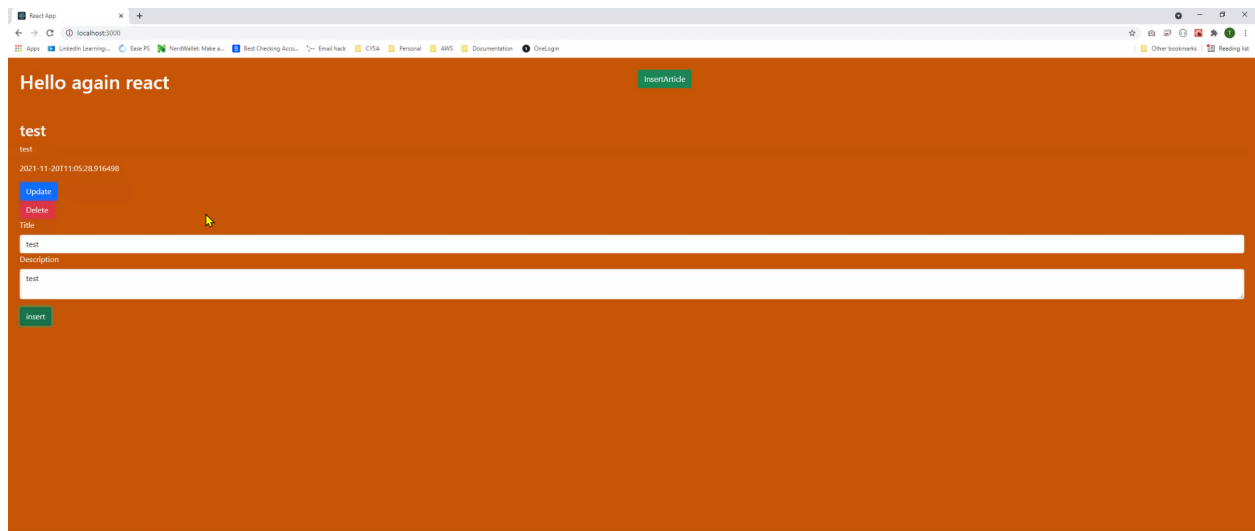




Deployment 08 - CICD

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.



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...

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-0b67e4d020c0d6d7
  InstanceType: t2.micro
  SubnetId: !Ref PublicSubnet1
  KeyName: !Ref KeyName
  SecurityGroups:
    - !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-0b67e4d020c0d6d7
  InstanceType: t2.micro
  SubnetId: !Ref PublicSubnet1
  KeyName: !Ref KeyName
  SecurityGroups:
    - !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-0b67e4d020c0d6d7
  InstanceType: t2.micro
  SubnetId: !Ref PublicSubnet1
  KeyName: !Ref KeyName
  SecurityGroups:
    - !Ref ProductionSecurityGroup
  Tags:
    - Key: "Name"
      Value: "Production"

DeployRDSSecurityGroup:
Type: AWS::EC2::SecurityGroup
Properties:
  GroupDescription: "Security group for the RDS MySQL database that allows access from ProductionAgent SG only"
  GroupName: "DeployRDS"
  SecurityGroupIngress:
    - IpProtocol: tcp
      FromPort: 0
      ToPort: 0000
      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 Public Address

