# Deploying a Docker Application on AWS Elastic Beanstalk with GitLab

Many customers rely on AWS Elastic Beanstalk to manage the infrastructure provisioning, monitoring, and deployment of their web applications. While Elastic Beanstalk supports several [development platforms and languages](https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/concepts.platforms.html), its support for Docker applications provides the most flexibility for developers to define their own stacks and achieve faster delivery cycles.

At the same time, organizations want to automate their build, test, and deployment processes and leverage continuous methodologies with modern DevOps platforms like GitLab. In this blog post, we will walk you through a process to build a simple Node.js application as a Docker container, host that container image in GitLab Container Registry, and use GitLab CI/CD and GitLab Runner to create a deployment pipeline to build the Docker image and push it to the Elastic Beanstalk environment.

## Solution Overview

The solution deployed in this blog post will complete the following steps in your AWS account:

1. Setup the initial GitLab Environment on Amazon EC2 in a new Amazon VPC and populate a GitLab code repository with a simple Node.js application. This step will also configure a deployment pipeline involving GitLab CI/CD, GitLab Runner and GitLab Container Registry.
2. Login and setup SSH access to your GitLab environment and configure GitLab CI/CD deployment tokens.
3. Provision a sample AWS Elastic Beanstalk application and environment.
4. Update the application code in the GitLab repository and automatically initiate the build and deployment to Elastic Beanstalk with GitLab CI/CD.

Figure 1 below illustrates the deployed solution.

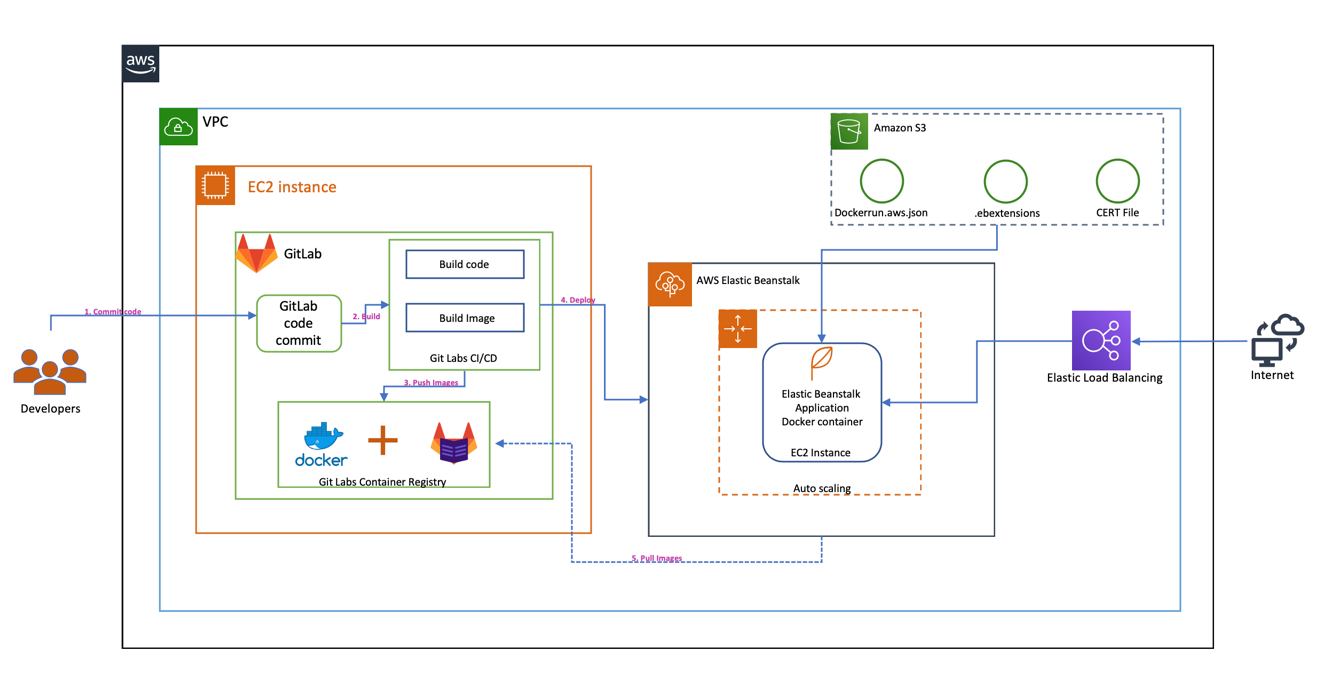


Figure 1

## Prerequisites and Assumptions

To follow the steps outlined in this blog post, you need the following:

