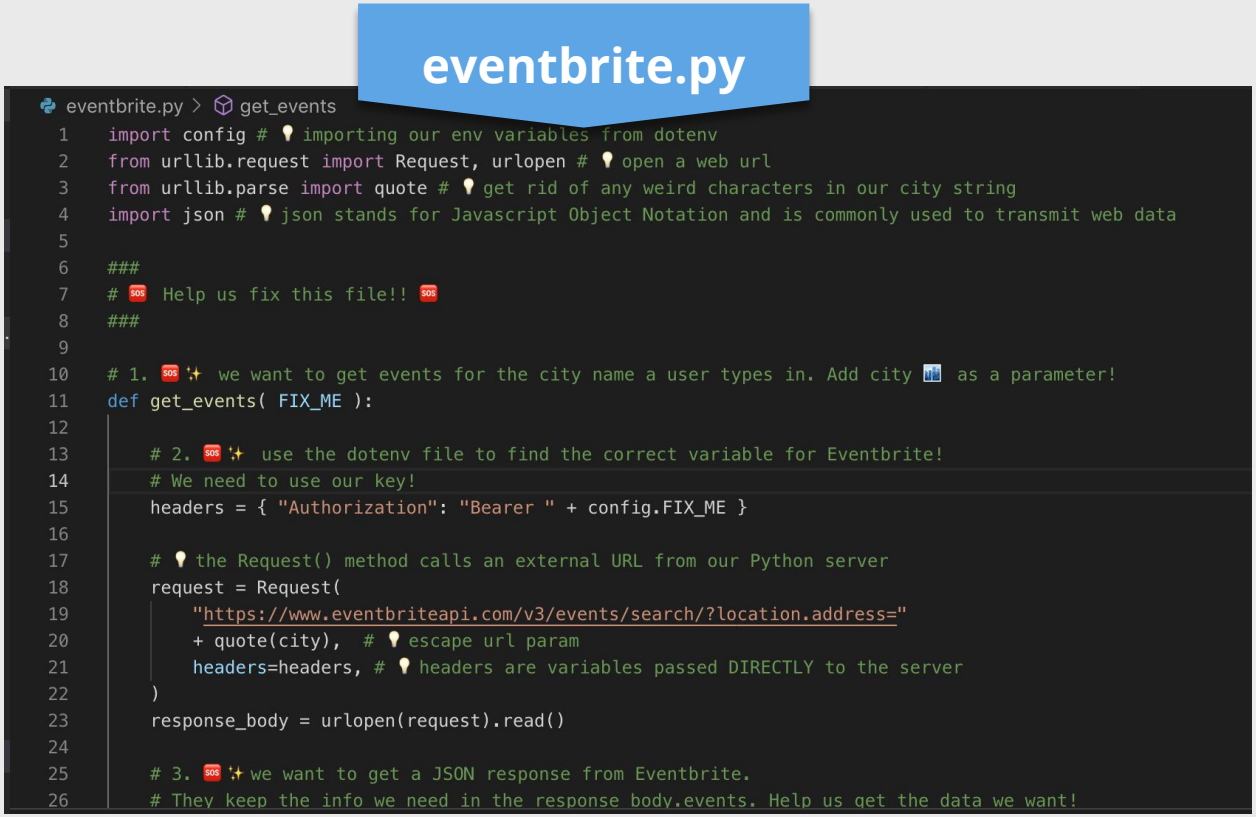


What do you think is going on?
What do you think the solution is?

Time to Debug!

Code Review

It seems we will need to fix up the code in our Eventbrite API. This code exists in the `eventbrite.py` file



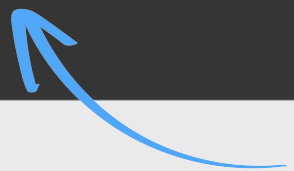
Write Code - Fixing get_events

It seems we will need to fix the code the Eventbrite API utilizes. This code exists in the **eventbrite.py file**

First, we need to fix the **argument** that our `get_events()` method is expecting

eventbrite.py

```
###
# 🆘 Help us fix this file!! 🆘
###

# 1. 🆘✨ we want to get events for the city name a user types in. Replace
the placeholder variable with city 🏙️ as a parameter!
def get_events( FIX_ME ):
```



Code Solution

1. Go into `eventbrite.py`
2. On **line 11**, change `"FIX_ME"` to `city`

eventbrite.py

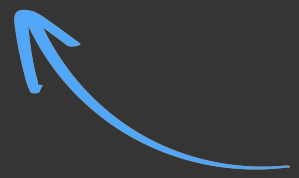
```
###
# 🆘 Help us fix this file!! 🆘
###

# 1. 🆘✨ we want to get events for the city name a user types in. Replace
the placeholder variable with city 🏙 as a parameter!
def get_events( city ):
```

Write Code - Adding our Environment Variable

Our **environment variable key** isn't correct here! How do you think we should fix it?

eventbrite.py

```
# 2. 🆘✨ use the dotenv file to find the correct variable for Eventbrite!
    # We need to use our key! Look in the .env file for the Eventbrite key
name
    headers = {  "Authorization": "Bearer " + config.FIX_ME }
```



Code Solution

We need to reference our environment variables we created in the **.env** file.

1. Change `FIX_ME` to `EVENTBRITE_AUTH_TOKEN`

eventbrite.py

```
# 2. 🆘✨ use the dotenv file to find the correct variable for Eventbrite!
    # We need to use our key! Look in the .env file for the Eventbrite key
name
    headers = {  "Authorization": "Bearer " + config.EVENTBRITE_AUTH_TOKEN }
```

Write Code - Fetching the Correct Data

Now we need to get the correct data from our **response body**.

