Wordpress Application Deployment Using Multiple Methods

Presented By

Amgoth Pravalika

AGENDA

1. Manual RDS Deployment: Deploy WordPress using AWS RDS (MySQL)
2. Docker Compose Deployment: Containerize WordPress with Docker Compose
3. Git +Jenkins CI/CD: Deploy using Git and Jenkins
4. User Data Automation: Automate deployment with EC2 User Data
5. Jenkins Bash Scripts: Use Jenkins and Shell Scripts for deployment
6. Jenkins Pipeline: Automate with SCM Polling and Periodic Builds
7. Terraform Deployment: Deploy WordPress using Terraform scripts
8. Jenkins +Terraform: Integrate Git, Jenkins, and Terraform for automation
9. Automated WordPress Deployment with Git, Jenkins pipeline and Terraform
10. Kubernetes Deployment: Deploy WordPress using Kubernetes manifests

Method-1

Step 1: Create MySQL Database with AWS RDS

Step 2: Set Up EC2 Instance

Step 3: Allow TraTc from EC2 to RDS

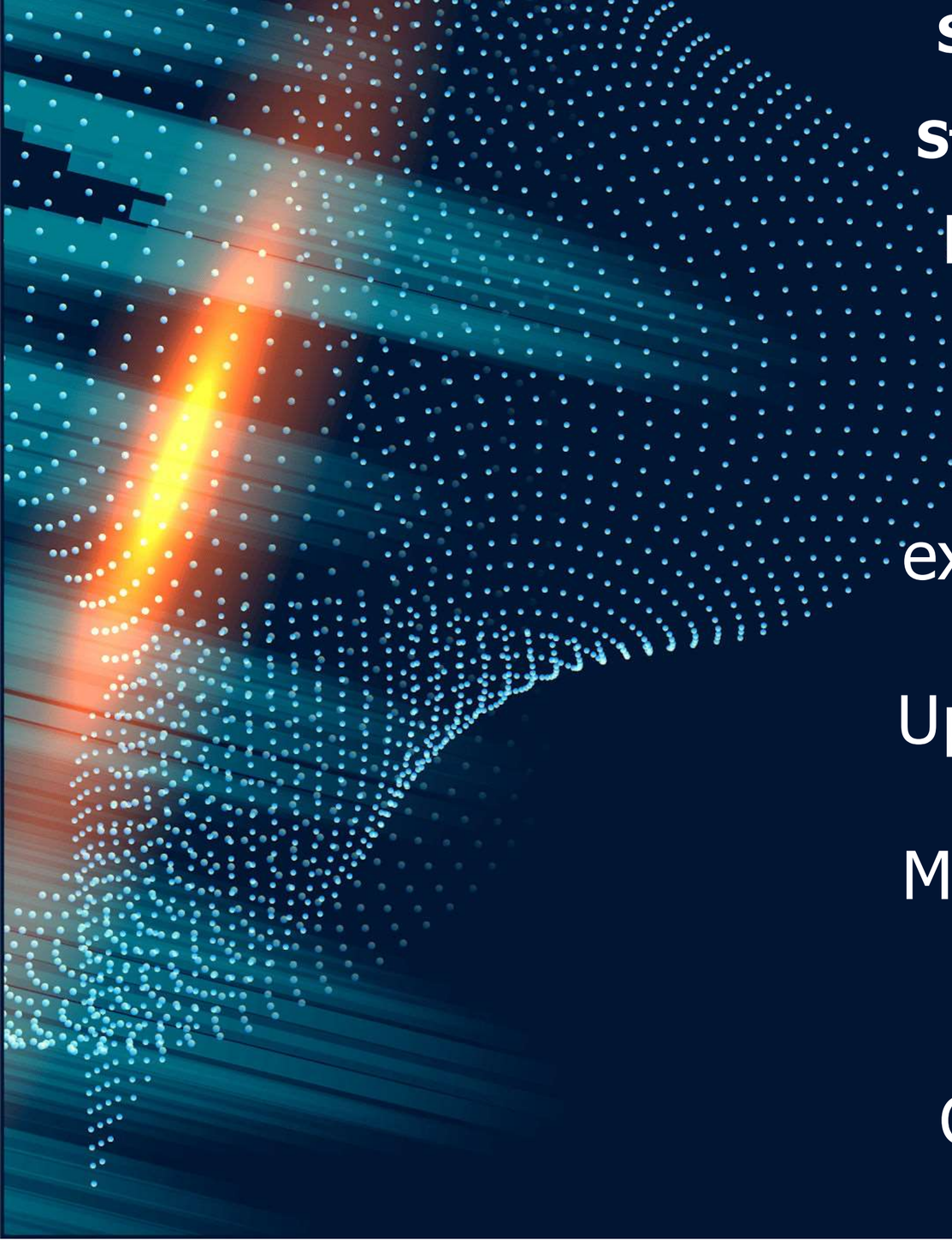
Step 4: Access MySQL Database

Export the RDS Endpoint: export MYSQL_HOST=<endpoint address>.Access MySQL with: mysql -h <endpoint> -u <username> -p.

