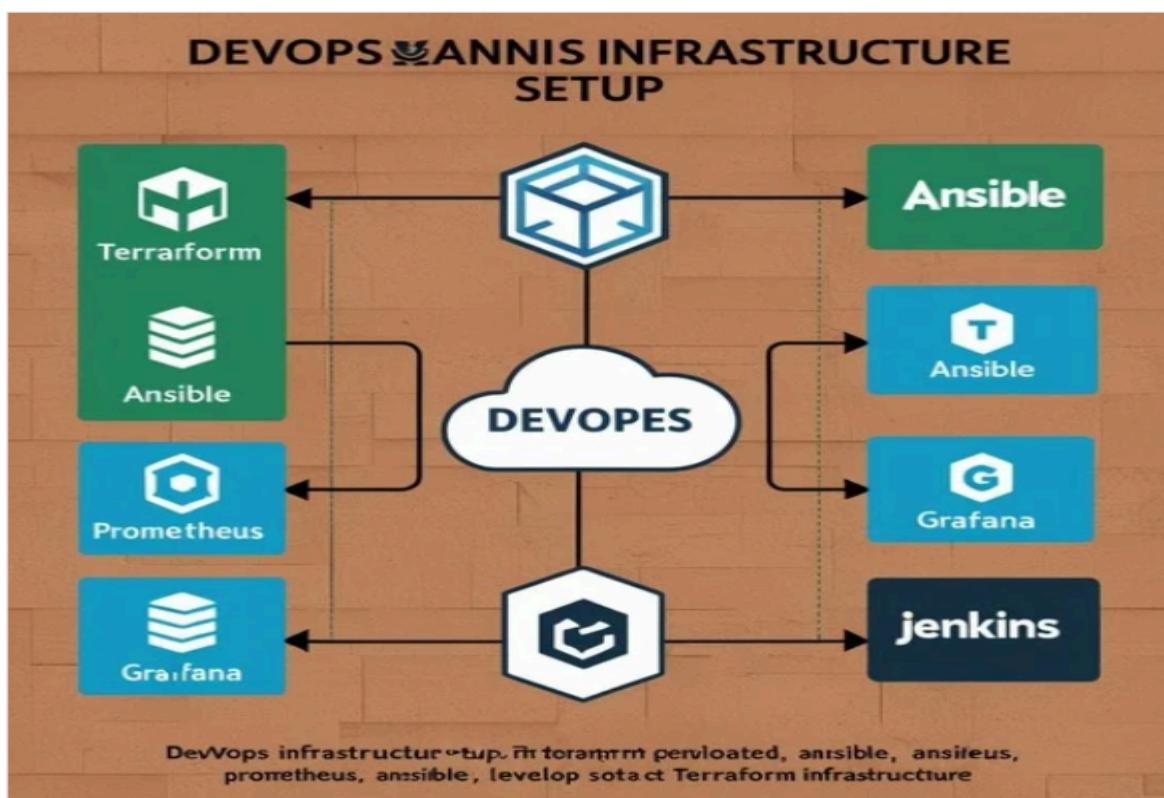


cprime

DevOps Infrastructure Setup with Terraform, Ansible, Prometheus, Grafana and Jenkins



NAME : T MANOJ

EMPID : LYAKE2KHS

Introduction

Modern software development requires robust, scalable, and automated infrastructure to support continuous integration, deployment, and monitoring. This document outlines a comprehensive DevOps infrastructure built on AWS, leveraging industry-standard tools such as Terraform, Ansible, Prometheus, Grafana, and Jenkins.

The architecture presented here enables teams to automate infrastructure provisioning, configure servers consistently, implement continuous integration/continuous deployment (CI/CD) pipelines, and monitor system performance in real-time. This integrated approach helps organizations achieve faster delivery cycles, improved reliability, and better visibility into their applications and infrastructure.

Infrastructure Components

Terraform

Description: Terraform is an Infrastructure as Code (IaC) tool that allows developers to define and provision data center infrastructure using a declarative configuration language.

Role in this setup: Terraform handles the automated provisioning of AWS resources, including EC2 instances, security groups, and networking components. It ensures infrastructure can be version-controlled, reused, and consistently deployed across environments.

Key features utilized:

- AWS provider integration for resource management
- Provisioner functionality to transfer files and execute remote commands
- Output values for convenient access to infrastructure information

Ansible

Description: Ansible is an open-source automation tool that simplifies configuration management, application deployment, and task automation.

Role in this setup: Ansible serves as the configuration management system, ensuring all servers have the correct software installed and configured according to specifications. It eliminates manual server setup, reducing human error and ensuring consistency.

Key features utilized:

- Agentless architecture requiring only SSH access
- Playbooks for defining server configurations
- Inventory management to organize and control target hosts

Prometheus

Description: Prometheus is an open-source systems monitoring and alerting toolkit originally built at SoundCloud.

Role in this setup: Prometheus collects and stores metrics from configured targets, providing a powerful data source for monitoring system health, performance, and behavior over time.

Key features utilized:

- Pull-based metrics collection model
- Flexible query language (PromQL) for data analysis
- Integration with Node Exporter for system metrics

Node Exporter

Description: Node Exporter is a Prometheus exporter that collects a wide variety of hardware and kernel-related metrics.

Role in this setup: Installed on each slave instance, Node Exporter exposes system-level metrics that Prometheus scrapes, enabling visibility into server health and resource utilization.

Key features utilized:

- Comprehensive system metrics collection
- Low overhead monitoring
- Standardized Prometheus integration

Grafana

Description: Grafana is an open-source visualization and analytics platform that allows users to query, visualize, and alert on metrics data.

Role in this setup: Grafana provides customizable dashboards that visualize the metrics collected by Prometheus, offering intuitive insights into system performance and application behavior.

Key features utilized:

- Rich visualization options
- Prometheus data source integration
- Customizable dashboards for different monitoring needs

Jenkins

Description: Jenkins is an open-source automation server that enables developers to build, test, and deploy their software reliably.

Role in this setup: Jenkins manages the CI/CD pipeline, automatically building, testing, and deploying code changes as they are committed to version control.

Key features utilized:

- Multibranch pipeline support for feature branch workflows
- Declarative pipeline syntax for defining build processes
- Integration with popular development tools and platforms

Architecture in Detail

The infrastructure consists of three Amazon EC2 instances:

1. Ansible Control Server:

- Hosts Ansible for configuration management
- Runs Prometheus for metrics collection
- Provides Grafana for metrics visualization
- Acts as the central management node

2. Slave Instance 1:

- Runs Node Exporter for system metrics
- Hosts Jenkins for CI/CD functionality
- Serves as a managed application server

3. Slave Instance 2:

- Runs Node Exporter for system metrics
- Serves as a managed application server

All instances operate within a shared security group that allows necessary communication between components while restricting access from the outside world. The architecture enables centralized management, monitoring, and deployment across the infrastructure.