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

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

```

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



Create a stack

Create a CloudFormation stack

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

Create stack

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 JSON or YAML file that describes your stack's resources and properties.

☒ Template is ready

☐ Upload a template file

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 include

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

KeyName
Name of an existing EC2 key pair

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


Next



Next

Create stack



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...

CloudFormation > Stacks > deploy08

Stacks (1) 

Active   View nested

< 1 >

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.

CloudFormation > Stacks > deploy08

Stacks (1)

Active ☒ View nested

< 1 >

deploy08

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

CREATE_COMPLETE

delete Update Stack actions Create stack

Stack info Events Resources **Outputs** Parameters

Change sets

Events (89)

Outputs (15)

Key	Value	Description
Deploy08DBSecurityGroup	sg-0b4668a6c9ffb1c6f	Security group for the RDS MySQL database that allows access from Production and Agent SG
JenkinsAgentEC2Instance	54.172.243.1	JenkinsAgentEC2Instance's PublicIP Address
JenkinsAgentSecurityGroup	sg-08dd289183857236f	Security group with SSH from anywhere ingress rule
JenkinsControllerEC2Instance	34.224.40.163	JenkinsController's PublicIP Address
JenkinsControllerSecurityGroup	sg-0012a92b9d79cd31a	Security group with SSH from anywhere ingress rule
NoIngressSecurityGroup	sg-0601c284a44fdc096	Security group with no ingress rule
PrivateSubnet1	subnet-04aea0f971030dec7	A reference to the private subnet in the 1st Availability Zone
PrivateSubnet2	subnet-06b753f319ff1f3b7	A reference to the private subnet in the 2nd Availability Zone
PrivateSubnets	subnet-04aea0f971030dec7,subnet-06b753f319ff1f3b7	A list of the private subnets
ProductionEC2Instance	54.210.242.75	ProductionEC2Instance's PublicIP Address

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.

[Controller]

54.173.152.217 ansible_user=ubuntu ansible_ssh_private_key_file=~/.ssh/rixardo.pem

[Agent]

54.87.130.44 ansible_user=ubuntu ansible_ssh_private_key_file=~/.ssh/rixardo.pem

[Production]

54.208.27.54 ansible_user=ubuntu ansible_ssh_private_key_file=~/.ssh/rixardo.pem

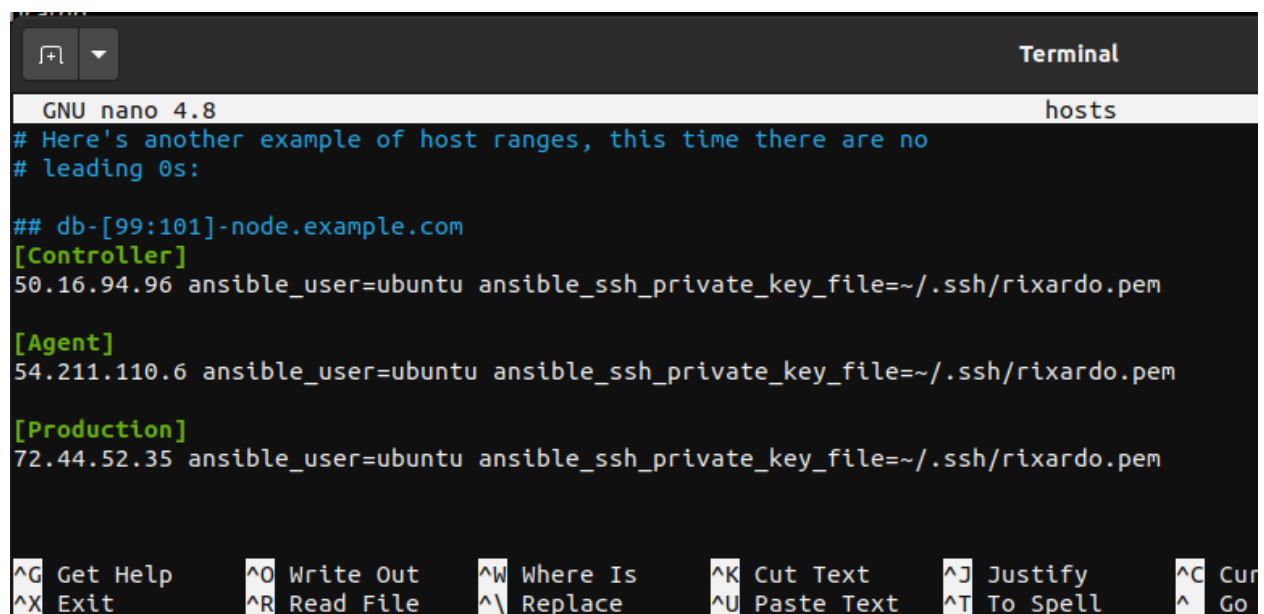
Once we set up all the resources and obtain the necessary data, we can proceed to setting up Ansible so we can configure the EC2s. Change directory to your SSH folder and make sure your Key that you use to SSH into EC2 instances is there.

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 hosts
# Here's another example of host ranges, this time there are no
# leading 0s:

## db-[99:101]-node.example.com
[Controller]
50.16.94.96 ansible_user=ubuntu ansible_ssh_private_key_file=~/.ssh/rixardo.pem

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

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

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


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

```

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 modules
  shell: sudo apt-get install \ ca-certificates \ curl \ gnupg \ lsb-release -y

- name: adding docker keys
  shell: curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

- 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 docker-ce-cli containerd.io -y

- 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

While testing the application locally, I decided use RDS

Go to AWS RDS service

RDS

Managed Relational Database Service

Create a database

Create 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 compatibility issues with older database versions.