Learn more about the **Eventbrite API** at this mlhlocal.host/eventbrite-api

eventbrite.py

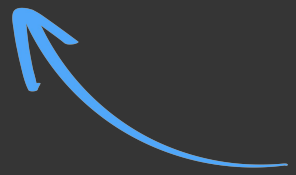
```
    # 3. 🆘✨we want to get a JSON response from Eventbrite.
     # They keep the info we need in the response_body.events. Help us get
the data we want!
     events = json.loads(response_body)[ "FIX_ME"]
```

Code Solution

1. Change `"FIX_ME"` to `"events"`

eventbrite.py

```
    # 3. 🆘✨we want to get a JSON response from Eventbrite.
     # They keep the info we need in the response_body.events. Help us get
the data we want!
     events = json.loads(response_body)[ "events"]
```



Re-run our Project

Great work fixing that code!

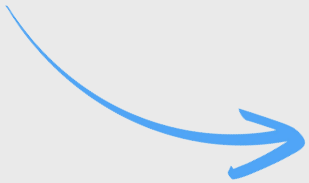
Let's rerun the project with the changes that we've made.

Rerun our Project

Remember we **run** the following command in your Terminal to start the project

```
(mlh-localhost-build-and-deploy-aws) bash-3.2$ FLASK_APP=application.py
FLASK_DEBUG=1 flask run
```

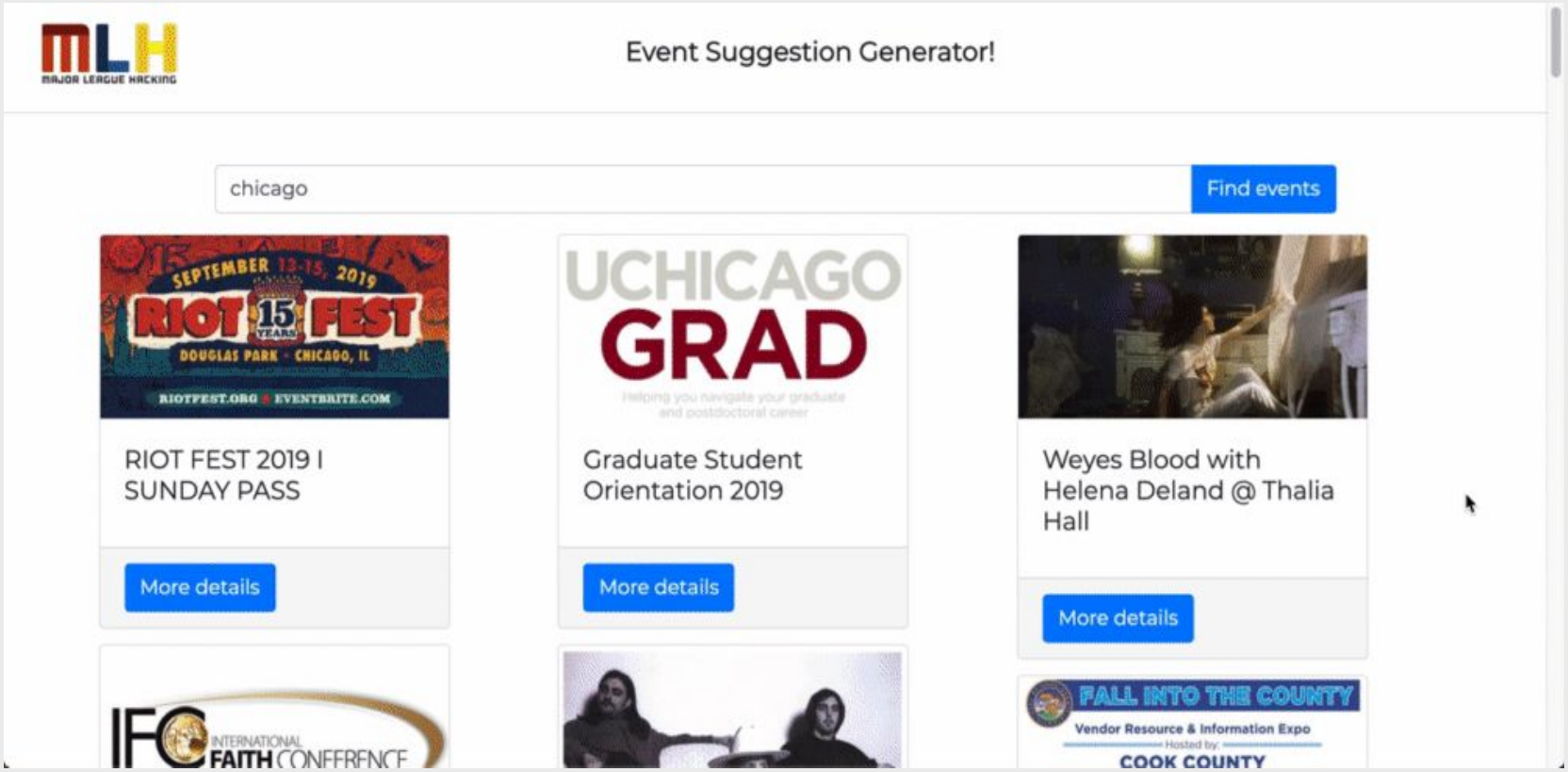
```
(mlh-localhost-build-and-deploy-aws) bash-3.2$ FLASK_APP=application.py FLASK_DEBUG=1 flask run
 * Serving Flask app "application.py" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 131-112-721
127.0.0.1 - - [23/Sep/2019 10:52:39] "GET / HTTP/1.1" 500 -
```



The local URL is http://127.0.0.1:5000/!

Run it back!

It works!



Great Work!

We just setup:

- Installed packages with a BASH file
- Ran our virtual environment

Next up:

- Create an AWS user
- Deploy our local project to production

Table of Contents

1. Full Stack & Application Introduction
2. Setting up our Dev Environment
3. Creating an AWS Account
4. Creating an IAM User and Group
5. Building and Running the Application
6. Connecting to the Eventbrite API
7. Deploying to AWS Elastic Beanstalk
8. Review & Next Steps

Deploying our project to production

Our app looks great, but there's one **BIG PROBLEM**. No one can see it!

In order to fix this, we need to **deploy** our project

We will be doing this with the **Elastic Beanstalk**

What is Elastic Beanstalk?

Elastic Beanstalk (EB) is a service that AWS offers. It **automates** the process of getting your app on AWS.

