**Deploy MERN App to AWS using AWS Cloudformation, Jenkins, and Ansible**

In this article, I show how to deploy a **MERN**(MongoDB, Express.js, React, and Node.js) application to AWS using AWS Cloudformation, Jenkins, and Ansible.

*This article is part of a series of articles on DevOps. If you missed the previous one, check it*[*here*](https://medium.com/aws-in-plain-english/deploy-mern-app-to-aws-using-cloudformation-and-ansible-5caeac42d502)*. This is the second article in the series.*

**Prerequisites**

To better follow along with this article, you would need the following:

* AWS CLI configured on your local machine
* An [AWS](https://aws.amazon.com/console/) account
* The previous article found [here](https://aws.plainenglish.io/deploy-mern-app-to-aws-using-cloudformation-and-ansible-5caeac42d502)

**Getting Started**

Building from the previous article, we would carry out the following steps:

1. Provision required resources using [AWS CloudFormation](https://aws.amazon.com/cloudformation/).
2. Create the helper Setup file
3. Create the Ansible Playbook
4. Create Jenkins file
5. Create .env file on Jenkins Server
6. Jenkins Setup
7. Install Necessary Plugins
8. Create Jenkins Pipeline
9. Add Webhook to GitHub
10. Test Pipeline for both branches

**Clone Repo**

Before proceeding further, we would need to get the source code we would work with. Proceed to fork an already existing application found [here](https://github.com/chyke007/Yum-food). It has a backend API built using NodeJs and a frontend built using ReactJs(This is the same app deployed in the previous article)

Head over to GitHub and click the **Fork** button to create a copy of this repo into your GitHub account. Then create a clone of your fork on your local machine.

**Step 1: Provisioning Resources using AWS CloudFormation**

This article introduces [Jenkins](https://www.jenkins.io/), an open-source automation server to reliably build, test, and deploy software, into the mix.

**We provision 3 EC2 Servers**:

1. 1 master server where Jenkins and Ansible would be installed
2. 2 Slaves servers(Production and Development) where application code would be deployed

Next, create a main.ymlfor CloudFormation template:

mkdir yum-food && cd yum-food && nano main.yml

Add the following content to the main.yml file:

AWSTemplateFormatVersion: '2010-09-09'  
Parameters:  
 KeyPairName:  
 Type: String  
 Description: Name of the Key Pair to create  
 Default: MyKeyPair  
Resources:  
 MyKeyPair:  
 Type: AWS::EC2::KeyPair  
 Properties:  
 KeyName: !Ref KeyPairName  
  
 EC2InstanceRole:  
 Type: AWS::IAM::Role  
 Properties:  
 RoleName: EC2-SSM-FullAccess-Role  
 AssumeRolePolicyDocument:  
 Version: '2012-10-17'  
 Statement:  
 - Effect: Allow  
 Principal:  
 Service: ec2.amazonaws.com  
 Action: sts:AssumeRole  
 Policies:  
 - PolicyName: EC2FullAccessPolicy  
 PolicyDocument:  
 Version: '2012-10-17'  
 Statement:  
 - Effect: Allow  
 Action: ec2:\*  
 Resource: '\*'  
 - PolicyName: SSMFullAccessPolicy  
 PolicyDocument:  
 Version: '2012-10-17'  
 Statement:  
 - Effect: Allow  
 Action: ssm:\*  
 Resource: '\*'  
  
 EC2InstanceProfile:  
 Type: AWS::IAM::InstanceProfile  
 Properties:  
 InstanceProfileName: EC2-SSM-FullAccess-InstanceProfile  
 Roles:  
 - !Ref EC2InstanceRole  
  
 VPC:  
 Type: AWS::EC2::VPC  
 Properties:  
 CidrBlock: 10.0.0.0/16  
 EnableDnsSupport: true  
 EnableDnsHostnames: true  
  
 InternetGateway:  
 Type: AWS::EC2::InternetGateway  
  
 InternetGatewayAttachment:  
 Type: AWS::EC2::VPCGatewayAttachment  
 Properties:  
 InternetGatewayId: !Ref InternetGateway  
 VpcId: !Ref VPC  
  
 PublicSubnet:  
 Type: AWS::EC2::Subnet  
 Properties:  
 VpcId: !Ref VPC  
 AvailabilityZone: !Select [ 0, !GetAZs '' ]  
 CidrBlock: 10.0.1.0/24  
 MapPublicIpOnLaunch: true  
  
 PublicRouteTable:  
 Type: AWS::EC2::RouteTable  
 Properties:  
 VpcId: !Ref VPC  
  
 DefaultPublicRoute:  
 Type: AWS::EC2::Route  
 DependsOn: InternetGatewayAttachment  
 Properties:  
 RouteTableId: !Ref PublicRouteTable  
 DestinationCidrBlock: 0.0.0.0/0  
 GatewayId: !Ref InternetGateway  
  
 PublicSubnetRouteTableAssociation:  
 Type: AWS::EC2::SubnetRouteTableAssociation  
 Properties:  
 RouteTableId: !Ref PublicRouteTable  
 SubnetId: !Ref PublicSubnet  
  
 MasterSecurityGroup:  
 Type: AWS::EC2::SecurityGroup  
 Properties:  
 GroupDescription: Enable SSH access  
 VpcId: !Ref VPC   
 SecurityGroupIngress:  
 - CidrIp: 0.0.0.0/0  
 FromPort: 22  
 ToPort: 22  
 IpProtocol: tcp  
 - CidrIp: 0.0.0.0/0  
 FromPort: 8080  
 ToPort: 8080  
 IpProtocol: tcp  
  