Version

MySQL 8.0.23

For the database instance size,

DB instance size



Production

db.r6g.xlarge
4 vCPUs
32 GiB RAM
500 GiB
1.017 USD/hour



Dev/Test

db.r6g.large
2 vCPUs
16 GiB RAM
100 GiB
0.231 USD/hour



Free tier

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-sensitive and can contain only alphanumeric characters or hyphens. First character cannot be a hyphen.

Take note of the username that you create

Master username [Info](#)

Type a login ID for the master user.

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) [Info](#)

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 [Info](#)

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

Create new DB Subnet Group

Public access [Info](#)

☒ Yes

Amazon EC2 instances and devices outside the VPC can connect to your database. Choose one or more subnets to 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 inside the VPC can connect to your database.

For VPC security groups, we can select an existing security group that was created during CloudFormation

VPC security group

Choose a VPC security group to allow access to your database. Ensure that the security group rules allow the appropriate 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-db



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 and Monitoring

Backup

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

Monitoring

- ☐ Enable Enhanced monitoring
Enabling Enhanced monitoring metrics are useful when

Disable maintenance

Maintenance

Auto minor version upgrade [Info](#)

- ☐ Enable auto minor version upgrade
Enabling auto minor version upgrade will allow you to upgrade your database when they are released. The automatic upgrades database.

Maintenance window [Info](#)

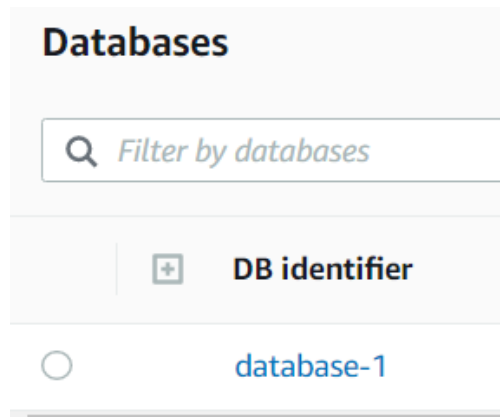
Select the period you want pending modification

- ☐ Select window
- ☒ No preference

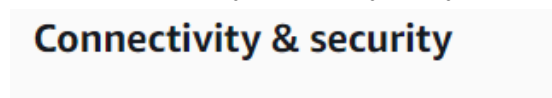
Once all that is configured, we can proceed and create the database

Create database

Once the database has been created, click on the database



Under connectivity & security, copy the Endpoint



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 add our connection information.

```
10 app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql://DBusername:DBpassword@DBEndpoint:3306/DatabaseName'
```

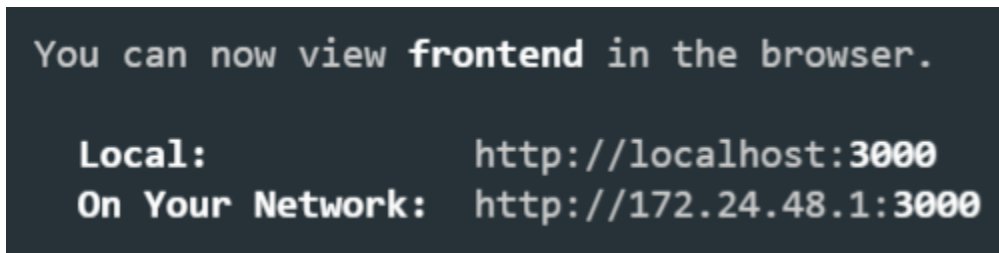
```
12 app.config[
13     "SQLALCHEMY_DATABASE_URI"
14 ] = "mysql://admin:abc123abc@database-1.cet4jo0trfys.us-east-1.rds.amazonaws.com:3306/deploy08"
15
16 app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False
```

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

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

We can then test the application locally.. For the front end application, run the following command

