LESSON 1

In lesson 1, you are going to set up the Serverless Framework and create an IAM user. You will create an empty Serverless project, run a Lambda function locally, and deploy it to AWS.

You need to have the following prerequisites in order to do this workshop:

- An AWS account. You can register at aws.amazon.com
- Node.js 6.10 (and npm) installed on your computer (nodejs.org)
- AWS SDK installed on your computer (https://aws.amazon.com/sdk-for-node-js/)

PLEASE CREATE ALL YOUR AWS RESOURCES IN THE N. VIRGINIA REGION (US-EAST-1)

1. INSTALL AWS SDK

The AWS SDK is a prerequisite for this workshop. So … if you haven't installed it, open your terminal and type:

```
npm install aws-sdk
```

You don't need to configure anything at this stage, so please move on to the next step.

2. INSTALL SERVERLESS FRAMEWORK

You are going to use the Serverless Framework for our chatbot, so you'll need to install it. Open your terminal and type:

```
npm install serverless@1.10.0 -g
```

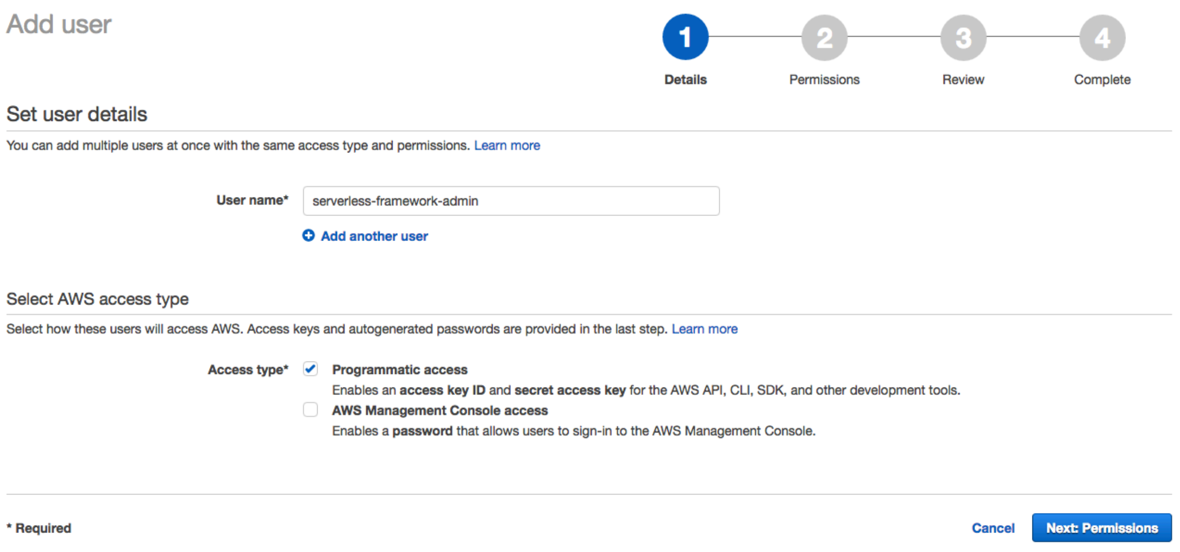
The above line installs Serverless Framework version 1.10.0 – this version is ***required*** for this workshop.

Pro tip: if you get Serverless Framework warnings about future upcoming changes, you can hide them by running `export SLS_IGNORE_WARNING=*` on Mac or `set SLS_IGNORE_WARNING=*` on Windows via the terminal.

3. SET UP YOUR AWS CREDENTIALS

Serverless Framework works with your AWS account. You are going to create a new Identity and Access Management User (IAM) that will allow Serverless Framework to perform actions (such as creating and updating Lambda functions) on your behalf.