You focus on building your web app, and then **deploy** it to Elastic Beanstalk

EB then takes care of **domain management**, auto **scaling your code** to manage load, and monitoring **security**

Installing awsebcli

To deploy to Elastic Beanstalk we to install **Elastic Beanstalk command line interface (EB CLI)**. We can do this in a few simple steps.

Activate our environment
(If it's not activated already)

```
pipenv shell
```

Install the library

```
pipenv install awsebcli
```

Key Terms

EB CLI: command line interface that can perform a variety of operations to deploy and manage your Elastic Beanstalk applications and environments.

Set Up Elastic Beanstalk

In the shell, run the command `eb init`. This will initialize a new Elastic Beanstalk instance. Follow the prompt that follows and enter the settings below

Region	Select the default option (3), and write it down (to view on console)
CodeCommit	No
Credentials	Log in with the IAM user we created at the start of the workshop
Application Name	Defaults to the directory name
Python Version	Select Python 3+ (however any version is fine)
SSH	Yes (Just hit **enter** through the prompts)

Set Up Elastic Beanstalk

Next, let's use `eb create` to create an environment. Follow the prompt again:

- **Environment name**: default is fine here
- **DNS Name**: select the default
- **Load Balancer Type**: select the default (Application)

Add RDS DB to Elastic Beanstalk Instance

Our next goal will be to add an RDS instance to our Elastic Beanstalk, so we can **save our favorite events on our production server**

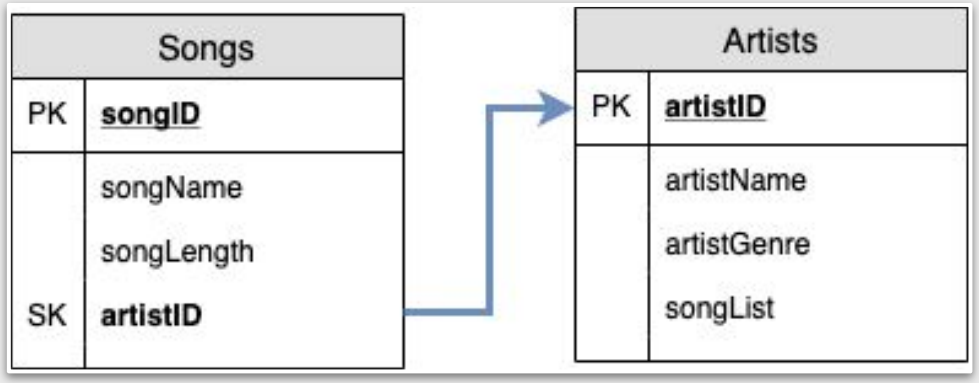
What is RDS?

Amazon Relational Database System (RDS) is an AWS service that allows you to spin up relational databases in the cloud.

Remember:

Relational databases are databases that have **tables** that refer to each other

For example if I have a table for **songs** and a table for **artists** I can link them as follows:



Set Up Elastic Beanstalk

Let's set our **RDS Engine** variable. In the pipenv shell, type:

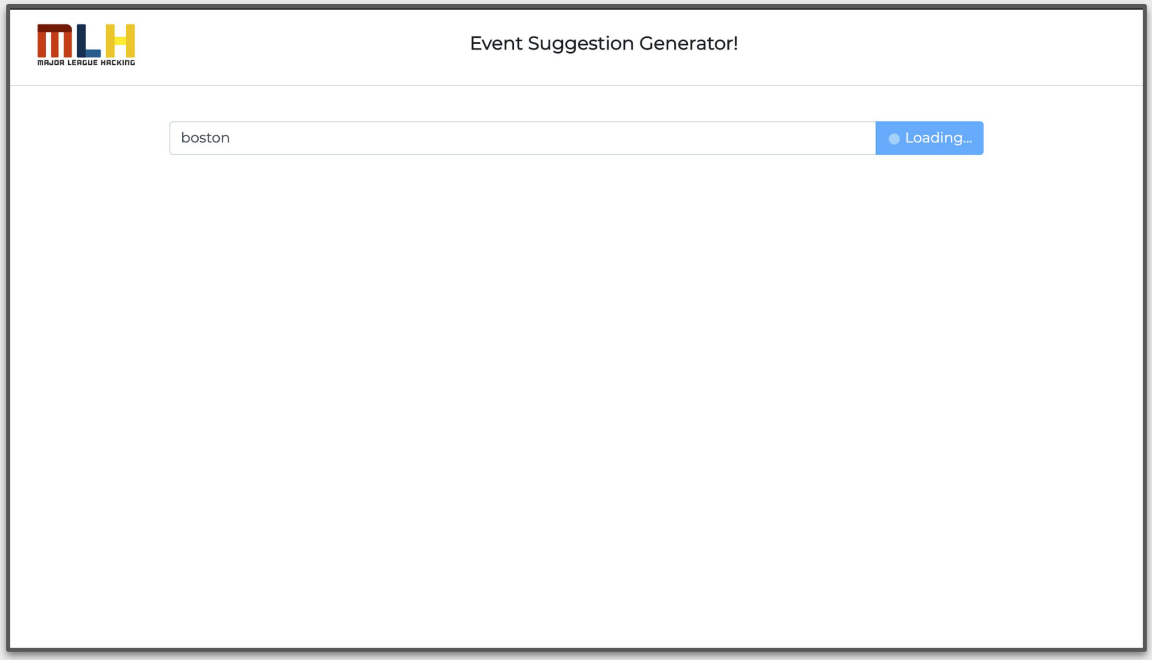
```
eb setenv RDS_ENGINE=mysql
```

```
bash-3.2$  . /Users/eddie/.local/share/virtualenvs/mlh-localhost-build-and-deploy-aws-starter-RL1OycUS/bin/activate
(mlh-localhost-build-and-deploy-aws-starter) bash-3.2$ eb setenv RDS_ENGINE=mysql
```