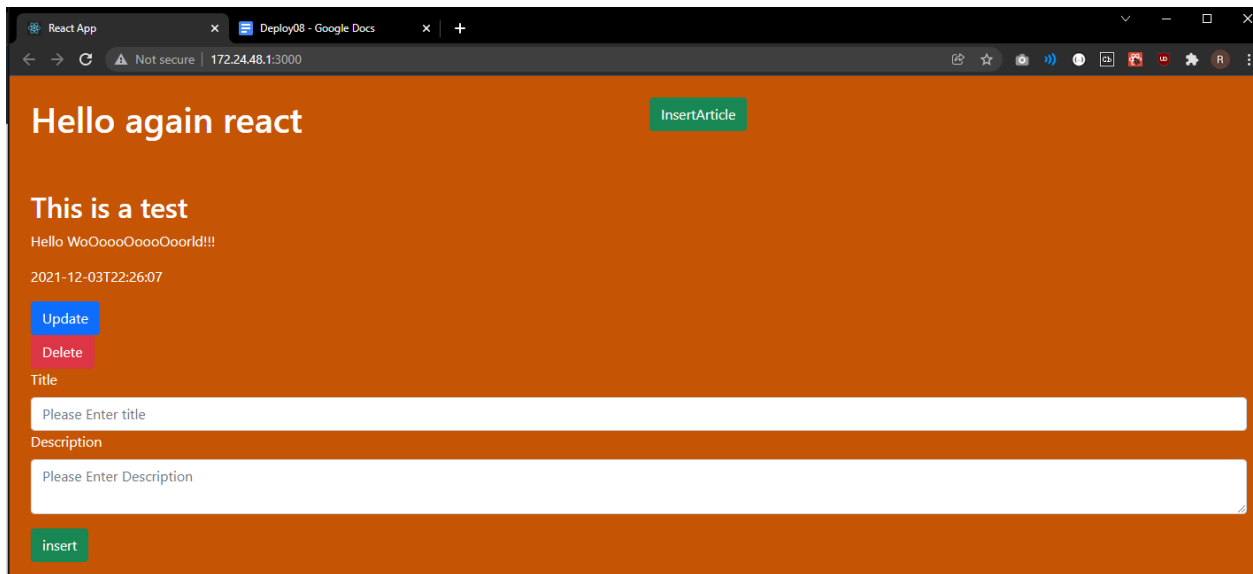
```
npm install
npm run start
```



Open another terminal and for the backend application use the following commands

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

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



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/>

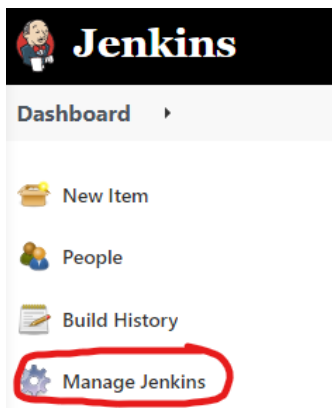
Enter the password that was outputted in the ansible playbook

```
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 and create a user

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.

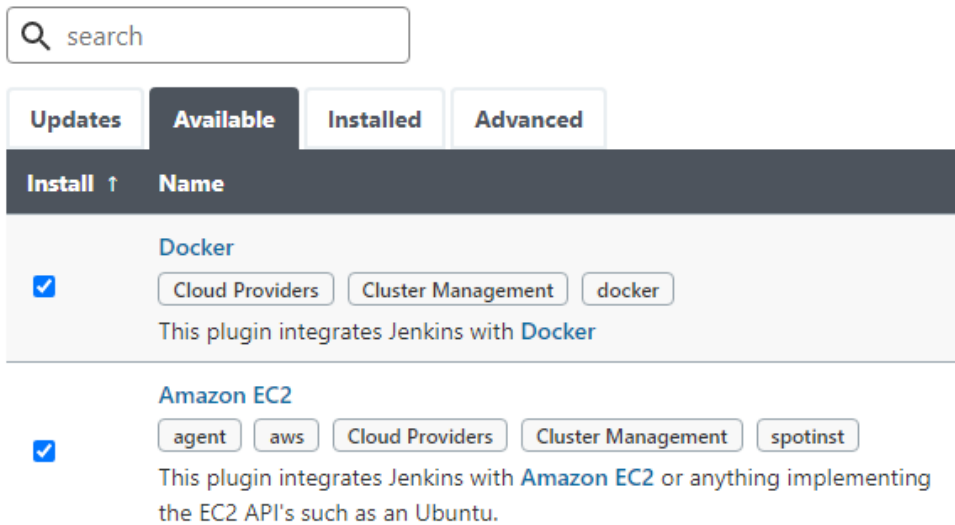


Manage Plugins
Add, remove, disable or enable plugins that can extend the functionality of Jenkins.



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



Download the following plugins

Download now and install after restart

Select Restart when installation is complete

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

Once Jenkins has restarted, sign back in. We can now configure the Agent/Node inside of Jenkins. 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.

Select New Node in the left side



New Node

Name the agent, “Node”

Node name

☒ **Permanent Agent**

Adds a plain, permanent agent, such as dynamic provisioned computer, virtual machines

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

Name

Description

Number of executors

Remote root directory

Labels

We will then need to set up how we will access our agent which is hosted on an EC2. When we created our EC2 we set up SSH access from anywhere in the security group.

For the Launch method, select via SSH. The Host IP is the public IPv4 of the EC2 we created.

Usage

Use this node as much as possible

Launch method

Launch agents via SSH

Host

54.87.130.44

We need to add credentials to access the EC2 instance.