* An AWS account that provides access to AWS services.
* The templates and code are intended to work in the us-east-1 region only and they are only for demonstration purposes. This is not for production use.
* Node.js and npm installed on your local machine. If installing Node.js and npm on mac, you can run “brew update” and “brew install node” commands on your terminal. You can also download Node.js from for windows from this page: <https://nodejs.org/en/download/>. The Node.js installer for Windows OS will also include NPM package manager. *<Sri to add Windows Instructions>*
* You also need to have the TypeScript compiler (tsc) installed on your local machine. Our sample application is developed using TypeScript which is a superset of JavaScript. To install the TypeScript compiler on your local mac, you can run npm install -g typescript in your terminal. On Windows OS machine, you can run the same command by opening any Console Window (cmd.exe). *<Sri to add Windows Instructions>*

Additionally, be aware of the following:

* We configure all services in the same VPC to simplify networking considerations.
* **Important**: The [AWS CloudFormation](https://aws.amazon.com/cloudformation/) templates and the sample code that we provide use hard-coded user names and passwords and open security groups. *DD: I am not sure how this will be received if going through a Security review.*

## Step 1: Setup the initial GitLab environment

In this step, we will setup the GitLab environment. To do so, we will provision a VPC with an Internet Gateway, a public subnet, a route table and a security group. The security group has one inbound rule to allow access to any TCP port from any VPC host configured to use the same security group. A Route 53 private hosted zone and an Amazon S3 bucket to store input data and processed data. The template will also download a sample application, push the code into the GitLab repository, and create a deployment pipeline with GitLab CI/CD.

### You can use this [downloadable](https://s3.amazonaws.com/aws-bigdata-blog/artifacts/awsblog-beanstalk-gitlab/cloudformations/step1-vpc-nw-ec2-gitlab.yaml) AWS CloudFormation template to set up these components. To launch directly through the console, choose Launch Stack.

### [launch_stack](https://console.aws.amazon.com/cloudformation/home?region=us-east-1#/stacks/new?stackName=Step1-Bean-GitLab-blog&templateURL=https://s3.amazonaws.com/aws-bigdata-blog/artifacts/awsblog-beanstalk-gitlab/cloudformations/step1-vpc-nw-ec2-gitlab.yaml)

*<Sri to make a single stack from these without the pipeline running and failing. Also we want to show the stack is complete only after GitLab is rebooted. Cfn-signal?> DD: I don’t think we need to call out the Outputs that aren’t specifically used in the blog*

### Provide a stack name and EC2 Key pair. After you specify the template parameters choose Next and create the Cloudformation stack. When the stack launch is complete, it should return outputs similar to the following.

|  |  |
| --- | --- |
| Key | Value |
| StackName | Name |
| VPCID | vpc-xxxxxxxx |
| SubnetIDA | subnet-xxxxxxxx |
| SubnetIDB | subnet-xxxxxxxx |
| SubnetIDC | subnet-xxxxxxxx |
| VPCSubnets | VPCSubnetsList |
| AWSBLOGBEANAccessSecurityGroup | Security group |
| GitEc2PublicDNS | ec2-xx-xx-xx-xx.compute-1.amazonaws.com |
| GitEc2PublicIp | xx-xx-xx-xx |
| ExpS3Bucket | <bucket-that-was-created> |

**Note:** To install and configure GitLab, it takes around ~10 minutes. Please wait while the GitLab is completely configured and running.

### Make a note of the output, because you use this information in the next step. You can [view the stack outputs](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-console-view-stack-data-resources.html) on the AWS Management Console or by using the following AWS CLI command:

aws cloudformation describe-stacks --stack-name *<stack\_name>* --region us-east-1 --query 'Stacks[0].Outputs'