Workflow Integration

The components work together in a seamless workflow:

1. Infrastructure Provisioning:

- Terraform scripts create AWS resources with appropriate configuration
- Security groups, instances, and networking are established

2. Configuration Management:

- Ansible configures all instances with required software
- Node Exporter, Prometheus, Grafana, and Jenkins are installed and configured

3. Monitoring Setup:

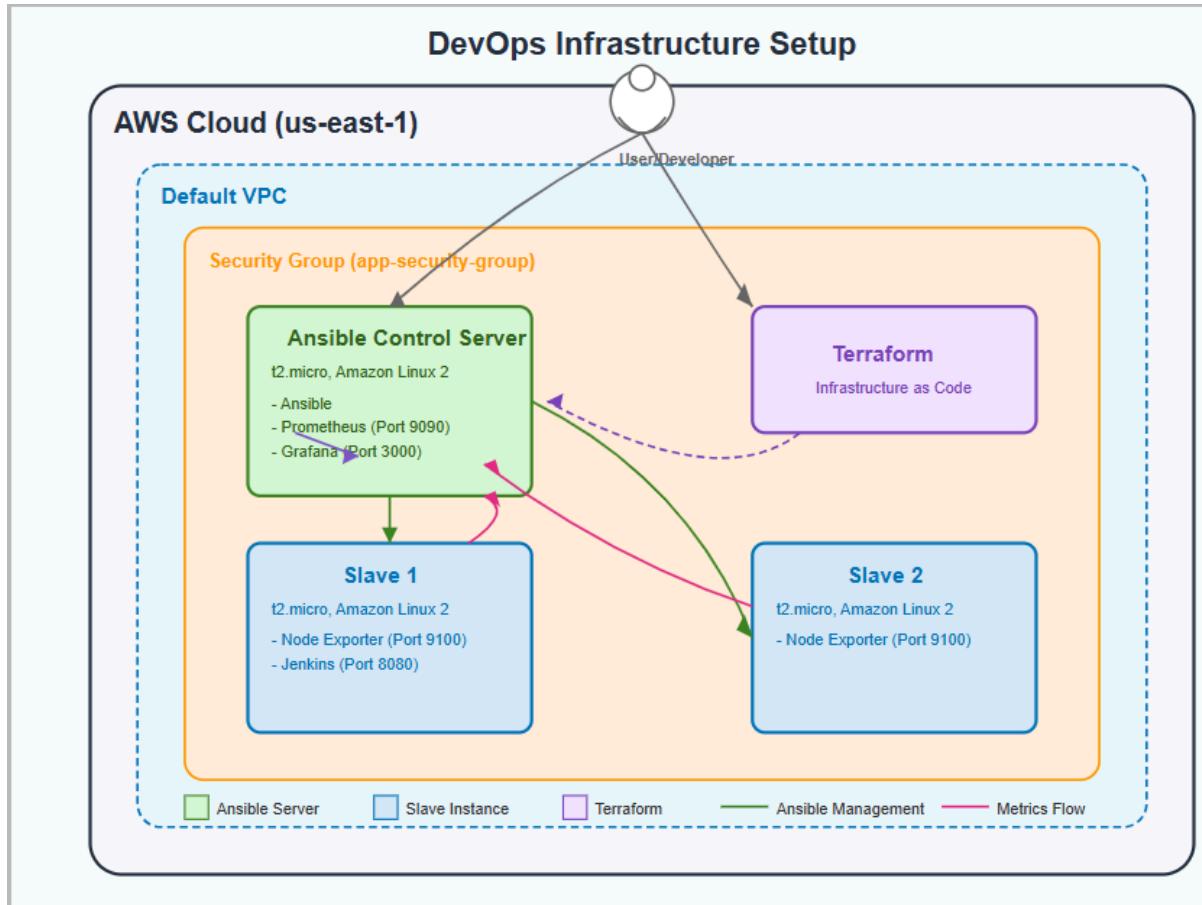
- Node Exporter exposes metrics on each slave
- Prometheus collects these metrics at regular intervals
- Grafana visualizes the collected data

4. CI/CD Implementation:

- Jenkins pipelines define build, test, and deployment stages

- Code changes trigger automated workflows
- Applications are deployed to the slave instances

Architecture Overview



The infrastructure consists of:

1. Three EC2 instances on AWS:
 - 1 Ansible control server (which also hosts Prometheus and Grafana)
 - 2 Slave instances (monitored by Node Exporter)
2. Key components:
 - Terraform: Used for infrastructure provisioning
 - Ansible: For configuration management
 - Prometheus: For metrics collection and monitoring
 - Grafana: For metrics visualization
 - Jenkins: For CI/CD pipelines
3. Network setup:
 - Uses AWS default VPC
 - A custom security group allowing access to ports:
 - SSH (22)
 - HTTP (80)

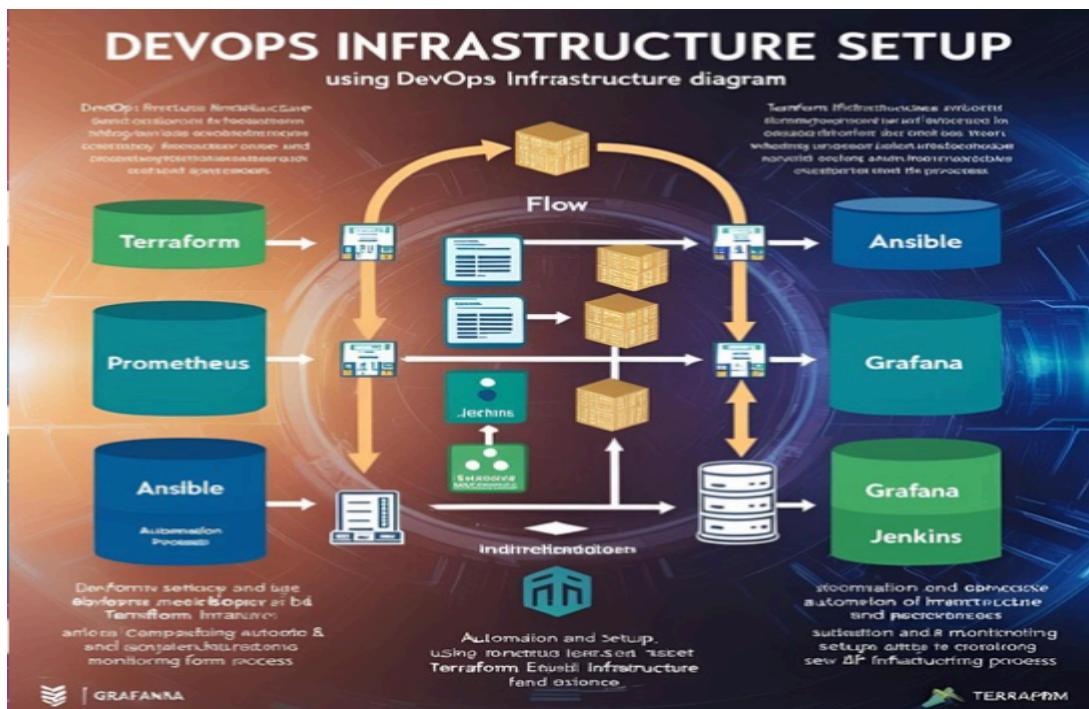
- Application port (4080)
- Node Exporter (9100)
- Prometheus (9090)
- Grafana (3000)

4. Communication flow:

- Ansible server manages the slave nodes
- Node Exporter runs on all slaves to collect metrics
- Prometheus scrapes metrics from Node Exporter endpoints
- Grafana connects to Prometheus for visualization
- Jenkins is set up with multibranch pipeline support

Prerequisites

- AWS Account
- Terraform installed locally
- SSH keypair named "ansible.pem"



1. Terraform Infrastructure Deployment

AWS Provider Configuration

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

Explanation: This block configures Terraform to use AWS as the provider and sets the region to US East (N. Virginia). All resources will be created in this region.