Credentials

- current -

Add



The selected



Jenkins

nc

For the settings, change the Kind to SSH username with a private key. The username is important, It will be the same as our AMI.

Add Credentials

Domain

Global credentials (unrestricted)

Kind

SSH Username with private key

Scope

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

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

Credentials

ubuntu (Controller SSH into Agent) ▼

 Add ▼

ubuntu (Controller SSH into Agent)

Host 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.



We will then need to create another node for the production



New Node

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 ECS that gets instructions from Controller

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

Labels

production

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




Launch method

Launch agents via SSH

Host

54.208.27.54

You should see your agent 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	

Refresh status

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](#)

New Access Token

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 can rotate 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.

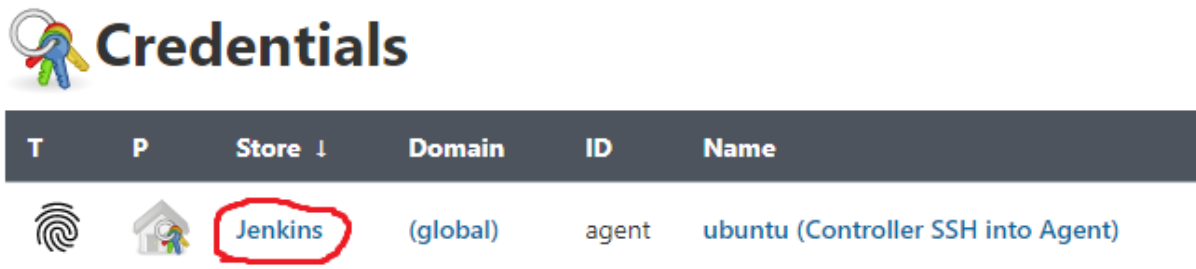
Generate your token and make SURE to copy the personal access token. Navigate back to the Jenkins Controller and go to the dashboard. Select Manage Jenkins. We then need to select Manage Credentials under security.



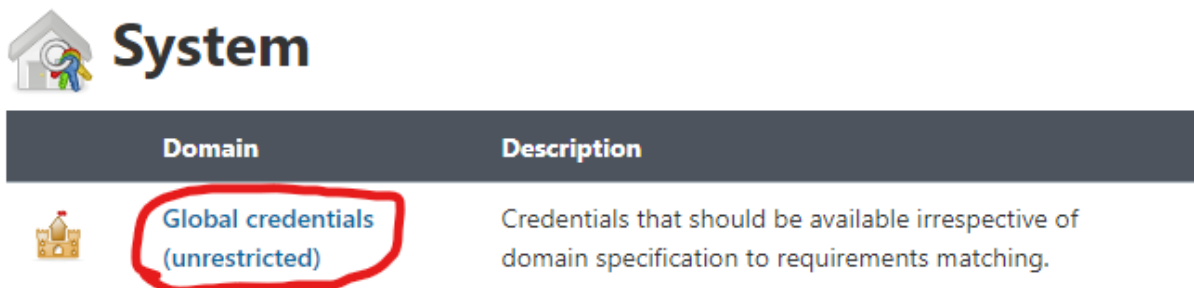
Manage Credentials

Configure credentials

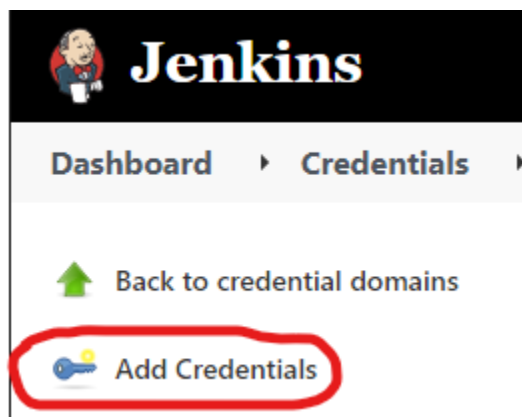
Select the following credentials



Once inside that credentials, we need to select Global Credentials



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

.....

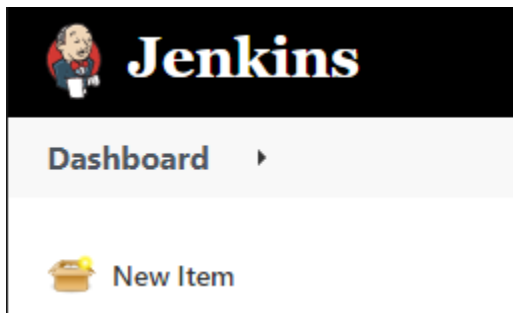
ID

rixardo-dockerhub

