Getting Started with AWS

Deploying a Web Application

Getting Started with AWS: Deploying a Web Application

Table of Contents

Welcome

Welcome to *Getting Started with AWS: Deploying a Web Application*.

This guide provides a tutorial on deploying a web application with Amazon Web Services (AWS). Using AWS, you can develop applications quickly and then deploy them to a cloud environment that scales on demand. And with several AWS deployment services to choose from, you can create a deployment solution that gives you the right mix of automation and control.

For this tutorial, we'll use AWS Elastic Beanstalk to deploy our application. Many of the concepts covered in this guide apply to the other AWS deployment services too. If you're currently running a web application on an on-premises server and you're thinking about migrating to the cloud, this guide is a great place to start. By deploying a simple but fully functional web app, you'll learn the basics of working with an AWS deployment service.

Our example app is built with Node.js, though you don't need any experience with Node to complete this tutorial. Our app stores user information in an Amazon DynamoDB table and sends notifications with Amazon Simple Notification Service, so you'll learn a little bit about DynamoDB and SNS. You'll also get a brief introduction to managing permissions with AWS Identity and Access Management.

To complete this tutorial, you just need an Amazon Web Services account and an hour or so of free time. If you haven't signed up for AWS, you can do that now. Go to the AWS home page and click the **Sign Up** button.

Okay, let's get started!

Overview of the Project

Let's say that you're a startup with a Really Big Idea for a new web application. And let's say that you're working hard on your web app, but it isn't ready for production yet. What do you do in the meantime to generate interest in your project? One option is to deploy a small placeholder app that collects contact info from interested site visitors. That's exactly the sort of application we'll deploy in this tutorial. Our signup app will help us get in touch with potential users—people who might become early adopters or take part in a private beta test. For this tutorial, our app only collects contact info. But we could add functionality that sends invitations or interacts with users in some other programmatic way.

Let's take a quick tour of AWS Elastic Beanstalk and the other technologies we'll be using. If you want to dive right into the hands-on part of the tutorial, feel free to skip ahead to the next section.

AWS Elastic Beanstalk

AWS Elastic Beanstalk is a high-level deployment tool that helps you get an application from your desktop to the web in a matter of minutes. Elastic Beanstalk automatically handles the details of your hosting environment—capacity provisioning, load balancing, scaling, and application health monitoring—so you don't have to.

Elastic Beanstalk supports applications developed in Java, PHP, .NET, Node.js, Python, and Ruby, as well as different container types for each language. A container defines the infrastructure and software stack to be used for a given environment. When you deploy your app, Elastic Beanstalk provisions one or more Amazon EC2 instances, among other resources. The software stack running on EC2 is dependent on the container type. For example, Elastic Beanstalk supports two container types for Node.js: a 32-bit Amazon Linux image and a 64-bit Amazon Linux image, each running a software stack tailored to hosting a Node.js application.

You can interact with Elastic Beanstalk by using the AWS Management Console, the AWS Command Line Interface (CLI) or eb, a high-level CLI designed specifically for Elastic Beanstalk. For this tutorial, we'll use the management console.

The Node.js Application

In this tutorial, we're going to deploy an application that a startup might use as a placeholder for their project. The example app lets customers submit contact information and express interest in a preview.

The app is built on Node.js, a relatively new platform that uses server-side JavaScript to build network applications. At its core, Node.js consists of a library and a runtime, the latter provided by the V8 JavaScript Engine. Node.js is designed around a nonblocking, event-driven I/O model, making it particularly useful for creating highly scalable web servers. Our app employs two external Node modules: Express, a web application framework, and Jade, a Node.js template engine that can be used to create HTML documents.

AWS provides a Node.js SDK, which helps take the complexity out of coding by providing JavaScript objects for AWS services. We've used the Node.js SDK to build our sample application. To learn more about the Node.js SDK, see AWS SDK for JavaScript in Node.js.

To make our app pretty, we're using Bootstrap, a mobile-first front-end framework that started as a Twitter project.