Set Up Elastic Beanstalk

Deploy our application with the command `eb deploy`

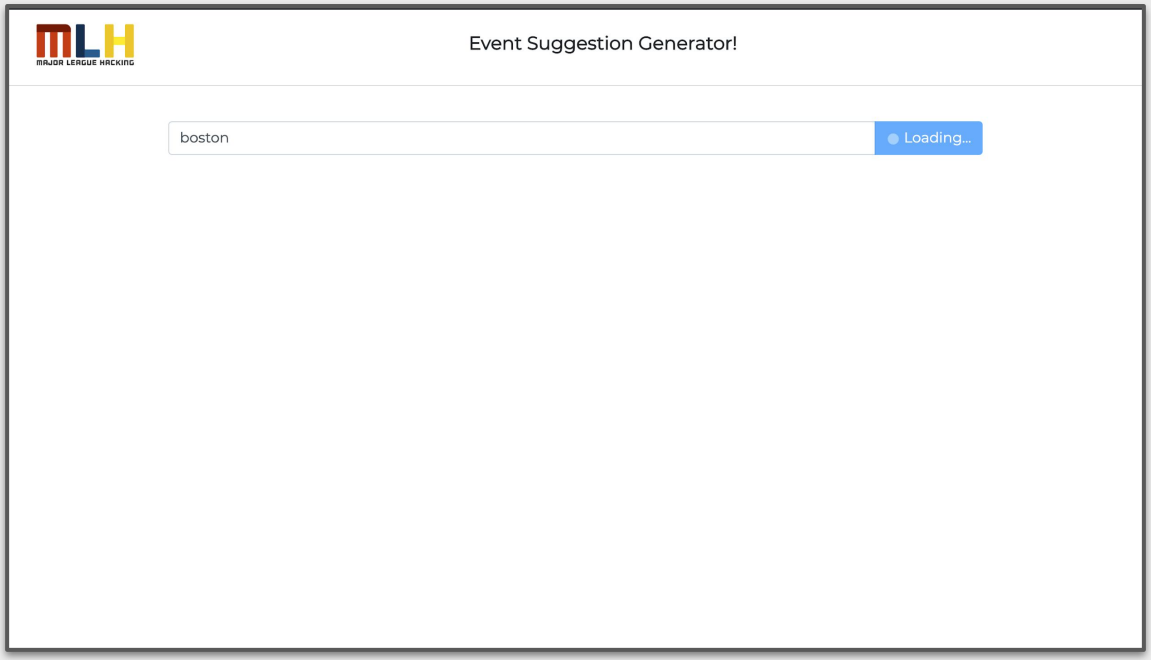
Open the application in your default browser with `eb open`



Oh No!

Nothing is loading!

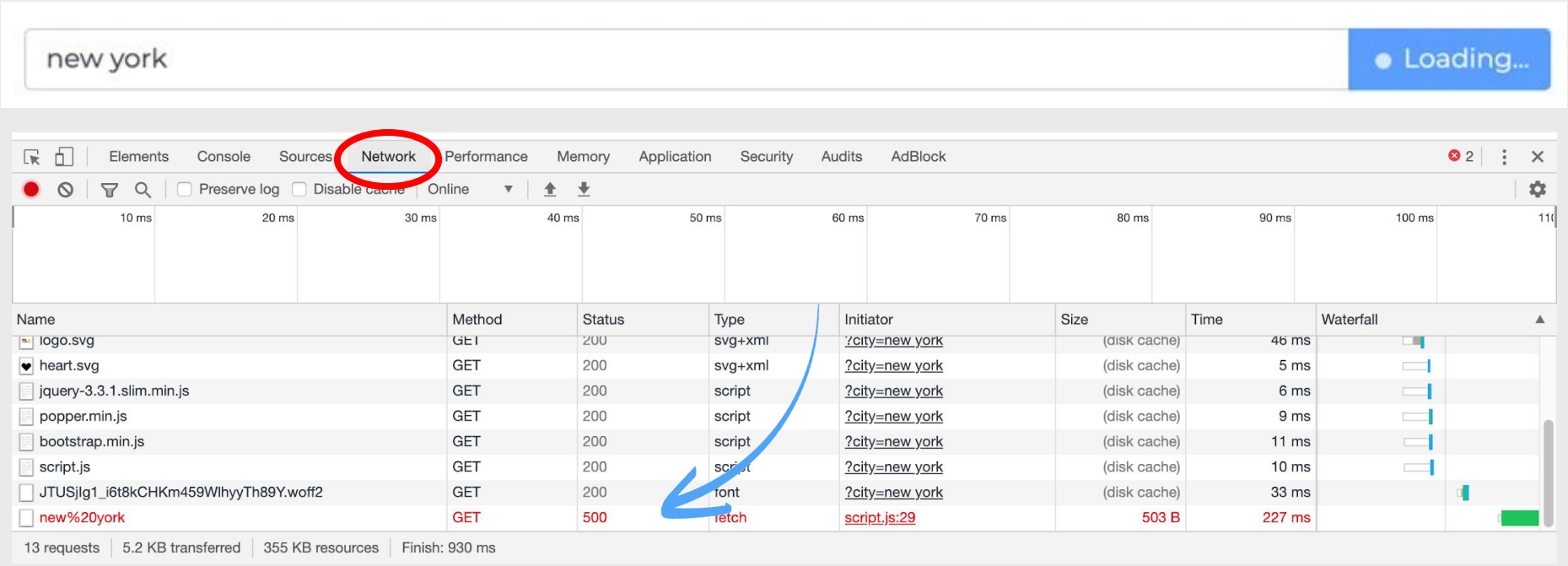
What do you think is happening?



Oh no!

Let's open the **console.** A **console** is a feature for developers to interact with the code the browser sees. You can open your console in the browser by using **View > Developer > Developer Tools** on Chrome and **CMD-SHIFT-J** or **CTRL-SHIFT-J** on Firefox.