- Open the AWS console by going to aws.amazon.com.
- Click **IAM.**
- Click **Users** and then **Add User.**
- Set the User name as **serverless-framework-admin.**
- Enable the **Programmatic access** checkbox.
- Click **Next: Permissions.**



Add user

Set user details

You can add multiple users at once with the same access type and permissions. Learn more

User name* serverless-framework-admin

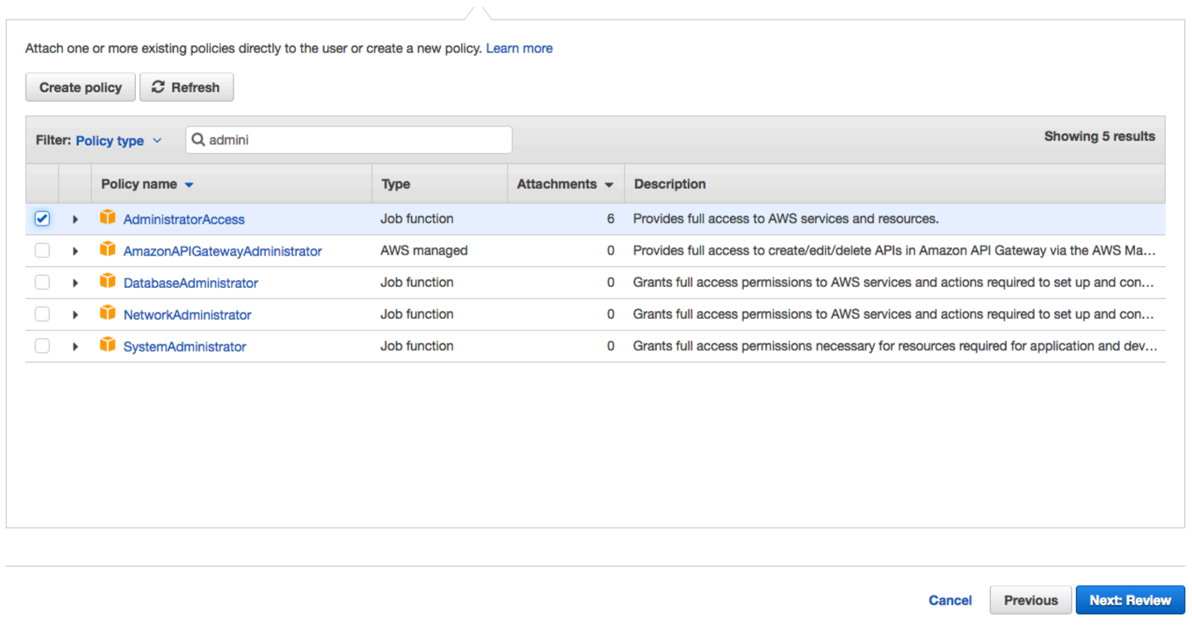
Select AWS access type

Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. Learn more

Access type* [✓] **Programmatic access**
Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.
[ ] **AWS Management Console access**
Enables a **password** that allows users to sign-in to the AWS Management Console.

* Required

- Click **Attach existing policies directly**.
- Search for **AdministratorAccess** and select it.
- Click **Next: Review**.



Attach one or more existing policies directly to the user or create a new policy. Learn more

Filter: Policy type admini — Showing 5 results

		Policy name	Type	Attachments	Description
✓	▸	AdministratorAccess	Job function	6	Provides full access to AWS services and resources.
	▸	AmazonAPIGatewayAdministrator	AWS managed	0	Provides full access to create/edit/delete APIs in Amazon API Gateway via the AWS Ma...
	▸	DatabaseAdministrator	Job function	0	Grants full access permissions to AWS services and actions required to set up and con...
	▸	NetworkAdministrator	Job function	0	Grants full access permissions to AWS services and actions required to set up and con...
	▸	SystemAdministrator	Job function	0	Grants full access permissions necessary for resources required for application and dev...

- Click **Create user**.
- Click the **Download .csv** button to download the Access Key ID and Secret Access Key of the user. Alternatively, you can copy and paste these keys to a safe place. ***You will need both of these keys later, so don't lose them.***
- Click **Close** when you are finished.