Amazon DynamoDB

Our application needs to store the contact information that users submit. To do so, we'll use Amazon DynamoDB, a NoSQL database service. DynamoDB is a schemaless database, so you need to specify only a primary key attribute. For our app, we'll use an `email` field as a key.

Amazon Simple Notification Service

We want to be notified when customers submit a form, so we'll use Amazon Simple Notification Service. Amazon SNS is a push messaging service that can deliver notifications over various protocols. For our app, we'll push notifications to an email address.

Deploying the App

First, we need to get the code for our application. We can do that at the AWS Labs GitHub repository. Go to the eb-node-express-signup repo and click **Download ZIP**. Or download the archive directly.

We're going to make a few changes to the code, so go ahead and unzip the archive. In the top-level directory you'll see a file called app_config.json. Open this file in a text editor. As you can see, the file contains a JSON object with three name-value pairs. The value strings are currently empty.

```
{
  "AWS_REGION": "",
  "STARTUP_SIGNUP_TABLE": "",
  "NEW_SIGNUP_TOPIC": ""
}
```

We're going to use this JSON to tell our application which region to connect to and which DynamoDB table and SNS topic to use. In order to do that, we first need to create the table and the topic.

Creating a DynamoDB Table

Let's create a DynamoDB table for our application.

1. Sign in to the AWS Management Console and open the Amazon DynamoDB console at https://console.aws.amazon.com/dynamodb/.
2. In the menu bar, in the region selector, click the region where you want to deploy your application. For this tutorial, let's use **US West (Oregon)**.
3. Click **Create Table**.
4. In the Create Table wizard, in the Table Name box, type **`my-startup-signup-table`**.
5. For the Primary Key Type, click **Hash**.
6. Under Hash Attribute Name, click String. In the corresponding box, type **`email`**. Click **Continue**.
7. We're not going to add any additional indexes, so on the Add Indexes page, click **Continue**.
8. For our sample app, we can use the minimum provisioned throughput capacity for our table. Both the read and write capacity units should be set to "1" by default, and the **Help me calculate how much throughput capacity I need to provision** check box should be cleared. Click **Continue**.

9. Our app won't need any Throughput Alarms, so clear the **Use Basic Alarms** check box and then click **Continue**.
10. On the Review page, take a look at the table specifications. The values should look something like this:

    If everything looks correct, click **Create**. DynamoDB will start the table creation process. When the process is complete, the table status will be listed as ACTIVE.
11. Now that we have a database table, we can add the table name to the configuration file for the app. If you haven't done so already, open the application folder and open app_config.json in a text editor. The value for "STARTUP_SIGNUP_TABLE" is an empty string. Let's insert the name of the new database. The name-value pair should look like this:

```
"STARTUP_SIGNUP_TABLE": "my-startup-signup-table",
```

    You can leave the file open, but be sure to save your edits.

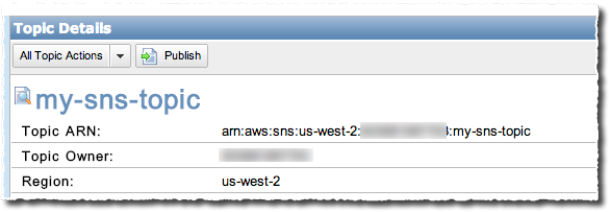
You've created the DynamoDB table. Now let's create the SNS topic.

Creating an SNS Topic

Our app is designed to notify the site owner of each new signup. When the data from the signup form is successfully written to the DynamoDB table, the app sends an SNS notification. To make this work, we need to create an SNS topic.

1. Sign in to the AWS Management Console and open the Amazon SNS console at https://console.aws.amazon.com/sns/.
2. We'll use the same region for SNS as we did for DynamoDB. Ensure that **US West (Oregon)** is still selected in the region selector, and then click **Create New Topic**.
3. In the Create New Topic dialog box, enter **`my-sns-topic`**. We can leave the Display Name blank. Click **Create Topic**.