Create WordPress user:CREATE USER 'wordpress' IDENTIFIED BY 'wordpress-pass';

GRANT ALL PRIVILEGES ON wordpress.* TO 'wordpress' ;FLUSH PRIVILEGES;

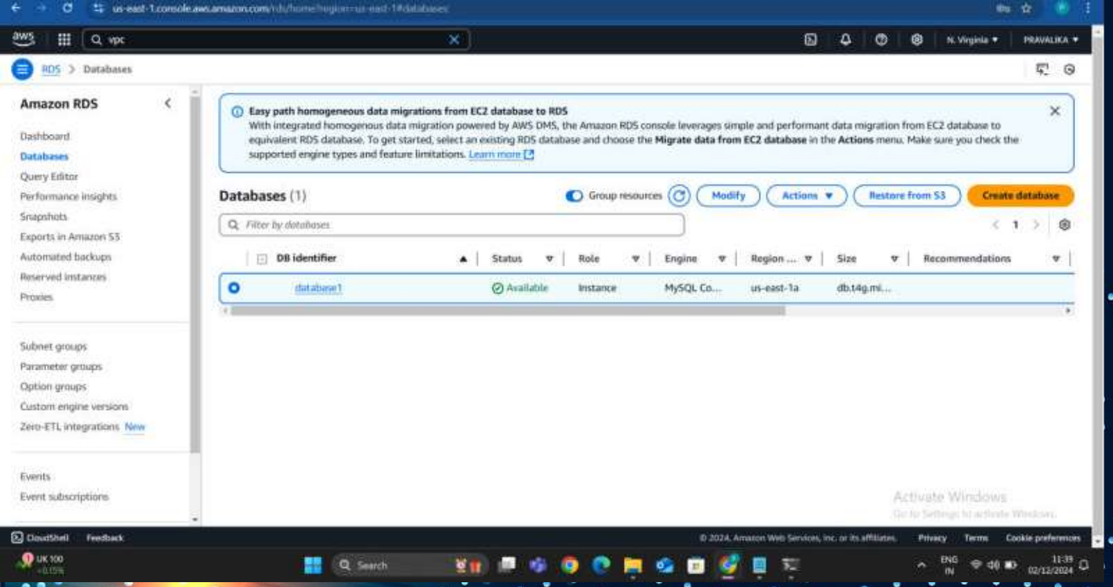
EXIT



Step 5: Install Apache Web Server

Step 6: Download and Configure WordPress

Download WordPress using wget <URL>.
Unzip WordPress: unzip <file>.
Install PHP: sudo amazon-linux-extras install -y lamp-mariadb10.2-php7.2.
Update EC2 instance if needed: sudo yum -y update.
Move wp-config-sample.php to wp-config.php: sudo mv wp-config-sample.php wp-config.php.
Configure WordPress: sudo vi wp-config.php.



Step 7: Final WordPress Setup

- SetDB details(database name, user, password, host) in.

-GenerateWordPress secret keysonline.

-Copy WordPress files to /var/www/html/: sudo cp -r /* /var/www/html/

-Restart Apache:sudo systemctl restart httpd

-Verify WordPress setup by checking EC2’s public IP in a browser.

METHOD-2:Docker-compose

Step 1: Launch EC2 Instance

Step 2: Install Git, Docker

Set Docker permissions:sudo usermod -aG ec2-user docker

sudo chmod 666 /var/run/docker.sock

Step 3: Install Docker Compose

Set executable permissions: sudo chmod +x /usr/local/bin/docker-compose

Create a symbolic link:ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose

Step 4: Configure Docker Compose File

```yaml
version: '3.3'
services:
  db:
    image: mysql:8.0.19
    command: --default-authentication-plugin=mysql_native_password
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      - MYSQL_ROOT_PASSWORD=wordpress
      - MYSQL_DATABASE=databaseword
      - MYSQL_USER=admin
      - MYSQL_PASSWORD=admin123

  wordpress:
    image: wordpress:latest
    ports:
      - "80:80"
    restart: always
    environment:
      - WORDPRESS_DB_HOST=db
      - WORDPRESS_DB_USER=admin
      - WORDPRESS_DB_PASSWORD=admin123
      - WORDPRESS_DB_NAME=databaseword

volumes:
  db_data:
```

Method-3-Using Git and Jenkins

Step 1: Launch EC2 Instance

Step 2: Install Required Tools on EC2 Instance

a.Git b.Docker,Docker-compose c.Jenkins d.mysql

Step 3: Create Jenkins Freestyle Job

Enter theGit repository URL that contains your Docker Compose files.

