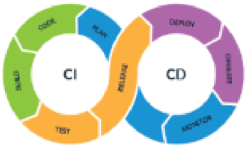
Jenkins docker

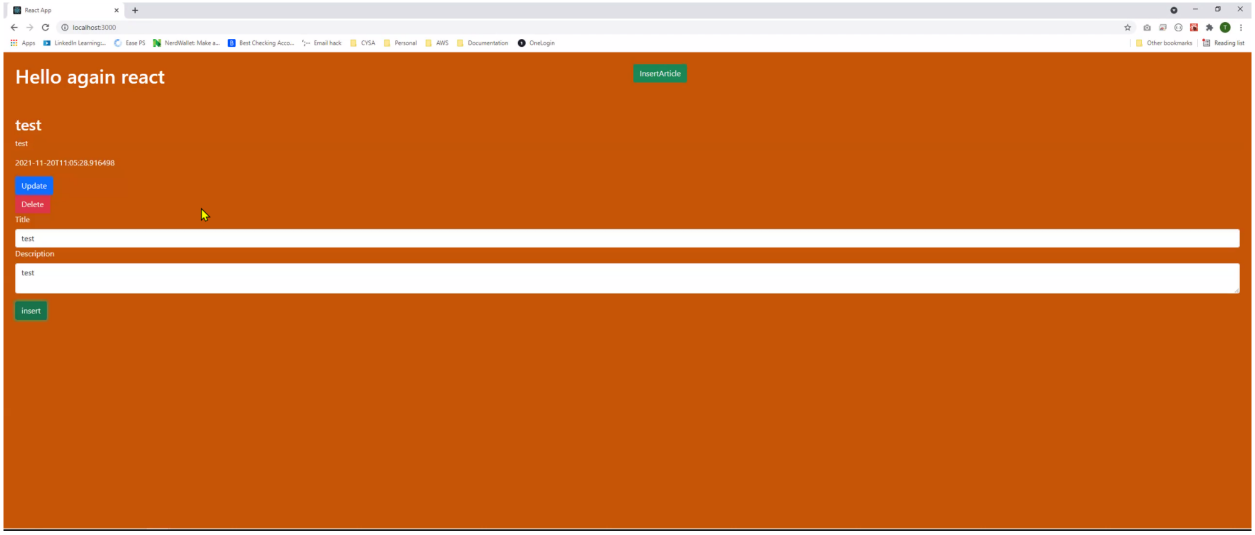


cypress ANSIBLE

Deployment 08 - CICD

For this assignment, we have to create an entire CICD pipeline

We will have to deploy an application. The application provided will insert data and populate the page. The data is then sent to a database. The application will use React as the front end and Python as the backend.



Hello again react

test

test

2021-11-20T11:05:28.916498

Title: test

Description: test

For this assignment, we will have to incorporate Docker, Cypress, and ansible. There will be 3 EC2 instances. The first one will be a jenkins controller, second will be a jenkins agent, and last will be a production instance.

Build app -> test front end (can do backend) -> when doing test, should have test results and encrypt it and put it on GitHub (using ansible) -> create a docker image of the application after its tested and know exactly what risk you have for application -> after created application do a security check on the application. Encrypt the report -> set up a cloudwatch alert and stress out CPU -> configure alerts and notice what to keep an eye of

Notes

Jenkins
What are the factors. Different ways to take app and send it to deployment pipeline

Is this for production and development?
Have to change the value and test
A lot of key values to change
Authentication keys and key value pairs
What are some things to keep in mind
Take app and send it through pipeline

Order of operations
Plan out how to put things together
Keep stuff in mind.
Requirements for test

Task 1

For this task, we need to provision all the resources that we need. We need 3 EC2 instances for this assignment. The first EC2 will be the Jenkins Controller. The second EC2 instance will be a Jenkins Agent that received instructions from the Jenkins Controller. Finally, the third EC2 will be used for a production environment. In this scenario, not all resources are required to be in a private subnet. Each EC2 requires different dependencies to be installed.

We will be using CloudFormation to provision all the resources and then use Ansible to configure each instance.

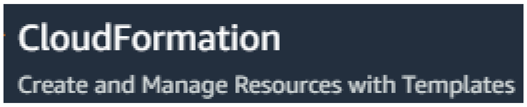
Before we begin, we need to make sure that our host system has Ansible. We also need to make sure that boto is installed and our AWS CLI is configured.

```
sudo apt install ansible
pip3 install boto
aws configure
```

Once that is set up, we can provision the resources using CloudFormation. Create a file called resource.yaml and paste the following inside the file…

```yaml
Description:
  This template deploys a VPC, with a pair of public and private subnets spread
  across two Availability Zones. It deploys an internet gateway, with a default
  route on the public subnets. It deploys a pair of NAT gateways (one in each AZ),
  and default routes for them in the private subnets.

Parameters:
  EnvironmentName:
    Description: An environment name that is prefixed to resource names
    Type: String

  VpcCIDR:
    Description: Please enter the IP range (CIDR notation) for this VPC
    Type: String
    Default: 192.168.0.0/16

  PublicSubnet1CIDR:
    Description: Please enter the IP range (CIDR notation) for the public subnet in the first Availability Zone
    Type: String
    Default: 192.168.0.0/18

  PublicSubnet2CIDR:
    Description: Please enter the IP range (CIDR notation) for the public subnet in the second Availability Zone
    Type: String
    Default: 192.168.64.0/18

  PrivateSubnet1CIDR:
    Description: Please enter the IP range (CIDR notation) for the private subnet in the first Availability Zone
    Type: String
    Default: 192.168.128.0/18

  PrivateSubnet2CIDR:
    Description: Please enter the IP range (CIDR notation) for the private subnet in the second Availability Zone
    Type: String
    Default: 192.168.192.0/18

  KeyName:
    Description: Name of an existing EC2 KeyPair to enable SSH access to the instance
    Type: AWS::EC2::KeyPair::KeyName

Resources:
  VPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: !Ref VpcCIDR
      EnableDnsSupport: true
      EnableDnsHostnames: true
      Tags:
        - Key: Name
          Value: !Ref EnvironmentName

  InternetGateway:
    Type: AWS::EC2::InternetGateway
    Properties:
      Tags:
        - Key: Name
          Value: !Ref EnvironmentName

  InternetGatewayAttachment:
    Type: AWS::EC2::VPCGatewayAttachment
    Properties:
      InternetGatewayId: !Ref InternetGateway
      VpcId: !Ref VPC

  PublicSubnet1:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [0, !GetAZs ""]
      CidrBlock: !Ref PublicSubnet1CIDR
      MapPublicIpOnLaunch: true
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Public Subnet (AZ1)

  PublicSubnet2:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [1, !GetAZs ""]
      CidrBlock: !Ref PublicSubnet2CIDR
      MapPublicIpOnLaunch: true
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Public Subnet (AZ2)

  PrivateSubnet1:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [0, !GetAZs ""]
      CidrBlock: !Ref PrivateSubnet1CIDR
      MapPublicIpOnLaunch: false
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Private Subnet (AZ1)

  PrivateSubnet2:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [1, !GetAZs ""]
      CidrBlock: !Ref PrivateSubnet2CIDR
      MapPublicIpOnLaunch: false
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Private Subnet (AZ2)

  NatGateway1EIP:
    Type: AWS::EC2::EIP
    DependsOn: InternetGatewayAttachment
    Properties:
      Domain: vpc

  NatGateway2EIP:
    Type: AWS::EC2::EIP
    DependsOn: InternetGatewayAttachment
    Properties:
      Domain: vpc

  NatGateway1:
    Type: AWS::EC2::NatGateway
    Properties:
      AllocationId: !GetAtt NatGateway1EIP.AllocationId
      SubnetId: !Ref PublicSubnet1

  NatGateway2:
    Type: AWS::EC2::NatGateway
    Properties:
      AllocationId: !GetAtt NatGateway2EIP.AllocationId
      SubnetId: !Ref PublicSubnet2

  PublicRouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Public Routes

  DefaultPublicRoute:
    Type: AWS::EC2::Route
    DependsOn: InternetGatewayAttachment
    Properties:
      RouteTableId: !Ref PublicRouteTable
      DestinationCidrBlock: 0.0.0.0/0
      GatewayId: !Ref InternetGateway

  PublicSubnet1RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PublicRouteTable
      SubnetId: !Ref PublicSubnet1

  PublicSubnet2RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PublicRouteTable
      SubnetId: !Ref PublicSubnet2

  PrivateRouteTable1:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Private Routes (AZ1)

  DefaultPrivateRoute1:
    Type: AWS::EC2::Route
    Properties:
      RouteTableId: !Ref PrivateRouteTable1
      DestinationCidrBlock: 0.0.0.0/0
      NatGatewayId: !Ref NatGateway1

  PrivateSubnet1RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PrivateRouteTable1
      SubnetId: !Ref PrivateSubnet1

  PrivateRouteTable2:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Private Routes (AZ2)

  DefaultPrivateRoute2:
    Type: AWS::EC2::Route
    Properties:
      RouteTableId: !Ref PrivateRouteTable2
      DestinationCidrBlock: 0.0.0.0/0
      NatGatewayId: !Ref NatGateway2

  PrivateSubnet2RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PrivateRouteTable2
      SubnetId: !Ref PrivateSubnet2

  JenkinsControllerSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: "Security group that allows SSH from anywhere"
      GroupName: "JenkinsController"
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 22
          ToPort: 22
          CidrIp: 0.0.0.0/0
        - IpProtocol: tcp
          FromPort: 8080
          ToPort: 8080
          CidrIp: 0.0.0.0/0
      VpcId: !Ref VPC

  JenkinsControllerEC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: ami-09e67e426f25ce0d7
      InstanceType: t2.micro
      SubnetId: !Ref PublicSubnet1
      KeyName: !Ref KeyName
      SecurityGroupIds:
        - !Ref JenkinsControllerSecurityGroup
      Tags:
        - Key: "Name"
          Value: "Jenkins Controller"

  JenkinsAgentSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: "Security group that allows SSH from anywhere"
      GroupName: "JenkinsAgent"
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 22
          ToPort: 22
          CidrIp: 0.0.0.0/0
      VpcId: !Ref VPC

  JenkinsAgentEC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: ami-09e67e426f25ce0d7
      InstanceType: t2.micro
      SubnetId: !Ref PublicSubnet1
      KeyName: !Ref KeyName
      SecurityGroupIds:
        - !Ref JenkinsAgentSecurityGroup
      Tags:
        - Key: "Name"
          Value: "Jenkins Agent"

  ProductionSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: "Security group that allows SSH from anywhere"
      GroupName: "Production"
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 22
          ToPort: 22
          CidrIp: 0.0.0.0/0
      VpcId: !Ref VPC

  ProductionEC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: ami-09e67e426f25ce0d7
      InstanceType: t2.micro
      SubnetId: !Ref PublicSubnet1
      KeyName: !Ref KeyName
      SecurityGroupIds:
        - !Ref ProductionSecurityGroup
      Tags:
        - Key: "Name"
          Value: "Production"

  Deploy08DBSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: "Security group for the RDS MySQL database that allows access from Production/Agent SG only"
      GroupName: "deploy08-db"
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 0
          ToPort: 65535
          CidrIp: 0.0.0.0/0
      VpcId: !Ref VPC

Outputs:
  VPC:
    Description: A reference to the created VPC
    Value: !Ref VPC

  JenkinsControllerEC2Instance:
    Value: !GetAtt JenkinsControllerEC2Instance.PublicIp
    Description: JenkinsController's PublicIp Address

  JenkinsAgentEC2Instance:
    Value: !GetAtt JenkinsAgentEC2Instance.PublicIp
    Description: JenkinsAgentEC2Instance's PublicIp Address

  ProductionEC2Instance:
    Value: !GetAtt ProductionEC2Instance.PublicIp
    Description: ProductionEC2Instance's PublicIp Address
```

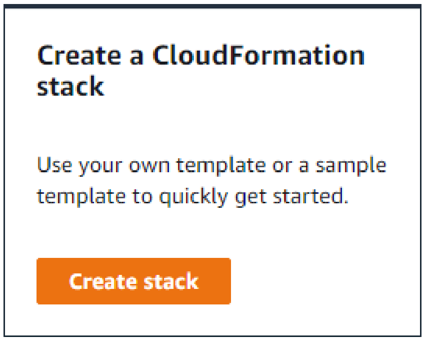
Once the file is saved, go to CloudFormation on AWS (Make sure you are in the correct region).



CloudFormation

Create and Manage Resources with Templates

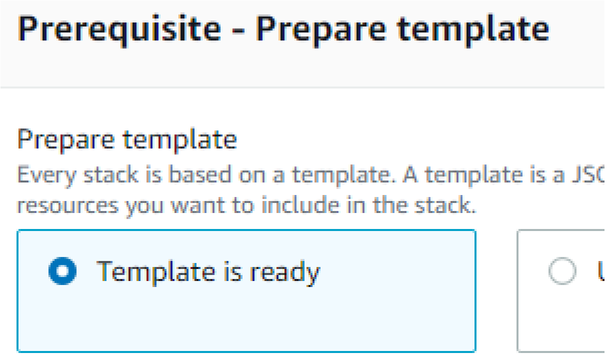
Create a stack



Create a CloudFormation stack

Use your own template or a sample template to quickly get started.

We will be using a template to create all the resources needed for this assignment. For the prerequisite select, Template is ready



Prerequisite - Prepare template

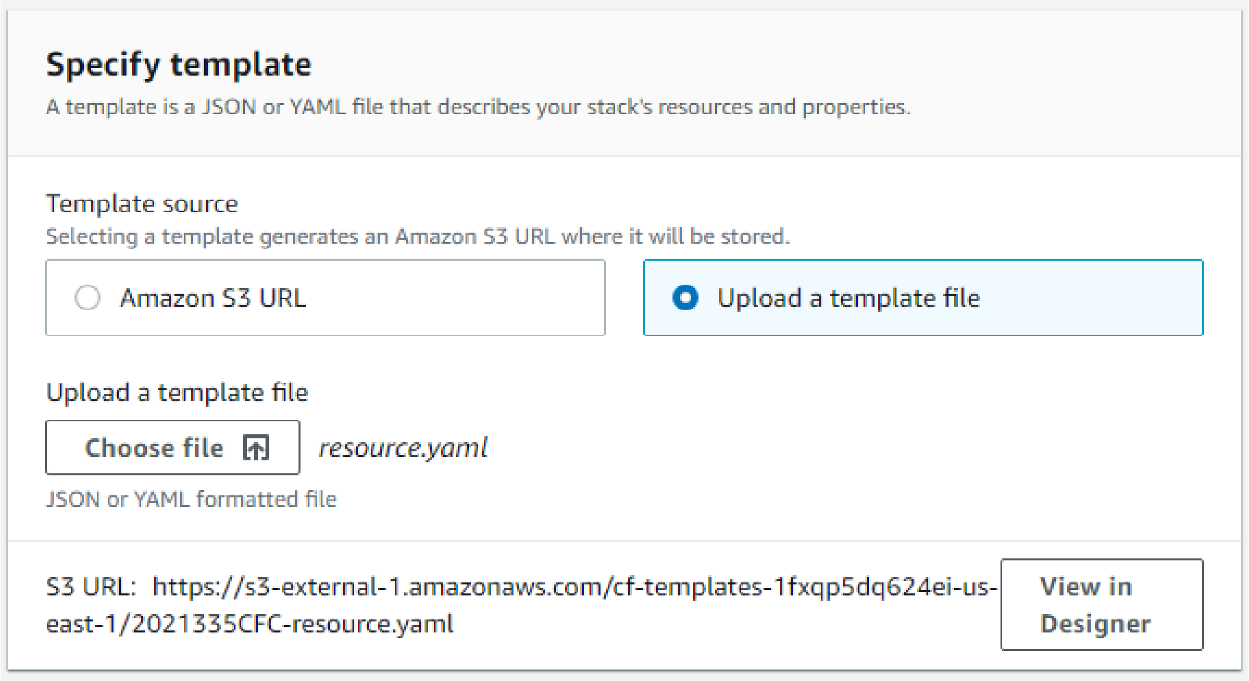
Prepare template

Every stack is based on a template. A template is a JSC
resources you want to include in the stack.

● Template is ready

○ U

We then need to upload our template from local to AWS. Select upload a template file and upload your resource.yaml file.



Specify template

A template is a JSON or YAML file that describes your stack's resources and properties.

Template source

Selecting a template generates an Amazon S3 URL where it will be stored.

○ Amazon S3 URL

● Upload a template file

Upload a template file

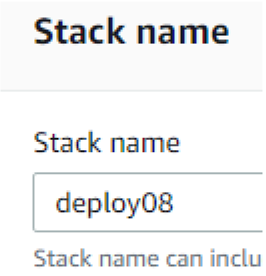
Choose file *resource.yaml*

JSON or YAML formatted file

S3 URL: https://s3-external-1.amazonaws.com/cf-templates-1fxqp5dq624ei-us-east-1/2021335CFC-resource.yaml

View in Designer

We will need to configure our stack details. Select a name



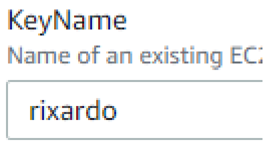
Stack name

Stack name

deploy08

Stack name can inclu

Under Parameters, select the KeyName that will be used to SSH into each EC2 instance.