VPC Configuration

```
# Fetch the default VPC
data "aws_vpc" "default" {
  default = true
}
```

Explanation: This data source retrieves the default VPC in your AWS account. Instead of creating a new VPC, we're using the existing default one for simplicity.

Security Group Configuration

```
# Create security group for our instances
resource "aws_security_group" "app_security_group" {
  name          = "app-security-group"
  description   = "Security group for servers with Node Exporter and Prometheus"
  vpc_id        = data.aws_vpc.default.id

  # SSH access from anywhere
  ingress {
    from_port  = 22
    to_port    = 22
    protocol   = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  # HTTP access
  ingress {
    from_port  = 80
    to_port    = 80
    protocol   = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  # Application port access
  ingress {
    from_port  = 4080
    to_port    = 4080
    protocol   = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

```

}

# Node Exporter port
ingress {
    from_port    = 9100
    to_port      = 9100
    protocol     = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
}

# Prometheus port
ingress {
    from_port    = 9090
    to_port      = 9090
    protocol     = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
}

# Grafana port
ingress {
    from_port    = 3000
    to_port      = 3000
    protocol     = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
}

# Allow all outbound traffic
egress {
    from_port    = 0
    to_port      = 0
    protocol     = "-1"
    cidr_blocks = ["0.0.0.0/0"]
}

tags = {
    Name = "AppSecurityGroup"
}
}

```

Explanation:

- This creates a security group that controls network access to our EC2 instances.
- The security group allows:
 - SSH access (port 22) for remote administration
 - HTTP access (port 80) for web services
 - Custom application port (4080)

- Node Exporter port (9100) for metrics collection
- Prometheus port (9090) for metrics storage and querying
- Grafana port (3000) for visualization
- The `egress` block allows all outbound traffic from our instances.
- Using `0.0.0.0/0` allows access from any IP address, which is convenient for testing but should be restricted in production.

EC2 Instance Configuration for Slave Nodes

```
# EC2 instance for Slave1
resource "aws_instance" "slave1" {
  ami                  = "ami-0c02fb55956c7d316"  # Amazon Linux 2 AMI
  instance_type        = "t2.micro"
  key_name             = "ansible"  # Replace with your key pair
  vpc_security_group_ids = [aws_security_group.app_security_group.id]

  tags = {
    Name = "SLAVE1"
  }
}

# EC2 instance for Slave2
resource "aws_instance" "slave2" {
  ami                  = "ami-0c02fb55956c7d316"  # Amazon Linux 2 AMI
  instance_type        = "t2.micro"
  key_name             = "ansible"  # Replace with your key pair
  vpc_security_group_ids = [aws_security_group.app_security_group.id]

  tags = {
    Name = "SLAVE2"
  }
}
```

Explanation:

- These blocks create two EC2 instances that will serve as our slave nodes.
- `ami-0c02fb55956c7d316` is the ID for Amazon Linux 2 in the us-east-1 region.
- `t2.micro` is a free-tier eligible instance type suitable for lightweight workloads.
- `key_name` specifies the SSH key pair for accessing the instances.
- `vpc_security_group_ids` attaches our previously created security group.
- The `tags` block assigns a readable name to each instance for identification in the AWS console.

Ansible Server Configuration

```

# EC2 instance for Ansible Server
resource "aws_instance" "ansible_server" {
    ami                  = "ami-0c02fb55956c7d316"  # Amazon Linux 2 AMI
    instance_type        = "t2.micro"
    key_name             = "ansible"
    vpc_security_group_ids = [aws_security_group.app_security_group.id]

    depends_on = [aws_instance.slave1, aws_instance.slave2]

```

Explanation:

- This block creates the EC2 instance that will serve as our Ansible control server.
- `depends_on` ensures this instance is created after the slave instances, which is important for the provisioning steps that follow.
- The configuration is similar to the slave instances, using the same AMI and instance type.

Transferring SSH Key File

```

# Upload the private key file to Ansible server
provisioner "file" {
    source      = "ansible.pem"  # Your local key file
    destination = "/home/ec2-user/ansible.pem"

    connection {
        type      = "ssh"
        user      = "ec2-user"
        private_key = file("ansible.pem")
        host      = self.public_ip
    }
}

```

Explanation:

- This `file` provisioner uploads the SSH private key to the Ansible server.
- The key is necessary for Ansible to connect to the slave nodes without password authentication.
- `source` is the local path to your key file.
- `destination` is where the key will be placed on the Ansible server.
- The `connection` block specifies how Terraform should connect to the EC2 instance:
 - Using SSH protocol
 - As the `ec2-user` user
 - Using the local private key
 - Connecting to the instance's public IP

Creating Ansible Playbook

```
# Create the playbook file for Node Exporter deployment
provisioner "file" {
    content      = <<-EOF
---
- hosts: all
  user: ec2-user
  become: yes
  tasks:
    - name: Install required system packages
      yum:
        name:
          - python3
          - python3-pip
          - git
        state: present

    - name: Create installation directory
      file:
        path: /opt/node_exporter
        state: directory
        mode: 0755

    - name: Download node_exporter
      get_url:
        url:
          "https://github.com/prometheus/node_exporter/releases/download/v1.9.1/node_exporter-1.9.1.linux-amd64.tar.gz"
        dest: "/tmp/node_exporter-1.9.1.linux-amd64.tar.gz"
        mode: 0644

    - name: Extract node_exporter
      unarchive:
        src: "/tmp/node_exporter-1.9.1.linux-amd64.tar.gz"
        dest: "/opt/node_exporter"
        remote_src: yes
        extra_opts: [--strip-components=1]

    - name: Start node_exporter in background
      shell: "nohup /opt/node_exporter/node_exporter > /opt/node_exporter/node_exporter.log 2>&1 &"
      args:
        creates: "/opt/node_exporter/node_exporter.log"

    - name: Clean up temporary files
```

```

file:
  path: "/tmp/node_exporter-1.9.1.linux-amd64.tar.gz"
  state: absent
EOF
destination = "/home/ec2-user/deploy.yml"

connection {
  type      = "ssh"
  user      = "ec2-user"
  private_key = file("ansible.pem")
  host      = self.public_ip
}
}

```

Explanation:

- This `file` provisioner creates an Ansible playbook file on the control server.
- The playbook contains tasks to install Node Exporter on the slave instances:
 - Installing required dependencies (Python, pip, git)
 - Creating installation directory
 - Downloading Node Exporter (version 1.9.1)
 - Extracting the downloaded archive
 - Starting Node Exporter as a background process
 - Cleaning up temporary files
- The `<<-EOF ... EOF` syntax is a heredoc, allowing multi-line content in Terraform.
- The playbook will be saved as `/home/ec2-user/deploy.yml` on the Ansible server.

Server Configuration and Software Installation