4. Now we need to create a subscription that tells SNS how and where to send the notification. Click **Create Subscription**.
5. In the Create Subscription dialog box, select **Email** as the protocol. In the Endpoint box, type the email address that will receive the notification. (For this tutorial, you'll probably want to send the notification to your own email account, as you'll need to respond to a confirmation message.) Click **Subscribe**. In the confirmation message that appears, click **Close**.
6. The email address you provided should receive a message titled "AWS Notification — Subscription Confirmation." It may take a moment for the email to appear. Open the message and click **Confirm subscription**.
7. Now that we've created an SNS topic, we can plug the Topic ARN into our config file, so that the application knows where to send notifications. An ARN, or Amazon Resource Name, is a unique identifier for an AWS resource. We can get the Topic ARN from the SNS Management Console.



Copy the Topic ARN and paste it into app_config.json, in the empty value corresponding to "NEW_SIGNUP_TOPIC". The name-value pair should look like this (but with your particular topic ARN):

```
"NEW_SIGNUP_TOPIC": "arn:aws:sns:us-west-2:123456789012:my-sns-topic"
```

While you're at it, set the region to "us-west-2". The final app_config file should look like this, but with your specific value for the signup topic ARN:

```
{
  "AWS_REGION": "us-west-2",
  "STARTUP_SIGNUP_TABLE": "my-startup-signup-table",
  "NEW_SIGNUP_TOPIC": "arn:aws:sns:us-west-2:123456789012:my-sns-topic"
}
```

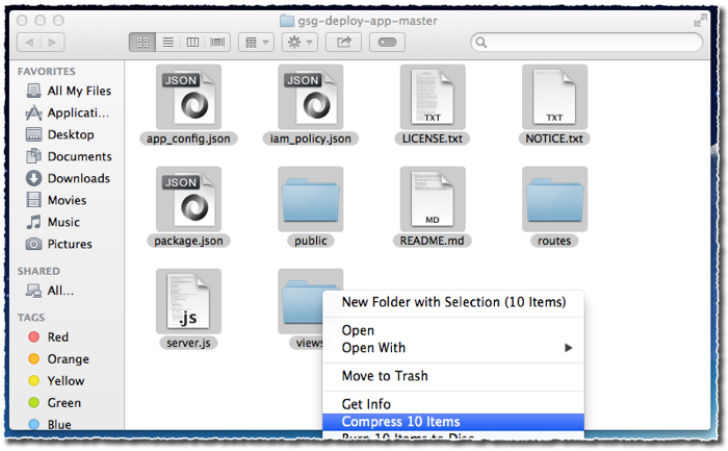
When you're finished updating the config file, save and close it. That's it for the SNS topic. Now let's use Elastic Beanstalk to roll out the app.

Deploying with Elastic Beanstalk

We've downloaded the source code and set the configuration values. Now, let's deploy the app with Elastic Beanstalk.