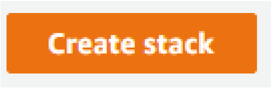


KeyName

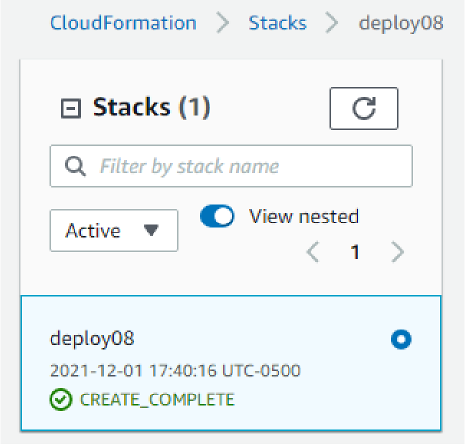
Name of an existing EC

rixardo

You can leave everything else default and go to the next page



The CloudFormation will take some time to create. You can refresh every few minutes to check completion. Once it's created, it should look like this…

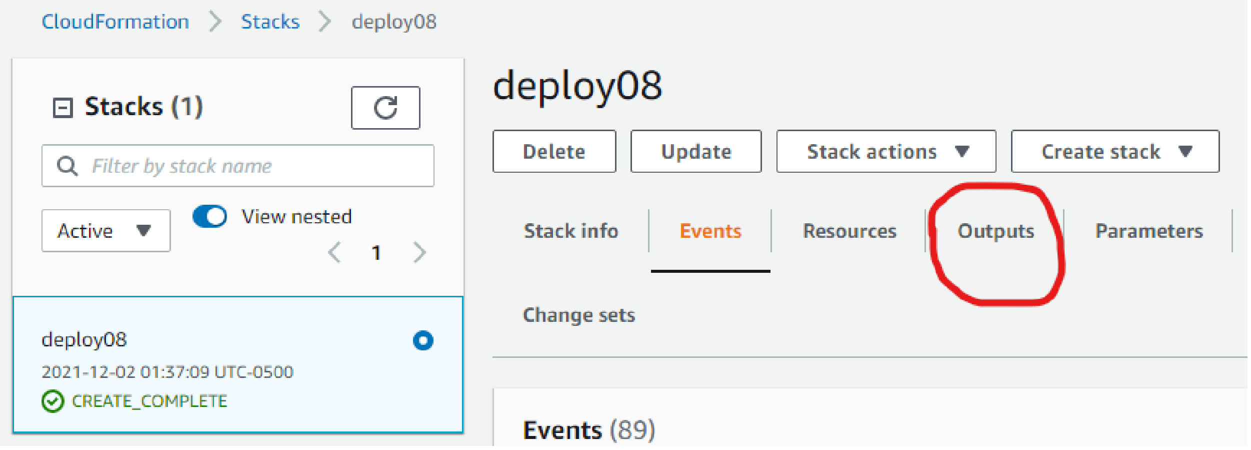


deploy08 ●

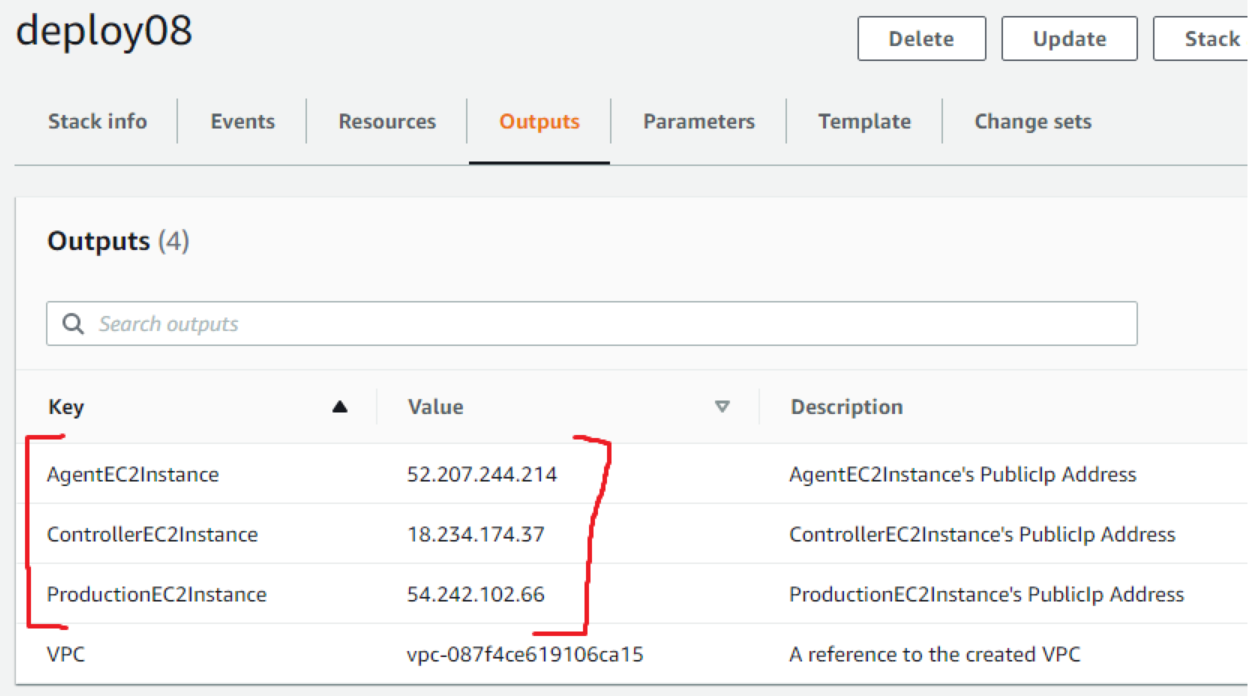
2021-12-01 17:40:16 UTC-0500

✓ CREATE_COMPLETE

Before moving on to configuring each of the instances, we need to obtain data. Navigate to Outputs tab in the CloudFormation Stack.



deploy08
2021-12-02 01:37:09 UTC-0500
CREATE_COMPLETE



Key	Value	Description
AgentEC2Instance	52.207.244.214	AgentEC2Instance's PublicIp Address
ControllerEC2Instance	18.234.174.37	ControllerEC2Instance's PublicIp Address
ProductionEC2Instance	54.242.102.66	ProductionEC2Instance's PublicIp Address
VPC	vpc-087f4ce619106ca15	A reference to the created VPC

Take a note of the Public IPv4 address of all the EC2 instances that were created in this format. We will need to edit our hosts file in the future with this format.

[Agent]
18.234.51.214 ansible_user=ubuntu ansible_ssh_private_key_file=~/.ssh/**rixardo.pem**

[Controller]
174.129.117.4 ansible_user=ubuntu ansible_ssh_private_key_file=~/.ssh/**rixardo.pem**

[Production]
52.55.217.196 ansible_user=ubuntu ansible_ssh_private_key_file=~/.ssh/**rixardo.pem**

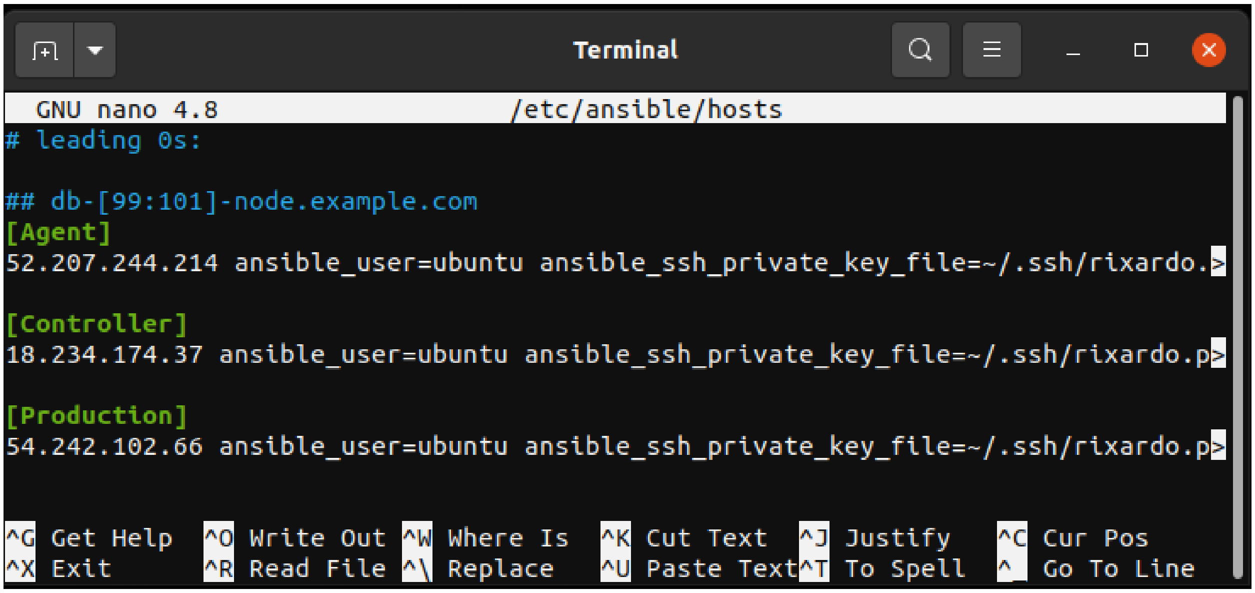
Once we set up all the resources and obtain the necessary data, we can proceed to setting up Ansible so we can configure the EC2s. Change directory to your SSH folder and make sure your Key that you use to SSH into EC2 instances is there.

cd ~/.ssh

We will then need to change the host file for Ansible so that we can SSH into our EC2s with Ansible and run commands.

sudo nano /etc/ansible/hosts

Scroll all the way to the bottom and attach the formatted text above that we created with the Public IPv4 for each of the instances at the bottom.



```
  GNU nano 4.8                   /etc/ansible/hosts
# leading 0s:

## db-[99:101]-node.example.com
[Agent]
52.207.244.214 ansible_user=ubuntu ansible_ssh_private_key_file=~/.ssh/rixardo.>

[Controller]
18.234.174.37 ansible_user=ubuntu ansible_ssh_private_key_file=~/.ssh/rixardo.p>

[Production]
54.242.102.66 ansible_user=ubuntu ansible_ssh_private_key_file=~/.ssh/rixardo.p>


^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos
^X Exit       ^R Read File  ^\ Replace    ^U Paste Text^T To Spell    ^  Go To Line
```

Save and exit the text editor

CTRL + O

<ENTER>

CTRL + X

We can now use ansible to configure our EC2 instances. For the first EC2, we will need to install Java and Jenkins. Once we install Jenkins, we need to obtain the password to set up jenkins on our browser. We will be able to access our Jenkins application using the Public IPv4 of the Jenkins Controller EC2 followed by port 8080. The second EC2 instance needs Java, npm, and nodejs. The third EC2 needs openjdk and docker.

We will need to create a set of YAML files with specific commands. There will be 3 ansible playbooks created to install dependencies in each EC2 instance. Once all the separated ansible playbooks are created, there will be one main ansible playbook that calls upon the other 3 and run them.

The first Ansible Playbook should be called "configure_controller.yaml". This playbook will install updates, java, and set up jenkins. A helpful addition that I added was to output if Jenkins is active and the password to log into the root account. Paste the following script inside…

```
---
- name: "Configuring the Controller EC2 instance"
  hosts: Controller
  gather_facts: false
  connection: ssh

  tasks:
    - name: updating the ec2 instance
      shell: sudo apt-get update && sudo apt-get upgrade -y

    - name: installing java
      shell: sudo apt install openjdk-11-jre-headless -y

    # https://www.jenkins.io/doc/book/installing/linux/
    - name: getting the long term support release of jenkins
      shell: curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo tee \ /usr/share/keyrings/jenkins-keyring.asc > /dev/null

    - name: signing the downloaded jenkins application and adding it to the repository
      shell: echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \ https://pkg.jenkins.io/debian-stable binary/ | sudo tee \ /etc/apt/sources.list.d/jenkins.list > /dev/null

    - name: upgrading the repository
      shell: sudo apt-get update && sudo apt-get upgrade -y

    - name: installing the jenkins application.
      shell: sudo apt-get install jenkins -y

    - name: installing git tool
      shell: sudo apt install git -y

    - name: checking if jenkins is active
      shell: sudo systemctl status jenkins | head -n 3
      register: command_output
    - debug:
        var: command_output.stdout_lines

    - name: outputting jenkins password
      shell: echo "The Jenkins password is $(sudo cat /var/lib/jenkins/secrets/initialAdminPassword)"
      register: command_output
    - debug:
        var: command_output.stdout_lines
```

The second Ansible Playbook should be called "configure_agent.yaml". This playbook simply installs updates, java, nodejs, and npm. Paste the following script inside…

```
---
- name: "Configuring the Agent EC2 instance"
  hosts: Agent
  gather_facts: false
  connection: ssh

  tasks:
    - name: updating the ec2 instance
      shell: sudo apt-get update && sudo apt-get upgrade -y

    - name: installing java
      shell: sudo apt install openjdk-11-jre-headless -y

    - name: installing nodejs
      shell: sudo apt install nodejs -y

    - name: installing npm
      shell: sudo apt install npm -y

    - name: using npm to install cypress
      shell: npm install cypress

    - name: using npm to install mocha
      shell: npm install mocha

    - name: installing dependencies for cypress testing
      shell: sudo apt install xvfb libatk1.0-0 libgtk-3-0 libgbm-dev libnotify-dev libgconf-2-4 libnss3 libxss1 libasound2 libxtst6 xauth -y

    - name: downloading dependencies for docker
      shell: sudo apt-get install apt-transport-https ca-certificates curl software-properties-common -y

    - name: adding gpg keys
      shell: curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

    - name: installing the docker repository
      shell: sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu  $(lsb_release -cs)  stable"

    - name: upgrading the repository
      shell: sudo apt-get update && sudo apt-get upgrade -y

    - name: installing latest version of docker
      shell: sudo apt-get install docker-ce -y

    - name: changing permissions for the docker socket
      shell: sudo chmod 666 /var/run/docker.sock

    - name: starting docker
      shell: sudo systemctl start docker

    - name: configure docker to start on boot
      shell: sudo systemctl enable docker

    - name: checking if docker is active
      shell: sudo systemctl status docker | head -n 3
      register: command_output
    - debug:
        var: command_output.stdout_lines
```

The third Ansible Playbook should be called “configure_production.yaml”. This playbook simply installs updates, java, and updates. A helpful addition that I added was to output if Docker is active Paste the following script inside…

```
---
- name: "Configuring the Production EC2 instance"
  hosts: Production
  gather_facts: false
  connection: ssh

  tasks:
  - name: updating the ec2 instance
    shell: sudo apt-get update && sudo apt-get upgrade -y

  - name: installing java
    shell: sudo apt install openjdk-11-jre-headless -y

  - name: downloading dependencies for docker
    shell: sudo apt-get install apt-transport-https ca-certificates curl software-properties-common -y

  - name: adding gpg keys
    shell: curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

  - name: installing the docker repository
    shell: sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu  $(lsb_release -cs)  stable"

  - name: upgrading the repository
    shell: sudo apt-get update && sudo apt-get upgrade -y

  - name: installing latest version of docker
    shell: sudo apt-get install docker-ce -y

  - name: changing permissions for the docker socket
    shell: sudo chmod 666 /var/run/docker.sock

  - name: starting docker
    shell: sudo systemctl start docker

  - name: configure docker to start on boot
    shell: sudo systemctl enable docker

  - name: checking if docker is active
    shell: sudo systemctl status docker | head -n 3
    register: command_output
  - debug:
      var: command_output.stdout_lines
```

Finally, the final Ansible playbook should be called “configure.yaml”. Paste the following script inside…

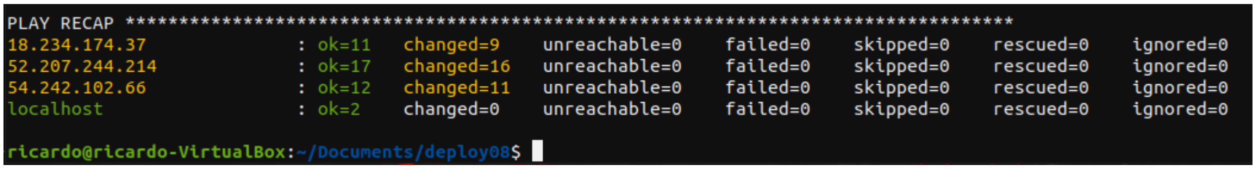
```
---
- hosts: localhost
  tasks:
  - debug:
      msg: Configuring All 3 EC2 Instances.
```

```
- name: configuring the controller ec2 instance
  import_playbook: configure_controller.yaml

- name: configuring the agent ec2 instance
  import_playbook: configure_agent.yaml

- name: configuring the production ec2 instance
  import_playbook: configure_production.yaml
```

Once we have created the required configuration files, we can execute one main ansible playbook that will run the other playbooks ansible playbook using the following command

```
ansible-playbook configure.yaml
```

A successful outcome would look like this

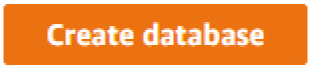


```
PLAY RECAP ****************************************************************************
18.234.174.37              : ok=11    changed=9     unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
52.207.244.214             : ok=17    changed=16    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
54.242.102.66              : ok=12    changed=11    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
localhost                  : ok=2     changed=0     unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

ricardo@ricardo-VirtualBox:~/Documents/deploy08$
```

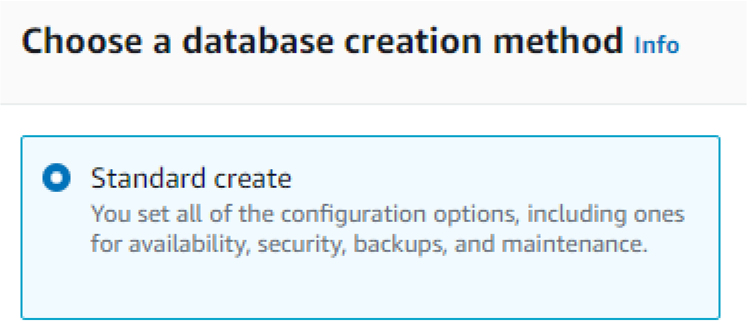
While testing the application locally, I decided use RDS

Go to AWS RDS service

Create a database



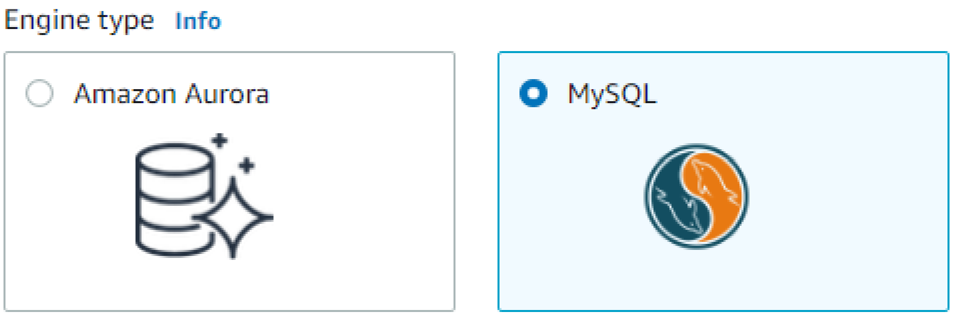
For database creation method, select easy create



Choose a database creation method Info

● Standard create
You set all of the configuration options, including ones for availability, security, backups, and maintenance.

For the engine type, select MySQL

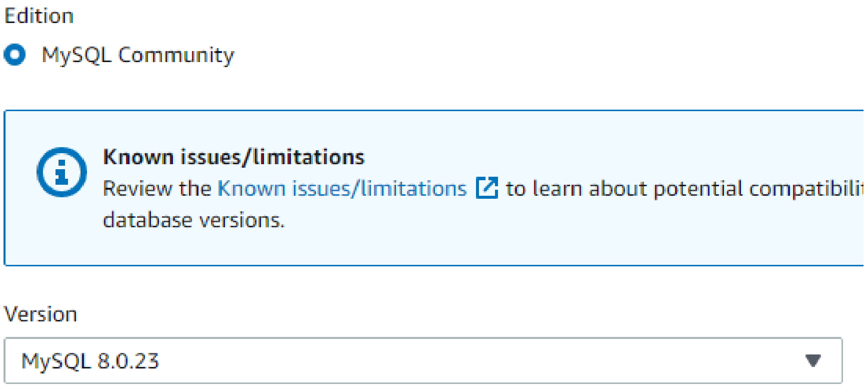


Engine type Info

○ Amazon Aurora

● MySQL

Keep the version and edition default



Edition

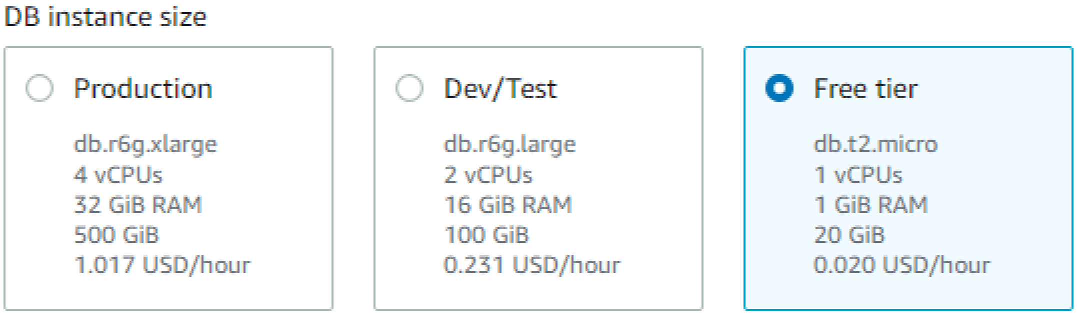
● MySQL Community

Known issues/limitations
Review the Known issues/limitations to learn about potential compatibili
database versions.