4. CONFIGURE AWS CREDENTIALS WITH THE SERVERLESS FRAMEWORK

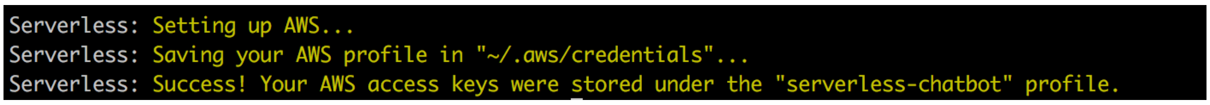
We have our AWS Access and Secret keys. Now, we need to make sure that the Serverless Framework can use these credentials.

Advanced note: you are going to create a custom profile for your credentials on your computer. This will allow you to avoid overwriting any of the other profiles you might already have set up (you might not have any other profiles which is OK, too).

In your terminal type the following, Replacing <YOUR ACCESS KEY> and <YOUR SECRET KEY> with the actual keys you created in the previous step:

```
serverless config credentials --provider aws --key <YOUR ACCESS KEY> --
secret <YOUR SECRET KEY> --profile serverless-chatbot
```

You should get the following success message:



```
Serverless: Setting up AWS...
Serverless: Saving your AWS profile in "~/.aws/credentials"...
Serverless: Success! Your AWS access keys were stored under the "serverless-chatbot" profile.
```

Pro tip: if you saved a CSV file in the previous step, open it in a ***plain text*** editor, like Sublime, Atom, or Text Wrangler. If you open it in Excel, it can (not so helpfully) change certain characters.

5. SLACK EVENTS VS REAL TIME MESSAGING API

There are 2 APIs for Slack bots – a Real Time Messaging API (https://api.slack.com/rtm) and an Events API (https://api.slack.com/events-api). We are going to use the Events API for this Bot project.

One advantage of using the Events API is that we can give Slack an endpoint to which it will deliver events as they happen. This makes integrating a Slack bot with AWS Lambda & API Gateway straightforward.

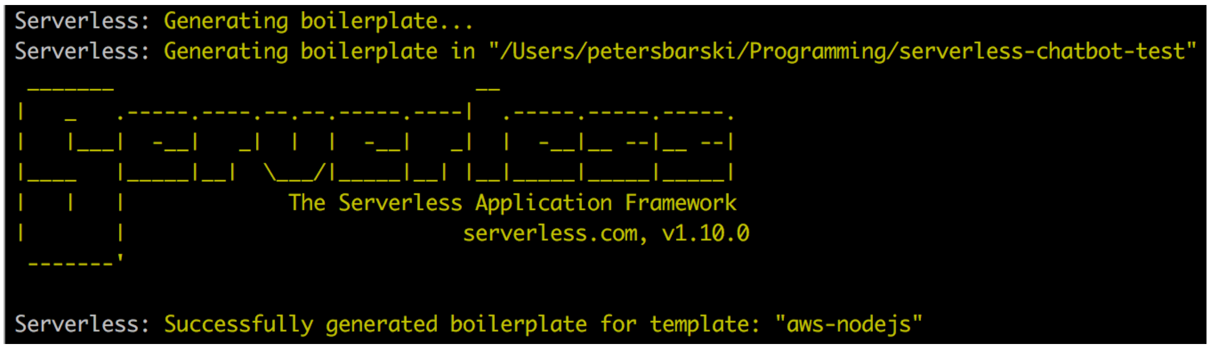
Check out https://api.slack.com/events-api for more information about the kinds of things it can do.