## Step 2: Login to Gitlab and setup SSH key and CI/CD token

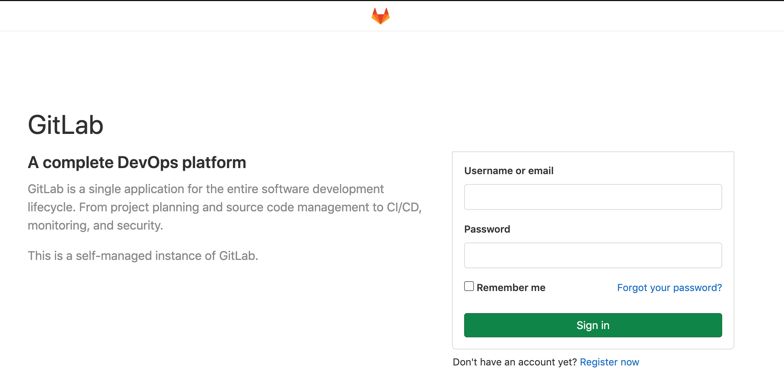
Next, login to your newly provisioned GitLab environment. Using the public DNS name that was shown in the Cloudformation stack output above, open your browser and enter the PublicDNS in the address bar. You will see the login page in Figure 2. Provide username as “root” and password as “changeme” to login to the GitLab environment. Please note that these are set in the “gitlab-setup.sh” script located at *Can we add the location of this gitlab-setup.sh script?*  
****

Figure 2

**Step 2a: Update SSH Key in GitLab:**

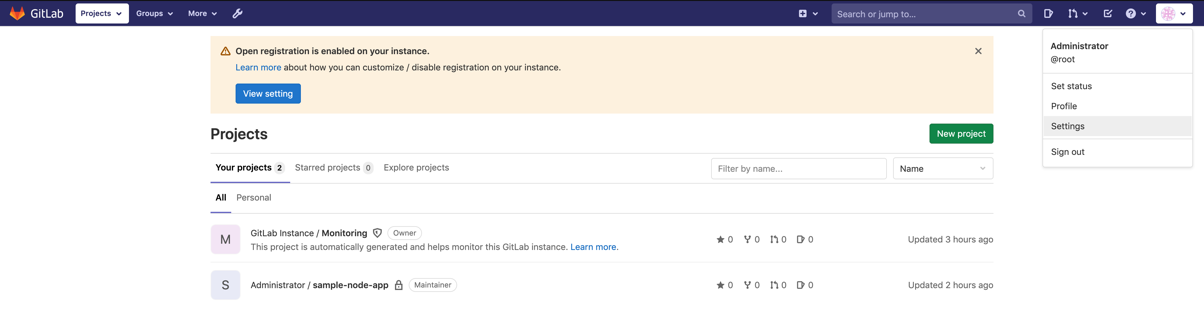
After successful login, we need to add your local host’s ssh key to establish a secure connection between your local computer and GitLab. We will need ssh access in order to clone the populated GitLab repository and push code changes in Step 4.   
  
Open settings page by selecting the “Settings” link as shown in Figure 3.  
****

Figure 3<Sri to update above image. I believe the UI has changed slightly>

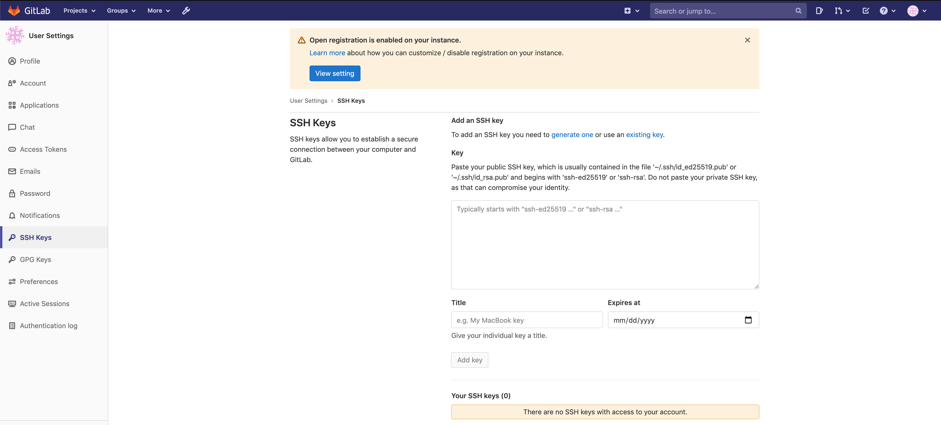
Click on “SSH Keys” on the left side navigation menu and you should the screen shown in Figure 4.  
****

Figure 4

Get your public ssh key from your local computer and paste it under the “Key” section. On a mac computer, you can get the public key by running the below command at the “terminal”. Once you paste the ssh key, click on “Add Key” button.