Version

MySQL 8.0.23 ▼

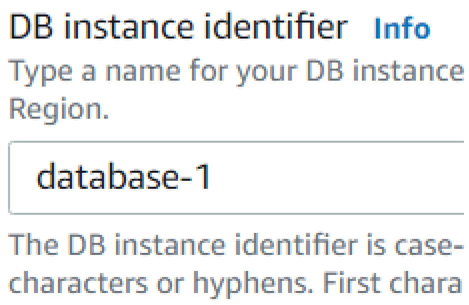
For the database instance size,



DB instance size

○ Production	○ Dev/Test	● Free tier
db.r6g.xlarge 4 vCPUs 32 GiB RAM 500 GiB 1.017 USD/hour	db.r6g.large 2 vCPUs 16 GiB RAM 100 GiB 0.231 USD/hour	db.t2.micro 1 vCPUs 1 GiB RAM 20 GiB 0.020 USD/hour

Under settings, for database name, a simply name such as “deploy08-db” will work



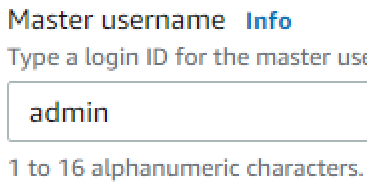
DB instance identifier Info

Type a name for your DB instance
Region.

database-1

The DB instance identifier is case-
characters or hyphens. First chara

Take note of the username that you create



Master username Info

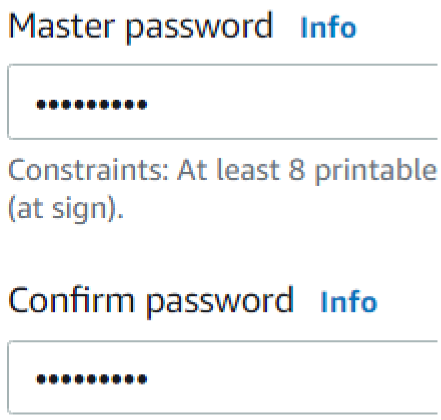
Type a login ID for the master use

admin

1 to 16 alphanumeric characters.

When creating a password, make sure to write it down

Password: abc123abc



Master password Info

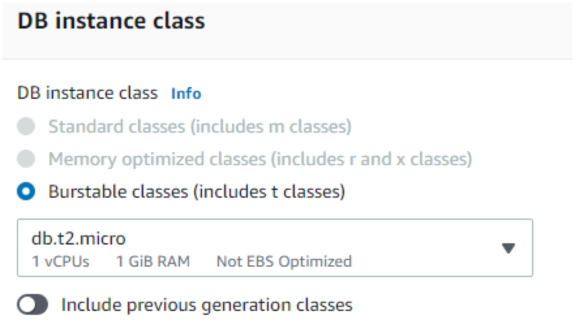
•••••••••

Constraints: At least 8 printable
(at sign).

Confirm password Info

•••••••••

DB Instance class can be default



DB instance class

DB instance class Info

○ Standard classes (includes m classes)

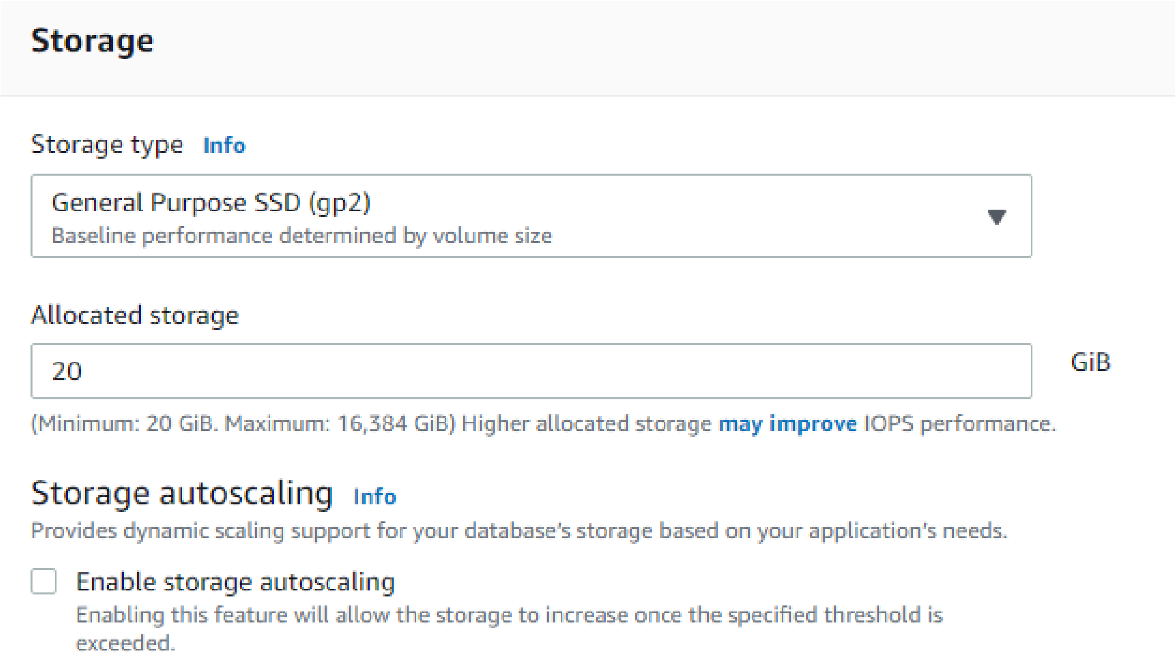
○ Memory optimized classes (includes r and x classes)

● Burstable classes (includes t classes)

db.t2.micro
1 vCPUs 1 GiB RAM Not EBS Optimized

Include previous generation classes

For storage, we can leave the default 20GiB General Purpose SSD and disable autoscaling



Storage

Storage type Info

General Purpose SSD (gp2)
Baseline performance determined by volume size

Allocated storage

20 GiB

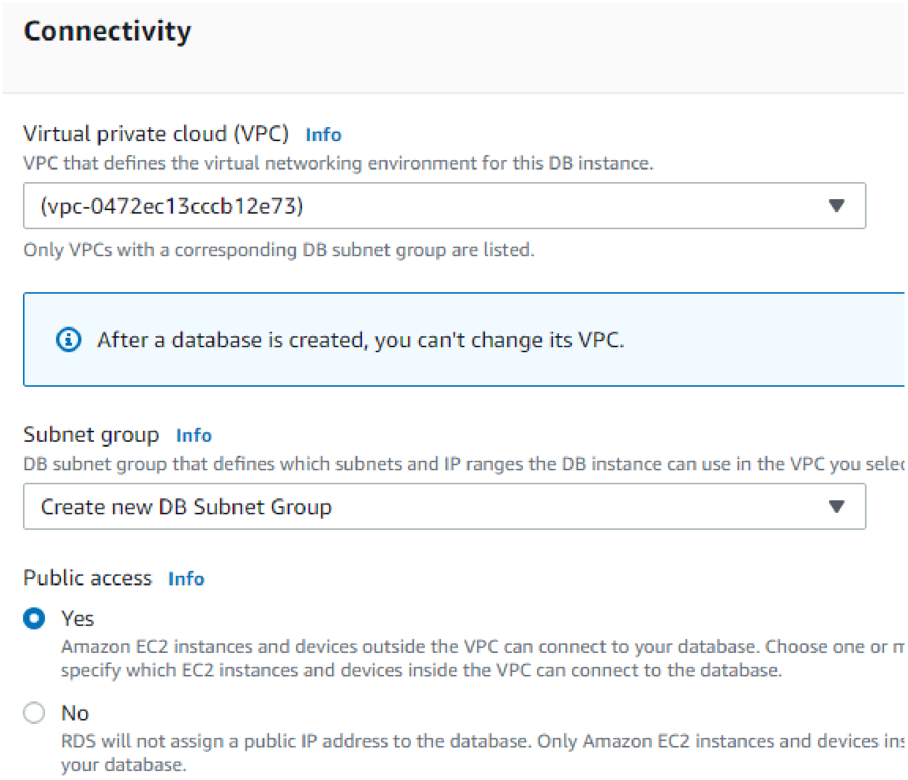
(Minimum: 20 GiB. Maximum: 16,384 GiB) Higher allocated storage **may improve** IOPS performance.

Storage autoscaling Info

Provides dynamic scaling support for your database's storage based on your application's needs.

☐ Enable storage autoscaling
Enabling this feature will allow the storage to increase once the specified threshold is exceeded.

Under Connectivity, we will need to select the VPC that was created during CloudFormation. You can find out under the VPC service. The IPv4 CIDR for the CloudFormation VPC is 192.168.0.0/16. We also need to make sure the database is publicly accessed.



Connectivity

Virtual private cloud (VPC)

VPC that defines the virtual networking environment for this DB instance.

(vpc-0472ec13cccb12e73) ▼

Only VPCs with a corresponding DB subnet group are listed.

After a database is created, you can't change its VPC.

Subnet group

DB subnet group that defines which subnets and IP ranges the DB instance can use in the VPC you selec

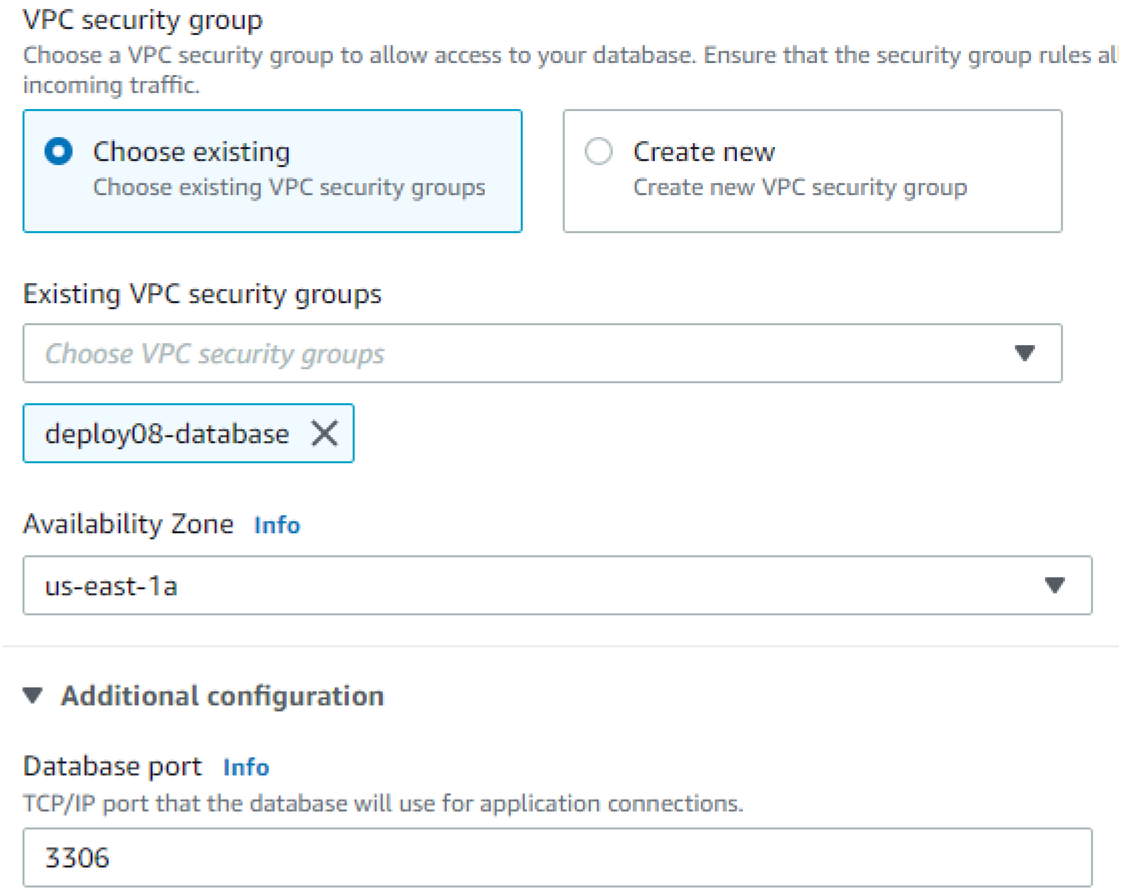
Create new DB Subnet Group ▼

Public access

● Yes
Amazon EC2 instances and devices outside the VPC can connect to your database. Choose one or m
specify which EC2 instances and devices inside the VPC can connect to the database.

○ No
RDS will not assign a public IP address to the database. Only Amazon EC2 instances and devices ins
your database.

For VPC security groups, we can select the existing security group that was created during CloudFormation called "deploy08-database"



VPC security group

Choose a VPC security group to allow access to your database. Ensure that the security group rules al incoming traffic.

● Choose existing
Choose existing VPC security groups

○ Create new
Create new VPC security group

Existing VPC security groups

Choose VPC security groups

deploy08-database ✕

Availability Zone Info

us-east-1a

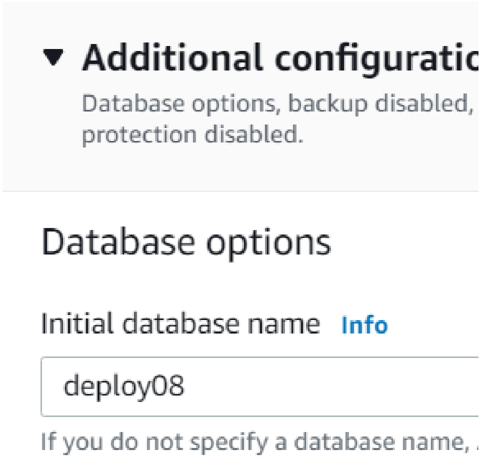
▼ Additional configuration

Database port Info

TCP/IP port that the database will use for application connections.

3306

Under Additional Configuration, put an initial database name



▼ Additional configuratic

Database options, backup disabled, protection disabled.

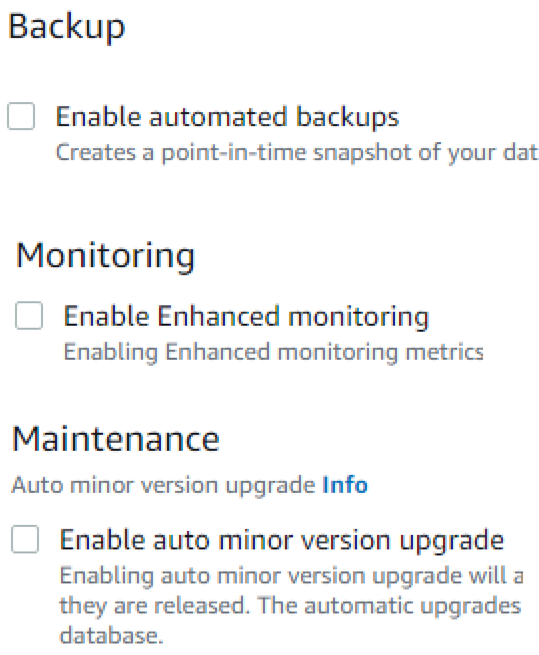
Database options

Initial database name Info

deploy08

If you do not specify a database name, .

Disable Backup, Monitoring, and Maintenance



Backup

□ Enable automated backups
Creates a point-in-time snapshot of your database

Monitoring

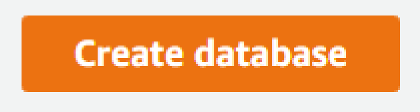
□ Enable Enhanced monitoring
Enabling Enhanced monitoring metrics

Maintenance

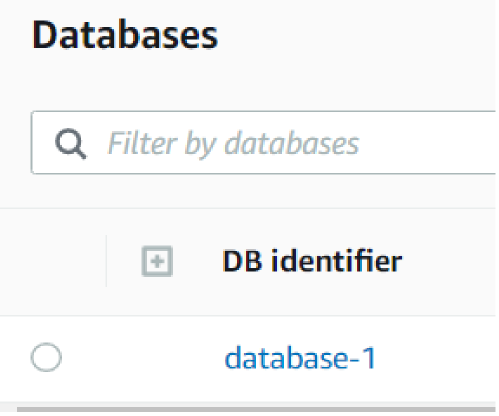
Auto minor version upgrade **Info**

□ Enable auto minor version upgrade
Enabling auto minor version upgrade will a
they are released. The automatic upgrades
database.

Once all that is configured, we can proceed and created the database



Once the database has been created, click on the database

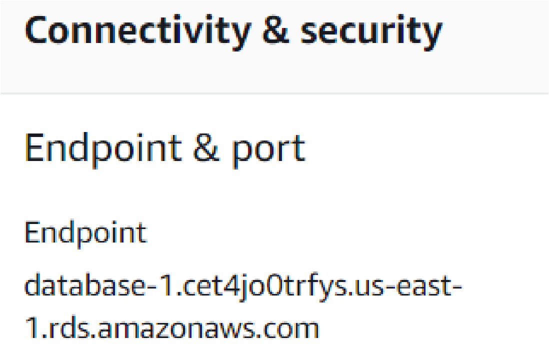


Databases

Filter by databases

	DB identifier
○	database-1

Under connectivity & security, copy the Endpoint down



Connectivity & security

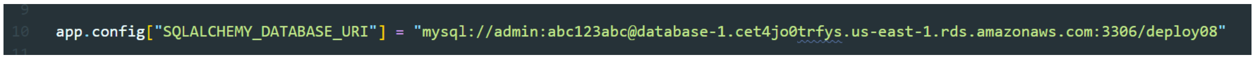
Endpoint & port

Endpoint

database-1.cet4jo0trfys.us-east-1.rds.amazonaws.com

Once the database is running, we can go to the backend code and alter it. Inside the backend folder, open the app.py file. In line 10, we need to update our connection information following the provided schema

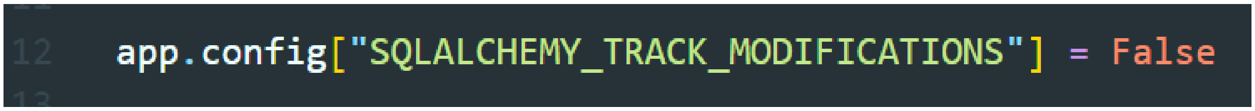
‘mysql://**DBUsername**:**DBPassword@DBEndpoint**:3306/**DatabaseInitialName**’



```
10   app.config["SQLALCHEMY_DATABASE_URI"] = "mysql://admin:abc123abc@database-1.cet4jo0trfys.us-east-1.rds.amazonaws.com:3306/deploy08"
```

In line 12, we will also need to change “app.config['SQLALCHEMY_DATABASE_TRACK_MODIFICATIONS'] = False” to “**app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False**”

Change the following **“app.config["SQLALCHEMY_DATABASE_TRACK_MODIFICATIONS"] = False**” to **“app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False”**