```

# Install and configure Ansible, Node Exporter, Prometheus and Grafana
provisioner "remote-exec" {
  inline = [
    # Install Ansible
    "sudo su -",
    "sudo amazon-linux-extras install ansible2 -y",
    "sudo yum install -y git",
    "chmod 400 /home/ec2-user/ansible.pem",
  ]
}

```

Explanation:

- The `remote-exec` provisioner runs commands directly on the EC2 instance after it's created.
- This section installs Ansible and sets proper permissions on the SSH key:

- `sudo su` - escalates to root privileges
- `amazon-linux-extras install ansible2 -y` installs Ansible from Amazon's repo
- `yum install -y git` installs Git
- `chmod 400` sets the correct permissions on the SSH key (read-only by owner)

Creating Ansible Inventory

```
# Create inventory file
"echo '[nodes]' > /home/ec2-user/hosts",
"echo '${aws_instance.slave1.private_ip} ansible_user=ec2-user
ansible_ssh_private_key_file=/home/ec2-user/ansible.pem
ansible_ssh_common_args=\"-o StrictHostKeyChecking=no\"' >>
/home/ec2-user/hosts",
"echo '${aws_instance.slave2.private_ip} ansible_user=ec2-user
ansible_ssh_private_key_file=/home/ec2-user/ansible.pem
ansible_ssh_common_args=\"-o StrictHostKeyChecking=no\"' >>
/home/ec2-user/hosts",
```

Explanation:

- These commands create an Ansible inventory file with the slave nodes:
 - First command creates the file with a `[nodes]` group header
 - Next commands add entries for each slave using their private IPs
 - Each entry includes connection parameters:
 - `ansible_user=ec2-user`: The user to connect as
 - `ansible_ssh_private_key_file`: Path to the SSH key
 - `ansible_ssh_common_args`: Additional SSH options (skipping host key verification)
- The `${aws_instance.slave1.private_ip}` syntax dynamically inserts the IP addresses of the instances created earlier.

Running Ansible Playbook

```
# Test connection
"ansible -i /home/ec2-user/hosts all -m ping",

# Run the playbook to deploy Node Exporter
"ansible-playbook -i /home/ec2-user/hosts
/home/ec2-user/deploy.yml",
```

Explanation:

- First, a simple ping test verifies connectivity to all nodes in the inventory.
- Then, the playbook is executed to deploy Node Exporter on all target hosts.
- `-i /home/ec2-user/hosts` specifies the inventory file to use.

Installing Prometheus and Grafana

```
# Install Prometheus
    "wget
https://github.com/prometheus/prometheus/releases/download/v2.44.0/prometheus-2.44.0.linux-amd64.tar.gz",
    "tar -xvzf prometheus-2.44.0.linux-amd64.tar.gz",
    "wget
https://dl.grafana.com/oss/release/grafana-11.6.0.linux-amd64.tar.gz",
    "tar -zxvf grafana-11.6.0.linux-amd64.tar.gz",

    "cd prometheus-2.44.0.linux-amd64",
    "sudo cp prometheus promtool /usr/local/bin/",
```

Explanation:

- These commands download and extract Prometheus and Grafana:
 - `wget` downloads the software packages directly from their official sources
 - `tar -xvzf` extracts the compressed archives
 - The Prometheus binaries are copied to `/usr/local/bin/` for system-wide access

Configuring Prometheus

```
# Create Prometheus config
"echo 'global:' > prometheus.yml",
"echo '  scrape_interval: 15s' >> prometheus.yml",
"echo '  evaluation_interval: 15s' >> prometheus.yml",
"echo 'scrape_configs:' >> prometheus.yml",
"echo '  - job_name: \"prometheus\"' >> prometheus.yml",
"echo '    static_configs:' >> prometheus.yml",
"echo '      - targets: [\"localhost:9090\"]' >> prometheus.yml",
"echo '  - job_name: \"node_exporter\"' >> prometheus.yml",
"echo '    static_configs:' >> prometheus.yml",
"echo '      - targets:
[\"${aws_instance.slave1.private_ip}:9100\",
\"${aws_instance.slave2.private_ip}:9100\"]' >> prometheus.yml",
```

Explanation:

- These commands create the Prometheus configuration file line by line:
 - `scrape_interval: 15s`: How often Prometheus collects metrics
 - `evaluation_interval: 15s`: How often rules are evaluated
 - First job monitors Prometheus itself (localhost:9090)
 - Second job monitors Node Exporter on both slave instances (port 9100)
- The slave IP addresses are dynamically inserted from the Terraform state.

Starting Prometheus and Grafana

```
# Start Prometheus
"nohup prometheus --config.file=prometheus.yml > prometheus.log
2>&1 &",
"cd /home/ec2-user/grafana-11.6.0/",
"nohup ./bin/grafana-server > grafana.log 2>&1 &,"
```

Explanation:

- `nohup` is used to start both services as background processes that persist after the SSH session ends.
- `> prometheus.log 2>&1 &` redirects both standard output and errors to a log file.
- Prometheus is started with our custom configuration file.
- Grafana is started from its installation directory.

Final Output Messages

```
# Print completion message
"echo 'Setup complete!'",
"echo 'Prometheus URL: http://${self.public_ip}:9090'",
"echo 'Grafana URL: http://${self.public_ip}:3000'"
]

connection {
    type      = "ssh"
    user      = "ec2-user"
    private_key = file("ansible.pem")
    host      = self.public_ip
}
}

tags = {
    Name = "AnsibleServer"
}
}
```

Explanation:

- These echo commands print confirmation messages with URLs for accessing the services.
- `${self.public_ip}` dynamically inserts the public IP of the Ansible server.
- The connection block specifies how to connect to run these commands.
- The final tags block assigns a name to the instance.

Output Values

```
# Output the public IPs and URLs for easy access
output "slave1_public_ip" {
    value = aws_instance.slave1.public_ip
}

output "slave2_public_ip" {
    value = aws_instance.slave2.public_ip
}

output "ansible_server_public_ip" {
    value = aws_instance.ansible_server.public_ip
}

output "prometheus_url" {
    value = "http://${aws_instance.ansible_server.public_ip}:9090"
}

output "grafana_url" {
    value = "http://${aws_instance.ansible_server.public_ip}:3000"
}
```

Explanation:

- These output blocks define values that Terraform will display after successful deployment.
- They provide convenient access to:
 - Public IP addresses of all instances
 - URLs for accessing Prometheus and Grafana web interfaces
- These values can be used for manual access or in other automation scripts.

Terraform init

```

manoj@IND-140:~/new$ terraform init
Initializing the backend...
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.94.1

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
manoj@IND-140:~/new$ |

```

Terraform plan

```

manoj@IND-140:~/new$ terraform plan
data.aws_vpc.default: Reading...
data.aws_vpc.default: Read complete after 2s [id=vpc-0c33a120e93d325f9]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the
following symbols:
+ create

Terraform will perform the following actions:

# aws_instance.ansible_server will be created
+ resource "aws_instance" "ansible_server" {
    ami                               = "ami-0c02fb55956c7d316"
    ami                                = (known after apply)
    arn                                = (known after apply)
    associate_public_ip_address        = (known after apply)
    availability_zone                  = (known after apply)
    cpu_core_count                     = (known after apply)
    cpu_threads_per_core              = (known after apply)
    disable_api_stop                  = (known after apply)
    disable_api_termination           = (known after apply)
    ebs_optimized                      = (known after apply)
    enable_primary_ipv6                = (known after apply)
    get_password_data                 = false
    host_id                            = (known after apply)
    host_resource_group_arn            = (known after apply)
    iam_instance_profile               = (known after apply)
    id                                 = (known after apply)
    instance_initiated_shutdown_behavior = (known after apply)
    instance_lifecycle                = (known after apply)
    instance_state                     = (known after apply)
    instance_type                      = "t2.micro"

```

```

+ outpost_arn                         = (known after apply)
+ password_data                        = (known after apply)
+ placement_group                      = (known after apply)
+ placement_partition_number           = (known after apply)
+ primary_network_interface_id         = (known after apply)
+ private_dns                          = (known after apply)
+ private_ip                           = (known after apply)
+ public_dns                           = (known after apply)
+ public_ip                            = (known after apply)
+ secondary_private_ips                = (known after apply)
+ security_groups                      = (known after apply)
+ source_dest_check                   = true
+ spot_instance_request_id             = (known after apply)
+ subnet_id                            = (known after apply)
+ tags                                = {
    + "Name" = "AnsibleServer"
}
+ tags_all                            = {
    + "Name" = "AnsibleServer"
}
+ tenancy                             = (known after apply)
+ user_data                           = (known after apply)
+ user_data_base64                     = (known after apply)
+ user_data_replace_on_change          = false
+ vpc_security_group_ids               = (known after apply)

+ capacity_reservation_specification (known after apply)
+ cpu_options (known after apply)
+ ebs_block_device (known after apply)

```

```
manoj@IND-140: ~/new      +  -  x
+ ami                      = "ami-0c02fb55956c7d316"
+ arn                      = (known after apply)
+ associate_public_ip_address = (known after apply)
+ availability_zone        = (known after apply)
+ cpu_core_count           = (known after apply)
+ cpu_threads_per_core     = (known after apply)
+ disable_api_stop         = (known after apply)
+ disable_api_termination   = (known after apply)
+ ebs_optimized             = (known after apply)
+ enable_primary_ipv6       = (known after apply)
+ get_password_data        = false
+ host_id                  = (known after apply)
+ host_resource_group_arn   = (known after apply)
+ iam_instance_profile      = (known after apply)
+ id                       = (known after apply)
+ instance_initiated_shutdown_behavior = (known after apply)
+ instance_lifecycle        = (known after apply)
+ instance_state            = (known after apply)
+ instance_type             = "t2.micro"
+ ipv6_address_count        = (known after apply)
+ ipv6_addresses            = (known after apply)
+ key_name                 = "ansible"
+ monitoring               = (known after apply)
+ outpost_arn               = (known after apply)
+ password_data             = (known after apply)
+ placement_group           = (known after apply)
+ placement_partition_number = (known after apply)
+ primary_network_interface_id = (known after apply)
+ private_dns               = (known after apply)
+ private_ip                = (known after apply)
+ public_dns                = (known after apply)
```

```
manoj@IND-140: ~/new      +  -  x
+   "Name" = "SLAVE2"
}
+ tenancy                  = (known after apply)
+ user_data                = (known after apply)
+ user_data_base64          = (known after apply)
+ user_data_replace_on_change = false
+ vpc_security_group_ids    = (known after apply)

+ capacity_reservation_specification (known after apply)
+ cpu_options (known after apply)
+ ebs_block_device (known after apply)
+ enclave_options (known after apply)
+ ephemeral_block_device (known after apply)
+ instance_market_options (known after apply)
+ maintenance_options (known after apply)
+ metadata_options (known after apply)
+ network_interface (known after apply)
+ private_dns_name_options (known after apply)
+ root_block_device (known after apply)
}
```

```
manoj@IND-140: ~/new      +  -  x
+ revoke_rules_on_delete = false
+ tags                   = {
+   "Name" = "AppSecurityGroup"
}
+ tags_all                = {
+   "Name" = "AppSecurityGroup"
}
+ vpc_id                  = "vpc-0c33a120e93d325f9"
}

Plan: 4 to add, 0 to change, 0 to destroy.

Changes to Outputs:
+ ansible_server_public_ip = (known after apply)
+ grafana_url              = (known after apply)
+ prometheus_url            = (known after apply)
+ slave1_public_ip          = (known after apply)
+ slave2_public_ip          = (known after apply)

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply" now.
```

Terraform apply

```
+ slave2_public_ip      = (known after apply)

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

Enter a value: yes
```

```
manoj@IND-140:~/new  x  +  ~
aws_instance.ansible_server (remote-exec): perl-Error.noarch 1:0.17020-2.amzn2
aws_instance.ansible_server (remote-exec): perl-Git.noarch 0:2.47.1-1.amzn2.0.2
aws_instance.ansible_server (remote-exec): perl-TermReadKey.x86_64 0:2.30-20.amzn2.0.2

aws_instance.ansible_server (remote-exec): Complete!
aws_instance.ansible_server (remote-exec): [WARNING]: Platform linux on host 172.31.31.233 is using the discovered Python
aws_instance.ansible_server (remote-exec): interpreter at /usr/bin/python, but future installation of another Python
aws_instance.ansible_server (remote-exec): interpreter could change this. See https://docs.ansible.com/ansible/2.9/referen
aws_instance.ansible_server (remote-exec): ce_appendices/interpreter_discovery.html for more information.
aws_instance.ansible_server (remote-exec): 172.31.31.233 | SUCCESS => {
aws_instance.ansible_server (remote-exec):   "ansible_facts": {
aws_instance.ansible_server (remote-exec):     "discovered_interpreter_python": "/usr/bin/python"
aws_instance.ansible_server (remote-exec):   },
aws_instance.ansible_server (remote-exec):   "changed": false,
aws_instance.ansible_server (remote-exec):   "ping": "pong"
aws_instance.ansible_server (remote-exec): }
aws_instance.ansible_server (remote-exec): [WARNING]: Platform linux on host 172.31.17.121 is using the discovered Python
aws_instance.ansible_server (remote-exec): interpreter at /usr/bin/python, but future installation of another Python
aws_instance.ansible_server (remote-exec): interpreter could change this. See https://docs.ansible.com/ansible/2.9/referen
aws_instance.ansible_server (remote-exec): ce_appendices/interpreter_discovery.html for more information.
aws_instance.ansible_server (remote-exec): 172.31.17.121 | SUCCESS => {
aws_instance.ansible_server (remote-exec):   "ansible_facts": {
aws_instance.ansible_server (remote-exec):     "discovered_interpreter_python": "/usr/bin/python"
aws_instance.ansible_server (remote-exec):   },
aws_instance.ansible_server (remote-exec):   "changed": false,
aws_instance.ansible_server (remote-exec):   "ping": "pong"
aws_instance.ansible_server (remote-exec): }

aws_instance.ansible_server (remote-exec): PLAY [all] ****
aws_instance.ansible_server (remote-exec): TASK [Gathering Facts] *****
```