On Mac:

* cat ~/.ssh/id\_rsa.pub

On Windows:

* C:\Users\[your user name]\.ssh. Make sure you replace [your user name] with your user name.

*<Sri to update with equivalent Windows command. Also, can we show an example of what the copied/pasted text loos like?>*

**Step 2b: Adding Deploy Tokens in the Gitlab console**

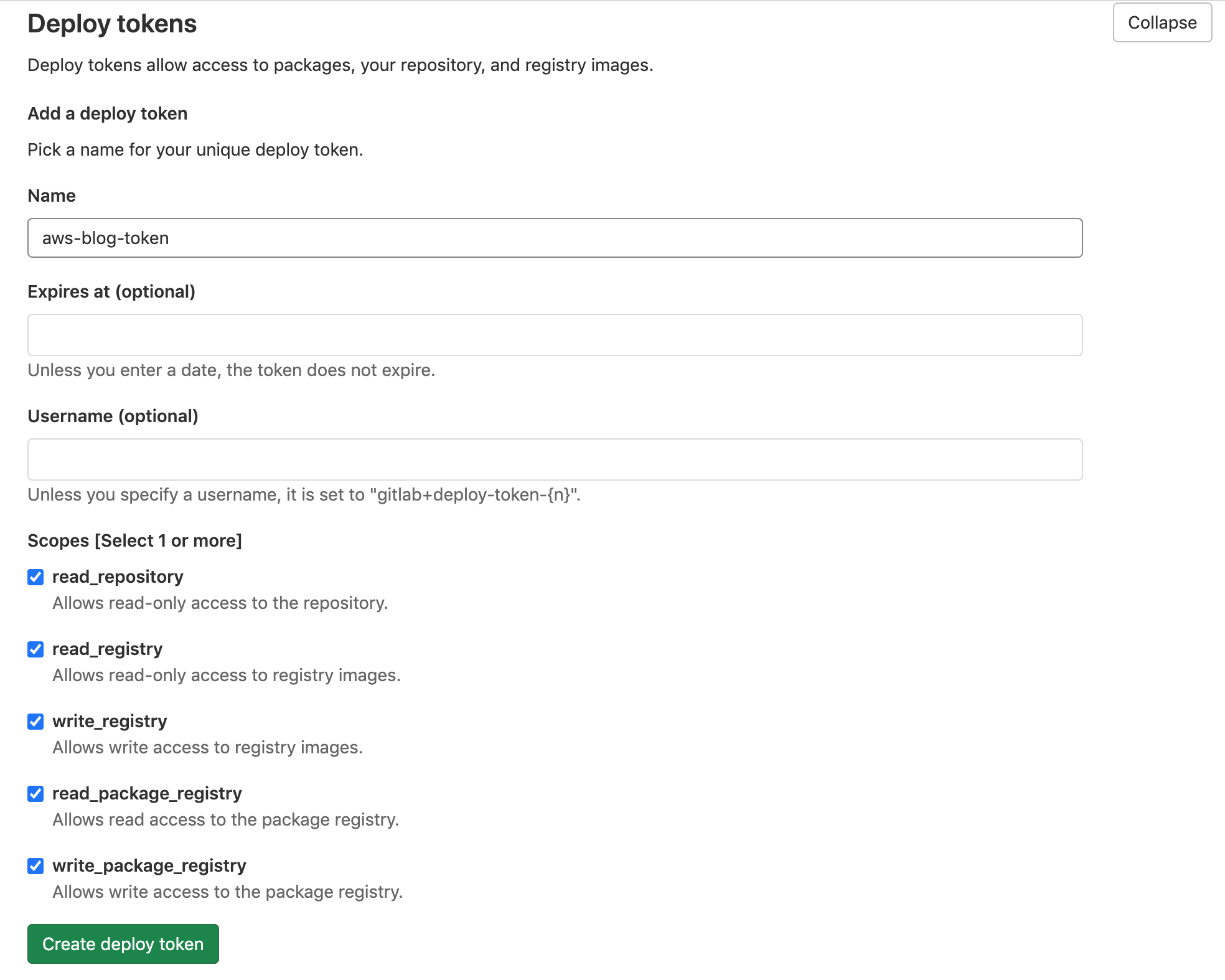
In order for Amazon Elastic Beanstalk to pull the Docker image containing our sample node.js app from the GitLab Conatiner Registry, we need to create [GitLab Deploy Tokens](https://docs.gitlab.com/ee/user/project/deploy_tokens/). Deploy tokens allow access to packages, your repository and registry images. Once you sign in to your Gitlab account, Select “sample-nodejs-app” -> Go Settings -> Repository. Expand “Deploy tokens”. Provide “Name” and select all options under “Scopes” section as shown in the Figure 5. Click on “Create Deploy token” button.  
  