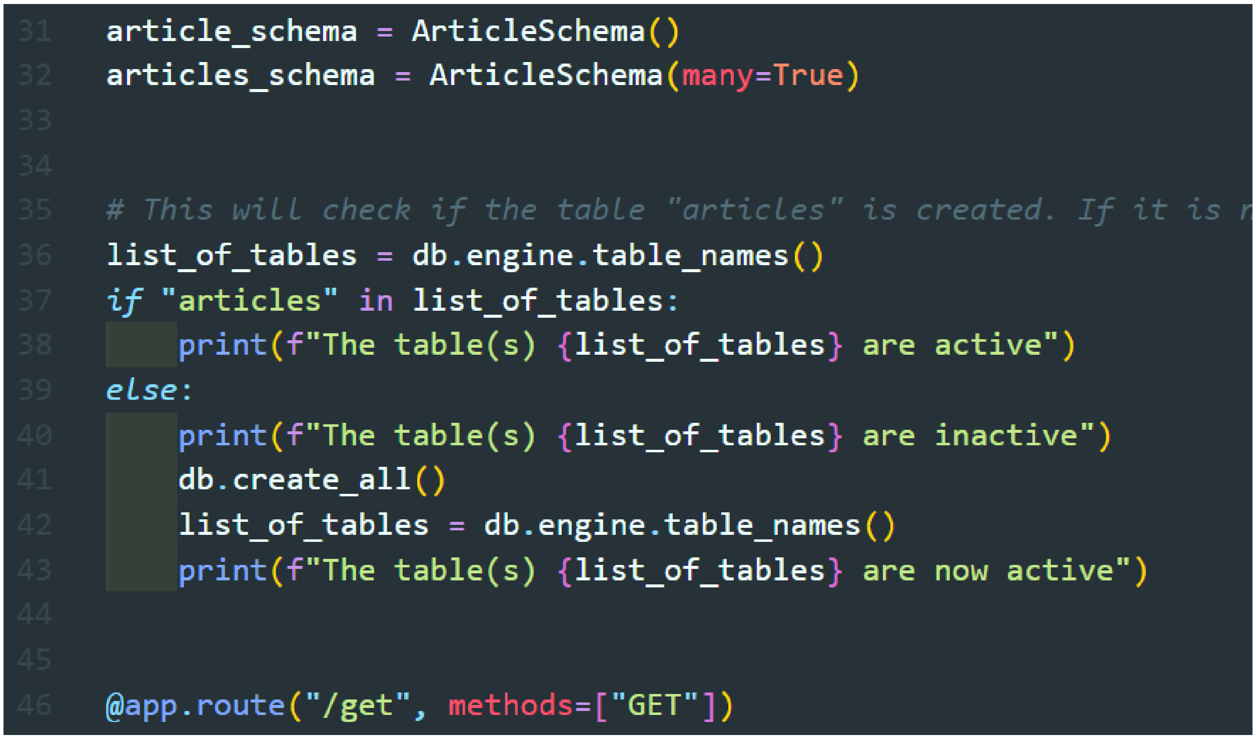


```
12   app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False
```

This code will not create a table. We will need to alter the code so it will check if the table named “articles” is created. If it is created, the code will continue. If it is not created, the program will create the table for us. Navigate to line 32 in the code. After that line, we will add our code

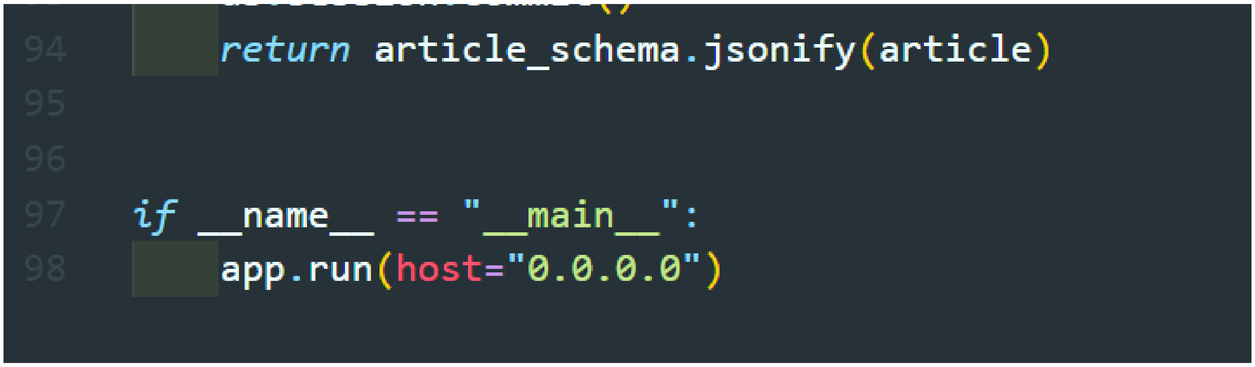
```
# This will check if the table "articles" is created. If it is not created, it will create it. If it is created, it will continue
list_of_tables = db.engine.table_names()
if "articles" in list_of_tables:
    print(f"The table(s) {list_of_tables} are active")
else:
    print(f"The table(s) {list_of_tables} are inactive")
    db.create_all()
```

```
list_of_tables = db.engine.table_names()
print(f"The table(s) {list_of_tables} are now active")
```



```
article_schema = ArticleSchema()
articles_schema = ArticleSchema(many=True)


# This will check if the table "articles" is created. If it is r
list_of_tables = db.engine.table_names()
if "articles" in list_of_tables:
    print(f"The table(s) {list_of_tables} are active")
else:
    print(f"The table(s) {list_of_tables} are inactive")
    db.create_all()
    list_of_tables = db.engine.table_names()
    print(f"The table(s) {list_of_tables} are now active")


@app.route("/get", methods=["GET"])
```

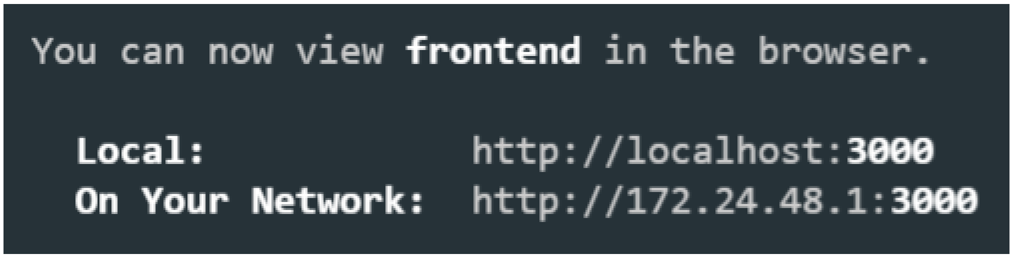
At the bottom of the backend application code, we have to alter the app.run() command. This will allow our flask application to run



```
    return article_schema.jsonify(article)


if __name__ == "__main__":
    app.run(host="0.0.0.0")
```

Once the code is updated, we can then test the application locally. For the front end application, change directory into the frontend folder and run the following command

cd application/frontend
npm install
npm run start

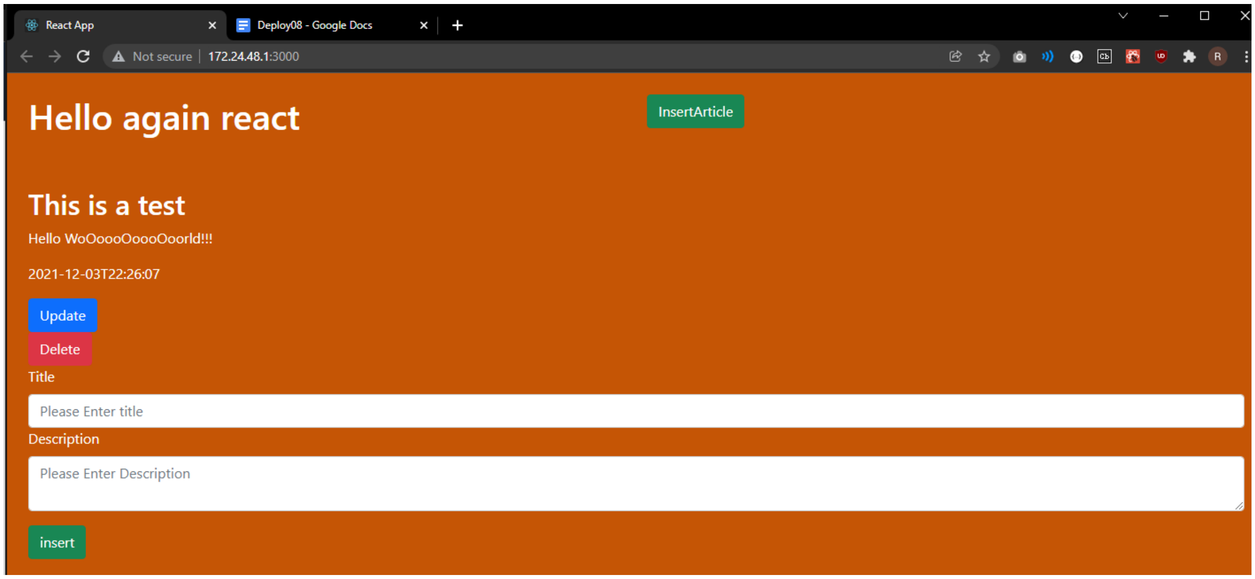


```
You can now view frontend in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://172.24.48.1:3000
```

Change directory into the backend application and open another terminal. We can run the following commands to start the backend application.

```
cd application/backend
pip3 install -r .\requirements.txt
$env:FLASK_APP = ".\application.py"
flask run
```

On the frontend, you should be able to add a new query to the table.



Hello again react

This is a test

Hello WoOoooOoooOoorld!!!

2021-12-03T22:26:07

Title

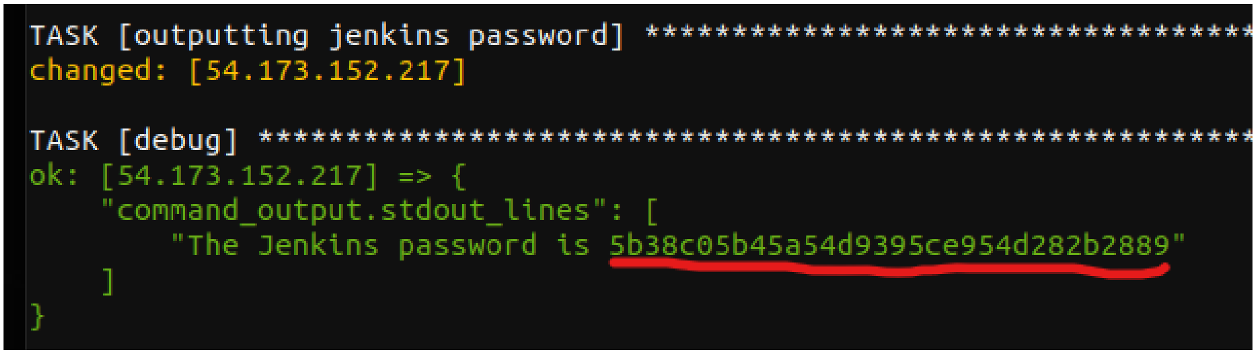
Please Enter title

Description

Please Enter Description

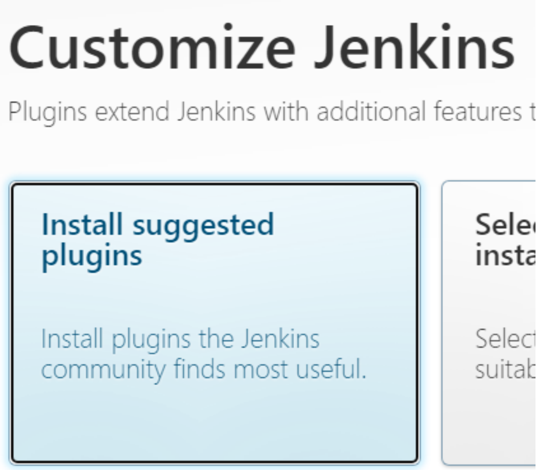
We can move on once we have created all the cloud resources, configured each EC2s, created the RDS database, and tested it locally. Access our Jenkins application using the Public IPv4 of the Controller EC2 Agent followed by port 8080.
http://54.173.152.217:8080/

Enter the password that was outputted in the ansible-playbook script earlier.



```
TASK [outputting jenkins password] ***********************************
changed: [54.173.152.217]

TASK [debug] *********************************************************
ok: [54.173.152.217] => {
    "command_output.stdout_lines": [
        "The Jenkins password is 5b38c05b45a54d9395ce954d282b2889"
    ]
}
```

Install the suggest plugins

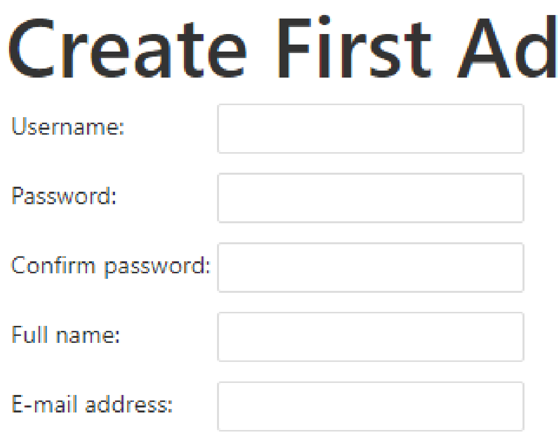


Customize Jenkins

Plugins extend Jenkins with additional features t

Install suggested plugins
Install plugins the Jenkins community finds most useful.
Selec insta
Select suitab

Create a basic Jenkins user.



Create First Admin User

Username:

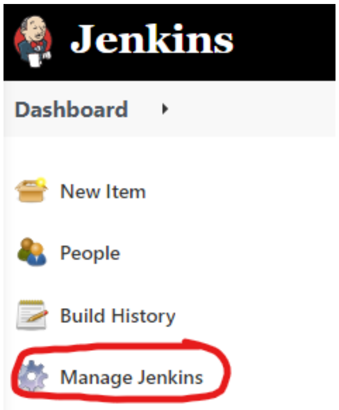
Password:

Confirm password:

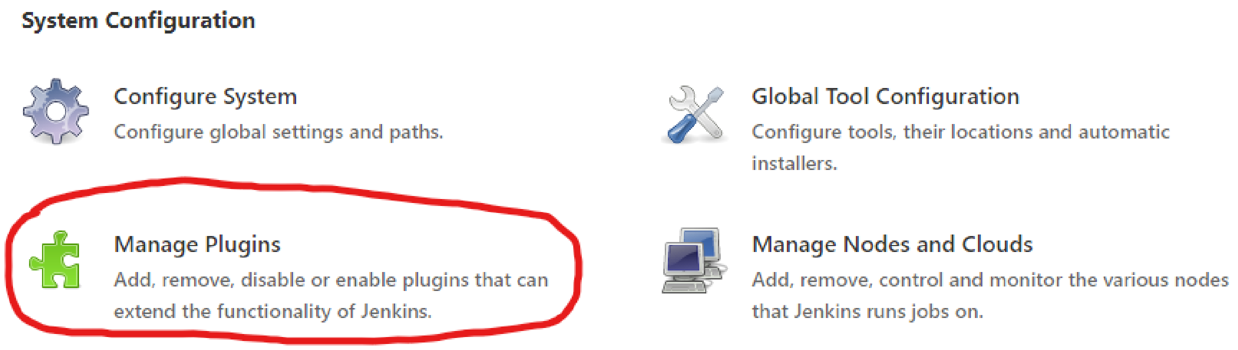
Full name:

E-mail address:

Inside of Jenkins, we will need to install a couple of plugins. Navigate to Manage Jenkins



Select Manage Plugins. This is where we will install and uninstall plugins



System Configuration

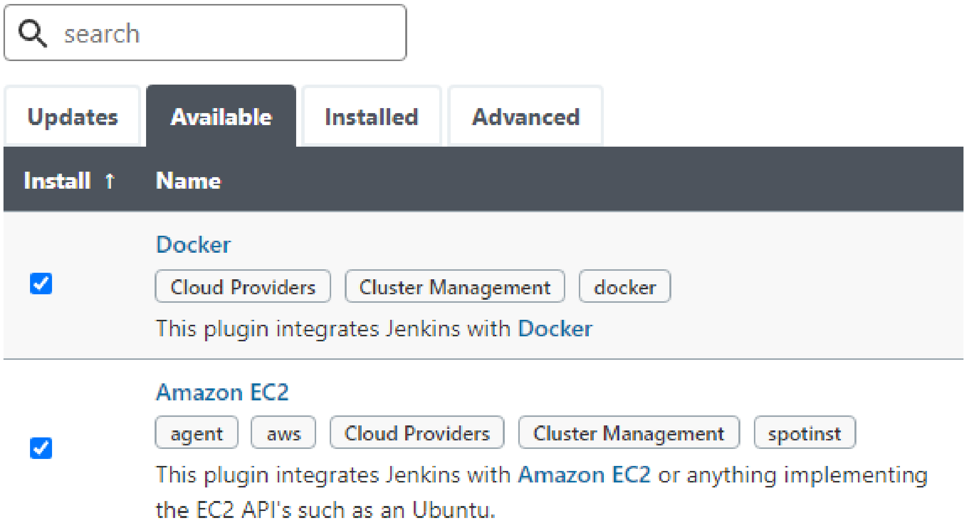
Configure System
Configure global settings and paths.

Global Tool Configuration
Configure tools, their locations and automatic installers.

Manage Plugins
Add, remove, disable or enable plugins that can extend the functionality of Jenkins.

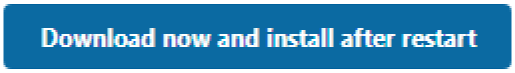
Manage Nodes and Clouds
Add, remove, control and monitor the various nodes that Jenkins runs jobs on.

We will be installing Amazon EC2, and Docker

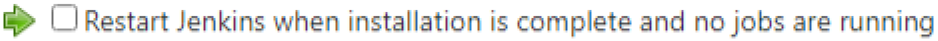


Install ↑	Name
☑	Docker Cloud Providers · Cluster Management · docker This plugin integrates Jenkins with Docker
☑	Amazon EC2 agent · aws · Cloud Providers · Cluster Management · spotinst This plugin integrates Jenkins with Amazon EC2 or anything implementing the EC2 API's such as an Ubuntu.

At the bottom of the screen, download the following plugins



Select Restart when installation is complete

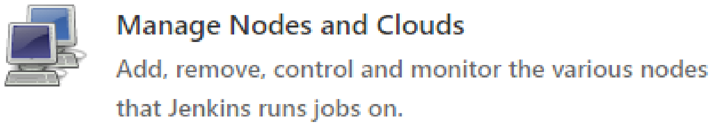


□ Restart Jenkins when installation is complete and no jobs are running

(This may take a couple of minutes. Keep refreshing the page occasionally)

Once Jenkins has restarted, sign back in. First we will remove all building powers on the controller agentWe can now configure Nodes inside of Jenkins. This will allow us to attach our agent and production EC2 to the controller EC2. Inside the Dashboard, select Manage Jenkins

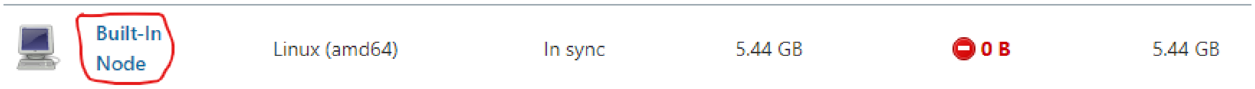
Under System Configurations, we need to add a node so select “Manage Nodes and Clouds”



Manage Nodes and Clouds

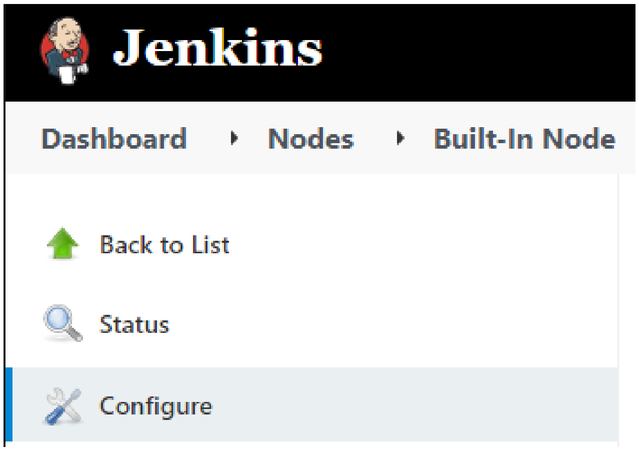
Add, remove, control and monitor the various nodes that Jenkins runs jobs on.

