3tier-Architechture with CI/CD Setup

3tier-Architechture with CI/CD Setup 1

Version History 1

Overview 1

Procedure 2

References 8

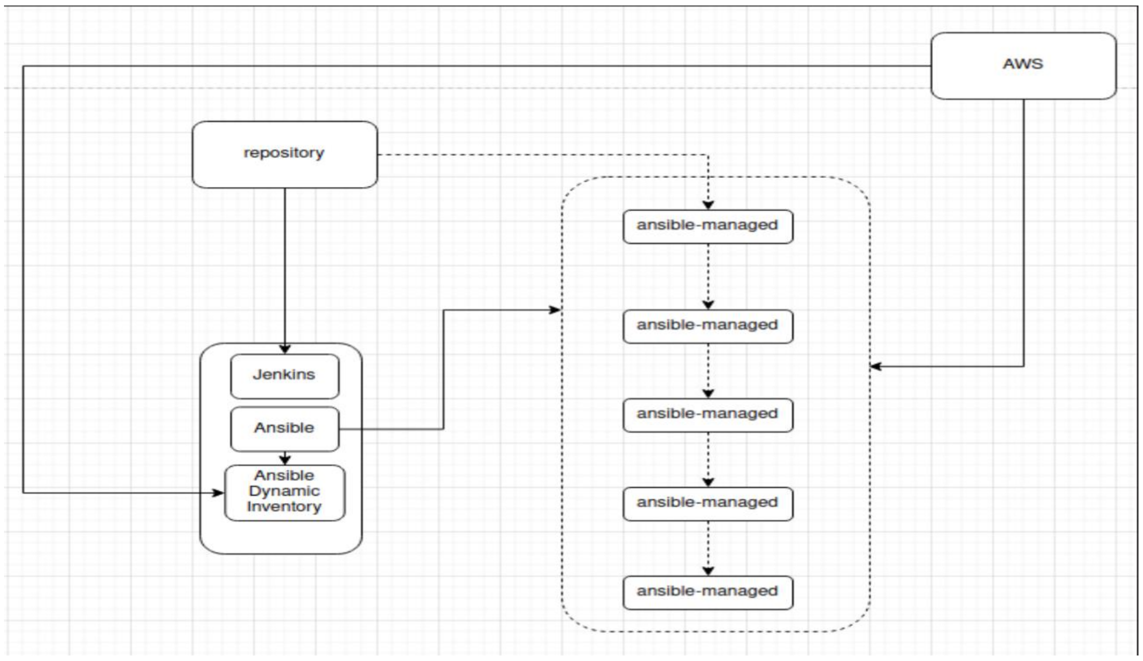
Errors 8

Version History

#	Date	Version	Author	Reviewer
1	15/09/2023	1.0.0	Umapoornima	

Overview

In this Proof of Concept (PoC), you've established a robust infrastructure for your front-end application. Your setup includes Git for version control, AWS Lambda functions and API Gateway for the backend, and Ansible for automation. With Ansible, you've configured an Ansible master server and target node server, enabling seamless application deployment and management. Additionally, you've integrated webhooks in your Git repository to trigger Jenkins freestyle jobs upon code updates. When changes are pushed to the Git repo, Jenkins orchestrates the execution of an Ansible playbook, ensuring that the updated front-end code is consistently deployed across all target instances. This end-to-end automation streamlines the development and deployment process, making it efficient and reliable.



Front-end Application: You have your front-end application code stored in a Git repository. This code represents the user interface and functionality of your application.
Backend to Database Connection: You've created Lambda functions that handle the backend logic of your application and connect to your database. This ensures the separation of the front-end and back-end components.
Backend to Front-end Connection: To serve your front-end application, you've set up an API Gateway. This API Gateway routes requests from the front-end to the appropriate Lambda functions.

Procedure

Step 1: Launching EC2 Instances

- In this step, you launch two Ubuntu servers: one for the Ansible master and another for the Ansible node.
- These servers will be used to manage and execute Ansible playbooks remotely.

Step 2: Copying PEM File

- You copy your AWS PEM file to the Ansible master server. This is necessary to establish SSH connections to other instances.
- which keypair you used to launch your ansible-master and ansible-node server copy that pem file
- first copy your pem file to your master server by using git-bash, In git-bash go to your pem-file (or) keypair folder path where you downloaded your key
- This is the command to copy your pem file
  scp -i C:/Users/admin/Downloads/awskey-oregon.pem awskey-oregon.pem ubuntu@54.188.23.133:/home/ubuntu

  C:/Users/admin/Downloads/awskey-oregon.pem- specifies the path to your local SSH key.

  awskey-oregon.pem- mention your pem file name.

ubuntu@54.188.23.133:/home/ubuntu- username of master server@public-ip:/ destination directory on the remote server where you want to copy the file.