6. CREATE A NEW SERVERLESS SERVICE

Now we begin creating our chatbot. Open your terminal and run the following command:

```
serverless create --template aws-nodejs --path serverless-chatbot
```

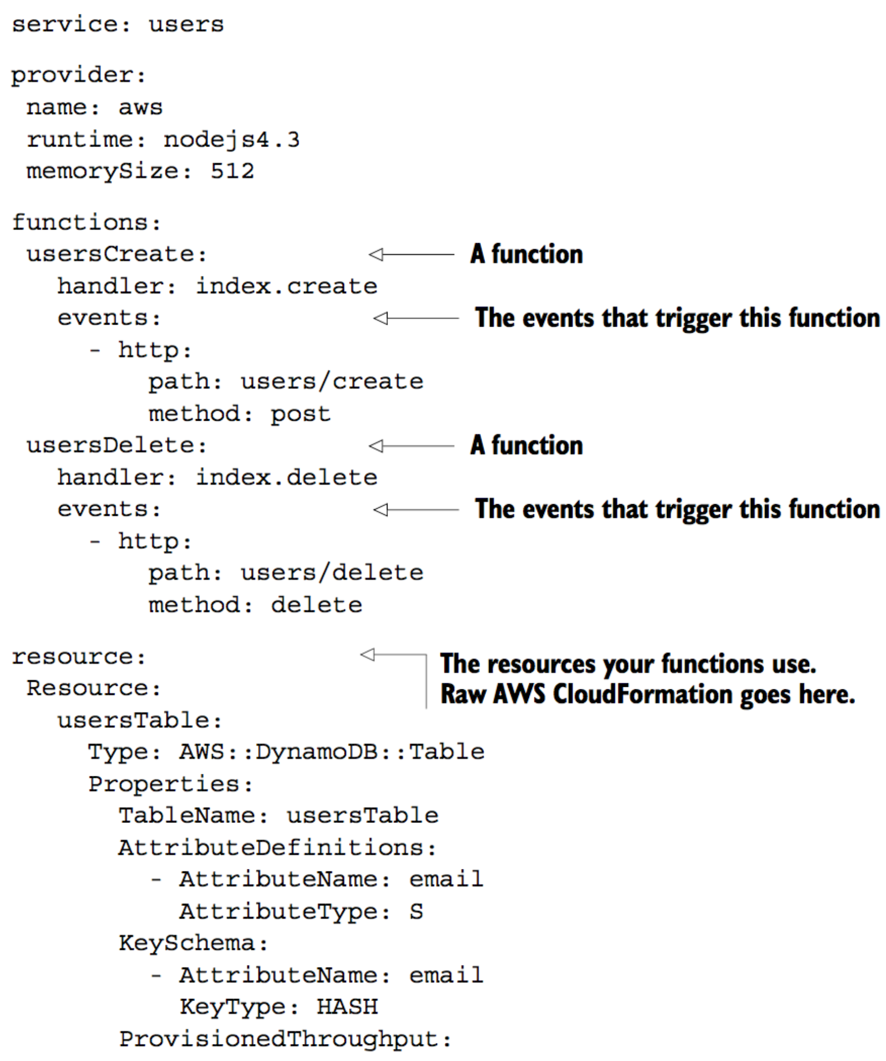
You should see the success message shown below this paragraph. The Serverless Framework has created a new directory called **serverless-chatbot** for you. Navigate to that directory and look at its contents. You should see three files: README.md, handler.js and serverless.yml.



```
Serverless: Generating boilerplate...
Serverless: Generating boilerplate in "/Users/petersbarski/Programming/serverless-chatbot-test"
 _______                             __
|   _   .-----.----.--.--.-----.----|  .-----.-----.-----.
|   |___|  -__|   _|  |  |  -__|   _|  |  -__|__ --|__ --|
|____   |_____|__|  \___/|_____|__| |__|_____|_____|_____|
|   |   |             The Serverless Application Framework
|       |                           serverless.com, v1.10.0
 -------'

Serverless: Successfully generated boilerplate for template: "aws-nodejs"
```

7. THE MAGIC OF SERVERLESS.YML

A **service** is the Framework's unit of organization. You can think of it as a project (though you can have multiple services for a single project or application). A service is where you define your functions, the events that trigger them, and the resource your functions use, all in a single file called **serverless.yml**. The following figure shows what a typical **serverless.yml** can look like.