First, select the controller node called (Built-In Node)

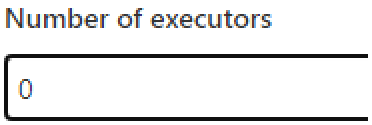


Built-In Node	Linux (amd64)	In sync	5.44 GB	0 B	5.44 GB

Select configure in the left side



Change number of executors to 0. This will allow us to not build on the controller agent.



Number of executors

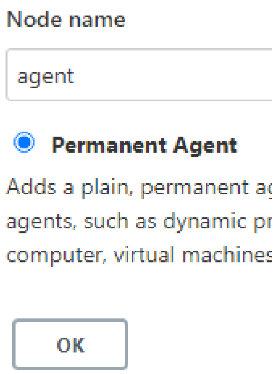
0

Save the changes

Now select go back to list

Select New Node in the left side

Name the agent, “Node”



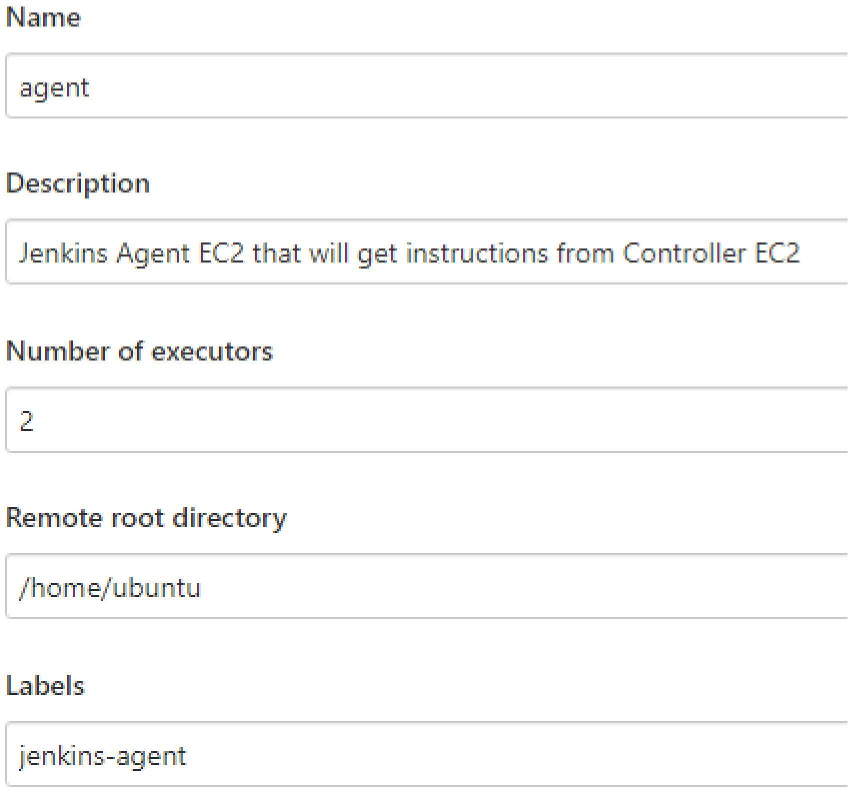
Node name

agent

● **Permanent Agent**

Adds a plain, permanent ag
agents, such as dynamic pr
computer, virtual machines

In the next steps we will need to fill out some configurations. Use the following…



Name

agent

Description

Jenkins Agent EC2 that will get instructions from Controller EC2

Number of executors

2

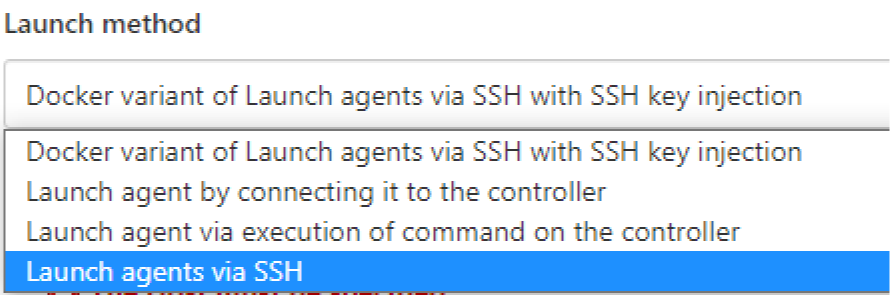
Remote root directory

/home/ubuntu

Labels

jenkins-agent

We will then need to set up how we will access our agent which is hosted on an EC2. The CloudFormation template sets up SSH access from anywhere. For the Launch method, select via SSH.

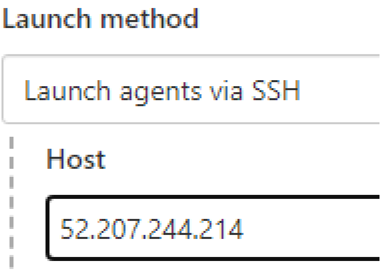


Launch method

Docker variant of Launch agents via SSH with SSH key injection

Docker variant of Launch agents via SSH with SSH key injection
Launch agent by connecting it to the controller
Launch agent via execution of command on the controller
Launch agents via SSH

The Host IP is the public IPv4 of the Agent EC2 we created.



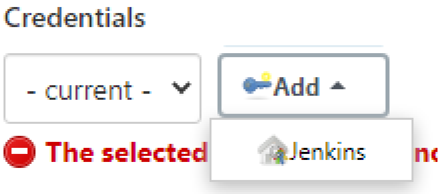
Launch method

Launch agents via SSH

Host

52.207.244.214

We need to add credentials to access the EC2 instance.



Credentials

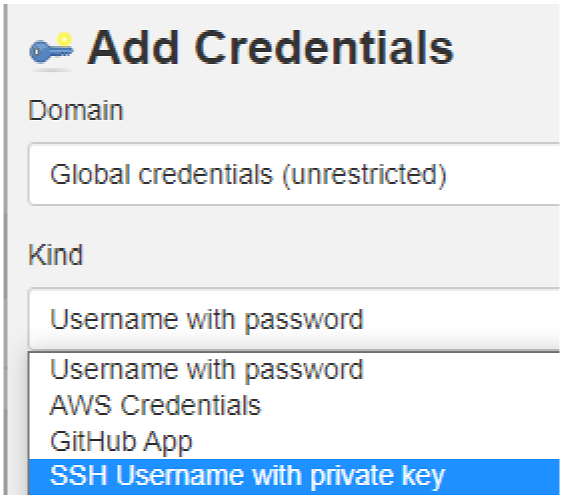
- current -

Add

Jenkins

The selected

For the settings, change the Kind to SSH username with a private key.



Add Credentials

Domain

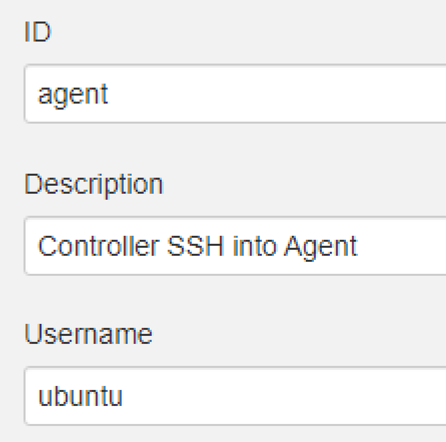
Global credentials (unrestricted)

Kind

Username with password

Username with password
AWS Credentials
GitHub App
SSH Username with private key

The username is important, It will be the same as our AMI.



ID

agent

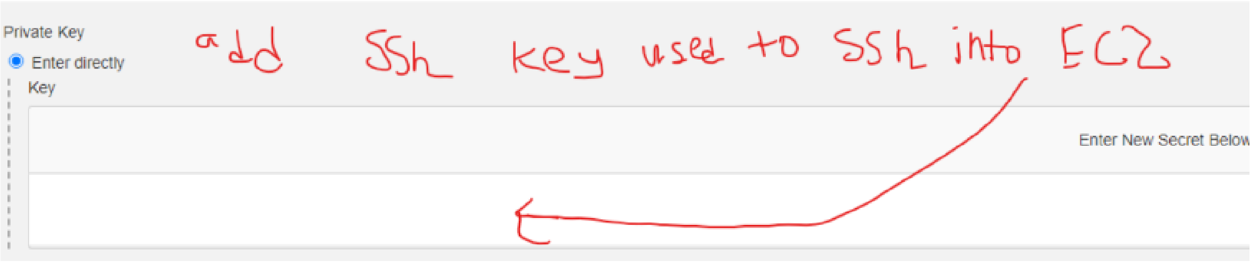
Description

Controller SSH into Agent

Username

ubuntu

We then need to add our private key directly so we can SSH into the EC2. Find your PEM key that you use to SSH into EC2 instances and paste the contents into the field



Private Key

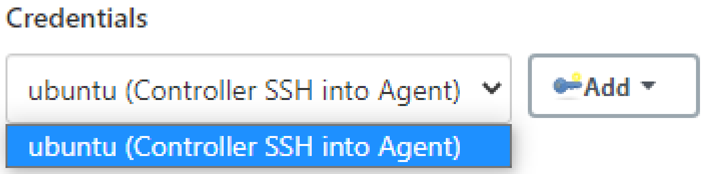
● Enter directly

Key

add SSh key used to SSh into EC2

Enter New Secret Below

Once the credentials are set, we need to select it.

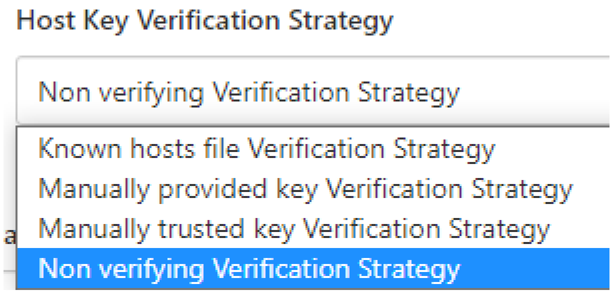


Credentials

ubuntu (Controller SSH into Agent)

ubuntu (Controller SSH into Agent)

We also need to change the host key verification strategy to “Non key verification strategy”.

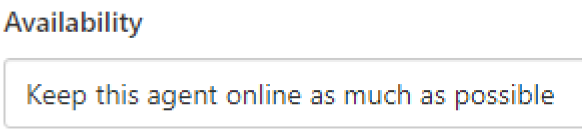


Host Key Verification Strategy

Non verifying Verification Strategy

- Known hosts file Verification Strategy
- Manually provided key Verification Strategy
- Manually trusted key Verification Strategy
- Non verifying Verification Strategy

We must keep this agent online as much as possible to avoid errors.

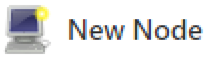


Availability

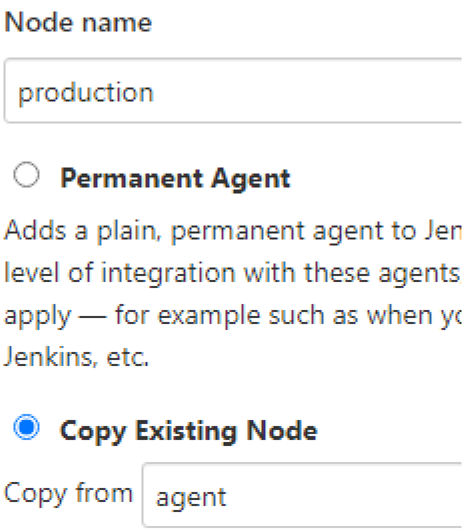
Keep this agent online as much as possible

Once everything is set up, save it.

Another node is needed for the production EC2. This will be used to deploy our application.



Name the agent and copy from the existing Node



Node name

production

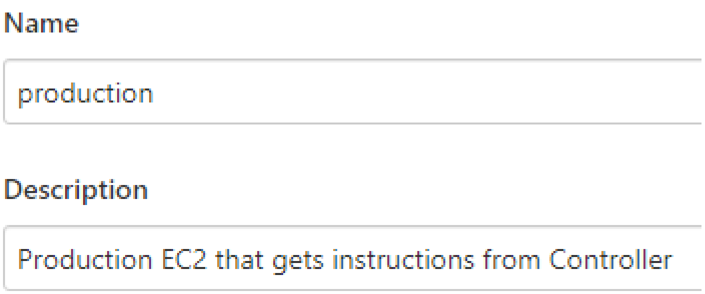
○ **Permanent Agent**

Adds a plain, permanent agent to Jen
level of integration with these agents
apply — for example such as when yc
Jenkins, etc.

● **Copy Existing Node**

Copy from agent

There are a couple of details that we need to configure



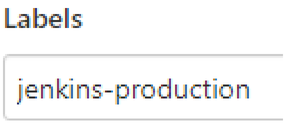
Name

production

Description

Production EC2 that gets instructions from Controller

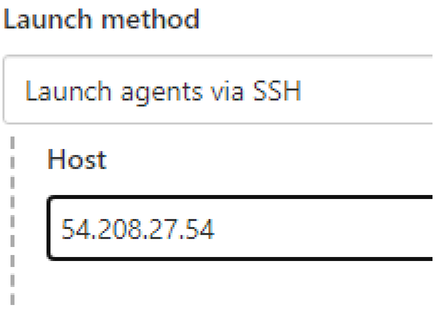
We will need to change the label so we can call different agents



Labels

jenkins-production

Under launch methods, we need to change the Public IPv4 to the Production EC2's IPv4



Launch method

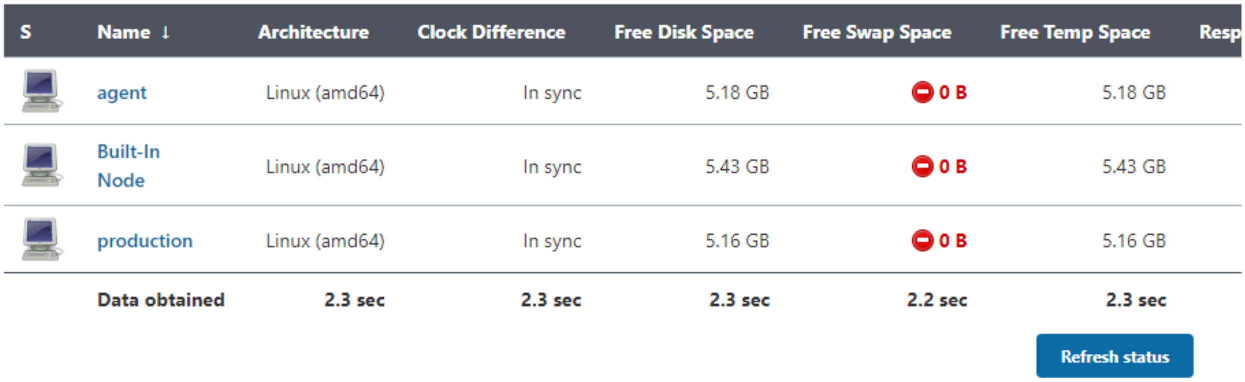
Launch agents via SSH

Host

54.208.27.54

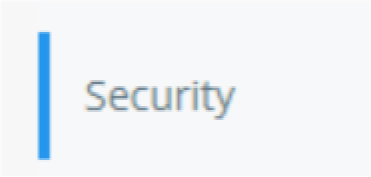
Those are all the settings we need to alter. Save the agent

You should see all your agents online

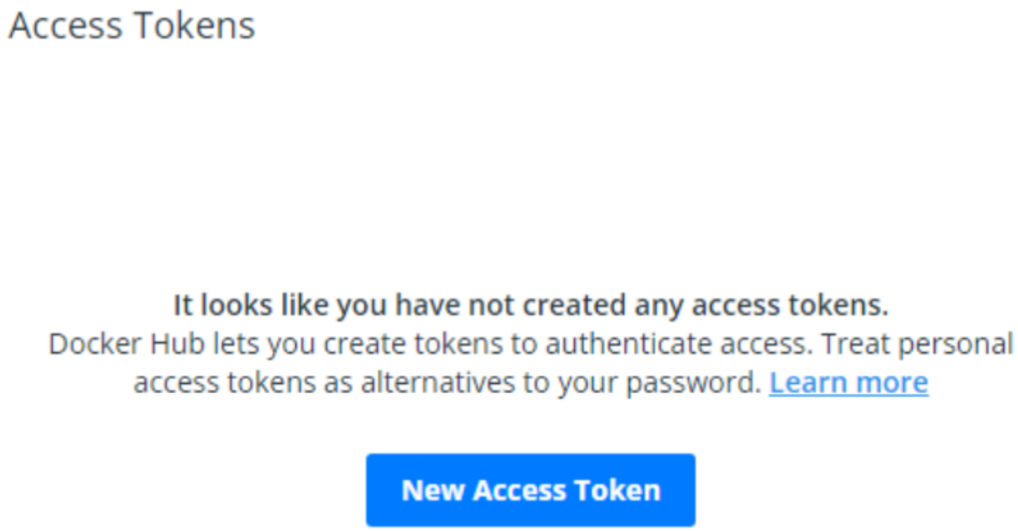


S	Name ↓	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Resp
	agent	Linux (amd64)	In sync	5.18 GB	⊖ 0 B	5.18 GB	
	Built-In Node	Linux (amd64)	In sync	5.43 GB	⊖ 0 B	5.43 GB	
	production	Linux (amd64)	In sync	5.16 GB	⊖ 0 B	5.16 GB	
	Data obtained	2.3 sec	2.3 sec	2.3 sec	2.2 sec	2.3 sec	

We will now need to set up Docker Credentials in Jenkins. This will allow our agent to push our docker images to DockerHub. First, we will need to create a Docker Access Token. Go to -> https://hub.docker.com/settings/security and signup/signin. Once you are logged in to DockerHub, select the security tab in Account Settings



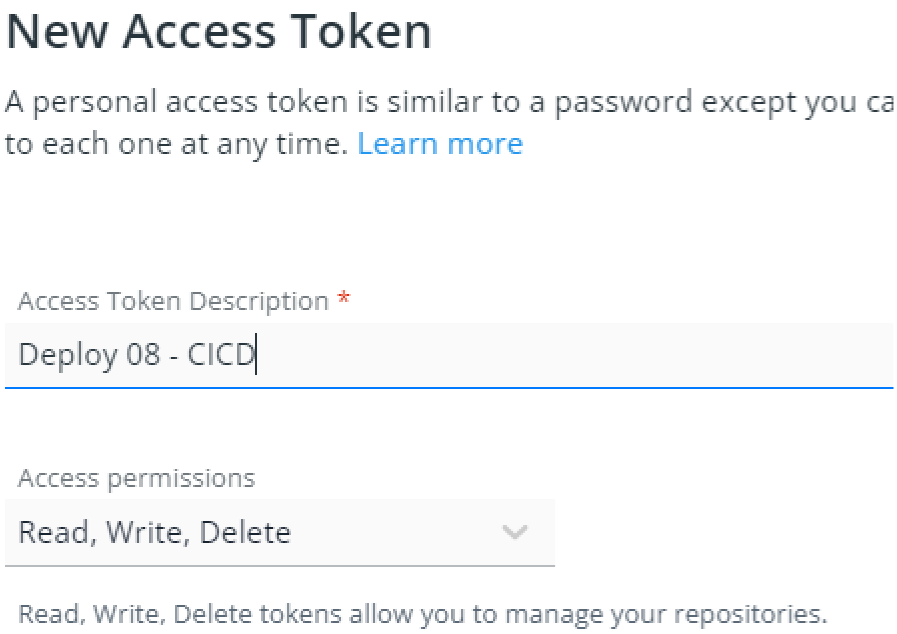
We need to create an access token which will be used to gain access to DockerHub without a password. Create a new access token



Access Tokens

It looks like you have not created any access tokens.
Docker Hub lets you create tokens to authenticate access. Treat personal access tokens as alternatives to your password. Learn more

When creating a new access token, you can put anything inside the description



New Access Token

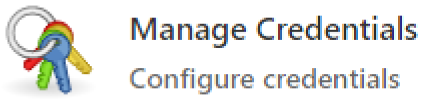
A personal access token is similar to a password except you ca
to each one at any time. Learn more

Access Token Description *
Deploy 08 - CICD

Access permissions
Read, Write, Delete

Read, Write, Delete tokens allow you to manage your repositories.