 SlaveSecurityGroup:  
 Type: AWS::EC2::SecurityGroup  
 Properties:  
 GroupDescription: Enable SSH access  
 VpcId: !Ref VPC   
 SecurityGroupIngress:  
 - CidrIp: 0.0.0.0/0  
 FromPort: 22  
 ToPort: 22  
 IpProtocol: tcp  
 - CidrIp: 0.0.0.0/0  
 FromPort: 4000  
 ToPort: 4000  
 IpProtocol: tcp  
  
 MasterEC2Instance:  
 Type: AWS::EC2::Instance  
 Properties:  
 InstanceType: t2.large  
 KeyName: !Ref MyKeyPair  
 Tags:  
 - Key: Name  
 Value: MasterHost  
 SubnetId: !Ref PublicSubnet  
 SecurityGroupIds:  
 - !GetAtt "MasterSecurityGroup.GroupId"  
 ImageId: ami-053b0d53c279acc90  
 IamInstanceProfile: !Ref EC2InstanceProfile  
 UserData:  
 Fn::Base64: !Sub |  
 #!/bin/bash  
 # Update package index and install required packages  
 sudo apt update -y  
 sudo apt install -y python3-pip  
  
 # Install Ansible via pip3  
 sudo pip3 install ansible  
  
 # Install Jenkins  
 sudo apt install openjdk-11-jdk -y  
  
 curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null  
  
 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  
  
 sudo apt update -y  
  
 sudo apt install jenkins -y  
  
 sudo chown -R jenkins:jenkins /home/ubuntu/  
   
 sudo usermod -aG sudo jenkins  
  
 sudo chmod -R 777 /home/ubuntu/  
  
 sudo systemctl start jenkins  
  
 # Install libssl1.1 for MongoMemoryServer package  
   
 ## To handle dependency installations  
 sudo apt-get install gdebi-core -y  
  
 ## Download libssl1.1  
 sudo wget http://ftp.us.debian.org/debian/pool/main/o/openssl/libssl1.1\_1.1.1n-0+deb11u4\_amd64.deb  
  
 ## Install libssl1.1  
 sudo gdebi --non-interactive libssl1.1\_1.1.1n-0+deb11u4\_amd64.deb  
  
 # Install boto via pip3  
 sudo pip3 install boto3  
  
 # Download and install AWS CLI v2 using curl, unzip, and sudo  
 sudo apt install unzip  
 curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"  
 unzip awscliv2.zip  
 sudo ./aws/install  
  
 # Create the /etc/ansible/ directory if it doesn't exist  
 sudo mkdir -p /etc/ansible  
 sudo mkdir -p /opt/ansible/inventory  
  
 # Create the /etc/ansible/ansible.cfg file with AWS dynamic inventory configuration  
 sudo cat > /etc/ansible/ansible.cfg << EOL  
 [inventory]  
 enable\_plugins = aws\_ec2  
 [defaults]  
 host\_key\_checking = False  
 inventory = /opt/ansible/inventory/aws\_ec2.yaml  
 remote\_user = ubuntu  
 private\_key\_file = /home/ubuntu/ssh\_key.pem  
 EOL  
  
 # Create the /opt/ansible/inventory/aws\_ec2.yaml file with AWS dynamic inventory configuration  
 sudo cat > /opt/ansible/inventory/aws\_ec2.yaml << EOL  
 ---  
 plugin: aws\_ec2  
  
 regions:  
 - us-east-1  
  
 keyed\_groups:   
 - key: tags  
 prefix: tag  
 EOL  
  
 SlaveDevEC2Instance:  
 Type: AWS::EC2::Instance  
 Properties:  
 InstanceType: t2.medium  
 KeyName: !Ref MyKeyPair  
 Tags:  
 - Key: Name  
 Value: SlaveDevelopmentHost  
 SubnetId: !Ref PublicSubnet  
 SecurityGroupIds:  
 - !GetAtt "SlaveSecurityGroup.GroupId"  
 ImageId: ami-053b0d53c279acc90  
 UserData: !Base64 |  
 #!/bin/bash  
 # Update package index and install required packages  
 sudo apt update -y  
  
 # Install git client on server  
 sudo apt-get install -y git  
  
 SlaveProdEC2Instance:  
 Type: AWS::EC2::Instance  
 Properties:  
 InstanceType: t2.medium  
 KeyName: !Ref MyKeyPair  
 Tags:  
 - Key: Name  
 Value: SlaveProductionHost  
 SubnetId: !Ref PublicSubnet  
 SecurityGroupIds:  
 - !GetAtt "SlaveSecurityGroup.GroupId"  
 ImageId: ami-053b0d53c279acc90  
 UserData: !Base64 |  
 #!/bin/bash  
 # Update package index and install required packages  
 sudo apt update -y  
  
 # Install git client on server  
 sudo apt-get install -y git  
  
Outputs:  
 MasterPublicIp:  
 Description: Master EC2 Instance Public Ip  
 Value: !GetAtt MasterEC2Instance.PublicIp  
  
 SlaveDevPublicIp:  
 Description: Slave Prodcution EC2 Instance Public Ip  
 Value: !GetAtt SlaveDevEC2Instance.PublicIp  
  
 SlaveProdPublicIp:  
 Description: Slave Development EC2 Instance Public Ip  
 Value: !GetAtt SlaveProdEC2Instance.PublicIp

Now run the deployment for the stack, and be patient a bit so the Jenkins server starts running.

aws cloudformation deploy --template-file main.yml --stack-name MyStack --capabilities CAPABILITY\_NAMED\_IAM