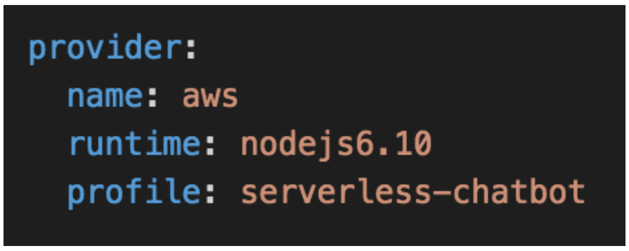

A function
The events that trigger this function
A function
The events that trigger this function
The resources your functions use. Raw AWS CloudFormation goes here.

```yaml
service: users

provider:
 name: aws
 runtime: nodejs4.3
 memorySize: 512

functions:
 usersCreate:
   handler: index.create
   events:
     - http:
         path: users/create
         method: post
 usersDelete:
   handler: index.delete
   events:
     - http:
         path: users/delete
         method: delete

resource:
 Resource:
   usersTable:
     Type: AWS::DynamoDB::Table
     Properties:
       TableName: usersTable
       AttributeDefinitions:
         - AttributeName: email
           AttributeType: S
       KeySchema:
         - AttributeName: email
           KeyType: HASH
       ProvisionedThroughput:
```

8. PROFILE

In step 4, you created a new AWS profile on your computer called serverless-chatbot. Your task here is to add this profile to **serverless.yml** so that it knows which credentials to use to deploy your bot.

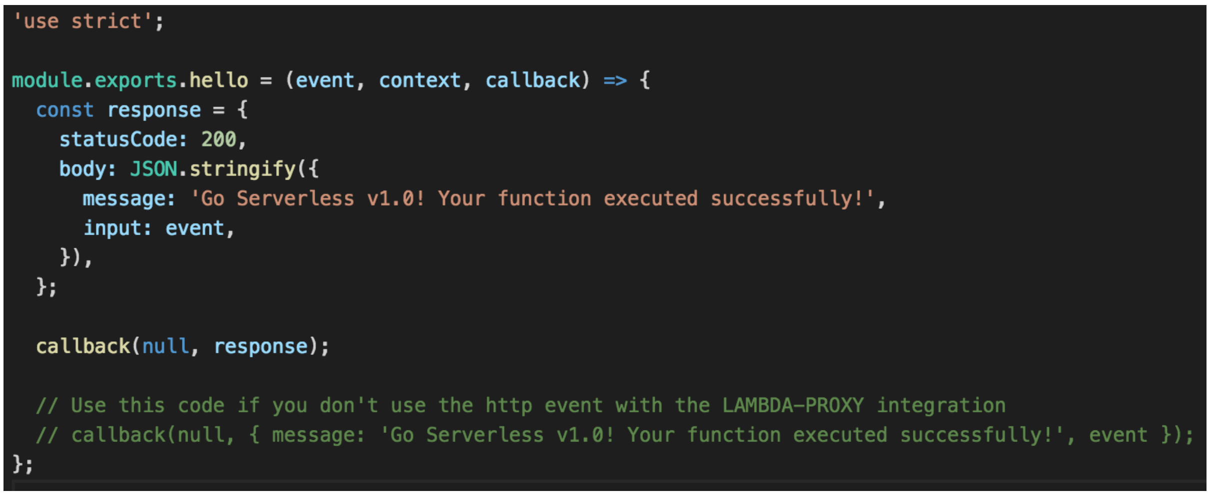
- Open **serverless.yml** in your favorite text editor.
- Under **runtime**, add **profile: serverless-chatbot** as seen below.



```yaml
provider:
  name: aws
  runtime: nodejs6.10
  profile: serverless-chatbot
```

9. HANDLER

If you look in the directory where you ran the `serverless create` command, there's another file called **handler.js**. This file contains your function code. The function definition in **serverless.yml** will point to this **handler.js** file and the function will be exported here. If you open the file in a text editor, you should see a function that returns a response with a message.