Figure 5

It will create the username as “gitlab+deploy-token-1” and token with some random alpha numeric characters. Save these values before navigating to some other screen as the token cannot be recovered. Upon creation you should see the deploy token creation message as shown in Figure 6.

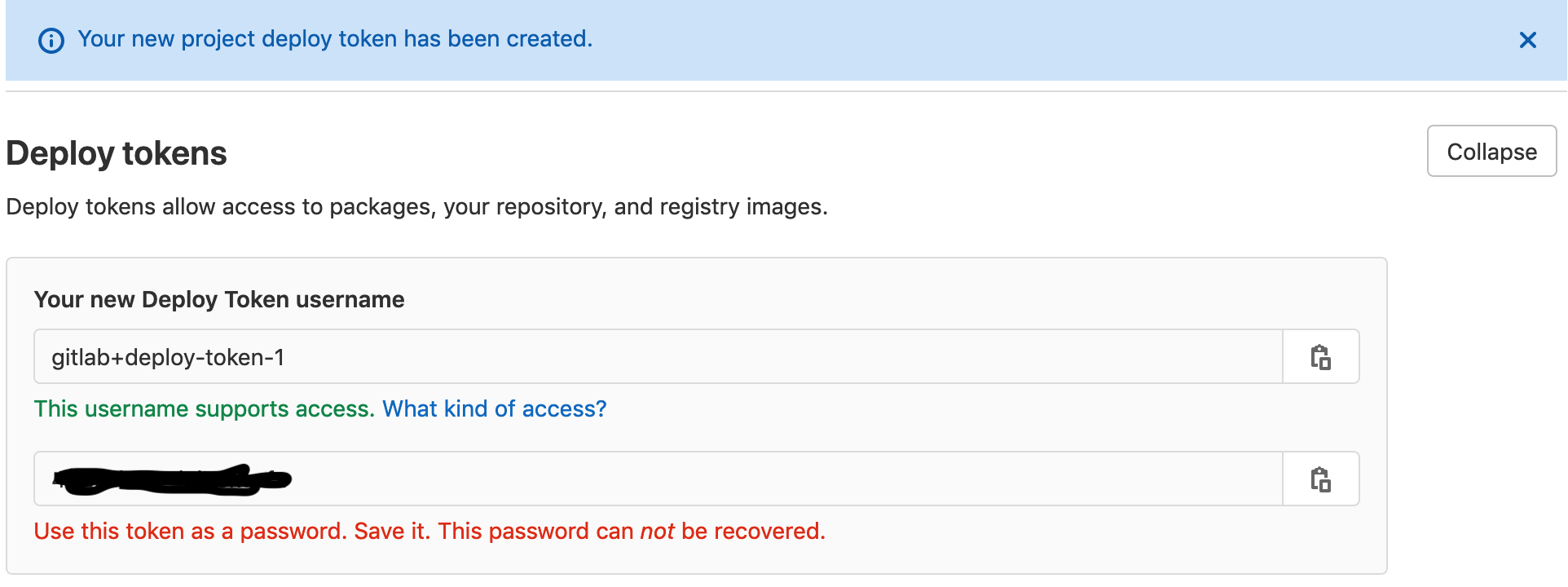


Figure 6

We use the “username” and “generated token” to connect to Docker using “docker login <registry.domain>” and it will generate the Docker “config.json” file. Once this file is generated, we use the populated token along with the domain name to S3 bucket. This part is of generating this config.json file and uploading to s3 bucket is automatically done inside the “.gitlab-ci.yml” file. You do not need to manually do these steps. Here is the code snippet in the “.gitlab-ci.yml” file.