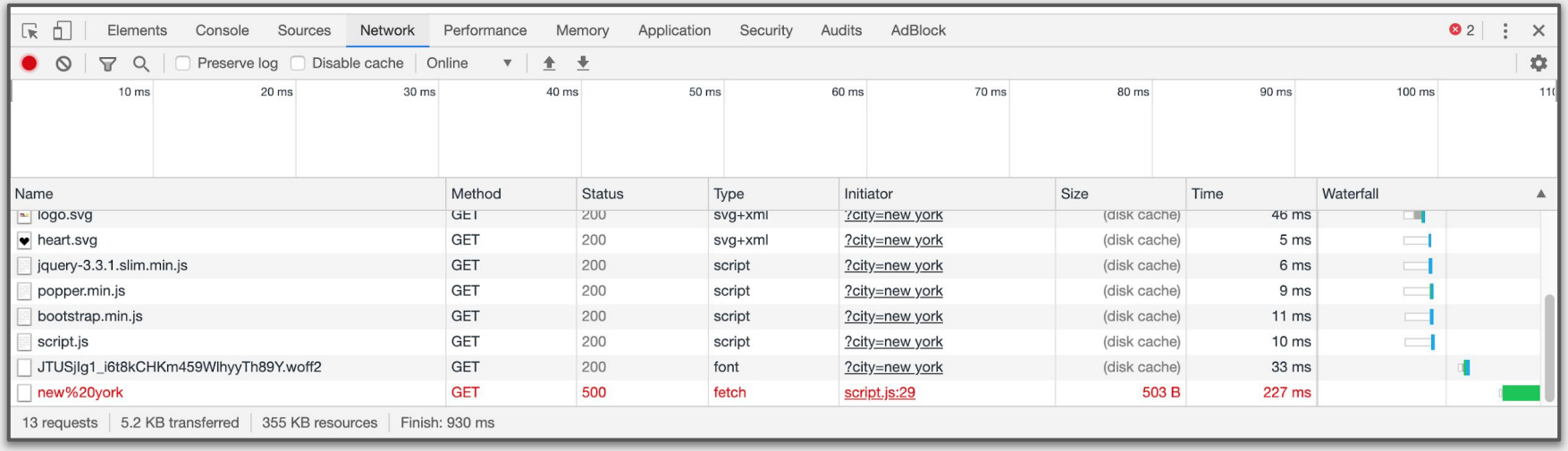
Open your console and go to the **Network** tab and see if you can spot any errors



Let's take a closer look

The **Network tab** is showing a **500 error** from Eventbrite. A **500 error** means that we have a **server error.**

Why do you think we're getting this error?



Let's take a closer look

This looks like the error we saw before! Maybe our API key for Eventbrite isn't working...**let's find out!**

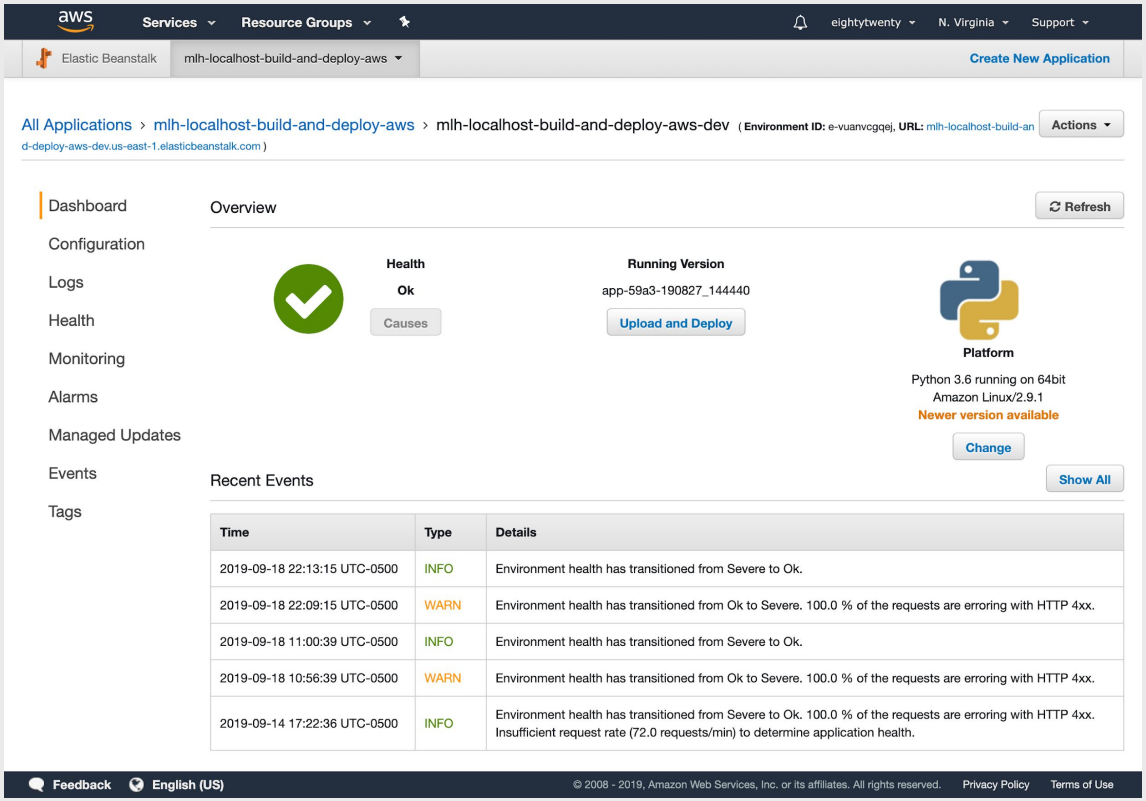
In your Terminal, type the following command to open your Elastic Beanstalk console:

```
eb console
```