**Step 2: Create the helper setup file**

Create a new file called setup.sh in the base of the project with the following content, which is the same file as in the previous article. It would be used to retrieve the ssh key from the AWS Parameter Store.

sudo nano setup.sh

Then enter the content below into the file:

AWS\_REGION="us-east-1"  
KEY\_NAME="MyKeyPair"  
KEY\_PAIR\_ID=$(aws ec2 describe-key-pairs --region "$AWS\_REGION" --query "KeyPairs[?KeyName=='$KEY\_NAME'].KeyPairId" --output text)  
# Set the SSM parameter name where the SSH private key is stored  
PARAMETER\_NAME="/ec2/keypair/$KEY\_PAIR\_ID"  
# Set the output file name for the .pem file  
OUTPUT\_FILE="/home/ubuntu/.ssh/ssh\_key.pem"  
# Retrieve the SSH private key from SSM Parameter Store  
SSH\_PRIVATE\_KEY=$(aws ssm get-parameter --region "$AWS\_REGION" --name "$PARAMETER\_NAME" --with-decryption --query "Parameter.Value" --output text)  
# Save the SSH private key to a .pem file  
echo "$SSH\_PRIVATE\_KEY" > "$OUTPUT\_FILE"  
# Set the correct file permissions for the .pem file  
chmod 400 "$OUTPUT\_FILE"

**Step 3: Create Ansible Playbook**

The next step involves creating a [Playbook](https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_intro.html) that Ansible would use to manage the server.

This is a similar playbook that was created in the previous article with minor changes.

Run this command in the project base:

sudo nano playbook.yml

Then enter the content below into the file:

---  
- name: Ansible Production  
 hosts: tag\_Name\_SlaveProductionHost  
 tags:  
 - prod  
 become: true  
   
 vars:  
 # Specify the path to your .env file on your local machine  
 backend\_env\_file: /home/ubuntu/prod/env/backend/.env  
 frontend\_env\_file: /home/ubuntu/prod/env/frontend/.env  
   
 tasks:  
 - name: Update apt cache  
 apt:  
 update\_cache: yes  
  
 - name: Install required packages  
 apt:  
 name:  
 - nodejs  
 - npm  
 - git  
 state: present  
  
 - name: Clone MERN app from GitHub repository  
 git:  
 repo: "https://github.com/chyke007/Yum-food"  
 dest: /home/ubuntu/yum-food/  
 version: master  
 force: yes  
 clone: yes  
  
 - name: Copy server .env file to the server  
 synchronize:  
 src: "{{ backend\_env\_file }}"  
 dest: /home/ubuntu/yum-food/.env  
 mode: push  
  
 - name: Copy Frontend .env file to the frontend folder  
 synchronize:  
 src: "{{ frontend\_env\_file }}"  
 dest: /home/ubuntu/yum-food/frontend/.env  
 mode: push  
  
 - name: Install Packages  
 command: npm install   
 args:  
 chdir: /home/ubuntu/yum-food/  
 become: yes  
 become\_user: root  
  
 - name: Install Forever package  
 command: npm install forever -g  
 args:  
 chdir: /home/ubuntu/yum-food/  
 become: yes  
 become\_user: root  
  
 - name: Run seeder function  
 command: npm run seed   
 args:  
 chdir: /home/ubuntu/yum-food/  
 become: yes  
 become\_user: root  
  
 - name: Removes cache  
 command: rm -rf node\_modules  
 args:  
 chdir: /home/ubuntu/yum-food/frontend/  
 become: yes  
 become\_user: root  
  
 - name: Installs required modules  
 command: npm install --cache /tmp/empty-cache  
 args:  
 chdir: /home/ubuntu/yum-food/frontend/  
 become: yes  
 become\_user: root  
  
 - name: Build frontend  
 command: npm run build  
 args:  
 chdir: /home/ubuntu/yum-food/frontend/  
 become: yes  
 become\_user: root  
  
 - name: Start Node.js app  
 command: forever start index.js  
 args:  
 chdir: /home/ubuntu/yum-food/  
  
- name: Ansible Development  
 hosts: tag\_Name\_SlaveDevelopmentHost  
 become: true  
 tags:  
 - dev  
  
 vars:  
 # Specify the path to your .env file on your local machine  
 backend\_env\_file: /home/ubuntu/dev/env/backend/.env  
 frontend\_env\_file: /home/ubuntu/dev/env/frontend/.env  
   
 tasks:  
 - name: Update apt cache  
 apt:  
 update\_cache: yes  
  
 - name: Install required packages  
 apt:  
 name:  
 - nodejs  
 - npm  
 - git  
 state: present  
  
 - name: Clone MERN app from GitHub repository  
 git:  
 repo: "https://github.com/chyke007/Yum-food"  
 dest: /home/ubuntu/yum-food/  
 version: master  
 force: yes  
 clone: yes  
  
 - name: Copy server .env file to the server  
 synchronize:  
 src: "{{ backend\_env\_file }}"  
 dest: /home/ubuntu/yum-food/.env  
 mode: push  
  
 - name: Copy Frontend .env file to the frontend folder  
 synchronize:  
 src: "{{ frontend\_env\_file }}"  
 dest: /home/ubuntu/yum-food/frontend/.env  
 mode: push  
  
 - name: Install Packages  
 command: npm install   
 args:  
 chdir: /home/ubuntu/yum-food/  
 become: yes  
 become\_user: root  
  