*#- docker login -u gitlab+deploy-token-1 -p "mPi9hmYWM6vsKNRne3T5" $CI\_REGISTRY*- docker login -u $GIT\_DEPLOYMENT\_USER -p $GIT\_DEPLOYMENT\_TOKEN $CI\_REGISTRY  
- cat /root/.docker/config.json  
- echo $(cat /root/.docker/config.json | jq 'del(.HttpHeaders)') > dockercfg.json  
- cat dockercfg.json  
- aws s3 cp dockercfg.json s3://$S3\_BUCKET\_NAME/.dockercfg  
- aws s3 ls s3://$S3\_BUCKET\_NAME/.dockercfg

**Step 2c: Adding CI/CD variables in the GitLab console**

The .gitlab-ci.yml file provides customized instructions for GitLab CI/CD. In our case, this file is configured to use GitLab CI/CD environment variables for the username, password and S3 bucket values needed during pipeline execution. To setup these environment variables, Go to Projects/Your Projects and choose the sample-nodejs-app. Then go to Settings -> CI/CD and expand the “Variables” section. Our pipeline needs three variables to be added in this section. Make sure you uncheck “Protected variable” for each. Figure 7 below shows what your screen should look like when you create the GIT\_DEPLOYMENT\_USER variable. Your username should be the same. Repeat this process for the following environment variables:

* GIT\_DEPLOYMENT\_TOKEN - the value of the password that was generated as part of creating the deployment token in Step 2b.
* S3\_BUCKET\_NAME - created during the CloudFormation stack deployed in Step 1. You can find the S3\_BUCKET\_NAME in the “Outputs” for that stack in the CloudFormation console.