Step 3: Copying Ansible Playbook

- You've created an Ansible playbook, which is used for automating tasks on your target servers (instances). In this case, the playbook is responsible for deploying and configuring your application on these servers.
- You copy your Ansible playbook to the Ansible master server. This playbook will be used to configure and manage your infrastructure.
- **** Refer to the below zip file to create ansible playbook.
  Mybas10bookAnsibleSetup.zip

  Mybas10bookAnsibleSetup.zip
- copy your ansible playbook zip file from windows to your ansible-master server
  ```
   scp -i awskey-oregon.pem Mybas10bookAnsibleSetup.zip
  ubuntu@35.88.141.95:/home/ubuntu
                  sudo apt install zip
                  unzip filename.zip
  ```
- Run **ls** command their you can able to see **"bas10book-application"** folder
- Go to this folder **"cd bas10book-application"** their you can able to see **"deploy_bas10book.yml"&"update_bas10book.yaml"** in those files instead of git repo url replace your git repository url

Step 4: Installing Ansible

- You install Ansible on the Ansible master server. This tool will be used to automate tasks on your managed instances.
  (**Reference link:** https://stackoverflow.com/questions/63672853/ansible-config-file-not-found-using-defaults)
  ```
  sudo apt update
  sudo apt install software-properties-common
  sudo apt-add-repository --yes --update ppa:ansible/ansible
  sudo apt install ansible
  ```

Step 5: Setting Up SSH Connection

- You've established SSH connections between the Ansible master server and the target node servers. This is how Ansible communicates with and manages the node servers.
- You configure SSH connections from the Ansible master to the Ansible node using your PEM file.
- You also ensure that the appropriate permissions are set for the PEM file.

- Now, we need to give the ssh connection for master to slave
  ```
  first copy your pem file
  sudo chmod 400 awskey-oregon.pem
  ```

```
ssh-agent bash
ssh-add /home/ubuntu/awskey-oregon.pem
sudo ssh -i /home/ubuntu/awskey-oregon.pem ubuntu@34.222.109.37
```
---for connecting master to slave (give your node server username@node-ip)

Step 6: Running Ansible Playbook

- You execute the Ansible playbook from the master instance to the managed-node instance using the `ansible-playbook` command.
- This playbook likely contains tasks to set up and configure your infrastructure.
- Run your ansible playbook remotely execute the playbook from the master instance on the managed-node instance

  "ansible-playbook -i 34.222.109.37, deploy_bas10book.yml" ------mention your managed node ip-address

Step 7: Checking Application

- You verify if the application is accessible on the managed-node instance's IP address. This step ensures that your Ansible playbook was successful in setting up the application.
- check whether the application is being served in the Ip address of the managed-node instance along with /register extension

  ```
  34.222.109.37/register
  ```

Step 8: Dynamic Inventory Setup

- Ansible dynamic inventory helps you manage your target servers dynamically. It allows Ansible to discover and list your target instances automatically.
- You create a dynamic inventory file to manage your EC2 instances dynamically.
- This inventory will be used by Ansible to determine which hosts to manage.
- create dynamic inventory file follow this document (**LINK:** https://devopscube.com/setup-ansible-aws-dynamic-inventory/)

Step 9: IAM Role and Policy

- You create an IAM role and policy to give the Ansible master server the necessary permissions to interact with AWS resources.
- This role will be attached to the Ansible master instance.
- create IAM Role and Policy for ec2 full access and attached that role to your master instance (where your ansible is running)

Step 10: Installing Python and Dependencies

- You ensure that Python 3 and necessary dependencies (like `boto3`) are installed on the Ansible server.
- Ensure you have python3 & pip3 installed in your Ansible server

  ```
  python3 --version
  sudo apt-get install python3 -y
  sudo apt-get install python3-pip -y
  ```
- If you have used the Ansible ppa for installation, install pip using the following command

```
sudo apt-get install python-boto3   (or)
sudo pip3 install boto3
sudo pip show boto3
```