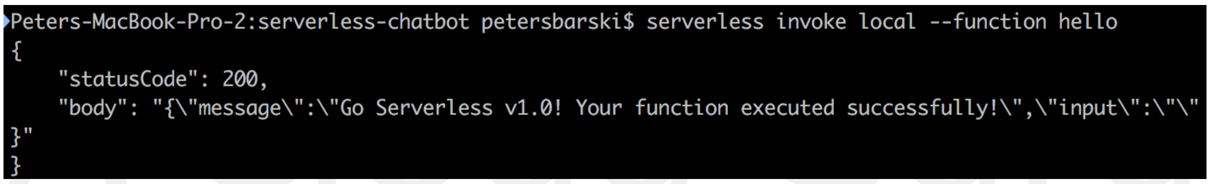


```
'use strict';

module.exports.hello = (event, context, callback) => {
  const response = {
    statusCode: 200,
    body: JSON.stringify({
      message: 'Go Serverless v1.0! Your function executed successfully!',
      input: event,
    }),
  };

  callback(null, response);

  // Use this code if you don't use the http event with the LAMBDA-PROXY integration
  // callback(null, { message: 'Go Serverless v1.0! Your function executed successfully!', event });
};
```

10. RUNNING LOCALLY

The Serverless Framework offers a command to run your AWS Lambda functions on AWS Lambda after they've been uploaded. Additionally, the Framework allows you to run your AWS Lambda functions locally via a powerful emulator, so you don't have to re-upload your functions every time you want to run your code. To invoke the hello function in **handler.js** enter the following in your terminal, as seen in the screenshot below:

```
serverless invoke local --function hello
```



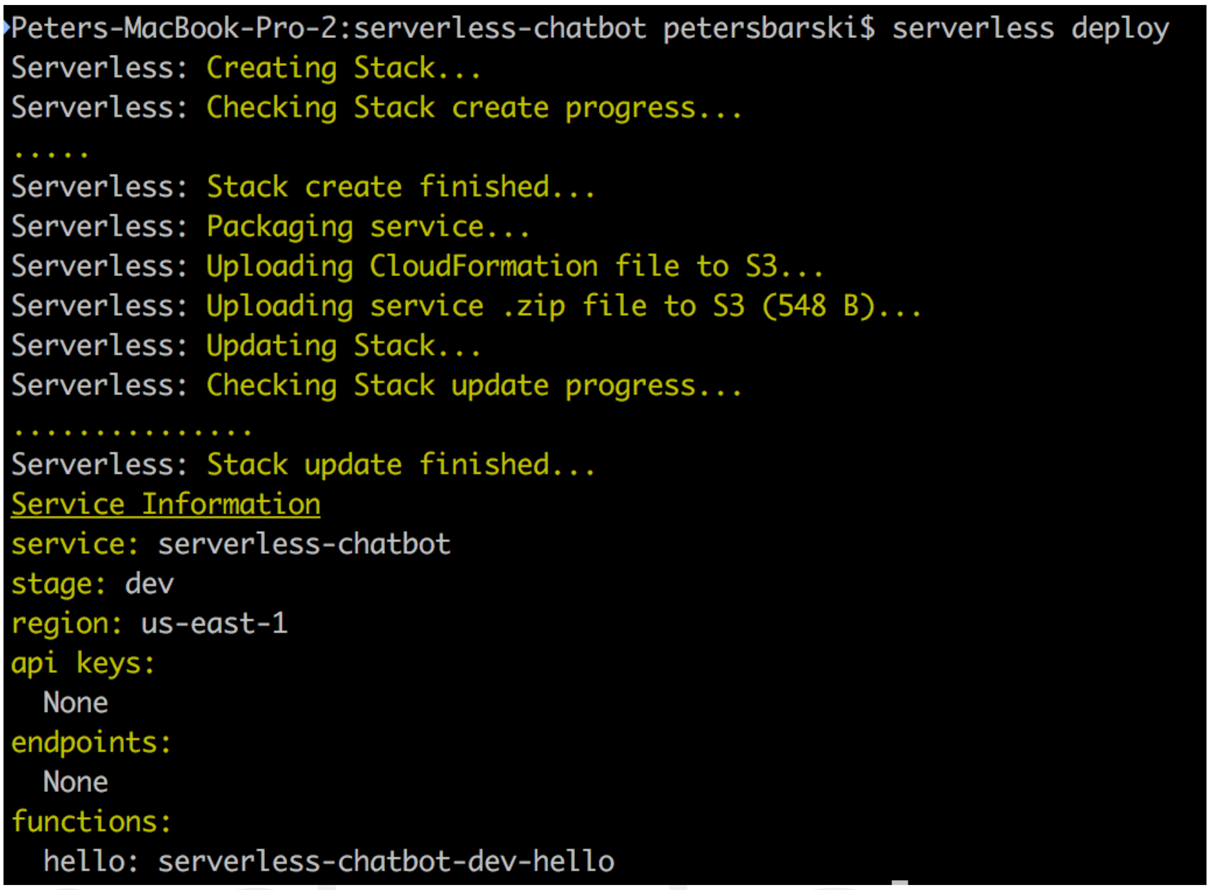
```
Peters-MacBook-Pro-2:serverless-chatbot petersbarski$ serverless invoke local --function hello
{
    "statusCode": 200,
    "body": "{\"message\":\"Go Serverless v1.0! Your function executed successfully!\",\"input\":\"\"
}"
}
```