Description

OK

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

Deployment 08

» 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...

General Branch Sources Build Configuration Scan Multibranch Pipeline Triggers

Orphaned Item Strategy Health metrics Properties Pipeline Libraries

Display Name ?

deploy08

Description

Multibranch pipeline script for deployment08

[Plain text] [Preview](#)

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

Branch Sources

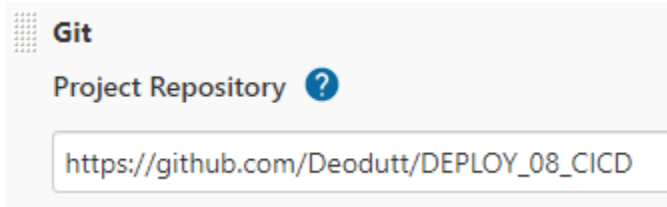
Add source ▲

Git

GitHub

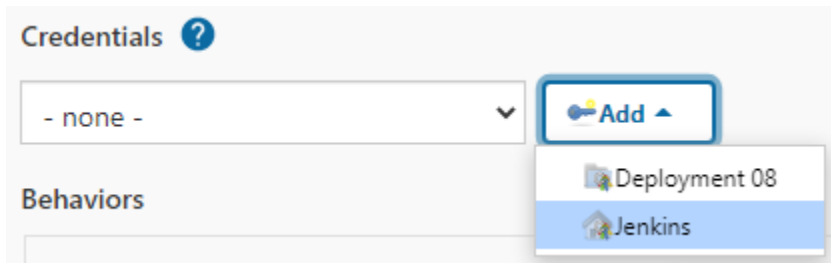
Single repository & branch

Under Project Repository, link the forked repository of the assignment



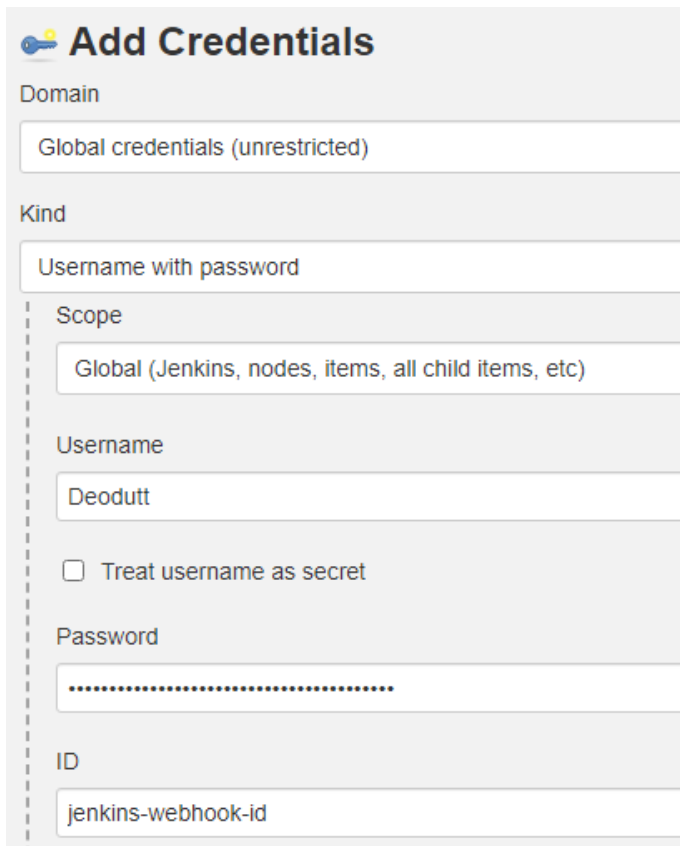
The screenshot shows the 'Git' section of the Jenkins configuration. Under 'Project Repository', there is a text input field containing the URL 'https://github.com/Deodutt/DEPLOY_08_CICD'. A blue question mark icon is visible next to the 'Project Repository' label.

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



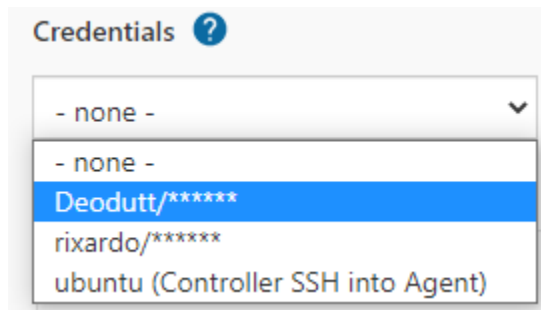
The screenshot shows the 'Credentials' section of the Jenkins configuration. A dropdown menu is open, displaying two options: 'Deployment 08' and 'Jenkins'. The 'Jenkins' option is highlighted in blue. An 'Add' button is visible next to the dropdown.

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

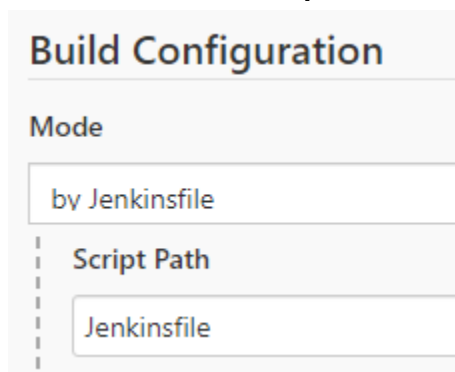


The screenshot shows the 'Add Credentials' form in Jenkins. The 'Domain' is set to 'Global credentials (unrestricted)'. The 'Kind' is set to 'Username with password'. The 'Scope' is set to 'Global (Jenkins, nodes, items, all child items, etc)'. The 'Username' is 'Deodutt'. The 'Password' field is masked with dots. The 'ID' is 'jenkins-webhook-id'. There is an unchecked checkbox for 'Treat username as secret'.