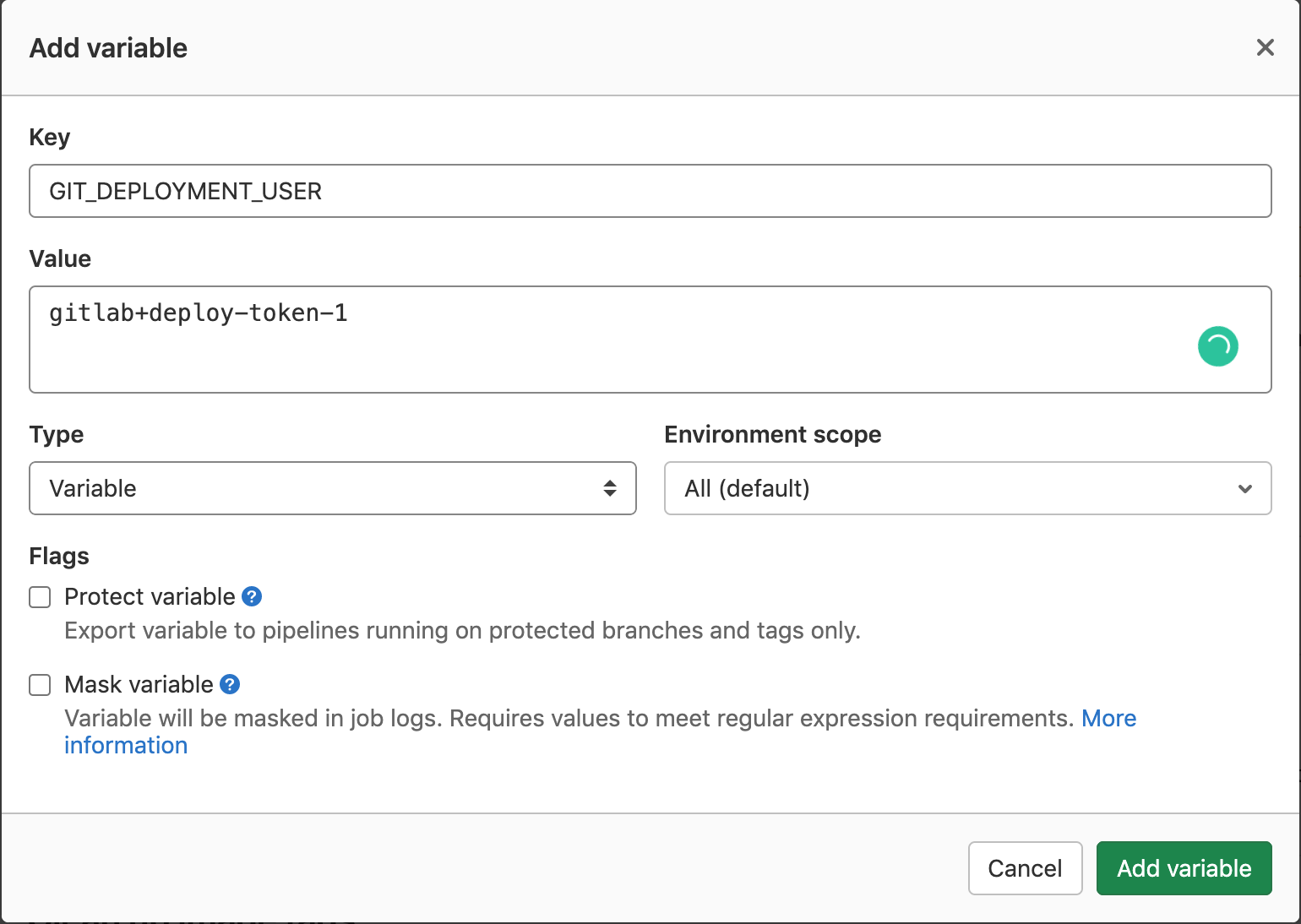


Figure 7

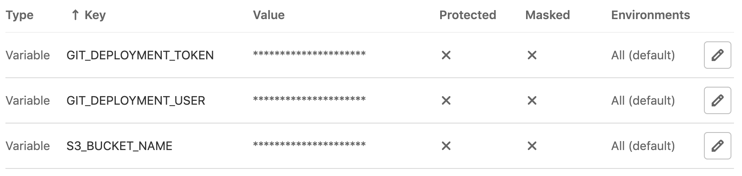
Once the three variables are created, you should see a summary similar to that shown in Figure 8.  


Figure 8

**Verify sample node.js application and the AWS Elastic Beanstalk configuration files**

The AWS CloudFormation stack deployed in Step 1 also downloads a sample application and pushes the code into your GitLab repository. To verify this, go to “Projects” menu in the GitLab console and click on “Your Projects” and you will see “sample-nodejs-app”.

## Step 3: Provision a sample Elastic Beanstalk application and environment

Now we will create a sample Elastic Beanstalk application and environment. Note, this step only creates an initial Elastic Beanstalk environment to which we will deploy, in Step 4. To begin, you can use this [downloadable](https://s3.amazonaws.com/aws-bigdata-blog/artifacts/awsblog-beanstalk-gitlab/cloudformations/step3-setup-beanstalk.yaml) AWS CloudFormation template. To launch directly through the console, choose Launch Stack.

[launch_stack](https://console.aws.amazon.com/cloudformation/home?region=us-east-1#/stacks/new?stackName=Step3-Bean-GitLab-blog&templateURL=https://s3.amazonaws.com/aws-bigdata-blog/artifacts/awsblog-beanstalk-gitlab/cloudformations/step3-setup-beanstalk.yaml)

After you specify the template details, choose Next. On the Review page, choose Create.

**Note:** This Cloudformation template will take around 10 minutes to complete.

Once the Cloudformation template’s execution is successful, you can see the newly created “Application” and “Environment” in the AWS Elastic Beanstalk service console as shown in Figure 9.

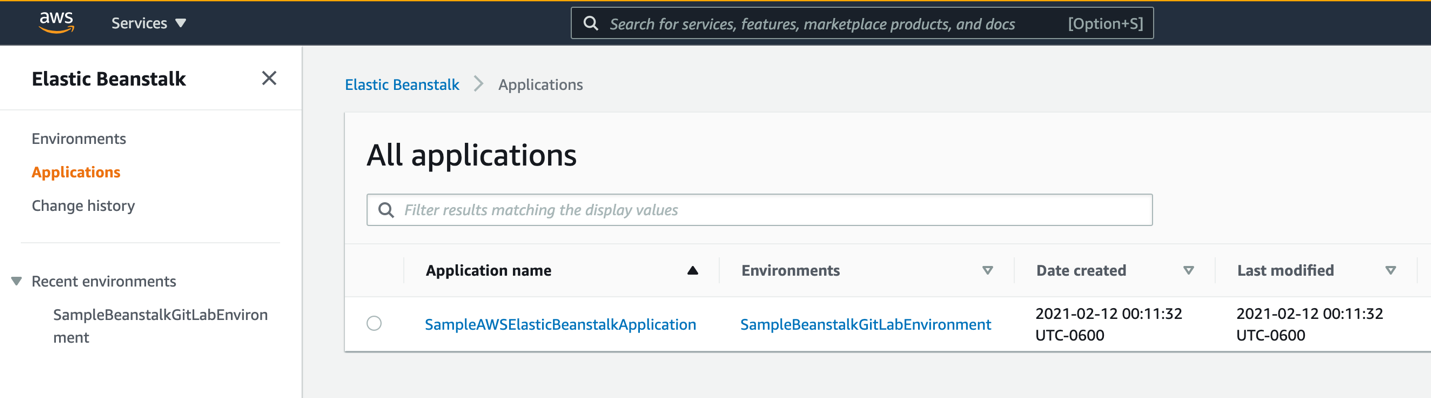


Figure 9

The stack also creates an Elastic Load Balancer (ELB) and we can use hostname of that ELB to connect to the application.