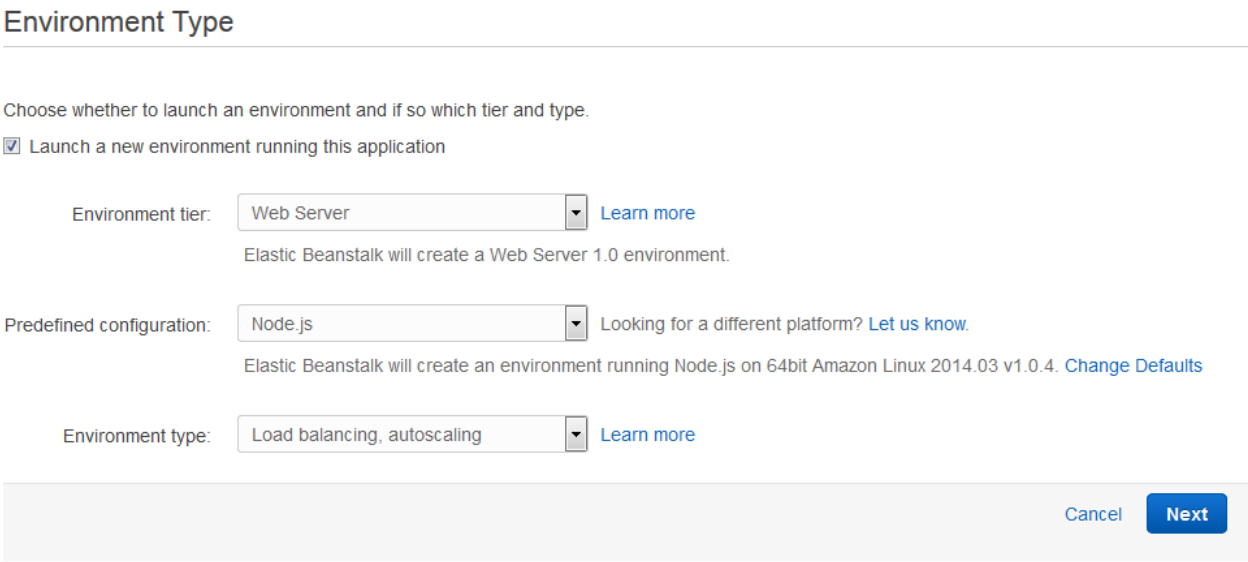
Elastic Beanstalk requires that your application be bundled as a .zip or .war file. But — and this is key —you have to compress the application files directly, rather than compressing the directory that contains them.

We already extracted the files from the sample app archive, so we'll need to compress them again. To do so, open the app folder (eb-node-express-signup-master), select all the items in the folder, and compress them.



For more detailed, platform-specific instructions on compressing the files, see Creating an Application Source Bundle.

1. Sign in to the AWS Management Console and open the AWS Elastic Beanstalk console at https://console.aws.amazon.com/elasticbeanstalk/.
2. Ensure that **US West (Oregon)** is still selected in the region selector, and click **Create New Application**.
3. Under Application Information, type a name for the application, and, if you'd like, a description. For this tutorial, let's call the application **`my-startup-app`**. When you're finished, click **Next**.
4. On the Environment Type page, In the **Predefined configuration** box, click **Node.js**. You shouldn't need to change any other settings. Your environment settings should look like this:



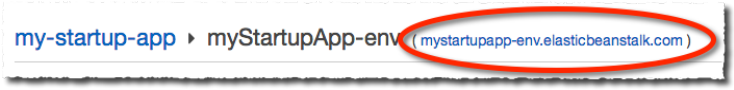
   Click **Next**.
5. On the Application Version page, select **Upload your own**, click **Browse**, and then select the compressed application file that you created. Click **Next**.
6. On the Environment Information page, type a unique environment name and environment URL. Click **Check Availability** to ensure your desired URL is available. Then, click **Next**.
7. On the Additional Resources page, ensure that both check boxes are cleared, and then click **Continue**.
8. The Configuration Details page presents a number of options, but for this tutorial we need to be aware of only one. In the **Instance profile** box, click **Create Default Profile**, and then click **Next**.

This will create the default instance profile for the application, which is **aws-elasticbeanstalk-ec2-role**. An instance profile is a container for an IAM role. You can use IAM to attach a policy to this instance profile. IAM policies grant permission to perform certain actions. In this case, we want an IAM policy that lets our app publish Amazon SNS notifications and put items to DynamoDB. Let's create that policy.

a. Leave the Configuration Details page as-is for now. We'll be back to it shortly. In the Services menu, find **IAM** and open it *in a new browser window*. In the IAM console, under **Dashboard** in the navigation pane, click **Roles**.
b. In the **Role Name** column, click the role **aws-elasticbeanstalk-ec2-role**.
c. Under **Permissions**, click **Attach Role Policy**.
d. On the **Set Permissions** page in the Manage Role Permissions wizard, click **Custom Policy**, and then click **Select**.
e. Click **Policy Name**, and then type **`my-startup-policy`**.
f. In the eb-node-express-signup-master directory (the downloaded, uncompressed archive), open iam_policy.json and copy its contents. Paste the contents into the **Policy Document** box.