After the credentials are credited, select the credentials.



When configuring Build Configuration, it is important to have the Jenkinsfile in your GitHub in the correct path



So I have completed...

creating resources using CloudFormation

Configuring each EC2 using ansible

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

Task 3

Task 4

Task 5

Errors

Was getting this error involving certificate not trusted

```
TASK [updating the ec2 instance] *****
fatal: [50.10.94.96]: 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-stable binary/ Release' does not have a Release file.", "stdout_lines": ["E: The repository 'http://pkg.jenkins.io/debian-stable binary/ Release' does 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.archive.ubuntu.com/ubuntu focal-updates InRelease\nHit:4 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-backports InRelease\nHit:5 http://security.ubuntu.com/ubuntu focal-security InRelease\nIgn:1 https://pkg.jenkins.io/debian-stable binary/ InRelease\nErr:6 https://pkg.jenkins.io/debian-stable binary/ Release\n  Certificate verification failed: The certificate is NOT trusted. The certificate chain uses expired certificate. Could 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 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-backports InRelease", "Hit:5 http://security.ubuntu.com/ubuntu focal-security InRelease", "Ign:1 https://pkg.jenkins.io/debian-stable binary/ InRelease", "Err:6 https://pkg.jenkins.io/debian-stable binary/ Release", "Certificate verification failed: The certificate is NOT trusted. The certificate chain uses expired certificate. Could not handshake: Error in the certificate verification. [IP: 146.75.30.133 443]", "Reading package lists..."]}
PLAY RECAP *****
```

Ansible-vault encrypt cloudlogs.csv