Add Build Step (Execute Shell):docker-compose up -d

Step 4: Build the Jenkins Job

Step 5: Access the WordPress Application

Method-4:Ec2 User-data

```
#!/bin/bash
# Update the system
yum update -y
yum install -y docker
systemctl start docker
systemctl enable docker
sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname
-s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
docker-compose --version
sudo usermod -aG docker ec2-user
sudo chmod 666 /var/run/docker.sock
sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
systemctl start docker
cat > /home/ec2-user/docker-compose.yml <<EOL
version: '3.3'
services:
  db:
    image: mysql:8.0.19
    command: --default-authentication-plugin=mysql_native_password
    volumes:
      - db_data:/var/lib/mysql
    restart: always
```

```
    environment:
      - MYSQL_ROOT_PASSWORD=wordpress
      - MYSQL_DATABASE=databaseword
      - MYSQL_USER=admin
      - MYSQL_PASSWORD=admin123
  wordpress:
    image: wordpress:latest
    ports:
      - "80:80"
    restart: always
    environment:
      - WORDPRESS_DB_HOST=db
      - WORDPRESS_DB_USER=admin
    - WORDPRESS_DB_PASSWORD=admin123
      - WORDPRESS_DB_NAME=databaseword
volumes:
  db_data:
EOL
chmod 644 /home/ec2-user/docker-compose.yml
cd /home/ec2-user
docker-compose up -d
docker ps
```

Method-5: Using Git and Jenkins bash script

Step 1: Launch EC2 Instance and Install Required Software

Step 2: Create Jenkins Job

Provide the Git repository URLthat contains yourDocker Compose file.

Step 3: Configure Jenkins Build Step

In the Build section of Jenkins, clickAdd build step>Execute Shell. add these script

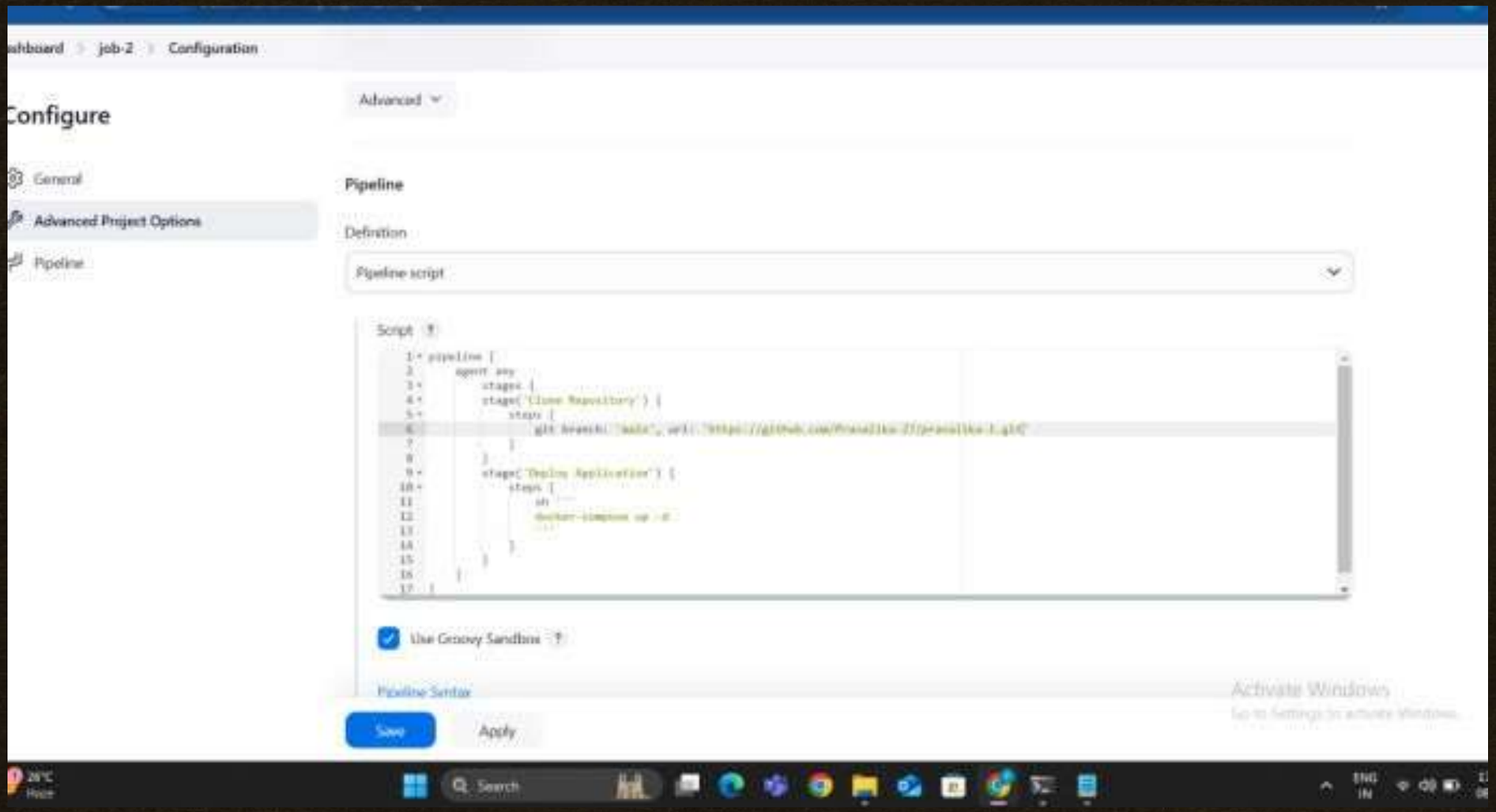
Git clone url
path of the file
docker-compose up -d

