Overview

During the last assignment, you built a highly available web application deployment by manually configuring a VPC, RDS database, and an EC2 auto scaling group. You probably figured out pretty quickly that this manual creation process involves many different components and it's easy to make a mistake during the configuration process.

You will use the Elastic Beanstalk service to deploy another scalable web application. You will learn how Platform as a Service can greatly simplify certain types of application deployments.

Elastic Beanstalk can greatly simplify an application deployment on AWS because it takes care of much of the underlying infrastructure configuration. While the service was originally designed for application developers with limited AWS knowledge, it's really a valuable service for any AWS user. As IT professionals, our goal is to build scalable, secure, and reliable IT services. A service architecture built around a Platform as a Service allows us to achieve these goals with greater repeatability and consistency.

Requirements

You need to have a personal AWS account and GitHub account for this assignment.

The assignment

It's time to launch some services using a Platform-as-a-Service (PaaS).

Launch management instance

We're going to take a different approach to creating AWS resources in this assignment. Until this point in the course, you have been working primarily with the AWS web console to create resources. In the real world we try to use web consoles as little as possible. Our preference is to define infrastructure as code rather than using point-and-click interfaces.

Why is this the case? The Elastic Beanstalk (EB) service provides a great example of why infrastructure as code is a preferred practice. AWS constantly updates and changes the web interface for the EB service. If you documented the web-based process for deploying an application on EB, you would have to update your documentation a couple times a year. Additionally, some EB users will see updates to the EB web console before other users. Good luck trying to keep all your team members happy.

The bottom line is that a code-based approach is easier to document, automate, and run. It is a much more scalable and reliable approach to building infrastructure. Let's start practicing what we preach.

Go ahead and create a new t2.micro linux instance called `mgmt1` in a default VPC in us-east-1 (N. Virginia) region. Yes, you can use the AWS console for this task (but this is the last time!).

Setup EB CLI configuration

Connect to the instance via ssh once it is available and install the Elastic Beanstalk CLI. The EB CLI is a command-line tool you can use to control the AWS EB service. It's kind of like a service-specific version of the AWS CLI. Type in the following command to install the EB CLI on the instance:

```
pip install awsebcli --upgrade --user
```

Next, clone the following Wordpress application git repository to the instance and change to the `seis665-wordpress` directory:

```
git clone https://github.com/rclc/seis665-wordpress.git
```

Inside the `seis665-wordpress` directory, run the `eb init` command:

```
eb init
```

First, the EB CLI prompts you to select a region. Type the number that corresponds to the region that you would like to use and press Enter.

```
us-east-1 (N. Virginia)
```

Next, provide your API access key and secret key so that the EB CLI can manage resources for you. Access keys are created in the AWS Identity and Access Management management console.

```
(aws-access-id): (type in your aws-access-id)
(aws-secret-key): (type in your aws-secret-key)
```

An application in Elastic Beanstalk is a resource that contains a set of application versions (source), environments, and saved configurations that are associated with a single web application. Each time you deploy your source code to Elastic Beanstalk using the EB CLI, a new application version is created.

The default application name is the name of the folder in which you ran `eb init`. Let's set this to `wordpress`:

```
Application name: wordpress
```

Next, the EB CLI should automatically detect that our application is using `PHP`. It will ask us what version of PHP our application needs. The PHP platform version we want is `5.6`.

We don't need to use Code Commit for this project, choose `no` when prompted.

We also don't need to connect to the instances created by EB, so choose `no` when asked to setup SSH.

Notice how the EB CLI setup a hidden subdirectory called `.elasticbeanstalk` within the current directory. The settings you just entered are stored in a YAML file within this subdirectory. Check it out:

```
cat .elasticbeanstalk/config.yml
```

Launch Wordpress environment

Now we need to create a new environment which will support our application. An environment consists of all the AWS resources needed to power the app — EC2 instances, ELB, RDS database, auto-scaling group, security groups, etc. Launch the environment by running the command:

```
eb create wordpress-dev --database --scale 2 --timeout 30
```

This command will ask a series of questions before launching the environment:

```
Load balancer type: classic
RDS DB name: wordpressdb
RDS DB master password: (choose one)
```

The EB CLI will automatically upload the Wordpress application to EB as a deployment package. Now watch as the EB service builds the infrastructure environment for your application. This process may take about 20 minutes.

NOTE: Elastic Beanstalk uses the access permissions associated with your IAM account to create resources such as EC2 instances and security groups. If the creation process fails, your IAM account may not have the proper permissions in place to create all the necessary resources. You should go into the IAM console and add the AdministratorAccess policy to your account.

After the launch process completes, check out the status of your EB environment by using the following command:

```
eb status
```

You should see something like:

```
Environment details for: wordpress-dev
Application name: wordpress
Region: us-east-1
Deployed Version: app-309e-170325_215522
Environment ID: e-2jxqcjvjkh
Platform: arn:aws:elasticbeanstalk:us-east-1::platform/PHP 5.6 running on
64bit Amazon Linux/2.3.2
Tier: WebServer-Standard
CNAME: jbaker-wordpress-dev.us-east-1.elasticbeanstalk.com
Updated: 2017-03-25 22:13:34.241000+00:00
Status: Ready
Health: Green
```

If the Status attribute is `Ready` and the Health is `Green` then you should be able to open up the Wordpress application in a browser using the CNAME address in the output above. For example:

```
http://jbaker-wordpress-dev.us-east-1.elasticbeanstalk.com
```

Feel free to log into the Wordpress application and play around with the interface a little bit. For example, try to create a new blog post. You just launched a highly-available Wordpress environment! What do you think? Was it easier to build the environment using the EB CLI and code versus using the AWS web console?

Modify the environment configuration

Go back to your terminal and check out the health of your deployed Wordpress environment:

```
eb health
```

You should see two healthy instances. When we executed the `eb create` command we specified that the environment should launch with 2 instances (`--scale 2`). Let's change the number of running instances in the environment to 1.

You can modify the EB application environment by typing in the following command:

```
eb config
```

This will open up a text editor containing the current application environment configuration. Look for the `aws:autoscaling:asg:` configuration section and change the MinSize attribute to `1`.

Save your file changes and exit the text editor. Notice how EB immediately begins to modify the deployed application environment. Wait for the update to complete.

Now, try checking the health of the EB environment again. The health command output might still show two running instances. If so, wait a few more minutes and check again. EB will eventually terminate one of the running instances.

Terminate the deployed application by running the command:

```
eb terminate --timeout 30
```

Watch as EB deletes all the infrastructure resources that were created to run your application.

Collect session data

Make a sub-directory in your home directory called `assignment7`. Change to that directory and create a git repository.

Next, configure the AWS CLI with your access key, secret key, and the current region. Review previous assignments and lectures if you don't recall how to configure the CLI.

Run the following command to retrieve the Elastic Beanstalk events and store the events in a JSON file:

```
$ aws elasticbeanstalk describe-events --application-name wordpress > eb-events.json
```

Check your work

Here is what the contents of your git repository should look like before final submission:

```
├ eb-events.json
```

Save your work

Add the `eb-events.json` files to the Git staging area and commit the files to the repository.

Create a new GitHub Classroom repository by clicking on this link:
https://classroom.github.com/a/XBdpyZMu

Configure your local Git repository to connect to this new GitHub repository. Push your work to GitHub and verify that the assignment files are located in the GitHub repository.

Terminate application environment

The last step in the assignment is to delete all the AWS resources you created. In this case, the only AWS resource still remaining is the `mgmt1` EC2 instance.

Submitting your assignment

I will review your published work on GitHub after the homework due date.