   Click **Apply Policy**.

9. Close the IAM console and go back to the Elastic Beanstalk console. On the Configuration Details page, click **Next**.
10. On the Environment Tags page, click **Next**.
11. On the Review page, verify that all the settings are as you want them, and then click **Launch**. On the AWS Elastic Beanstalk dashboard, you can watch in real time as Elastic Beanstalk creates an environment and deploys the app. The process will take a few minutes. When the deployment is finished and the environment health is listed as "Green", click the URL of the app.



You should see the deployed sample application. You can test it by filling out the signup form and verifying that you receive a notification.

Changing an Environment Variable

Now that our application is deployed, let's try setting an environment variable. You can access Node.js environment variables using `process.env.`*ENVVAR*, where ENVVAR is an environment variable used by the app. You can set the value of ENVVAR in the Elastic Beanstalk console. Let's see how it works.

Our application's server.js file contains the following line of code:

```
app.locals.theme = process.env.THEME;
```

This makes the THEME environment variable available to the application. We use this variable in one of our views, layout.jade.

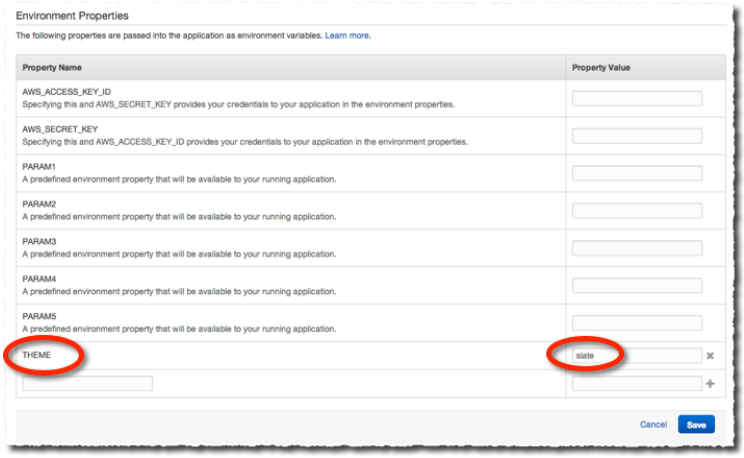
```
- var setTheme = theme
if setTheme
  link(href="../static/bootstrap/css/theme/" + setTheme + "/bootstrap.css",
```

```
rel="stylesheet")
  else
    link(href="../static/bootstrap/css/theme/flatly/bootstrap.css",
rel="stylesheet")
```

The theme variable changes the CSS that's applied to our app. By default, the `flatly` theme is used. But we can pass in other themes: `amelia`, `default`, `slate`, and `united`. To see how this works, explore public/static/bootstrap/css/ in the archive.

Okay, let's try changing a variable on our deployed app.

1. In the Elastic Beanstalk console, in the navigation pane for your environment, click **Configuration**.
2. Open **Software Configuration** and scroll down to the table of Environment Properties. At the bottom of the table, add the name **`THEME`** and the value **`slate`**.



   Click **Save**.
3. When the environment is done updating, click the URL for the app. We have a brand new look!

Troubleshooting with Logs

What if there's a problem? Say you followed all of the steps above, and you clicked the URL, and ... no app. With our sample app, which runs on an nginx server, a deployment problem is likely to result in a "502 Bad Gateway" message. The "502" message is not very informational. To troubleshoot a deployment issue, you may need to use the logs that are provided by AWS Elastic Beanstalk.

For example, let's say that, in the process of updating app_config.json, you accidentally leave off a quotation mark. Then, when you finish the deployment, you see the 502 error. How do you determine what happened, and how do you fix the problem? Let's check the logs.