 - name: Install Forever package  
 command: npm install forever -g  
 args:  
 chdir: /home/ubuntu/yum-food/  
 become: yes  
 become\_user: root  
  
 - name: Run seeder function  
 command: npm run seed   
 args:  
 chdir: /home/ubuntu/yum-food/  
 become: yes  
 become\_user: root  
  
 - name: Removes cache  
 command: rm -rf node\_modules  
 args:  
 chdir: /home/ubuntu/yum-food/frontend/  
 become: yes  
 become\_user: root  
  
 - name: Installs required modules  
 command: npm install --cache /tmp/empty-cache  
 args:  
 chdir: /home/ubuntu/yum-food/frontend/  
 become: yes  
 become\_user: root  
  
 - name: Build frontend  
 command: npm run build  
 args:  
 chdir: /home/ubuntu/yum-food/frontend/  
 become: yes  
 become\_user: root  
  
 - name: Start Node.js app  
 command: forever start index.js  
 args:  
 chdir: /home/ubuntu/yum-food/

**Step 4: Create a Jenkins file**

Create a new file called Jenkinsfile in the base of the project. This would contain the Jenkins declarative script the Jenkins server would run.

sudo nano Jenkinsfile

Then enter the content below into the file:

pipeline {  
 agent any  
   
 tools {nodejs "node"}  
 stages {  
   
 stage('Install dependencies') {  
 steps {  
 sh 'npm install'   
 echo 'Installing dependecies found in branch: ' + env.BRANCH\_NAME  
 }  
 }  
   
 stage('Unit Test stage') {  
   
 steps {  
 sh 'npm run test'  
 }  
 }  
  
 stage('Integration Test stage') {  
   
 steps {  
 sh 'npm run integration-test'  
 }  
 }  
  
 stage('Deploy stage:Dev') {  
  
 when {  
 branch 'dev'  
 }  
  
 steps {  
 echo "Running Dev Deploy"  
 sh 'chmod +x ./setup.sh'  
 sh './setup.sh && ansible-playbook playbook.yml --tags dev && rm -rf /home/ubuntu/ssh\_key.pem'  
 }  
 }  
  
 stage('Deploy stage:Prod') {  
  
 when {  
 branch 'master'  
 }  
  
 steps {  
 echo "Running Prod Deploy"  
 sh 'chmod +x ./setup.sh'  
 sh './setup.sh && ansible-playbook playbook.yml --tags prod && rm -rf /home/ubuntu/ssh\_key.pem'  
 }  
 }  
 }  
}

**Step 5: Create .env file on Jenkins Server**

Head over to the **AWS Management Console** and connect to the EC2 instance (using the **EC2 Instance connect**) with the tag name of MasterHost, which is the server with Jenkins and Ansible so as to run some configuration.

Once connected to the instance, run this command to create the respective directory the .env files would be in:

mkdir -p /home/ubuntu/dev/env/backend && mkdir -p /home/ubuntu/prod/env/backend/ && mkdir -p /home/ubuntu/dev/env/frontend/ && mkdir -p /home/ubuntu/prod/env/frontend/

**Create env for the backend**

Run the command below to create the .env file that would be used by the backend part of the application for the dev credentials.

sudo nano /home/ubuntu/dev/env/backend/.env

Then enter the content below into the file:

SECRET=1234567  
DB\_HOST=your\_mongo\_user:your\_mongo\_password@cluster0.xxx.mongodb.net  
DB\_USER=your\_mongo\_user  
PORT=4000  
DB\_PASS=your\_mongo\_password  
DB\_NAME=yum-food  
DB\_EXTENSION=+srv://  
NODE\_ENV='development'  
API\_KEY=anystringcangohere  
CLOUDINARY\_CLOUD\_NAME=your\_cloudinary\_cloud\_name  
CLOUDINARY\_API\_KEY=your\_cloudinary\_api\_key  
CLOUDINARY\_API\_SECRET=your\_cloudinary\_api\_secret

Repeat the process for the prod credentials, this approach saves 2 sets of credentials for the backend(/home/ubuntu/prod/env/backend/.env) so they each can be moved to their respective servers.

After that is done, proceed to do the same for the front end.

**Create env for frontend**

Run the command below to create the .env file that would be used by the frontend part of the application.

sudo nano /home/ubuntu/dev/env/frontend/.env

Then enter the content below into the file:

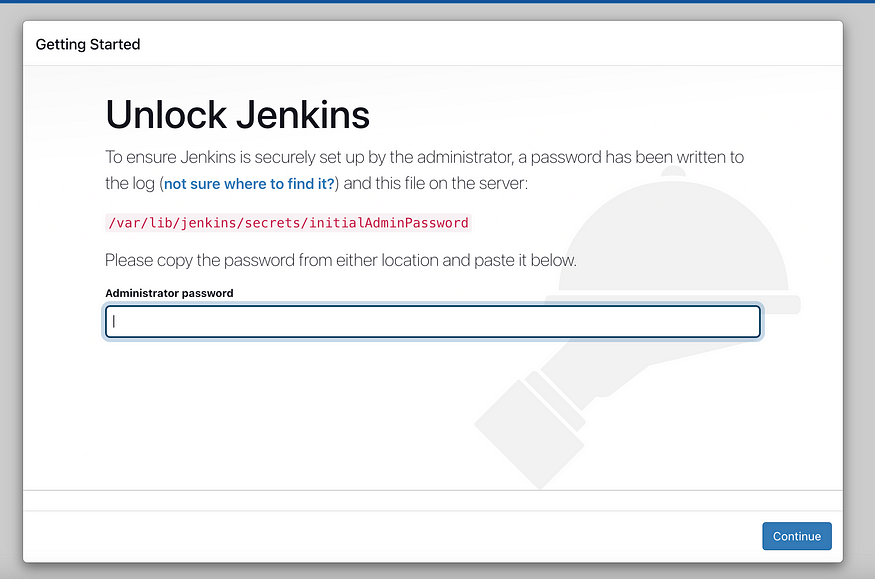
SKIP\_PREFLIGHT\_CHECK=true  
REACT\_APP\_API\_KEY=anystringcangohere

Repeat the process for the prod credentials (/home/ubuntu/prod/env/frontend/.env), as done for the backend credentials.