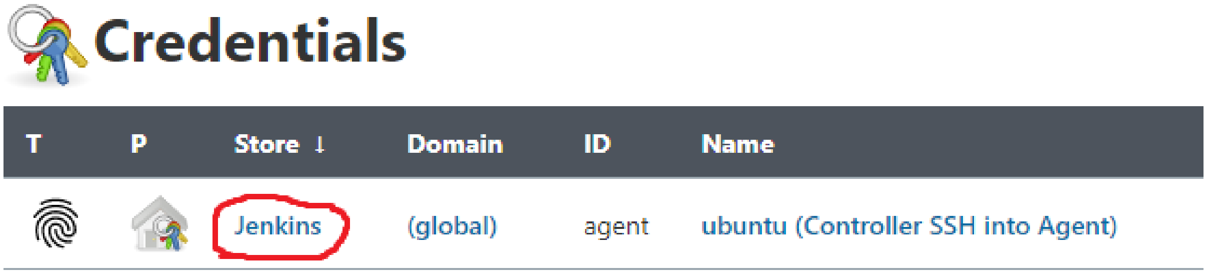
Generate your token and make SURE to copy the personal access token. Navigate back to the Jenkins Controller and go to the dashboard. Select Manage Jenkins. We then need to select Manage Credentials under security.



Manage Credentials

Configure credentials

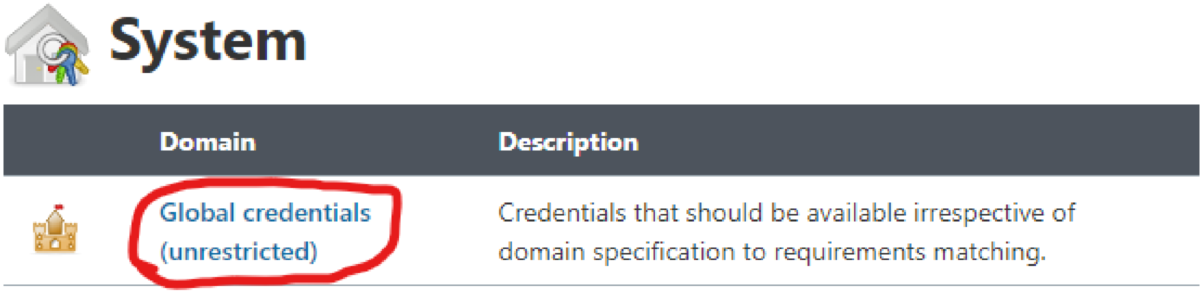
Select the following credentials



Credentials

T	P	Store ↓	Domain	ID	Name
		Jenkins	(global)	agent	ubuntu (Controller SSH into Agent)

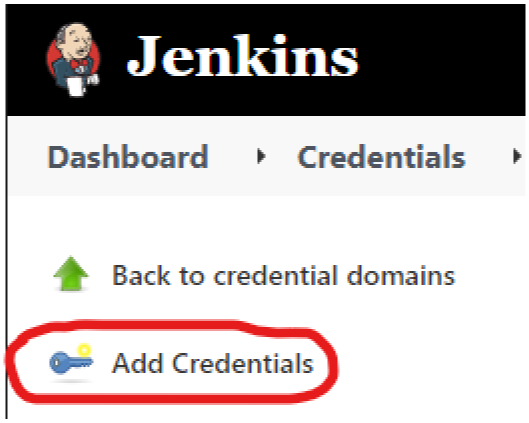
Once inside that credentials, we need to select Global Credentials



System

	Domain	Description
	Global credentials (unrestricted)	Credentials that should be available irrespective of domain specification to requirements matching.

This page is where we will configure our credentials to DockerHub. Choose Add Credentials



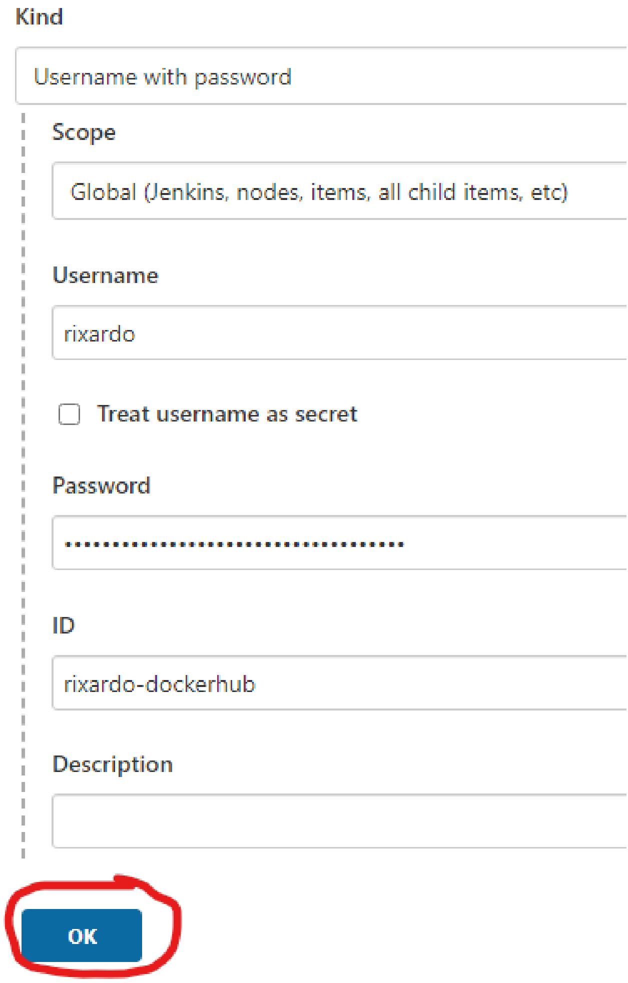
Jenkins

Dashboard ▸ Credentials ▸

Back to credential domains

Add Credentials

For the following settings, the Username should be your DockerHub username. The password is the personal access token that you created on DockerHub. The ID will simply have your DockerHub username followed by -dockerhub. (ex. rixardo-dockerhub)



Kind

Username with password

Scope

Global (Jenkins, nodes, items, all child items, etc)

Username

rixardo

□ Treat username as secret

Password

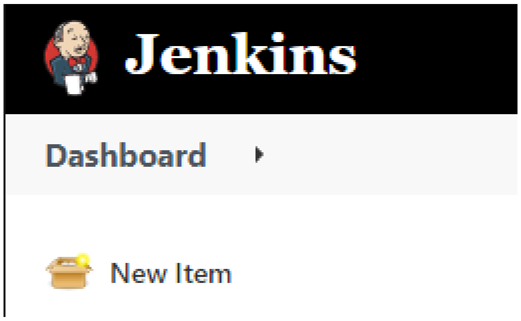
••••••••••••••••••••••••••••••••••••

ID

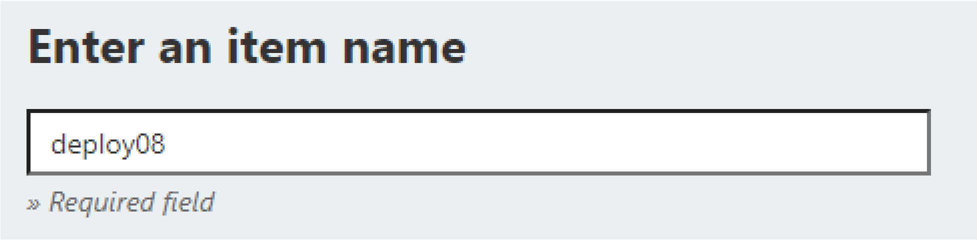
rixardo-dockerhub

Description

We can now move onto creating the actual Multibranch pipeline that will do all our stages. Navigate to the Jenkins Dashboard and select New Items



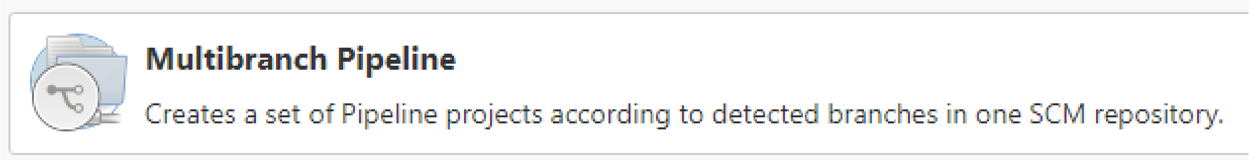
When creating an item, we will be making a multibranch pipeline. Any name for this item will be acceptable.



Enter an item name

deploy08

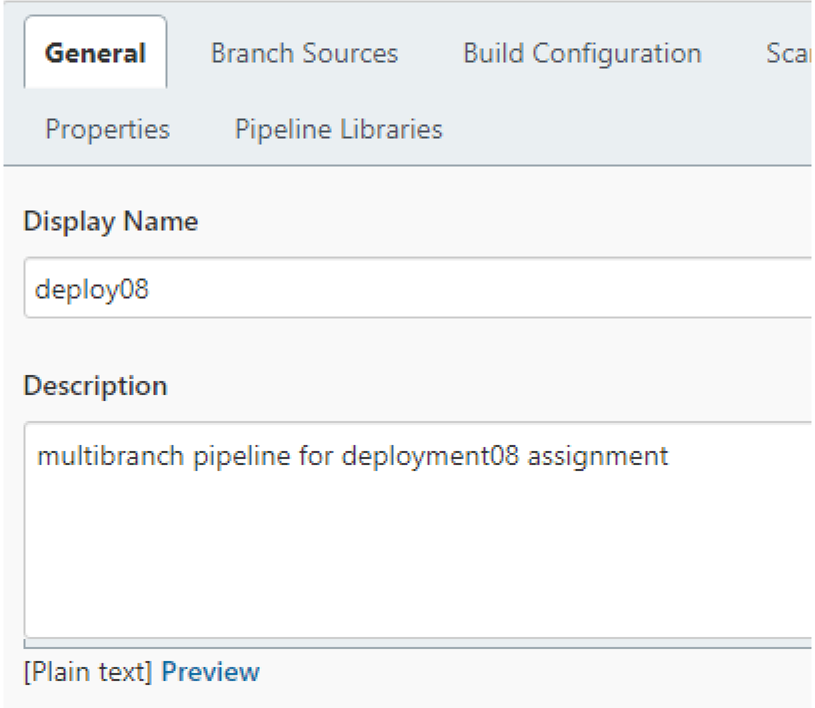
» *Required field*



Multibranch Pipeline

Creates a set of Pipeline projects according to detected branches in one SCM repository.

Inside the pipeline, we will need to configure multiple sections. For display name and description select a simple response…



Display Name

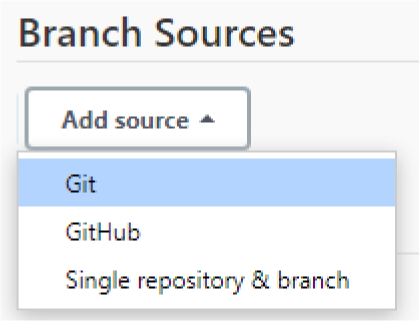
deploy08

Description

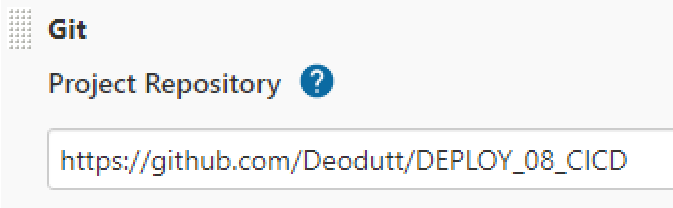
multibranch pipeline for deployment08 assignment

[Plain text] Preview

For Branch sources we will use GitHub. We are pulling our application from GitHub

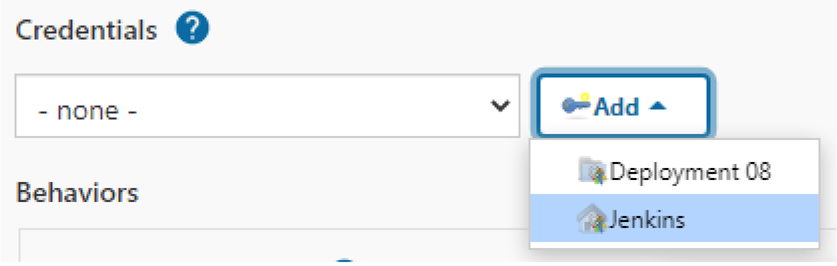


Under Project Repository, link the forked repository of the assignment



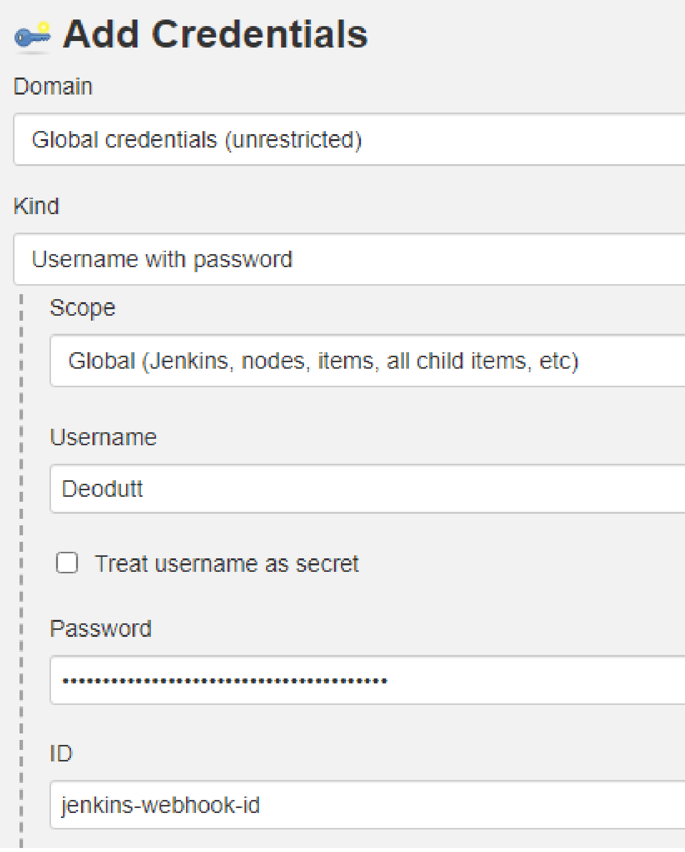
Project Repository: https://github.com/Deodutt/DEPLOY_08_CICD

We will need to add credentials that will allow Jenkins to interact with GitHub. Select Jenkins in the dropdown



Credentials: - none -

When adding the credentials, the username will be our GitHub Username. The password will be your GitHub personal access token that you can create using -> here. The ID can be a simple jenkins-webhook-id



Add Credentials

Domain: Global credentials (unrestricted)

Kind: Username with password

Scope: Global (Jenkins, nodes, items, all child items, etc)

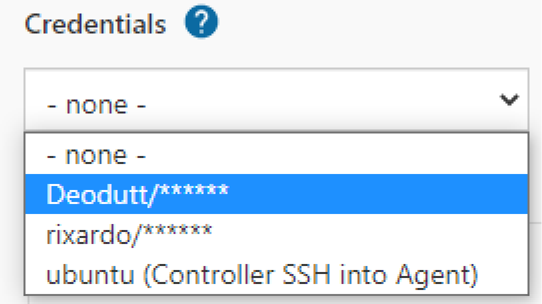
Username: Deodutt

□ Treat username as secret

Password: ••

ID: jenkins-webhook-id

After the credentials are credited, select the credentials.

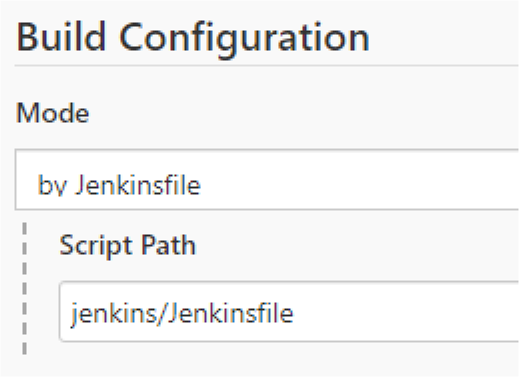


Credentials

- none -

- none -
Deodutt/******
rixardo/******
ubuntu (Controller SSH into Agent)

When configuring Build Configuration, it is important to have the Jenkinsfile in your GitHub in the correct path. In my case, I have my jenkins script inside a folder called jenkins in my repository.



Build Configuration

Mode

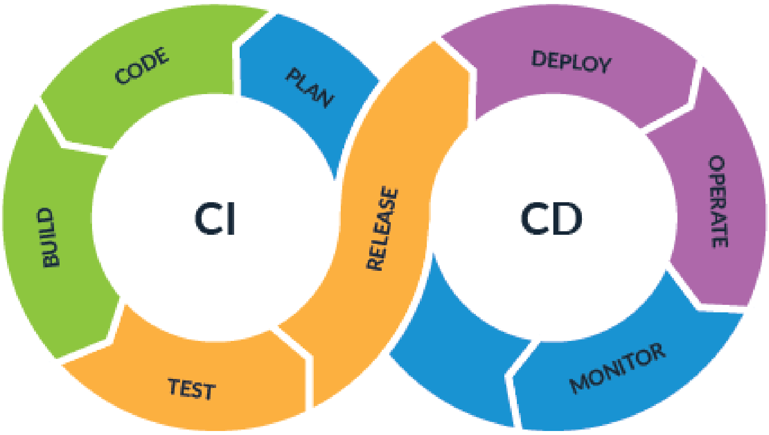
by Jenkinsfile

Script Path

jenkins/Jenkinsfile

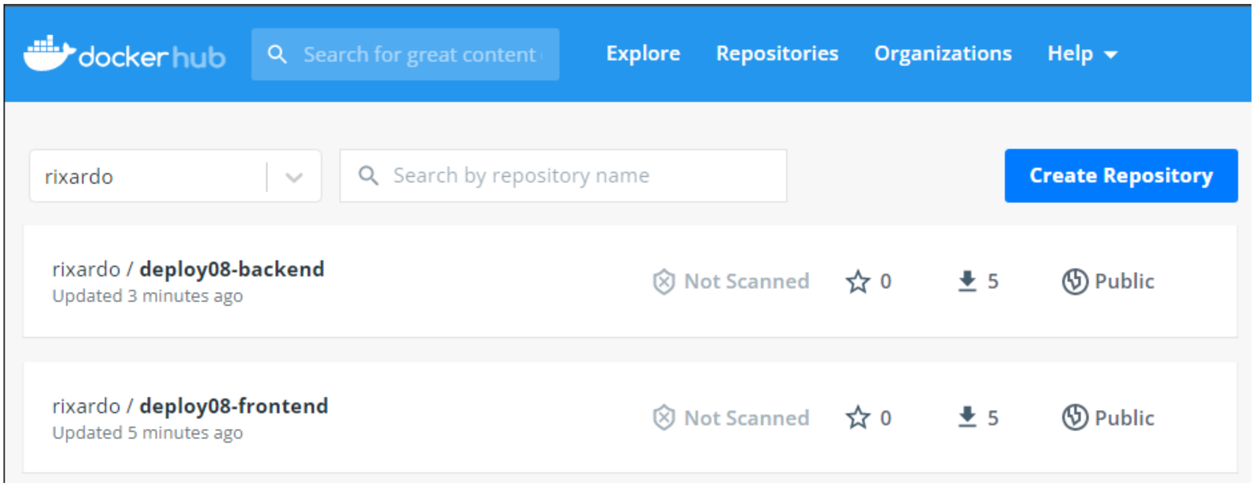
Once that is configured we can save

Inside the Jenkinsfile (in GitHub) we can edit the file and attach our pipeline script.