EB Console to the Rescue

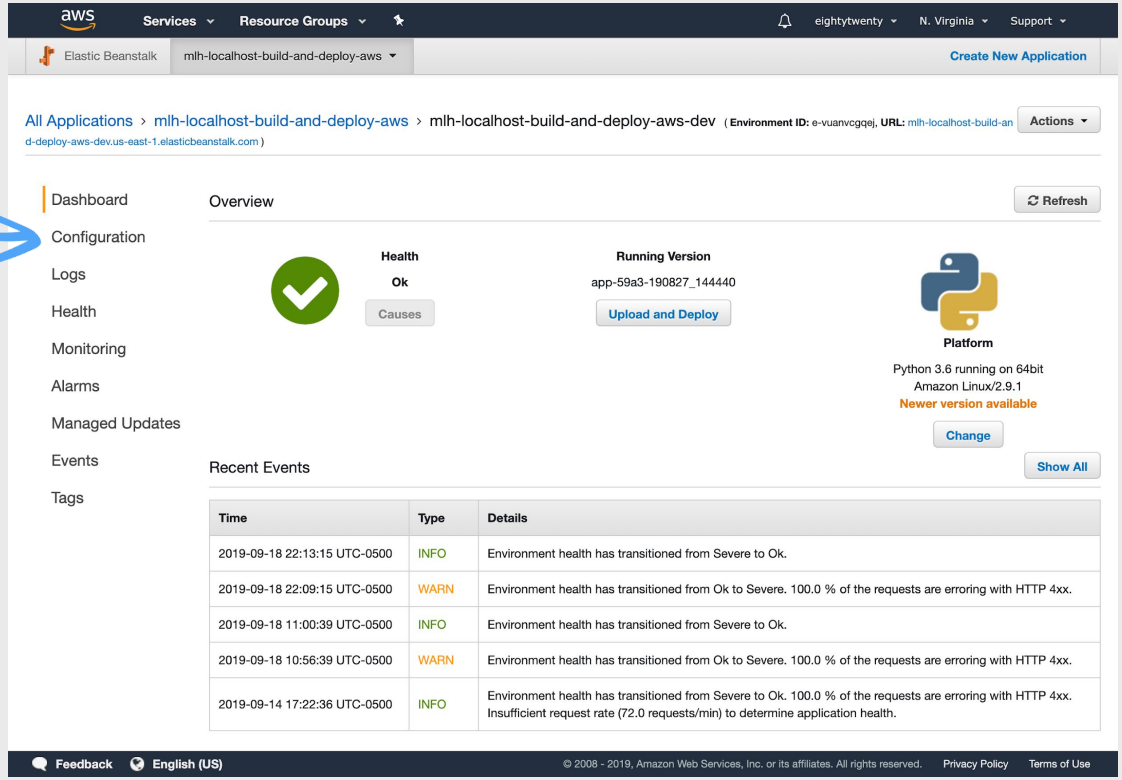
The EB Console is a powerful resource in interacting with our app in **realtime.**

You can always open the EB Console using the command **eb console** in the terminal. It will automatically open your browser to the console.



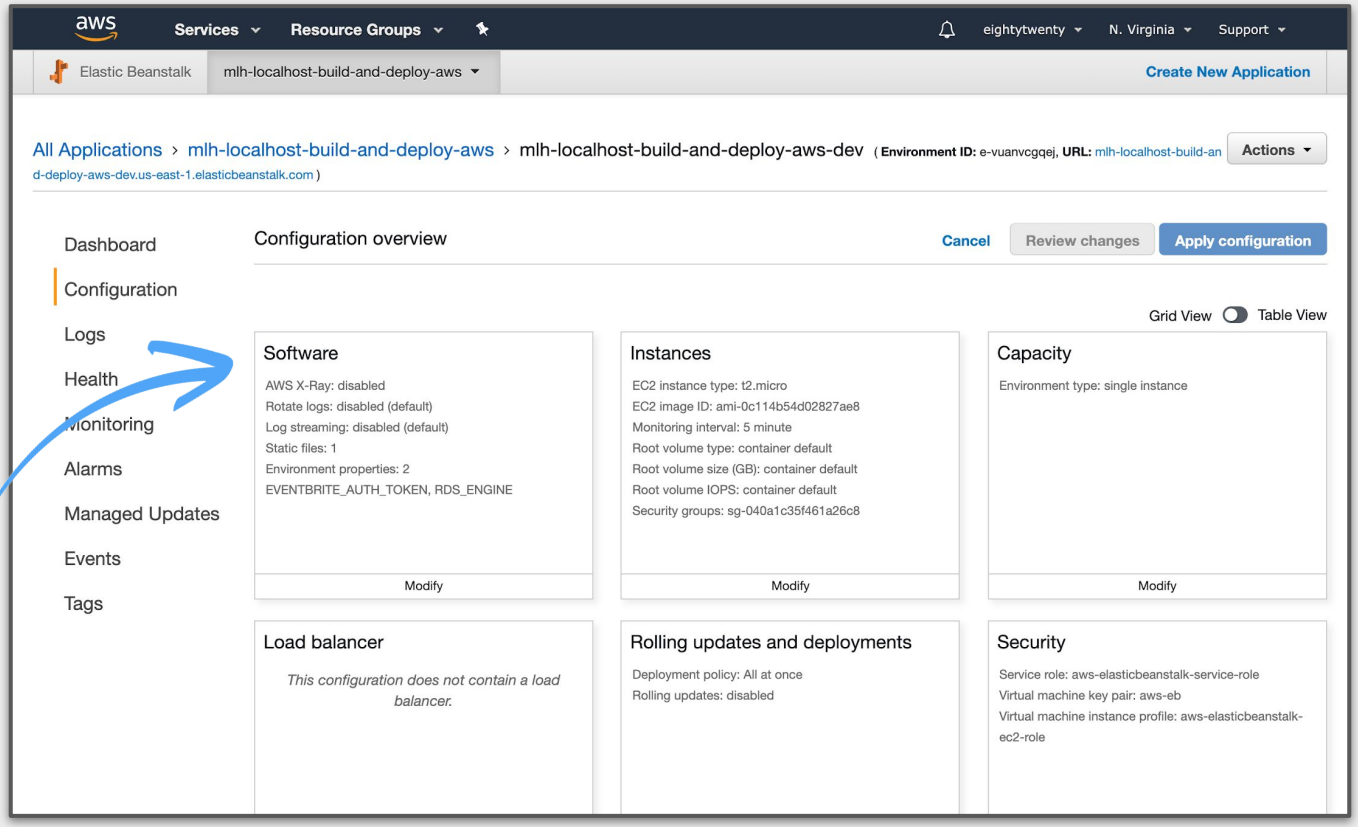
EB Console to the Rescue

The EB Console will automatically show us our web app instance. It also allows check our Environment variables as well. Let's go ahead and do this. Click on **Configuration** on the left-hand side.