11. FIRST DEPLOYMENT

Let's deloy you "hello world" function to AWS. You need to make sure deployments work before we go any futher. Type and execute the following in your terminal:

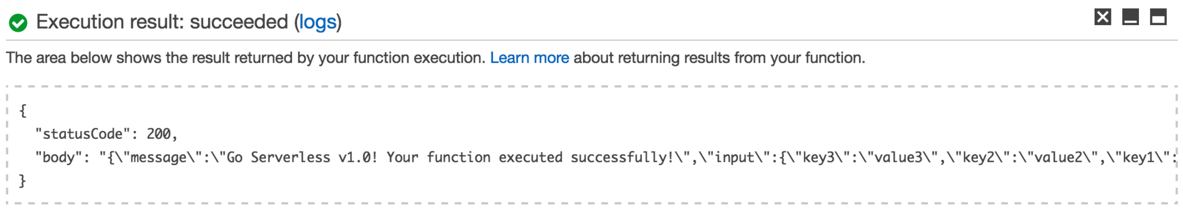
```
serverless deploy
```

The deployment may take a couple of minutes, but you will be able to see the steps the Framework takes in executing it. Follow along in your Terminal as the deployment executes.



```
Peters-MacBook-Pro-2:serverless-chatbot petersbarski$ serverless deploy
Serverless: Creating Stack...
Serverless: Checking Stack create progress...
.....
Serverless: Stack create finished...
Serverless: Packaging service...
Serverless: Uploading CloudFormation file to S3...
Serverless: Uploading service .zip file to S3 (548 B)...
Serverless: Updating Stack...
Serverless: Checking Stack update progress...
...............
Serverless: Stack update finished...
Service Information
service: serverless-chatbot
stage: dev
region: us-east-1
api keys:
  None
endpoints:
  None
functions:
  hello: serverless-chatbot-dev-hello
```

- After the deployment has finished, jump in to the **AWS console** and click **Lambda**.
- Find your function in the list (it should be called **serverless-chatbot-dev-hello**) and click on it.
- Click **Test**, then click **Save and test** in the window that appears.
- You should see the same execution result as before (when you ran the function locally.)



Execution result: succeeded (logs)

The area below shows the result returned by your function execution. Learn more about returning results from your function.

```
{
  "statusCode": 200,
  "body": "{\"message\":\"Go Serverless v1.0! Your function executed successfully!\",\"input\":{\"key3\":\"value3\",\"key2\":\"value2\",\"key1\":
}
```

That's it for this lesson. See you in the next!