```
manoj@IND-140:~/new  x  +  ~
aws_instance.ansible_server (remote-exec): ● grafana.service - Grafana instance
aws_instance.ansible_server (remote-exec):   Loaded: loaded (/etc/systemd/system/grafana.service; enabled; vendor preset: disabled)
aws_instance.ansible_server (remote-exec):     Active: activating (auto-restart) (Result: exit-code) since Mon 2025-04-07 06:40:42 UTC; 23ms ago
aws_instance.ansible_server (remote-exec):       Process: 7980 ExecStart=/home/ec2-user/grafana-11.6.0/bin/grafana-server --confi
aws_instance.ansible_server (remote-exec): Main PID: 7980 (code=exited, status=200/CHDIR)

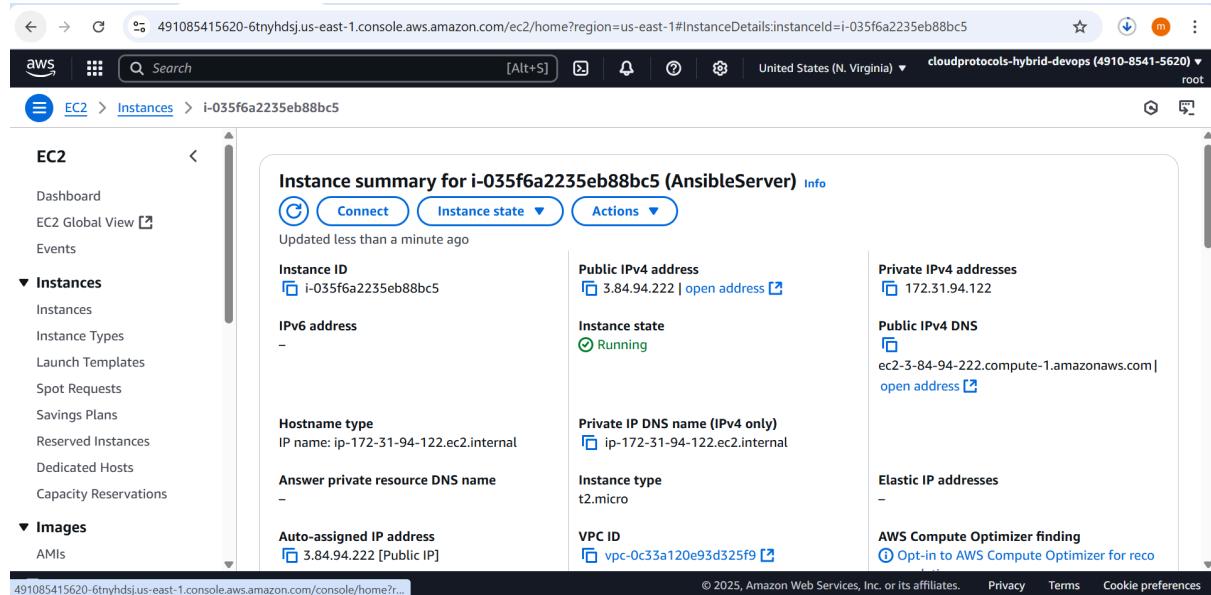
aws_instance.ansible_server (remote-exec): Apr 07 06:40:42 ip-172-31-90-231.ec2.internal systemd[1]: ...
aws_instance.ansible_server (remote-exec): Apr 07 06:40:42 ip-172-31-90-231.ec2.internal systemd[1]: ...
aws_instance.ansible_server (remote-exec): Apr 07 06:40:42 ip-172-31-90-231.ec2.internal systemd[1]: ...
aws_instance.ansible_server (remote-exec): Hint: Some lines were ellipsized, use -l to show in full.
aws_instance.ansible_server (remote-exec): Setup complete!
aws_instance.ansible_server (remote-exec): Prometheus URL: http://3.83.39.181:9090
aws_instance.ansible_server (remote-exec): Grafana URL: http://3.83.39.181:3000 (default login: admin/admin)
aws_instance.ansible_server: Creation complete after 2m19s [id=i-0d06d414ca2bc5cd8]

Apply complete! Resources: 4 added, 0 changed, 0 destroyed.

Outputs:

ansible_server_public_ip = "3.83.39.181"
grafana_url = "http://3.83.39.181:3000"
prometheus_url = "http://3.83.39.181:9090"
slave1_public_ip = "44.202.97.72"
slave2_public_ip = "107.23.195.247"
manoj@IND-140:~/new$ curl 3.83.39.181:9090
<a href="/graph">Found</a>