**Step 6: Jenkins Setup**

Jenkins runs on port 8080. Open the Public IP address of the Jenkins server.

[http://ec2-3-81-24-223.compute-1.amazonaws.com:8080](http://ec2-3-81-24-223.compute-1.amazonaws.com:8080/login?from=%2F)  
  
[http://ec2-3-81-24-223.compute-1.amazonaws.com:8080](http://ec2-3-81-24-223.compute-1.amazonaws.com:8080/login?from=%2F)

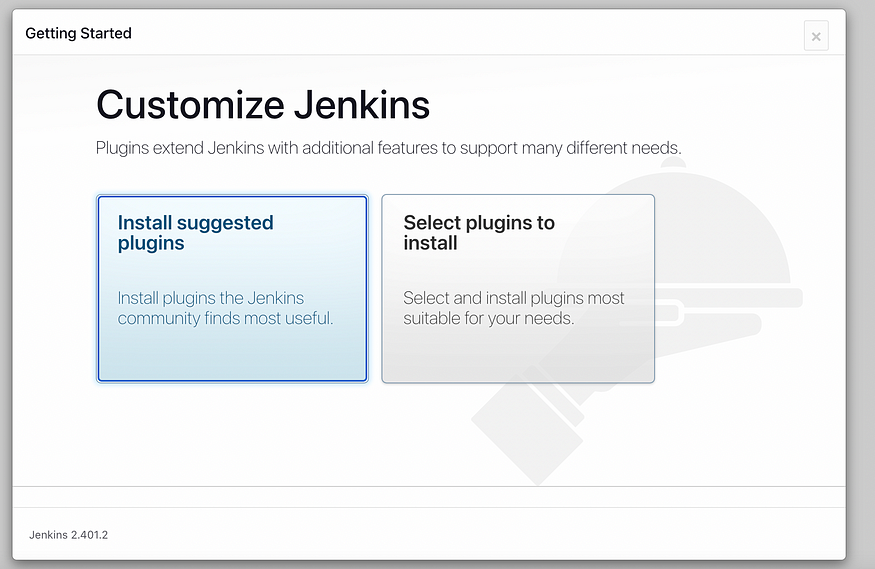


To get the administrator password, run this command:

sudo cat /var/lib/jenkins/secrets/initialAdminPassword



Copy the outputted password and enter. You would get to a page similar to the one below, click the Install Suggested Plugins to install.



A screenshot of a computer

Description automatically generated

Would take a couple of minutes to install all suggested plugins. After that create your first admin user.

Once done entering details, click the **Save and Continue** button

A screenshot of a computer

Description automatically generated

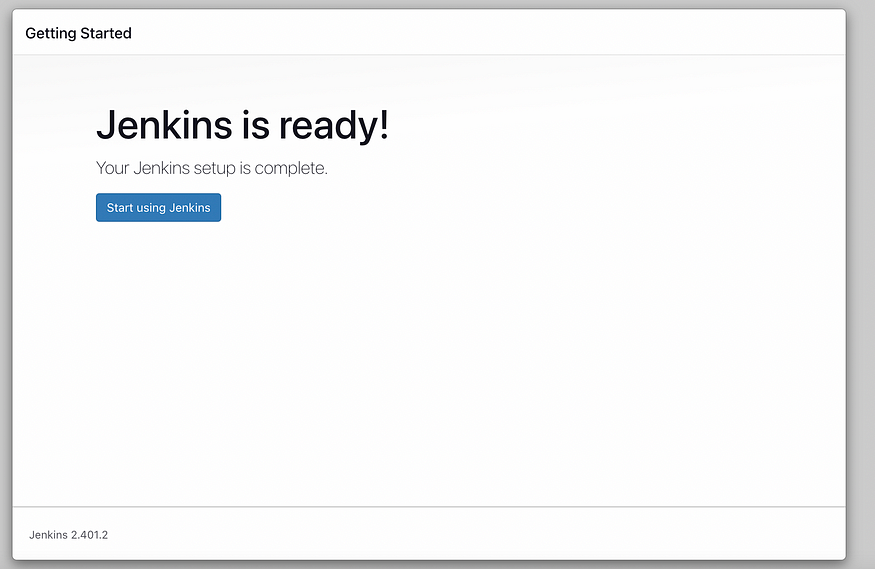
Leave the default Jenkins URL specified, and click the **Save and Finish**:

A screenshot of a computer

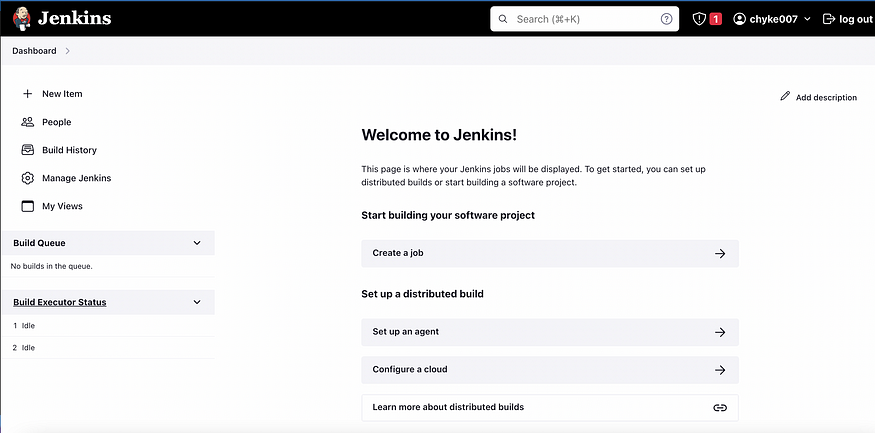
Description automatically generated

You should now get a confirmation message “Jenkins is ready!”. Click Start using Jenkins:

Click the **Start using Jenkins** to get started!



You are now in:



**Step 7: Install Necessary Plugins**

The following Jenkins plugins need to be installed:

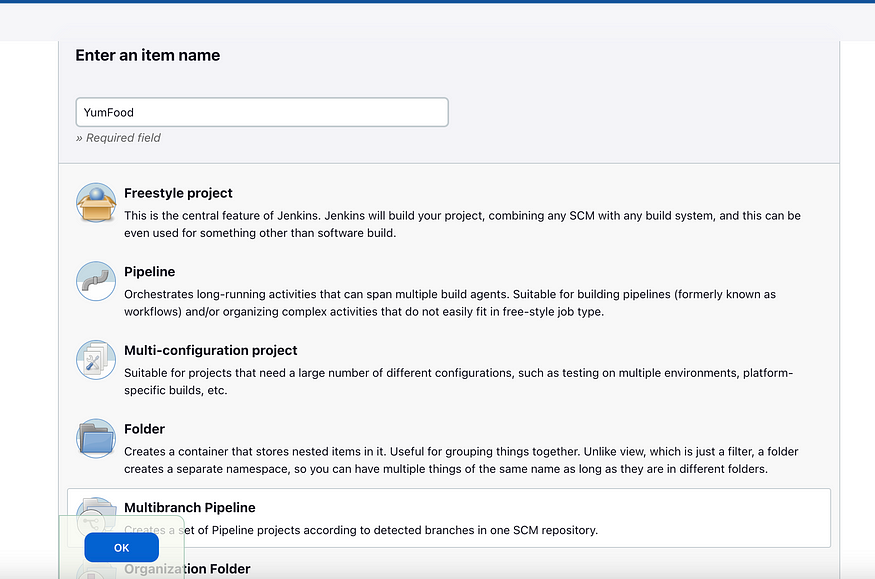
1. [Strict-Crumb-issuer](https://plugins.jenkins.io/strict-crumb-issuer/): For Authorisation and provide a valid token for requests.
2. [Nodejs](https://plugins.jenkins.io/nodejs/): To use Nodejs in pipeline.

Head over to the respective pages for the installation guide

**Step 8: Create Jenkins Pipeline**

Click the **Create a Job** to create a pipeline. Enter YumFood as the name and select **Multibranch Pipeline.**

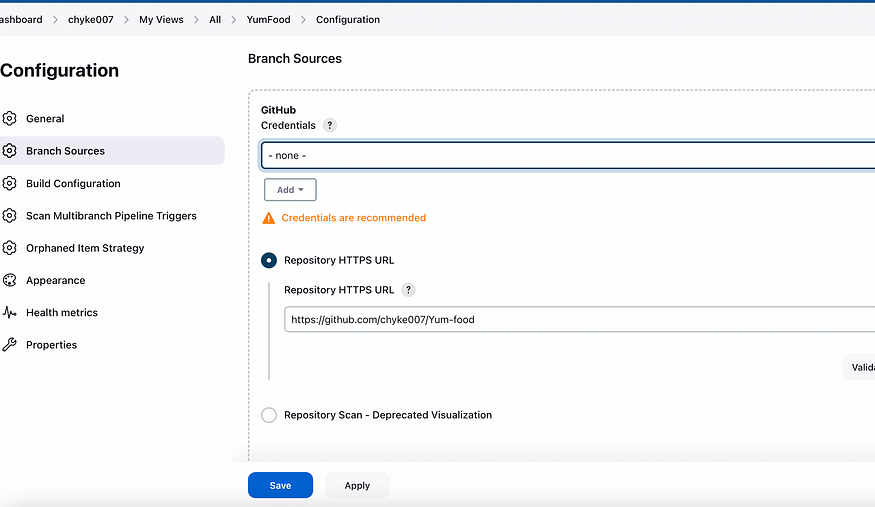
Using this would allow us to track and run the Jenkins pipeline for all branches in the repo:



Create Job

Click OK.