EB Console to the Rescue

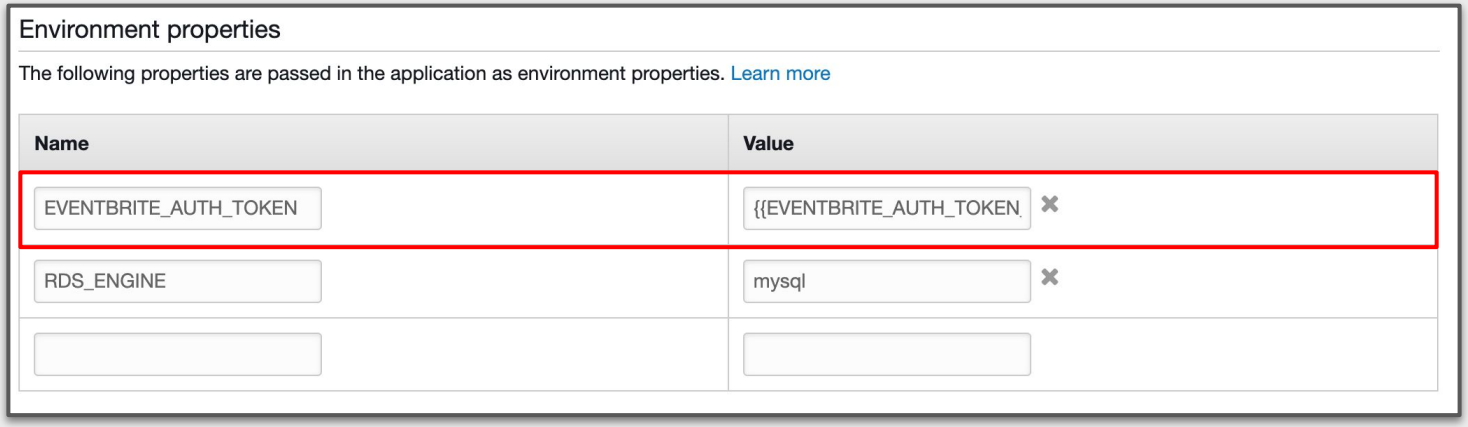
Select "**Grid View**" with the toggle in the top right



Click **Software**

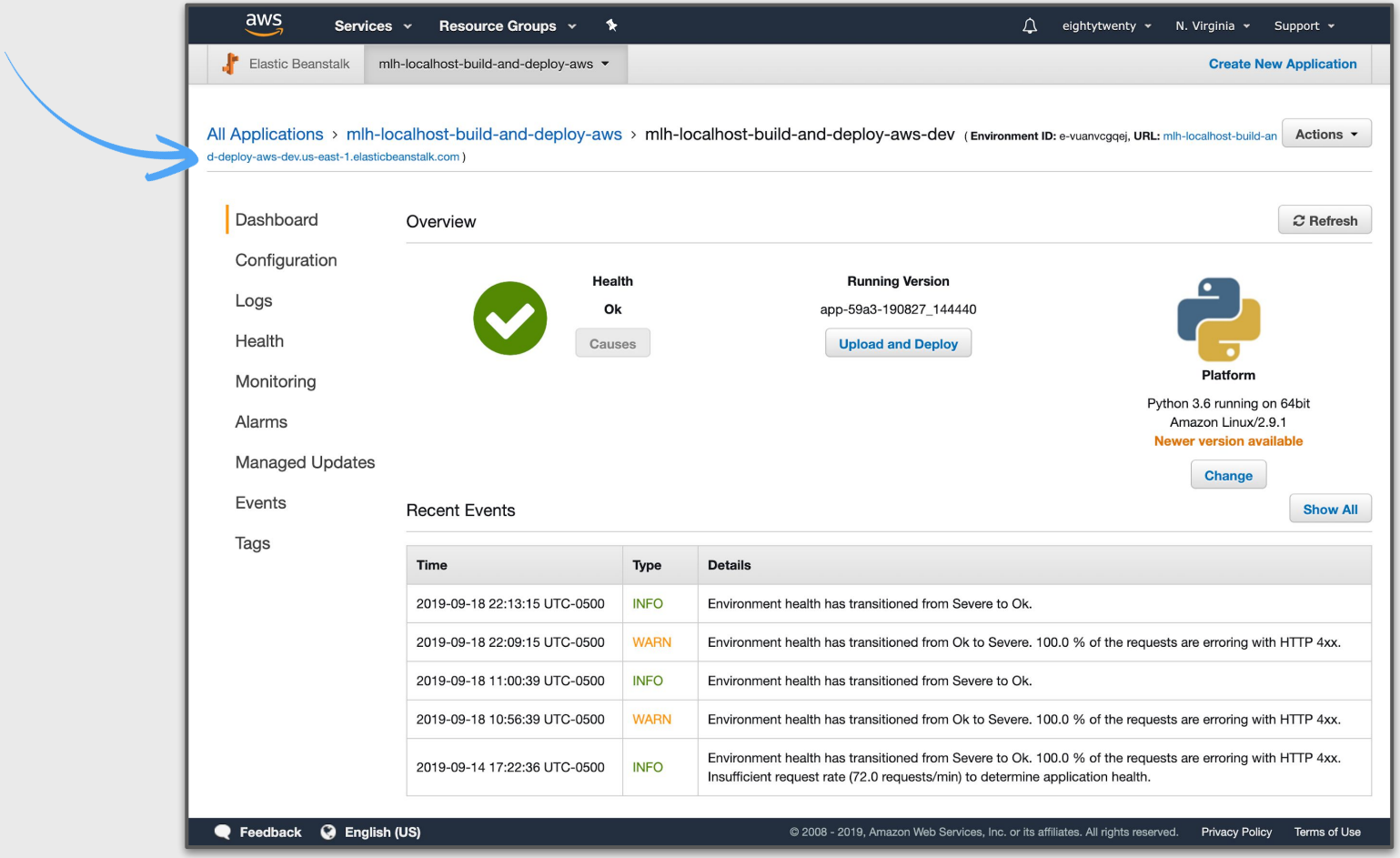
EB Console to the Rescue

1. Scroll down to **Environment Properties**
2. We need to add our Eventbrite API Key, the one we used **locally**
3. Add Name as **`EVENTBRITE_AUTH_TOKEN`** and Value as your **API key** (can be found in your .env file in VSC)