Step 11: Creating Dynamic Inventory File

- You create an `aws_ec2.yaml` file in a specific directory to define your dynamic inventory.
- This file specifies how Ansible should gather information about your AWS EC2 instances.
- create one directory ---> sudo mkdir -p /opt/ansible/inventory

```
cd /opt/ansible/inventory
sudo touch aws_ec2.yaml
sudo vi aws_ec2.yaml            # add this below script and save the file
  ---
  plugin: aws_ec2
  regions:
    - us-west-2      #Add your desired region here
  filters:
    "tag:Name": "node"     #select your ec2-target instance and click on tags their you can
  able to see key and value add key, value names under tag
```

Step 12: Ansible Configuration File

- You configure Ansible to use the dynamic inventory file you created.
- This step ensures that Ansible can dynamically discover and manage AWS EC2 instances.
- Go to ansible home directory under ansible.cfg file **"cd /etc/ansible"** sudo vi ansible.cfg   ## add below script and save it

```
[defaults]
host_key_checking = False
inventory       = /opt/ansible/inventory/aws_ec2.yaml

[inventory]
enable_plugins: aws_ec2
```

Step 13: Testing Dynamic Inventory

- You test the dynamic inventory configuration using the `ansible-inventory` command.
- This verifies that Ansible can discover your EC2 instances.
- Now let's test the dynamic inventory configuration by listing the ec2 instances.

```
ansible-inventory -i /opt/ansible/inventory/aws_ec2.yaml --list
```

Step 14: Testing Ansible Ping

- You use the `ansible` command to ping all the machines returned by the dynamic inventory.
- This tests whether Ansible can communicate with your EC2 instances.
- Execute the following command to test if Ansible is able to ping all the machines returned by the dynamic inventory

```
ansible all -m ping
```

Step 15: Running Playbook with Dynamic Inventory

- You run your Ansible playbook using the dynamic inventory configuration.

```
ansible-playbook -i /opt/ansible/inventory/aws_ec2.yaml deploy_bas10book.yml
```

- This allows Ansible to manage AWS instances automatically.

Step 16: Installing Jenkins

- You install Jenkins on the Ansible master server.
- Jenkins will be used to automate CI/CD processes.

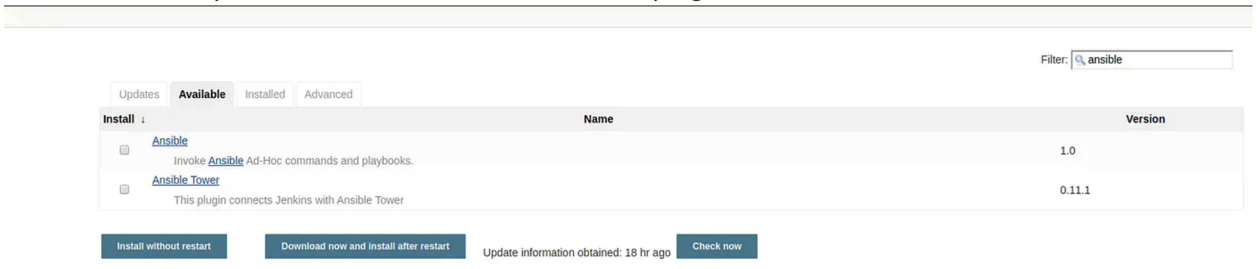
```
sudo apt update
sudo apt install openjdk-17-jre
java -version

sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key

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-get update
sudo apt-get install jenkins
sudo systemctl status jenkins
sudo systemctl start jenkins
sudo systemctl enable jenkins
```

- Copy your ansible-master public-p "35.88.141.95:8080" along with port no 8080 you can able to see the jenkins dashboard now unlock the jenkins

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

Step 17: Install Ansible Plugin

- You install the Ansible plugin, to enable Jenkins to execute Ansible playbooks.
- For Jenkins Pipeline we need to install the Ansible plugin.



Install ↓	Name	Version
☐	Ansible Invoke Ansible Ad-Hoc commands and playbooks.	1.0
☐	Ansible Tower This plugin connects Jenkins with Ansible Tower	0.11.1