Then proceed to configure the Job properties by selecting the repo from GitHub and **Strategy** to be **All branches**



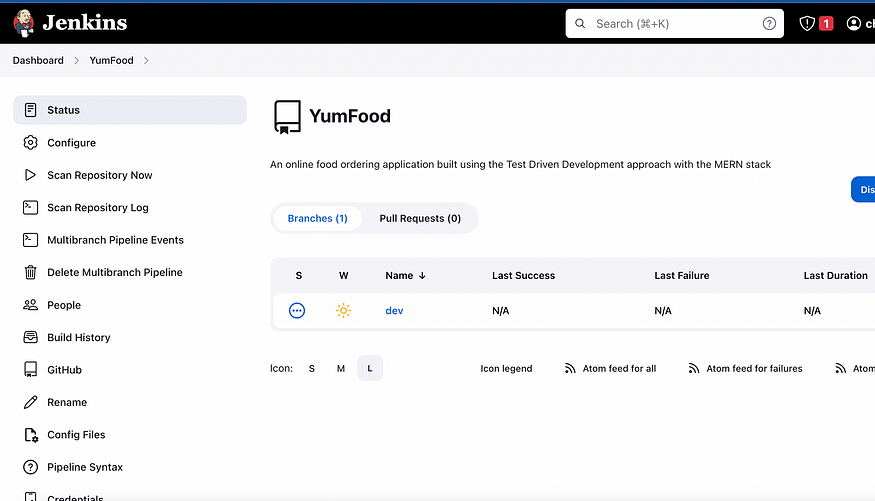
From GitHub

A screenshot of a computer

Description automatically generated

All Branches

Click the **Apply** button and then the **Save**button.



Jenkins would begin to Branch Index the branches in the repo with an active JenkinsFile, as can be seen in the image, only dev currently has.

Clicking on the dev branch, we see the Indexing is still ongoing, hence, no build has started yet.

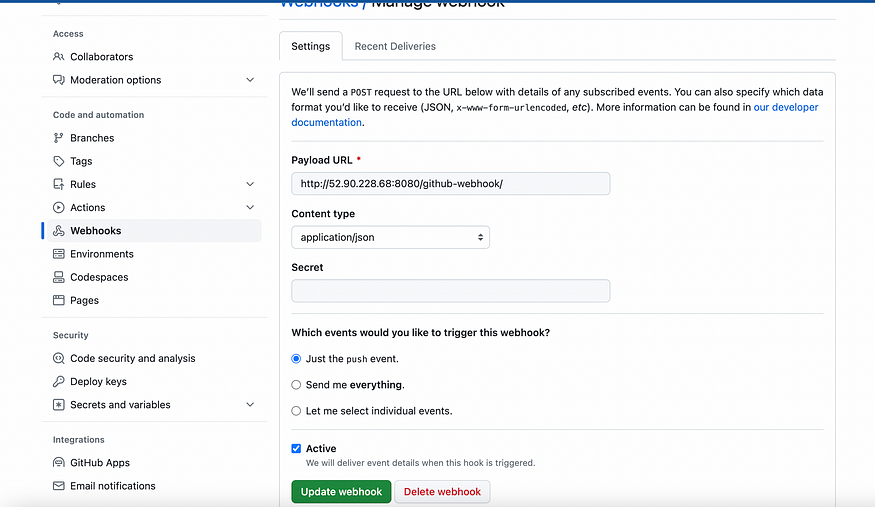
A screenshot of a computer

Description automatically generated

**Step 9: Add Webhook to Github**

Next, head over to GitHub and add a webhook, this would enable Github to trigger the endpoint provided once there is a push to the repo, in this case, this endpoint is Jenkins.

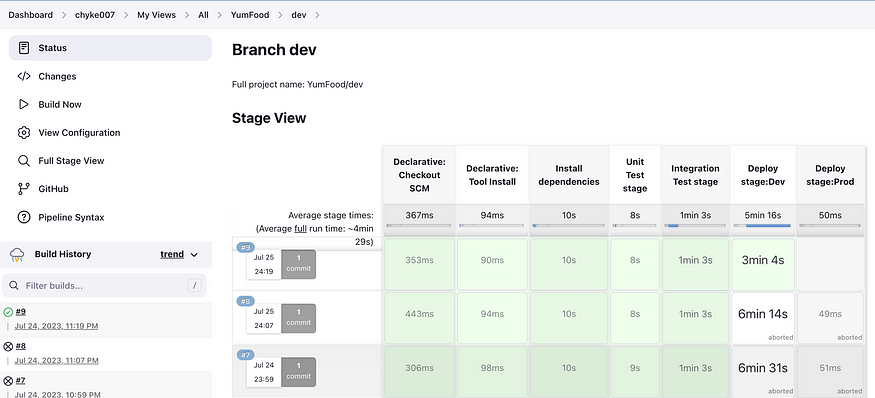
Replace http://52.90.228.68:8080 with your Jenkins server public IP address



**Step 10: Test Pipeline(DEV branch)**

Create a dev branch locally makes some minor changes to the codebase and push to your repo. This would trigger the pipeline if everything is configured well.

If you notice, the Deploy stage:Prod stage doesn't run, this is because we added a condition in the Jenkins file to only run that stage on the master branch.



Also, notice below that the Ansible playbook only runs the Ansible Development play and skips the Ansible Production play. This makes Ansible deploy to the dev server and skip the prod server since the branch that triggered the build job was the dev branch.

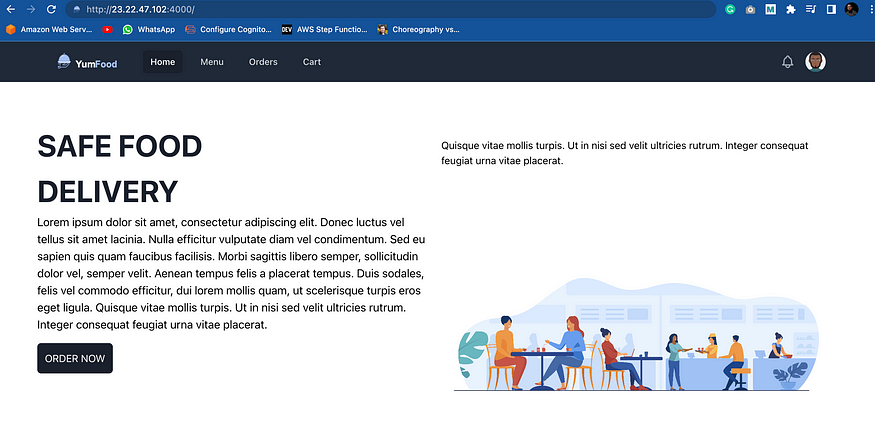
The same behavior would happen if the master branch was the activating branch, the use of --tags dev in the command is what helps achieve this.