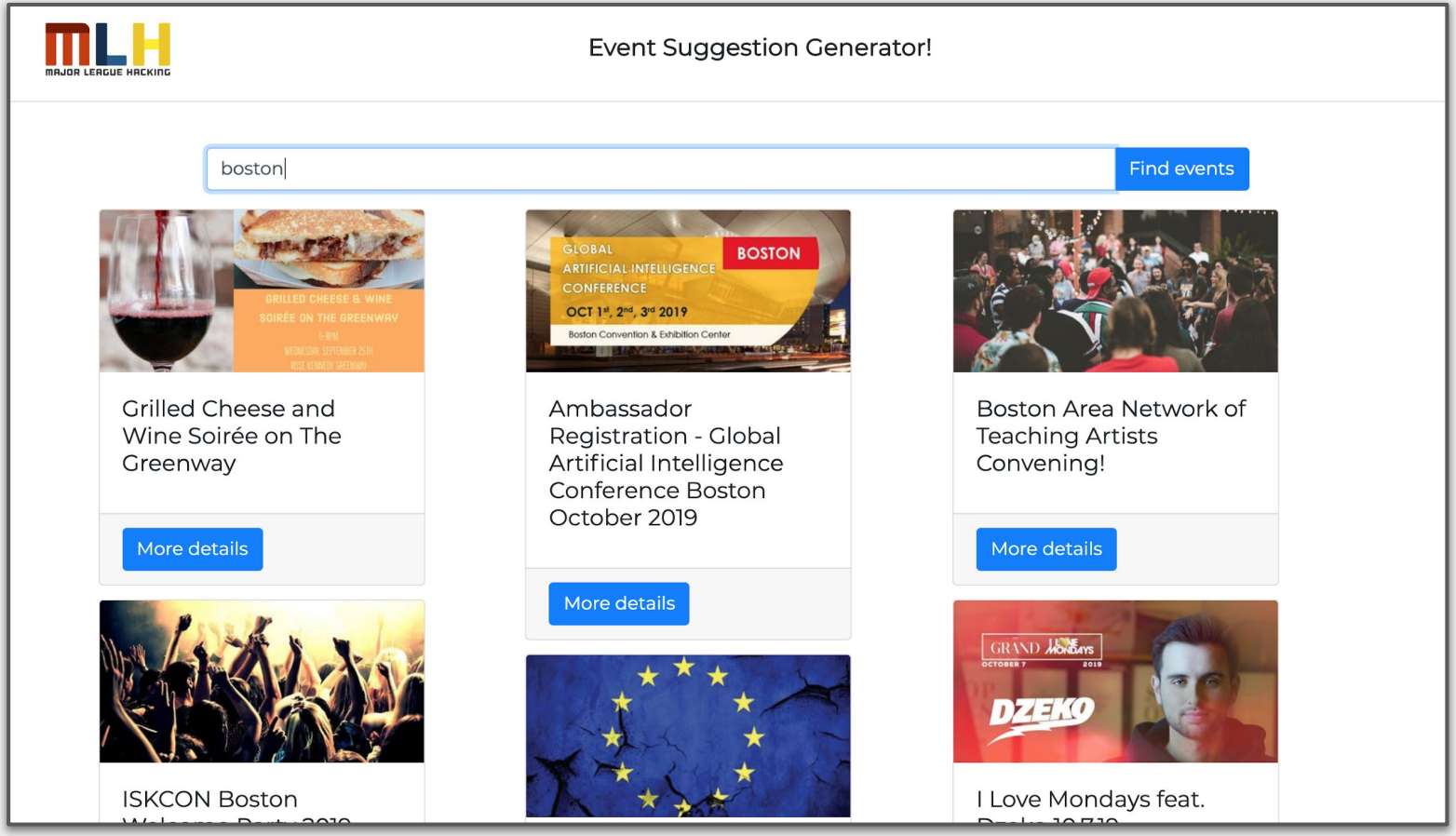
Let's check our app again

If you go back to your Dashboard, you can visit your app using the web URL AWS provides you!



Sweet Victory!

It works!



Whew! Take a breather!

We did a lot so pat yourself on the back. Let's recap what we did.

- Set up our Elastic Beanstalk instance.
- Created our environment
- Modified our environment variables on the EB Console

Table of Contents

1. Full Stack & Application Introduction
2. Setting up our Dev Environment
3. Creating an AWS Account
4. Creating an IAM User and Group
5. Building and Running the Application
6. Connecting to the Eventbrite API
7. Deploying to AWS Elastic Beanstalk
8. Review & Next Steps

Let's Recap

What did you learn?

1. How to run a full-stack web app with **Flask**, a Python web framework, in our Local Environment
2. How to deploy apps to **AWS Elastic Beanstalk** using the AWS EB Web Console and the **Eventbrite API**
3. How to configure our **AWS Elastic Beanstalk** with an **RDS DB** and Environment variables.

What did you learn today?

We created a fun quiz to test your knowledge and see what you learned from this workshop.

http://mlhlocal.host/quiz

Learning shouldn't stop when the workshop ends...

Check your email for access to:

- These workshop slides
- Practice problems to keep learning
- Deeper dives into key topics
- Instructions to join the community
- More opportunities from MLH!


MLH localhost
Workshop
Build and Deploy a Full Stack Web App with AWS
Presented by
aws educate
MLH
MAJOR LEAGUE HACKING
v.0.0.1