Step 4: Build the Jenkins Job and access the application

Method-6 Jenkins pipeline

1. Setup jenkins
2. install required plugins
3. create a pipeline job write pipeline and add build periodically
4. next create another job and write pipeline add by using poll scm
5. Access the application using public id

pipeline


ashboard > job-2 > Configuration
Configure
General
Advanced Project Options
Pipeline
Advanced
Pipeline
Definition
Pipeline script
Script
pipeline {
agent any
stages {
stage('Clone Repository') {
steps {
}
}
}
stage('Deploy Application') {
steps {
sh '''
docker-compose up -d
'''
}
}
}
}
Use Groovy Sandbox
Pipeline Syntax
Save
Apply
Activate Windows

Method-7:Using Terraform

1. Create the ec2 instance install terraform in terminal and configure aws

2. Create ec2.tf and data.sh files

3. Run terraform commands a.terraform init b.terraform plan c.terraform apply

4. Access the application using newly created instance ip address

```
provider "aws" {
  region = "us-east-1"
}

resource "aws_instance" "my_instance" {
  ami                    ="ami-0166fe664262f664c"
  instance_type           = "t2.micro"
  count                   = 1
  key_name                = "pinku"
  associate_public_ip_address = true
  user_data               = file("data.sh")
  subnet_id = "subnet-04e7d31710d9a5178"
  tags = {
    Name = "My public Instance 1"
  }
}

resource "aws_security_group" "demosg" {
  name        = "new-sg"
  description = "Security group for WordPress and MySQL"

  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress{
    from_port=443
    to_port=443
    protocol="tcp"
    cidr_blocks=["0.0.0.0/0"]
  }
```

Method-8:Using Git, Jenkins, Terraform, and AWS

1. install terraform and setup jenkins
2. download plugins and add aws credentials
3. create one free style job add git url
4. add aws credentials -build environment-secret text-aws credentials
5. add terraform commands in execute shell
6. build job and access the application

Method-9:git ,jenkins pipeline and terraform

1. setup jenkins and install terraform

2. install plugins-aws credentials and build pipeline

3. create job-1-add git url and build the job

4. create deploy-job take add job-1as reference near build other jobs add job-1add aws credentials in execute shell add terraform installation commands and path of the ffile and build the job

5. Go to job-1near post build actions add job-1add build periodically once and pollscm next and build the job

6. now near view build add name to and build pipeline there select which is initial job then apply it

7. access the application using newly created public ip

Method-10:Kubernetes manifest file

Step 1– Kubectl Installation

Step 2 – Kops Installation

Step 3: Conffigure AWS CLI

Step 4: Create an S3 Bucket for Kops State Store

Step 5: Create the Kubernetes Cluster Using

Step 6: Write Kubernetes YAML Manifest File for WordPress

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress
  labels:
    app: wordpress
spec:
  replicas: 1
  selector:
    matchLabels:
      app: wordpress
  template:
    metadata:
      labels:
        app: wordpress
    spec:
      containers:
      - name: wordpress
        image: pravalika27/wordpress-docker-compose:pinku
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: wordpress-service
  labels:
    app: wordpress
spec:
  type: NodePort
  ports:
  - port: 80
    targetPort: 80
    nodePort: 30022
  selector:
    app: wordpress

~
~
"wordpress-deployment-service.yaml" 38L, 598B
```


4,3

Step 7: Deploy WordPress Using kubectl

```
kubectl apply -f deployment.yaml
```

Step 7: Deploy WordPress Using kubectl

Step 8: Access the WordPress Application Using NodePort


Thank
You