A screenshot of a computer

Description automatically generated

Head over to the EC2 with the tag name **SlaveDevelopmentHost** via the EC2 Console. Click the Public IP address, and add the port to test:

You should have a similar IP address: <http://23.22.47.102:4000/>



**Step 11: Test Pipeline(Master branch)**

If you visit the EC2 instance with the tag name of **SlaveProductionHost**, the site won't be up as we haven't deployed there yet.

Let's fix, that by doing a deployment to the Production server.

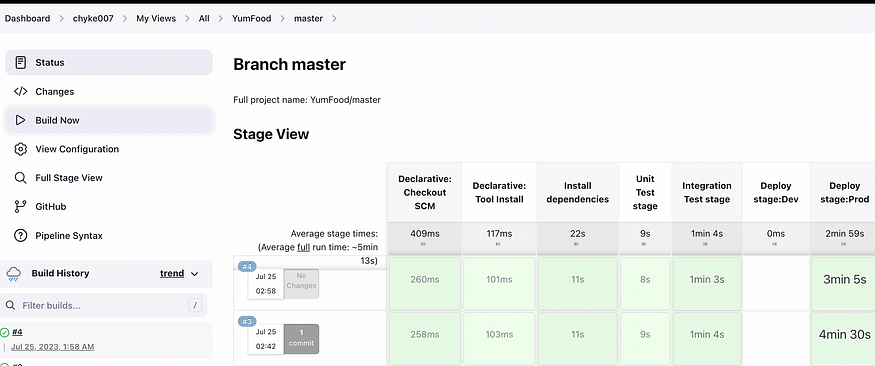
Create a Pull Request from the **Dev** branch to the **Master.**Then merge in that PR to trigger the pipeline to build for**Production.**

The Pipeline picks it up and begins the master branch pipeline:

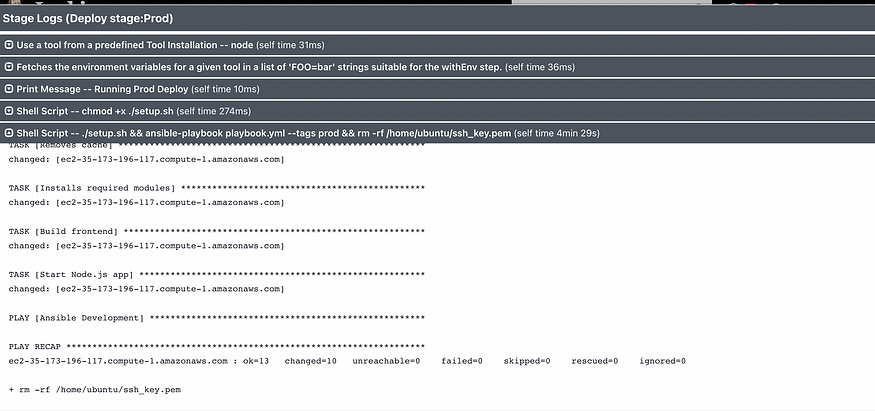
A screenshot of a computer

Description automatically generated

If you notice below also, the Deploy stage:Dev stage doesn't run, this is because we added a condition in the Jenkins file to only run that stage on the dev branch.

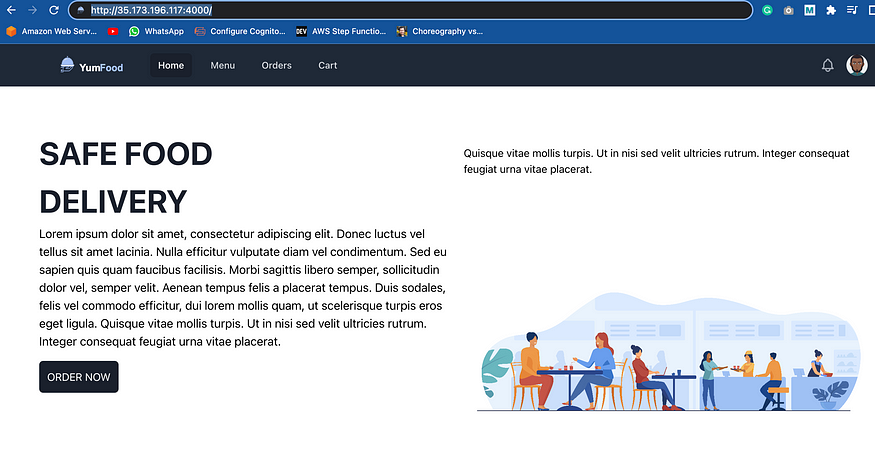


Hence Ansible playbook only runs the Ansible Production play and skips the Ansible Development play too due to --tags prod passed to it.



Head over to the EC2 with the tag name **SlaveProductionHost** via the EC2 Console. Click the Public IP address, and add the port to test:

You should have a similar IP address: <http://35.173.196.117:4000/>



To test real-time change, I updated the Landing Page content of the **Dev** branch and pushed it to the repo.

After running successfully we now have a different Landing Page for the Dev server. This quick test is very useful for checking out features built during development before actually merging them into the live production server.