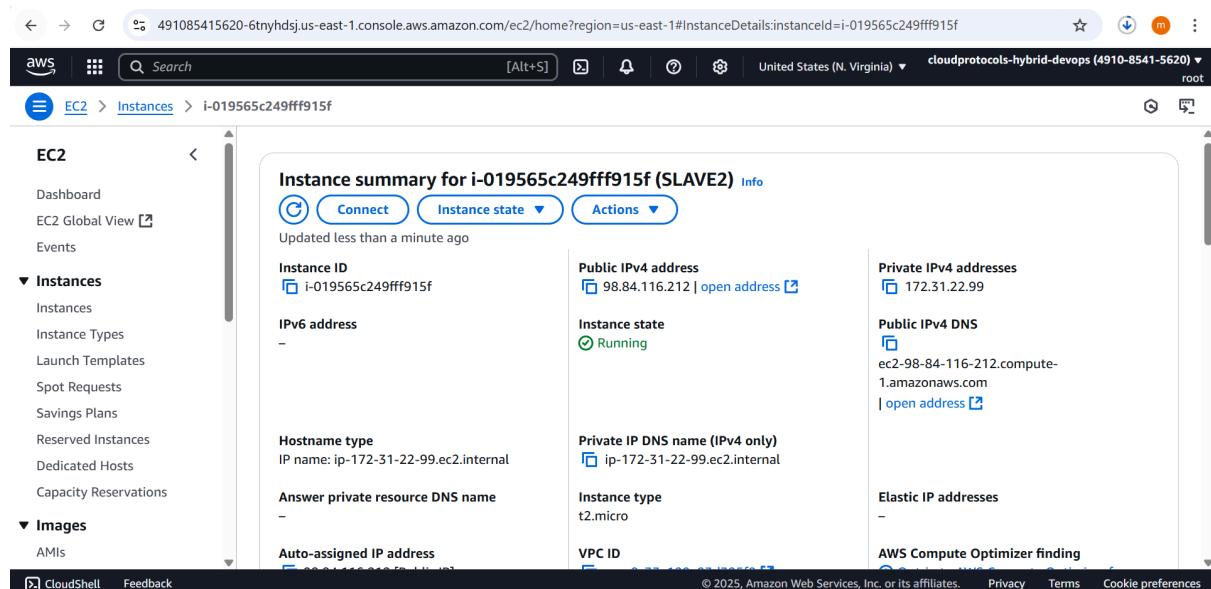
manoj@IND-140:~/new$ curl 3.83.39.181:9090
```



The screenshot shows the AWS EC2 Instances page for an instance named "AnsibleServer". The instance ID is i-035f6a2235eb88bc5. The instance is running and has a public IPv4 address of 3.84.94.222 and a private IPv4 address of 172.31.94.122. It is associated with a VPC ID vpc-0c33a120e93d325f9. The instance type is t2.micro.

Instance summary for i-035f6a2235eb88bc5 (AnsibleServer) Info		
Connect Instance state Actions		
Updated less than a minute ago		
Instance ID i-035f6a2235eb88bc5	Public IPv4 address 3.84.94.222 open address	Private IPv4 addresses 172.31.94.122
IPv6 address -	Instance state Running	Public IPv4 DNS ec2-3-84-94-222.compute-1.amazonaws.com open address
Hostname type IP name: ip-172-31-94-122.ec2.internal	Private IP DNS name (IPv4 only) ip-172-31-94-122.ec2.internal	Elastic IP addresses -
Answer private resource DNS name -	Instance type t2.micro	AWS Compute Optimizer finding Opt-in to AWS Compute Optimizer for reco
Auto-assigned IP address 3.84.94.222 [Public IP]	VPC ID vpc-0c33a120e93d325f9	

ANSIBLE SERVER



The screenshot shows the AWS EC2 Instances page for an instance named "SLAVE2". The instance ID is i-019565c249fff915f. The instance is running and has a public IPv4 address of 98.84.116.212 and a private IPv4 address of 172.31.22.99. It is associated with a VPC ID vpc-0a55a120e93d325f9. The instance type is t2.micro.

Instance summary for i-019565c249fff915f (SLAVE2) Info		
Connect Instance state Actions		
Updated less than a minute ago		
Instance ID i-019565c249fff915f	Public IPv4 address 98.84.116.212 open address	Private IPv4 addresses 172.31.22.99
IPv6 address -	Instance state Running	Public IPv4 DNS ec2-98-84-116-212.compute-1.amazonaws.com open address
Hostname type IP name: ip-172-31-22-99.ec2.internal	Private IP DNS name (IPv4 only) ip-172-31-22-99.ec2.internal	Elastic IP addresses -
Answer private resource DNS name -	Instance type t2.micro	AWS Compute Optimizer finding Opt-in to AWS Compute Optimizer for reco
Auto-assigned IP address 98.84.116.212 [Public IP]	VPC ID vpc-0a55a120e93d325f9	

SLAVESERVER 1

491085415620-6tnyhdjs.us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#InstanceDetails:instanceId=i-0ccf76f0120a072ed

aws Search [Alt+S] United States (N. Virginia) cloudprotocols-hybrid-devops (4910-8541-5620) root

EC2 Instances i-0ccf76f0120a072ed

Instance summary for i-0ccf76f0120a072ed (SLAVE1)

Updated less than a minute ago

Instance ID i-0ccf76f0120a072ed	Public IPv4 address 98.81.79.144 open address	Private IPv4 addresses 172.31.21.9
IPv6 address -	Instance state Running	Public IPv4 DNS ec2-98-81-79-144.compute-1.amazonaws.com open address
Hostname type IP name: ip-172-31-21-9.ec2.internal	Private IP DNS name (IPv4 only) ip-172-31-21-9.ec2.internal	Elastic IP addresses -
Answer private resource DNS name -	Instance type t2.micro	AWS Compute Optimizer finding
Auto-assigned IP address 172.31.21.9	VPC ID vpc-013e1501f01111111111111111111111	

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

SLAVESERVER 2

Not secure 3.84.94.222:9090/targets?search=

Prometheus Alerts Graph Status Help

Targets

All scrape pools ▾ All Unhealthy Collapse All Filter by endpoint or labels Unknown Unhealthy Healthy

node_exporter (2/2 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://172.31.21.9:9100/metrics	UP	instance="172.31.21.9:9100" job="node_exporter"	1.438s ago	11.418ms	
http://172.31.22.99:9100/metrics	UP	instance="172.31.22.99:9100" job="node_exporter"	11.919s ago	11.550ms	

prometheus (1/1 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:9090/metrics	UP	instance="localhost:9090" job="prometheus"	122.000ms ago	4.051ms	

PROMETHEUS DASHBOARD

Node Exporter

Prometheus Node Exporter

Version: (version=1.9.1, branch=HEAD, revision=f2ec547b49af53815038a50265aa2adcd1275959)

- Metrics

Download a detailed report of resource usage (pprof format, from the Go runtime):

- [heap usage \(memory\)](#)
- [CPU usage \(60 second profile\)](#)

To visualize and share profiles you can upload to [pprof.me](#)

NODE EXPORTER 1

Node Exporter

Prometheus Node Exporter

Version: (version=1.9.1, branch=HEAD, revision=f2ec547b49af53815038a50265aa2adcd1275959)

- Metrics

Download a detailed report of resource usage (pprof format, from the Go runtime):

- [heap usage \(memory\)](#)
- [CPU usage \(60 second profile\)](#)

To visualize and share profiles you can upload to [pprof.me](#)

Prometheus Node Exporter

Version: (version=1.9.1, branch=HEAD, revision=f2ec547b49af53815038a50265aa2adcd1275959)

- Metrics

Download a detailed report of resource usage (pprof format, from the Go runtime):

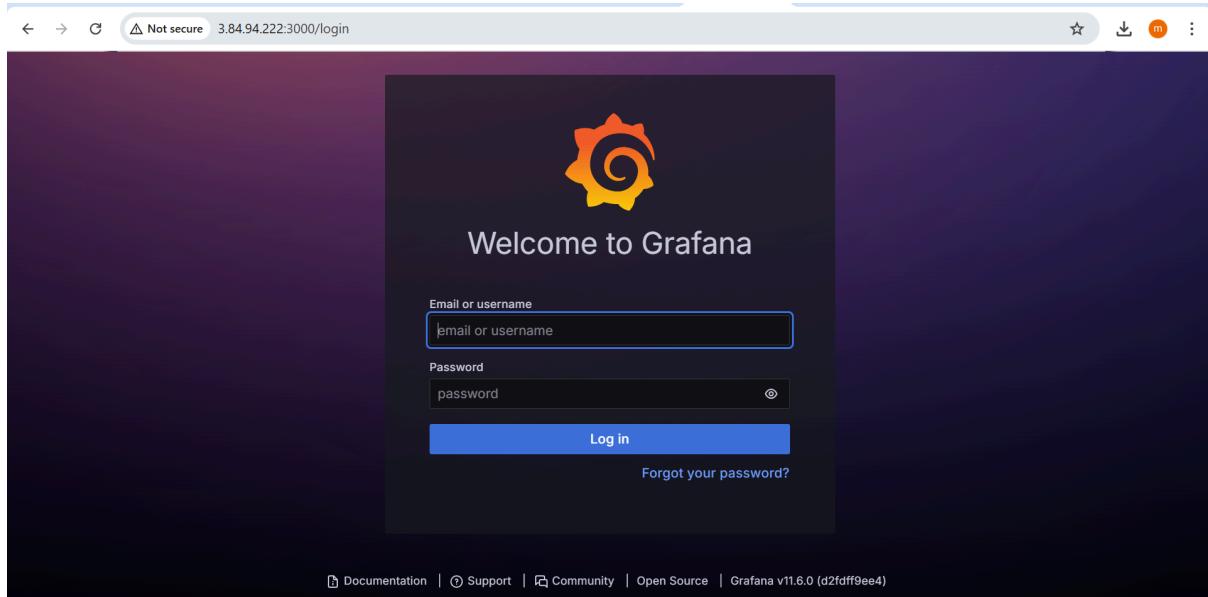
- [heap usage \(memory\)](#)
- [CPU usage \(60 second profile\)](#)

To visualize and share profiles you can upload to [pprof.me](#)

NODE EXPORTER 2