Go to Amazon Elastic Load Balancer console, select the newly created ELB, and copy the DNS name. Paste the DNS name in the browser address bar. It should open a default page similar to Figure 10.

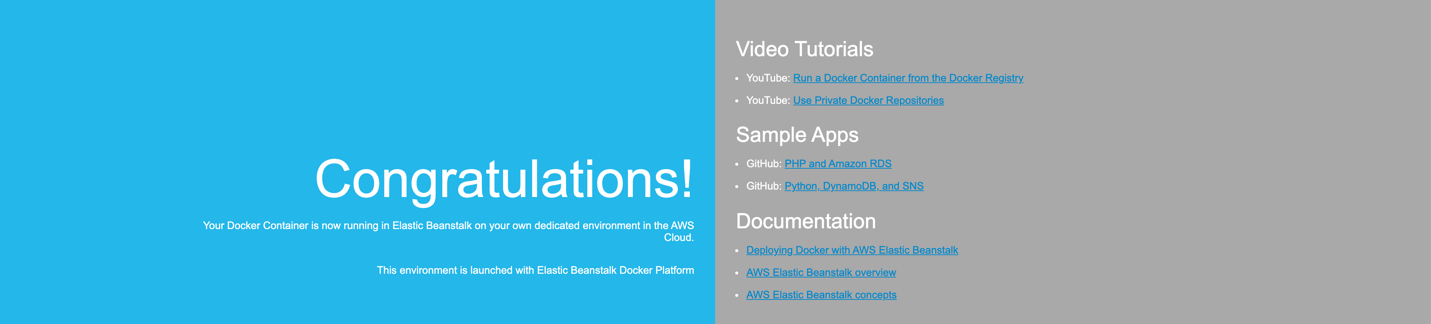


Figure 10

## Step 4: Update and deploy application changes

*DD: Step 4 is first running the pipeline manually now that EB is setup. As long as repo includes dist directory, it should deploy successfully. <Sri to add screenshot and directions to manually start the pipeline and verify success by hitting same ELB endpoint mentioned above.>*  
Next, we will clone, update and push changes back. Below will be the setup for that portion.

Now we will clone the **sample-nodejs-app** repository to your local machine, make a change and push it back to the Gitlab repo. Sign in to GitLab and go to “Your projects” and select the “sample-nodejs-app.

On your local machine (showing the below steps on mac), create a directory where you want to download your repository and then clone your repository.

mkdir -p ~/test/  
cd ~/test/  
git clone [git@ec2-XX-XX-XX-XX.compute-1.amazonaws.com:root/sample-nodejs-app.git](mailto:git@ec2-XX-XX-XX-XX.compute-1.amazonaws.com:root/sample-nodejs-app.git)

Make sure port 22 is opened to this instance from your network). Update the host name in the command above with the public DNS name of your GitLab EC2 instance that was created in Step 1.  
Next, run the below command to install the typescript module dependencies.

cd ~/test/sample-nodejs-app/

npm install @types/node @types/express @types/body-parser --save-dev

**Modify the sample Node.js application:**

*<Sri to add steps to make a simple change to the application>*

**Compiling the application using “tsc” command:**

The tsc command invokes the typescript compiler. It uses the tsconfig.json file to compile the application. Once the compilation is complete, it generates the “dist” directory.

cd ~/test/sample-nodejs-app/

tsc

Now login to your local machine and navigate to the directory where you copied the sample application.

cd ~/test/sample-nodejs-app

git add .

git commit -m "compiled application changes"

git push -u origin master

Once the code push is completed, sign in to the “Gitlab” console and select “sample-nodejs-app” and go to CI/CD->Pipelines. You will see the pipeline gets executed and it will deploy the new version to Elastic Beanstalk environment. *<Sri to add screenshot of successful deployment>*

## Conclusion