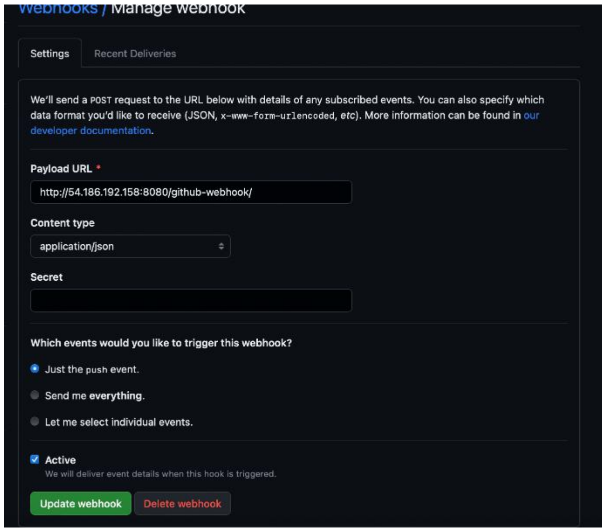
Update information obtained: 18 hr ago

- Go to manage Jenkins under that section click on **"Tools"** & Go to **"Ansible installations"** section and give Name as like **"ansible-playbook"** and "Path to ansible executables directory" under this section give path as like **"/usr/bin/"** and click on save

Step 18: Setting Up Webhooks

- You've set up webhooks in your Git repository. Webhooks are HTTP callbacks that are triggered when certain events occur in your repository, such as code pushes. These webhooks are configured to notify Jenkins whenever updates are pushed to the Git repository.
- You configure webhooks in your Git repository to trigger Jenkins whenever changes are pushed to the repository.
- This sets up automated builds and deployments.
- Setup webhook from the git repository to trigger the jenkins on each commit pushed to the main branch repo.
    - Go to your bastion book application git repository and go to settings there we can able to see webhooks click on that create a new webhook
    - under payload url give your jenkins dashboard url along with "github-webhooks" extension

      Ex: http://35.88.141.95:8080/github-webhook/



Webhooks / Manage webhook

Settings | Recent Deliveries

We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, etc). More information can be found in our developer documentation.

Payload URL *

http://54.186.192.158:8080/github-webhook/

Content type

application/json

Secret

Which events would you like to trigger this webhook?

● Just the push event.

○ Send me everything.

○ Let me select individual events.

✓ Active

We will deliver event details when this hook is triggered.

Step 19: Configuring Jenkins Job

- The Jenkins job is responsible for triggering the Ansible playbook. When it runs, it communicates with the Ansible master server and instructs it to deploy the updated front-end code to all the target instances.
- In Jenkins, you've created a freestyle job that listens for webhook notifications from your Git repository. When the webhook is triggered (e.g., due to a code update), Jenkins starts the job.
- You create and configure a Jenkins job (freestyle project) to monitor your Git repository for changes.
- You set up the Ansible plugin to run your Ansible playbooks and propagate changes to your target instances.

Steps to Configure Free Style Job: configure a build job (free-style) to monitor repo for changes and configure the ansible plugin for the build job to run the required playbooks and propagate the repo changes to the target instances

- create one free-style job in your Jenkins dash-board and click on that job

- click on configuration
- click on **Source Code Management** as a GIT and copy and paste your git-hub repository URL and change your branch as a "main"
- under **Build Triggers** section enable "GitHub hook trigger for GITScm polling" option
- under **Build Steps** section click on add build step and select **"Invoke Ansible Playbook"** and give your ansible playbook path "/var/lib/jenkins/workspace/bas10book-application/update_bas10book.yaml"
- Please remember to configure your Jenkins project to run your 'update' ansible playbook
- under credentials section click "Add" and click on "Jenkins OR Jenkins credentials provider"

  -->under kind section select "SSH User name with private key option"

  -->under ID give "ssh-privatekey" for identification

  -->under Description give some description like "credentials for Jenkins"

  -->under user-name section give user-name of your target node server "ubuntu"

  -->under private key section click on "Enter directly" option and click on add and

  -->copy your private-key pem file key and paste (which you used to launch your ansible-master and ansible-node servers)

  -->click on Add and add your credentials which you created in previous step and

  -->click on save

your setup allows for continuous integration and deployment of your front-end application. Any code updates pushed to your Git repository will automatically trigger a series of actions, ultimately updating your application on all target instances. This ensures that your application stays up to date and consistent across all servers.

References

Umapoornima/bas10book-poc6-Reference (github.com)

In this git repo url I have uploaded the ansible-playbook folder and I have added entire steps and procedure of this poc for 3tier-Architechture with Manul setup and 3tier-Architechture with CICD set-up (by using ansible playbook). If you have any queries regarding this poc and playbook just refer this Git repository.

Errors

If you face any errors while doing this poc, refer this below documentation
06_3tierarchitechture with CICD Setup_Errors.docx (sharepoint.com)