Inside the Jenkinsfile, we will have multiple stages. The planning stage starts fromAll code will be hosted on GitHub.

Successful screenshots



rixardo / **deploy08-backend**
Updated 3 minutes ago
Not Scanned ☆ 0 ⬇ 5 Public

rixardo / **deploy08-frontend**
Updated 5 minutes ago
Not Scanned ☆ 0 ⬇ 5 Public

So I have completed…
creating resources using CloudFormation
Configuring each EC2 using ansbil;e
Setting up RDS database

Editing the application locally and testing if connection to database works

Setting up Jenkins
Downloading Docker and Amazon EC2 Plugin
Adding the two nodes Agent and Production
Set up Docker Credentials

What I need to do…
Set up Multibranch Pipeline
Create Groovy Script

1st Stage is Building

```
npm install
npm run build
sudo npm install -g serve
serve -s build &
```

2nd Stage is testing

```
npm install cypress
npm install mocha
sudo apt-get install xvfb -y
npx cypress run --spec ./cypress/integration/test.spec.js
```

Then after these two stages will get the results using Junit
(Figure out how to implement ansible to encrypt it)

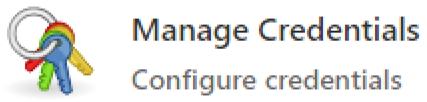
```
post {
        always {
        junit 'results/cypress-report.xml'
  }
```

Then build the application using docker and push it to dockerhub

cloudwatch?
Stress test
Security step

Finally, we can create a pipeline that will do all our steps such as building, testing, pushing, etc.

We can now move onto creating our Docker Credentials which will allow our agent to push to DockerHub. In the dashboard, select Manage Jenkins. We can then select Manage Credentials.



Manage Credentials

Configure credentials

Aws ecr
Put command to docker jenkins file
Top 2 commands use aws cli to grab standard

Create the entire application in container
Have ansible pull the container
Create it on an ec2

Ansible to configure all the packages needed to create the container on the ec2.
Download docker on the testing ec2. Download docker on the production ec2.
Have ansible deploy to the ec2 on the docker

Have ansible install docker. After it installs, create the container

Use ansible for configuration management.
Node docker and get ansible to make the container. react downloaded

A yaml file for testing environment

Testing environment worked with react and javascript application
When I want to run a test, know what to do

Containerize the application to move it around
Do jenkins do each step.
Test
Deployment

Ansible for one or two things

Cloudformation create environment
Ansible configure

Can use cloudformation to create the vpcs

Ansible to install docker

Ansible to configure a server

Install all the programs you need

Ansible for the secrets
Encrypt it

Security group

Split database

Can use rds and connect to application
ok
Just install ec2 on docker

Has application on container
Its portable
Can pass it over to testing environment.

Made it before, pull it down from dockerhub and put it into production environment

Created application on actual server then decided to create the image.

Create the image first,

Dockerfile to create application, take it, test it and then need to update it, update it

Log into ec2 and application and configure it

Create two application
Application connecting to test database
Application connect to rds database

Not testing the database

Create container

Use ansible to download node
Use anbile to configure docker and run the images

Use docker scp

Copy file
Ansible.builtin.copy

To another server

Can push over dockerfile and have ansible run that dockerfile
Have ansible make the dockerfile
Dont have to log inot the system and run dockerbuild
Jenkins build command

Jenkins use ansible
Can do it on multiple systems

Jenkins can do it on system at time.
Ansible to run it on multiple ec2s

Use ansible to push out insutrctuoin of whatever you want to build out in multiple systems

Configure ssh for

Top 10 modules for ansible

Package manager command
Service that can change service of system. Restart nginx/apache
copy
Debug

Ansible installed on production and test environment

Task 2

In this task, we have to figure out how to build the application and then create a build step in Jenkins pipeline. For this case, I will be splitting up my application. There will be a frontend application and a backend application

Task 3

For this task, we will need to create a test step that will test our front end application. I will be using cypress to test my front end application. Since ansible installed all the required cypress tools we will just have to run a simple command.

First, we will need to edit/create our test.spec.js file that is located in the cypress/integration folder in our root work directory. This file will explain the test that cypress will complete. The file will simply visit our front end application and check an HTML element header tag if it is equals to a certain text.

Inside your cypress/integration/test.spec.js file, paste the following test code

```
describe('Title', () => {
  it('has the right title', () => {
    // The application is hosted on port 3000
    cy.visit('http://localhost:3000')

    cy.get('h1')
      .invoke('text')
      .should("equal", "Hello again react")
    Cypress.Screenshot.defaults({
    capture: 'runner',
    })
    cy.screenshot();
    });
});
```

Inside the jenkins script, we will add a new stage called “Test”. This stage will run a cypress test at a specific path and then say it's completed. After the test has completed, the script will use junit to export a test report file

```
stages {
    // Testing the front end application
    stage('Test') {
        steps {
            sh '''
            npx cypress run --spec ./cypress/integration/test.spec.js
            '''
            sh 'echo "completed build"'
        }
        post {
            always {
            junit 'results/cypress-report.xml'
            }
        }
    }
}
```

Task 4

In this task, we will have to create an image of both our applications and push that image to Dockerhub. To do this task, we have set up DockerHub credentials earlier. This will aid us in the process of logging into DockerHub, building our image using Dockerfiles and pushing to DockerHub. Since we split up the frontend and backend there will also be two Dockerfiles located in each application directory

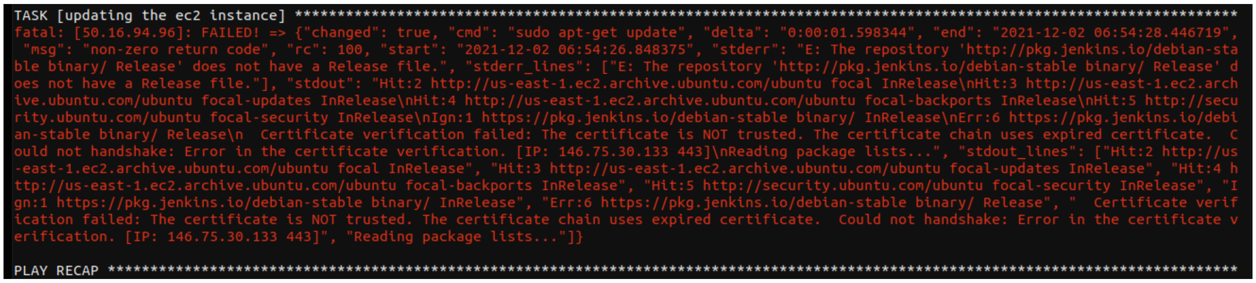
Inside the application/frontend/ directory, we will need to create a Dockerfile and paste the following contents. This will use an ubuntu image for the docker container. We will then update our OS and copy all the frontend application files from our agent to a new directory in the container. We will then change our working directory to the directory with our application and install a couple of vital applications. We will then use npm to install more dependencies, run our application, and run it in the background.

```
FROM ubuntu:latest
RUN apt-get update && apt-get upgrade -y
COPY ./application/frontend/ /home/ubuntu/app/
WORKDIR /home/ubuntu/app/
RUN DEBIAN_FRONTEND="noninteractive" apt-get -y install tzdata
RUN apt-get install nodejs -y
RUN apt-get install npm -y
RUN npm install
RUN npm run build
RUN npm install -g serve
EXPOSE 3000
CMD ["serve", "-s", "build"]
```

Task 5

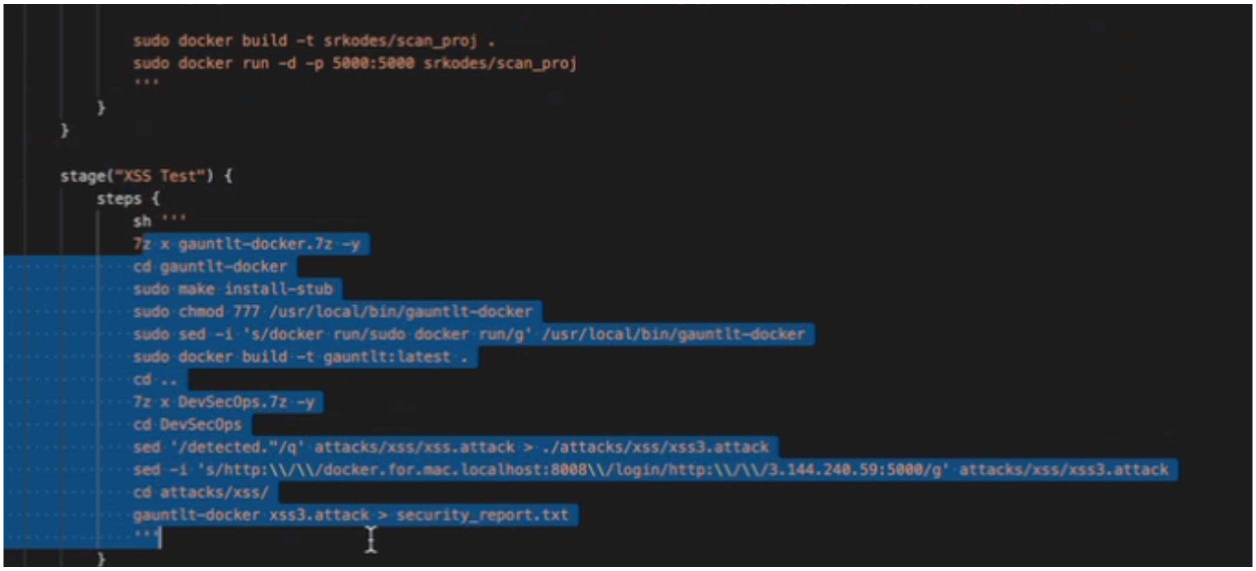
Errors

Was getting this error involving certificate not trusted



```
TASK [updating the ec2 instance] ***************************************************************************************************************
fatal: [50.16.94.96]: FAILED! => {"changed": true, "cmd": "sudo apt-get update", "delta": "0:00:01.598344", "end": "2021-12-02 06:54:28.446719",
 "msg": "non-zero return code", "rc": 100, "start": "2021-12-02 06:54:26.848375", "stderr": "E: The repository 'http://pkg.jenkins.io/debian-sta
ble binary/ Release' does not have a Release file.", "stderr_lines": ["E: The repository 'http://pkg.jenkins.io/debian-stable binary/ Release' d
oes not have a Release file."], "stdout": "Hit:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal InRelease\nHit:3 http://us-east-1.ec2.arch
ive.ubuntu.com/ubuntu focal-updates InRelease\nHit:4 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-backports InRelease\nHit:5 http://secu
rity.ubuntu.com/ubuntu focal-security InRelease\nIgn:1 https://pkg.jenkins.io/debian-stable binary/ InRelease\nErr:6 https://pkg.jenkins.io/debi
an-stable binary/ Release\n  Certificate verification failed: The certificate is NOT trusted. The certificate chain uses expired certificate.  C
ould not handshake: Error in the certificate verification. [IP: 146.75.30.133 443]\nReading package lists...", "stdout_lines": ["Hit:2 http://us
-east-1.ec2.archive.ubuntu.com/ubuntu focal InRelease", "Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-updates InRelease", "Hit:4 h
ttp://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-backports InRelease", "Hit:5 http://security.ubuntu.com/ubuntu focal-security InRelease", "I
gn:1 https://pkg.jenkins.io/debian-stable binary/ InRelease", "Err:6 https://pkg.jenkins.io/debian-stable binary/ Release", "  Certificate verif
ication failed: The certificate is NOT trusted. The certificate chain uses expired certificate.  Could not handshake: Error in the certificate v
erification. [IP: 146.75.30.133 443]", "Reading package lists..."]}

PLAY RECAP *************************************************************************************************************************************
```

Ansible-vault encrypt cloudlogs.csv



```
            sudo docker build -t srkodes/scan_proj .
            sudo docker run -d -p 5000:5000 srkodes/scan_proj
            '''
        }
    }

    stage("XSS Test") {
        steps {
            sh '''
            7z x gauntlt-docker.7z -y
            cd gauntlt-docker
            sudo make install-stub
            sudo chmod 777 /usr/local/bin/gauntlt-docker
            sudo sed -i 's/docker run/sudo docker run/g' /usr/local/bin/gauntlt-docker
            sudo docker build -t gauntlt:latest .
            cd ..
            7z x DevSecOps.7z -y
            cd DevSecOps
            sed '/detected."/q' attacks/xss/xss.attack > ./attacks/xss/xss3.attack
            sed -i 's/http:\\/\\/docker.for.mac.localhost:8008\\/login/http:\\/\\/3.144.240.59:5000/g' attacks/xss/xss3.attack
            cd attacks/xss/
            gauntlt-docker xss3.attack > security_report.txt
            '''
        }
```

Run the container

```
docker run
```

After you start the container, you have to bride it so it can talk together.

```
docker network create linkage
docker run -d -p 3000:3000 --name frontend --rm --net linkage rixardo/deploy08-frontend
docker run -d -p 5000:5000 --name backend --rm --net linkage rixardo/deploy08-backend
```

```
docker network connect linkage frontend
docker network connect linkage backend
```

```
docker stop frontend
docker stop backend
```

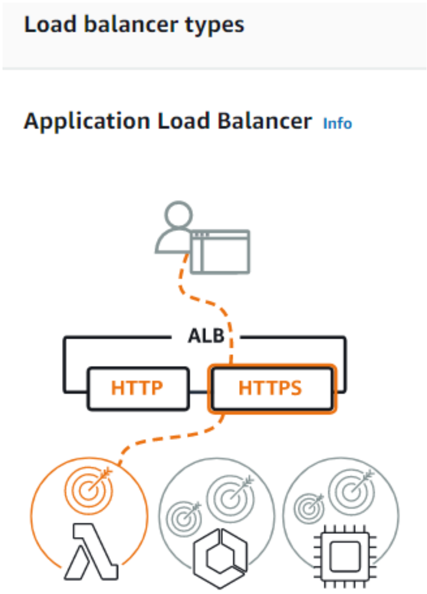
```
docker start frontend
docker start backend
```

```
docker rm $(docker ps -a -f status=exited -f status=created -q)
```

To deploy our application we will need to create an Application Load Balancer. Navigate to EC2 and select Load balancers

Create a load balancer

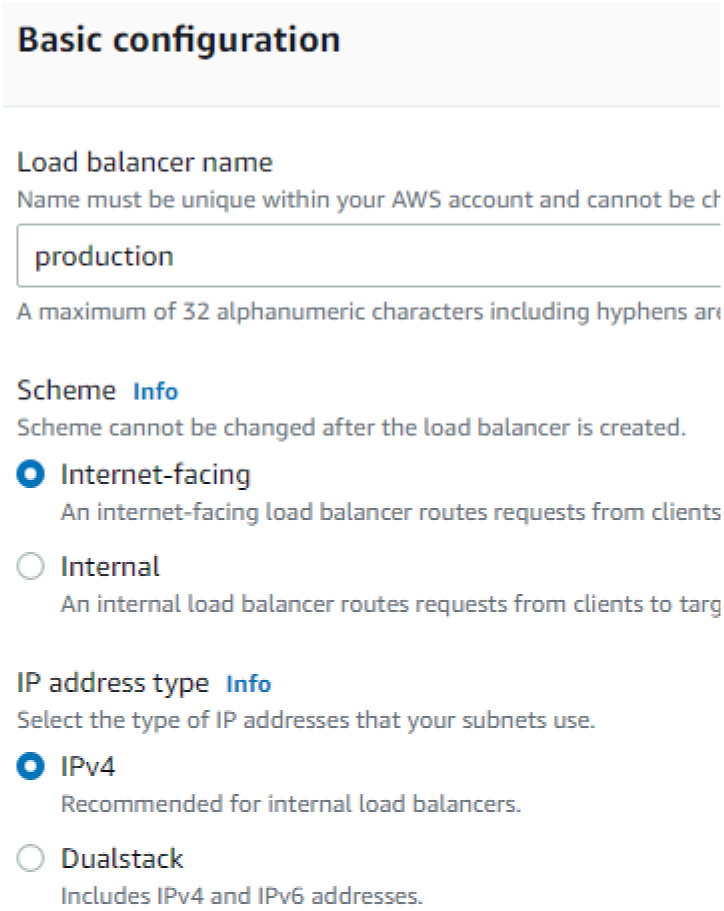
We will be creating an application load balancer



Load balancer types

Application Load Balancer Info

For the basic configuration



Basic configuration

Load balancer name

Name must be unique within your AWS account and cannot be ch

production

A maximum of 32 alphanumeric characters including hyphens are

Scheme Info

Scheme cannot be changed after the load balancer is created.

● Internet-facing

An internet-facing load balancer routes requests from clients

○ Internal

An internal load balancer routes requests from clients to targ

IP address type Info

Select the type of IP addresses that your subnets use.