1. In the Elastic Beanstalk console, in the navigation pane for your environment, click **Logs**.
2. At the Logs page, click **Snapshot Logs**. Wait for your environment to update, and then click **View log file**. In the log file, you can see just what happened on the server side during deployment and runtime. There's a lot of material to sort through, and we're not going to cover the different sections of the log in this tutorial. But if you did indeed leave out a quotation mark in the config file and you scrolled through the log to `/var/log/nodejs/nodejs.log`, you'd find an error similar to this:

```
SyntaxError: Unexpected token u
    at Object.parse (native)
```

```
    at Object.<anonymous> (/var/app/current/server.js:23:15)
    at Module._compile (module.js:449:26)
    at Object.Module._extensions..js (module.js:467:10)
    at Module.load (module.js:356:32)
    at Function.Module._load (module.js:312:12)
    at Module.runMain (module.js:492:10)
    at process.startup.processNextTick.process._tickCallback (node.js:245:9)

undefined:2
  "AWS_REGION": us-west-2",
                ^
```

In this case, the "Unexpected token u" message appears because the parser expected a quotation mark instead of a "u" in the string `us-west-2"`. Now that we've found the problem, we can fix it.

3. If you were actually troubleshooting this issue, you'd go back to the application code in your local environment and fix the missing quotation mark. Then you'd need to create a new .zip archive to upload.
4. To redeploy the app, go to Dashboard, click **Upload and Deploy**, and choose your updated .zip file.
5. If necessary, change the version label to something new. For example, if your first deployment was labeled "Archive", you can label this second deployment "Archive-2" or something like that.
6. Click **Deploy**. Elastic Beanstalk will update the environment.
7. When the environment is updated and listed as "Green", go to the app URL and test your app.

Of course, you'd normally try to catch a config error in development. But if an error does get through to production or you just want to update your app, Elastic Beanstalk makes it fast and easy to redeploy.

Additional Resources

This tutorial has focused on AWS Elastic Beanstalk, but that's only one of the deployment solutions available with AWS. You can also use AWS OpsWorks to deploy and manage applications, and if you want a do-it-yourself experience you can use Amazon Elastic Compute Cloud, AWS CloudFormation, Amazon CloudWatch, and Auto Scaling to build your own deployment framework. To learn more about choosing the right deployment solution, see Application Management for AWS.

To learn more about Elastic Beanstalk and other AWS deployment options, check out the resources below.

More on AWS Elastic Beanstalk

- For complete documentation on managing applications with AWS Elastic Beanstalk, see the AWS Elastic Beanstalk Developer Guide.
- To learn about eb, the updated Elastic Beanstalk CLI, see Eb Command Line Interface.
- To learn how to deploy a Node.js app with eb, see Deploying AWS Elastic Beanstalk Applications in Node.js Using Eb and Git.
- To learn more about the components and architecture of an Elastic Beanstalk app, see How Does AWS Elastic Beanstalk Work?.
- By including a configuration file with your source bundle, you can customize your environment at the same time that you deploy your application. To learn more, see Customizing and Configuring AWS Elastic Beanstalk Environments.

Other AWS Deployment Solutions

- To learn more about AWS OpsWorks, see AWS OpsWorks User Guide.
- To learn more about the individual services you'd typically use for a do-it-yourself deployment, see Amazon Elastic Compute Cloud Getting Started Guide, AWS CloudFormation User Guide, Amazon CloudWatch Getting Started Guide, and Auto Scaling Getting Started Guide.
- The AWS Toolkit for Visual Studio includes a deployment tool. To learn more, see Standalone Deployment Tool.
- The AWS Toolkit for Eclipse also provides deployment options. To learn more, see Getting Started with the AWS Toolkit for Eclipse.

Document History

The following table describes important changes to *Getting Started with AWS: Deploying a Web Application*.

- **Latest documentation update:** January 9, 2014

Change	Description	Date Changed
Initial release	This is the first release of *Getting Started with AWS: Deploying a Web Application*.	January 9, 2014