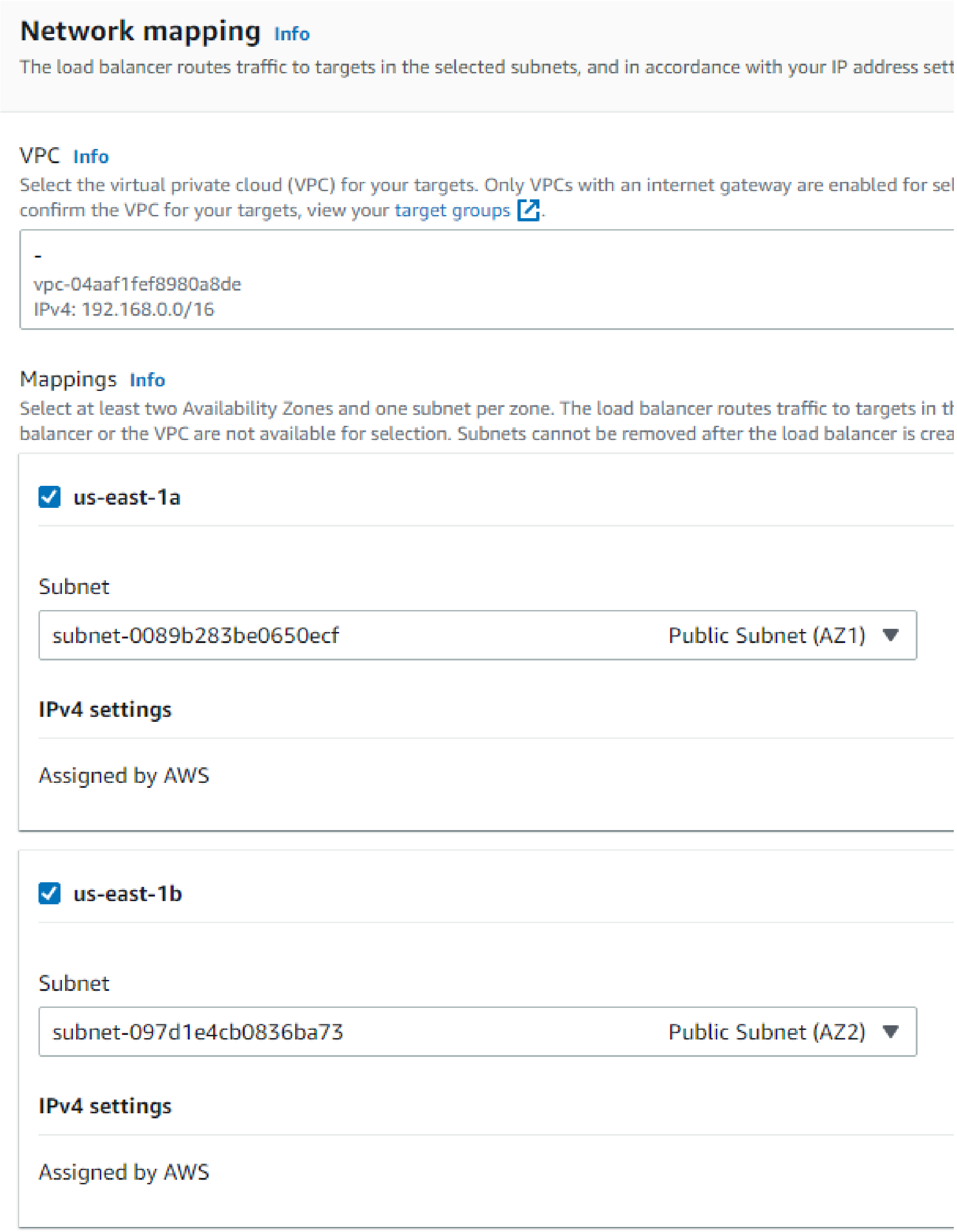
● IPv4

Recommended for internal load balancers.

○ Dualstack

Includes IPv4 and IPv6 addresses.

For the network mapping, we will need to select the VPC that cloudformation created and two two availability zones for resilience. It is important that we select the public subnets



Network mapping Info

The load balancer routes traffic to targets in the selected subnets, and in accordance with your IP address sett

VPC Info

Select the virtual private cloud (VPC) for your targets. Only VPCs with an internet gateway are enabled for sel confirm the VPC for your targets, view your target groups.

-
vpc-04aaf1fef8980a8de
IPv4: 192.168.0.0/16

Mappings Info

Select at least two Availability Zones and one subnet per zone. The load balancer routes traffic to targets in tl balancer or the VPC are not available for selection. Subnets cannot be removed after the load balancer is crea

☑ us-east-1a

Subnet

subnet-0089b283be0650ecf Public Subnet (AZ1)

IPv4 settings

Assigned by AWS

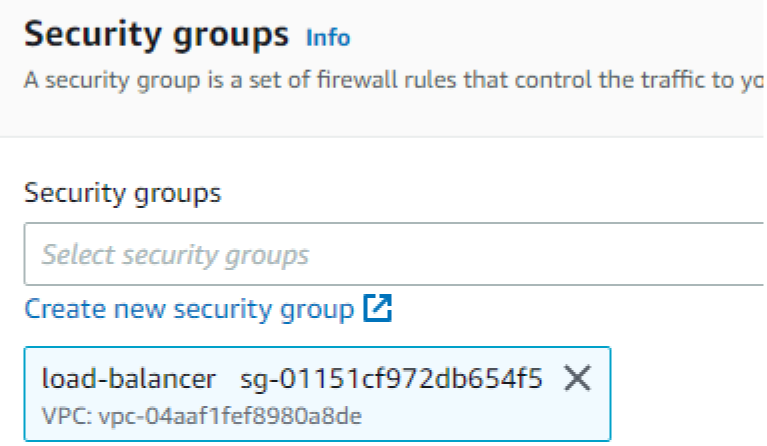
☑ us-east-1b

Subnet

subnet-097d1e4cb0836ba73 Public Subnet (AZ2)

IPv4 settings

Assigned by AWS



Security groups Info

A security group is a set of firewall rules that control the traffic to yo

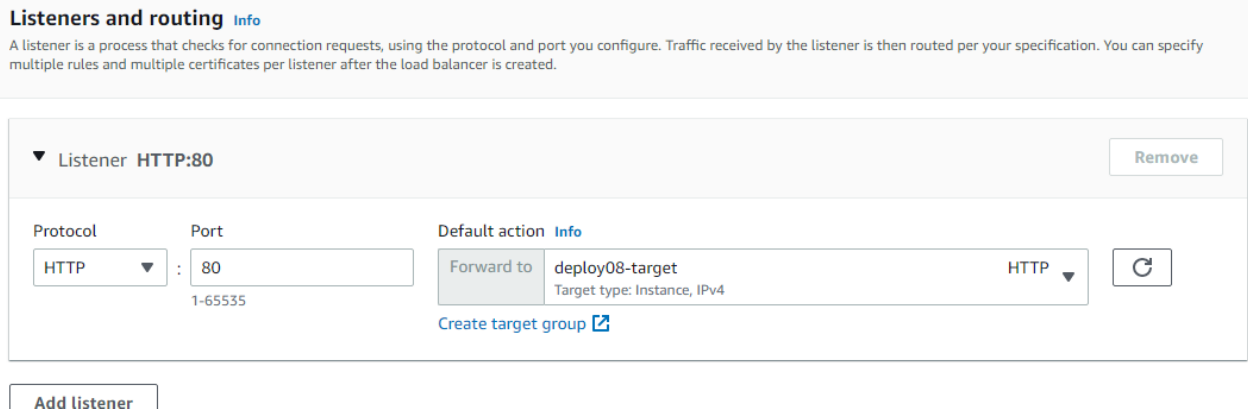
Security groups

Select security groups

Create new security group

load-balancer sg-01151cf972db654f5

VPC: vpc-04aaf1fef8980a8de



Listeners and routing Info

A listener is a process that checks for connection requests, using the protocol and port you configure. Traffic received by the listener is then routed per your specification. You can specify multiple rules and multiple certificates per listener after the load balancer is created.

Listener **HTTP:80**

Protocol: HTTP

Port: 80

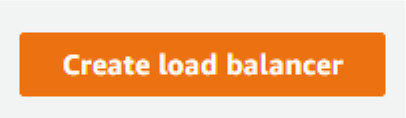
1-65535

Default action Info

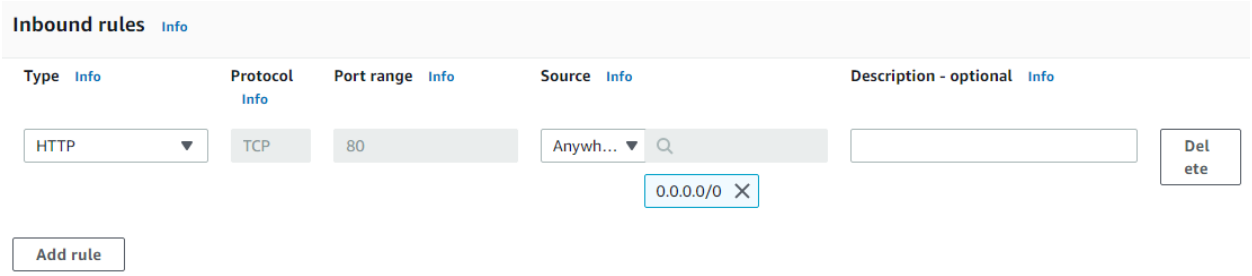
Forward to deploy08-target HTTP

Target type: Instance, IPv4

Create target group



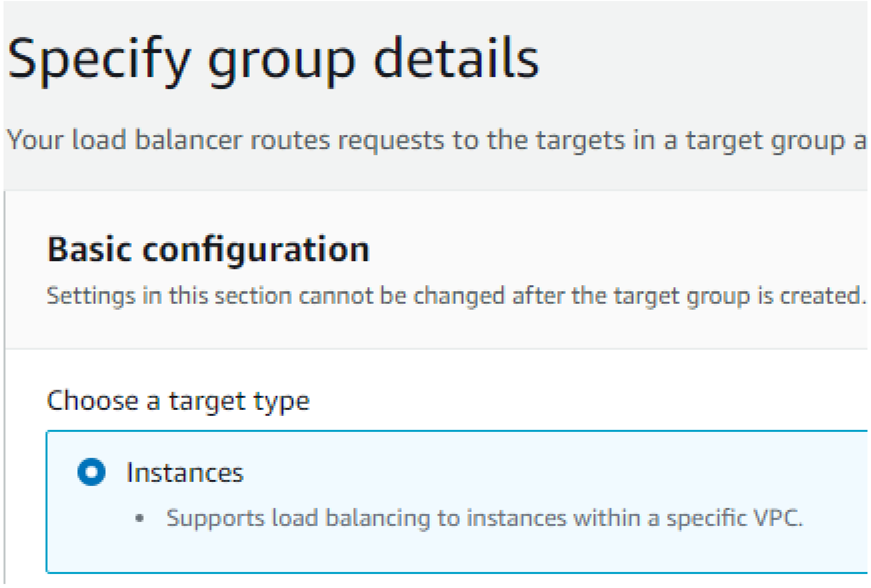
Load balancer sg



Inbound rules

Type	Protocol	Port range	Source	Description - optional
HTTP	TCP	80	Anywh… 0.0.0.0/0	

Target group selects the EC2 to route traffic



Specify group details

Your load balancer routes requests to the targets in a target group a

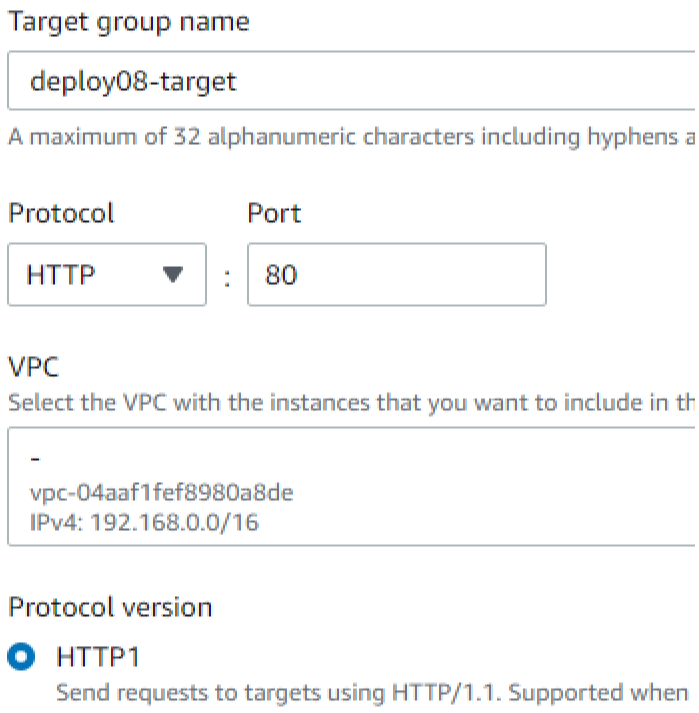
Basic configuration

Settings in this section cannot be changed after the target group is created.

Choose a target type

● Instances

- Supports load balancing to instances within a specific VPC.



Target group name

deploy08-target

A maximum of 32 alphanumeric characters including hyphens a

Protocol : Port

HTTP ▼ : 80

VPC

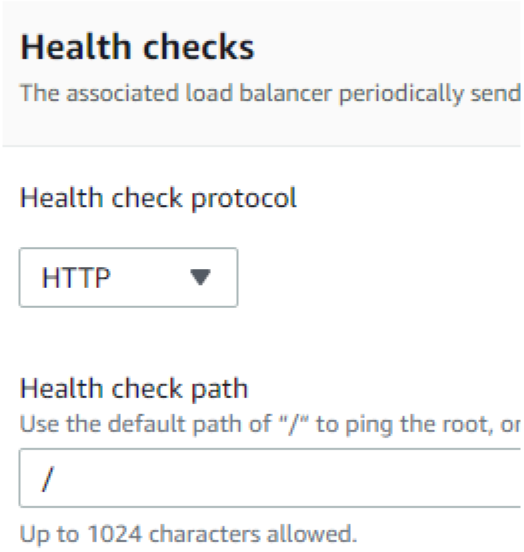
Select the VPC with the instances that you want to include in th

-
vpc-04aaf1fef8980a8de
IPv4: 192.168.0.0/16

Protocol version

● HTTP1

Send requests to targets using HTTP/1.1. Supported when



Health checks

The associated load balancer periodically send

Health check protocol

HTTP ▼

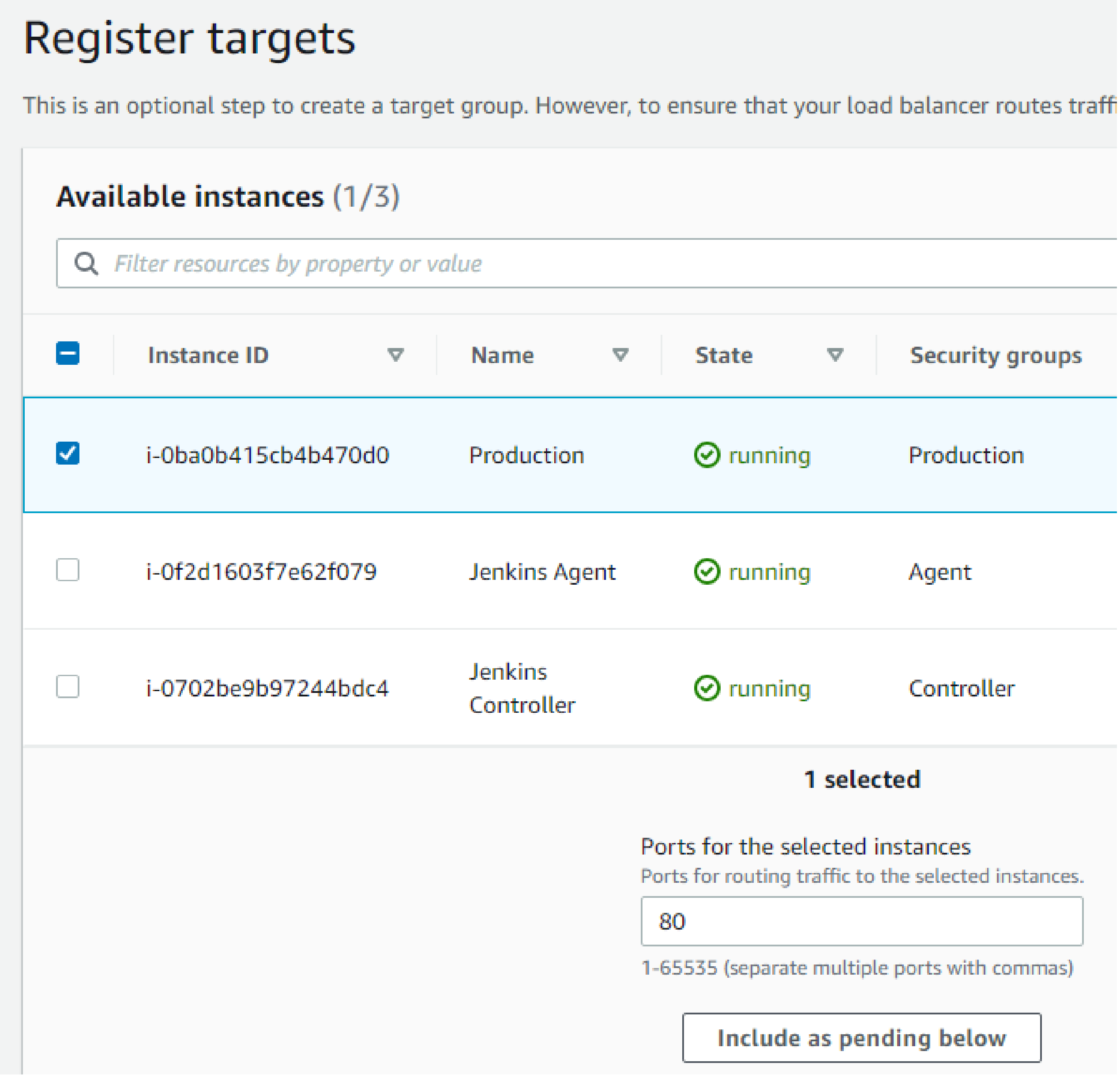
Health check path

Use the default path of "/" to ping the root, or

/

Up to 1024 characters allowed.

Register targets



Register targets

This is an optional step to create a target group. However, to ensure that your load balancer routes traffi

Available instances (1/3)

Filter resources by property or value

☐	Instance ID	Name	State	Security groups
☑	i-0ba0b415cb4b470d0	Production	✓ running	Production
☐	i-0f2d1603f7e62f079	Jenkins Agent	✓ running	Agent
☐	i-0702be9b97244bdc4	Jenkins Controller	✓ running	Controller

1 selected

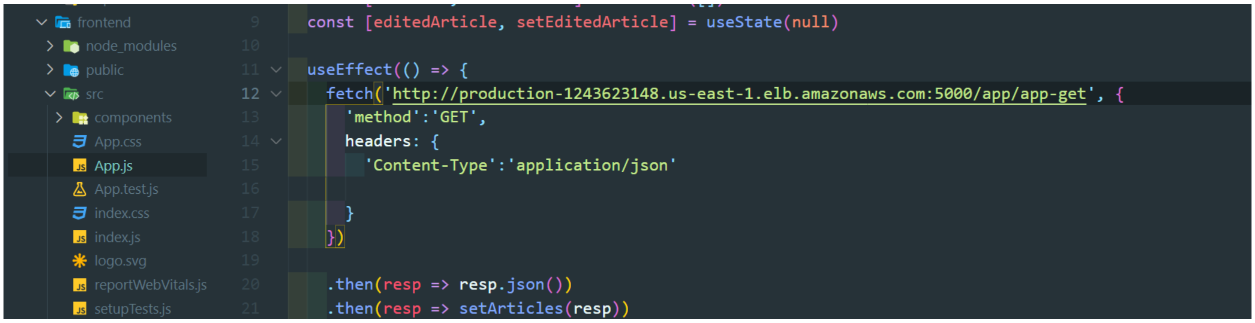
Ports for the selected instances
Ports for routing traffic to the selected instances.

80

1-65535 (separate multiple ports with commas)

Docker versions of kuberentes using swarm

To deploy our application, we are going to be creating a targetgroup/



```
const [editedArticle, setEditedArticle] = useState(null)

useEffect(() => {
  fetch('http://production-1243623148.us-east-1.elb.amazonaws.com:5000/app/app-get', {
    'method':'GET',
    headers: {
      'Content-Type':'application/json'

    }
  })

  .then(resp => resp.json())
  .then(resp => setArticles(resp))
```

To deploy our application we are going to use a docker stack. This feature is commonly used with docker swarm which is, a built in docker native container orchestration tool Similar to that of kuberentes. The benefits of this is that, if we wanted to scale our servers, all we would have to do is do a docker swarm join command, and connect our production servers together. We can also scale up and down both our containers and servers as we desire.

We created a load balancer which port forwards user traffic to our target groups. The target group then directs traffic to the specific server at a specific port.

For production ec2 allow http request

ERROR
CPU UTILIZATION KEPT HITTING 100% When i did docker build with the front end application
https://docs.docker.com/engine/reference/commandline/build/#options