```
[ec2-user@ip-172-31-94-122 ~]$ sudo systemctl status grafana-server
● grafana-server.service - Grafana instance
  Loaded: loaded (/usr/lib/systemd/system/grafana-server.service; enabled; vendor preset: disabled)
  Active: active (running) since Mon 2025-04-07 08:09:46 UTC; 19s ago
    Docs: http://docs.grafana.org
 Main PID: 8916 (grafana)
   CGroup: /system.slice/grafana-server.service
           └─8916 /usr/share/grafana/bin/grafana server --config=/etc/grafana/grafana.ini --pidfile=/var/run/grafana/grafana-server.pid -...
Apr 07 08:09:55 ip-172-31-94-122.ec2.internal systemd[1]: [/usr/lib/systemd/system/grafana-server.service:29] Unknown lvalue 'Prot...rvice'
Apr 07 08:09:55 ip-172-31-94-122.ec2.internal systemd[1]: [/usr/lib/systemd/system/grafana-server.service:31] Unknown lvalue 'Prot...rvice'
Apr 07 08:09:55 ip-172-31-94-122.ec2.internal systemd[1]: [/usr/lib/systemd/system/grafana-server.service:32] Unknown lvalue 'Prot...rvice'
Apr 07 08:09:55 ip-172-31-94-122.ec2.internal systemd[1]: [/usr/lib/systemd/system/grafana-server.service:33] Unknown lvalue 'Prot...rvice'
Apr 07 08:09:55 ip-172-31-94-122.ec2.internal systemd[1]: [/usr/lib/systemd/system/grafana-server.service:34] Unknown lvalue 'Prot...rvice'
Apr 07 08:09:55 ip-172-31-94-122.ec2.internal systemd[1]: [/usr/lib/systemd/system/grafana-server.service:35] Unknown lvalue 'Prot...rvice'
Apr 07 08:09:55 ip-172-31-94-122.ec2.internal systemd[1]: [/usr/lib/systemd/system/grafana-server.service:37] Unknown lvalue 'Remov...rvice'
Apr 07 08:09:55 ip-172-31-94-122.ec2.internal systemd[1]: [/usr/lib/systemd/system/grafana-server.service:39] Unknown lvalue 'Restr...rvice'
Apr 07 08:09:55 ip-172-31-94-122.ec2.internal systemd[1]: [/usr/lib/systemd/system/grafana-server.service:40] Unknown lvalue 'Restr...rvice'
Apr 07 08:09:55 ip-172-31-94-122.ec2.internal systemd[1]: [/usr/lib/systemd/system/grafana-server.service:41] Unknown lvalue 'Restr...rvice'
Hint: Some lines were ellipsized, use -l to show in full.
[ec2-user@ip-172-31-94-122 ~]$
```

VERIFYING THE SERVICE



GRAFANA DASHBOARD

A screenshot of the Grafana dashboard homepage. The top navigation bar shows the URL 3.84.94.222:3000/?orgId=1&from=now-6h&to=now&timezone=browser. The main content area has a dark background with a central panel titled 'Welcome to Grafana'. To the right, there's a 'Need help?' section with links to Documentation, Tutorials, Community, and Public Slack. Below this are three cards: 'Basic' (with a sub-section 'TUTORIAL DATA SOURCE AND DASHBOARDS' containing 'Grafana fundamentals' and a description of the tutorial), 'DATA SOURCES' (with a sub-section 'Add your first data source'), and 'DASHBOARDS' (with a sub-section 'Create your first dashboard'). At the bottom of the main panel, there are 'Dashboards' and 'Latest from the blog' buttons.

CREATION OF DASHBOARD AND ADDING DATA SOURCE

The screenshot shows the 'Add data source' page in Grafana. The URL is 3.84.94.222:3000/connections/datasources/new. The top navigation bar includes 'Home', 'Connections', 'Data sources', and 'Add data source'. A search bar at the top right contains 'Search or jump to...'. Below the navigation, the title 'Add data source' is displayed, followed by the sub-instruction 'Choose a data source type'. A filter bar says 'Filter by name or type'. The main content area is titled 'Time series databases' and lists three options:

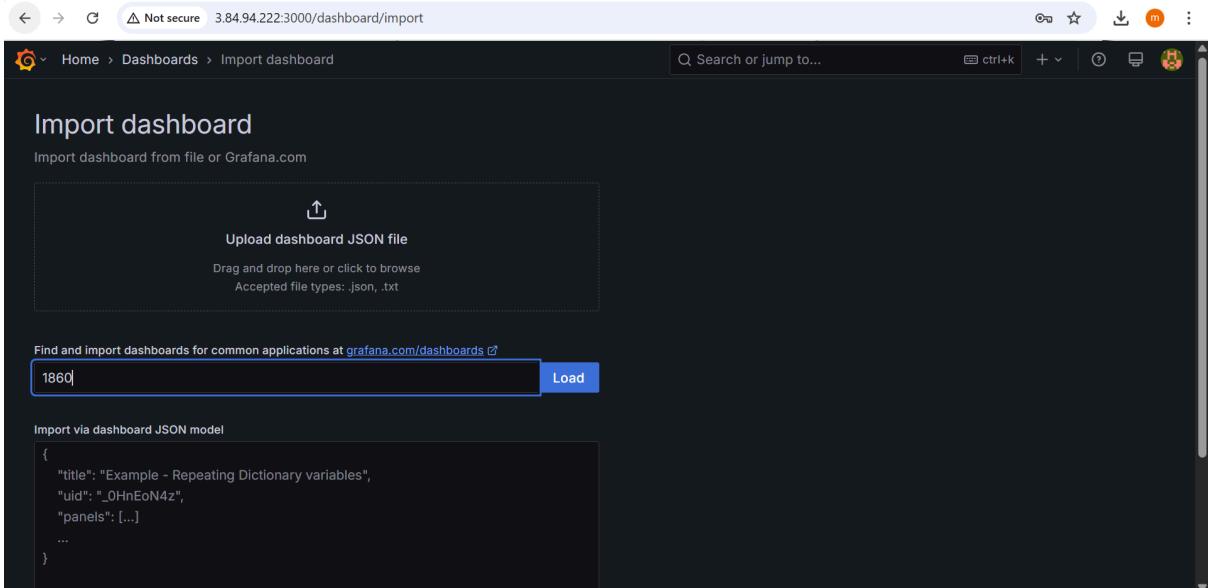
- Prometheus**: Open source time series database & alerting. Core.
- Graphite**: Open source time series database. Core.
- InfluxDB**: Open source time series database. Core.

ADD PROMETHEUS AS THE DATA SOURCE

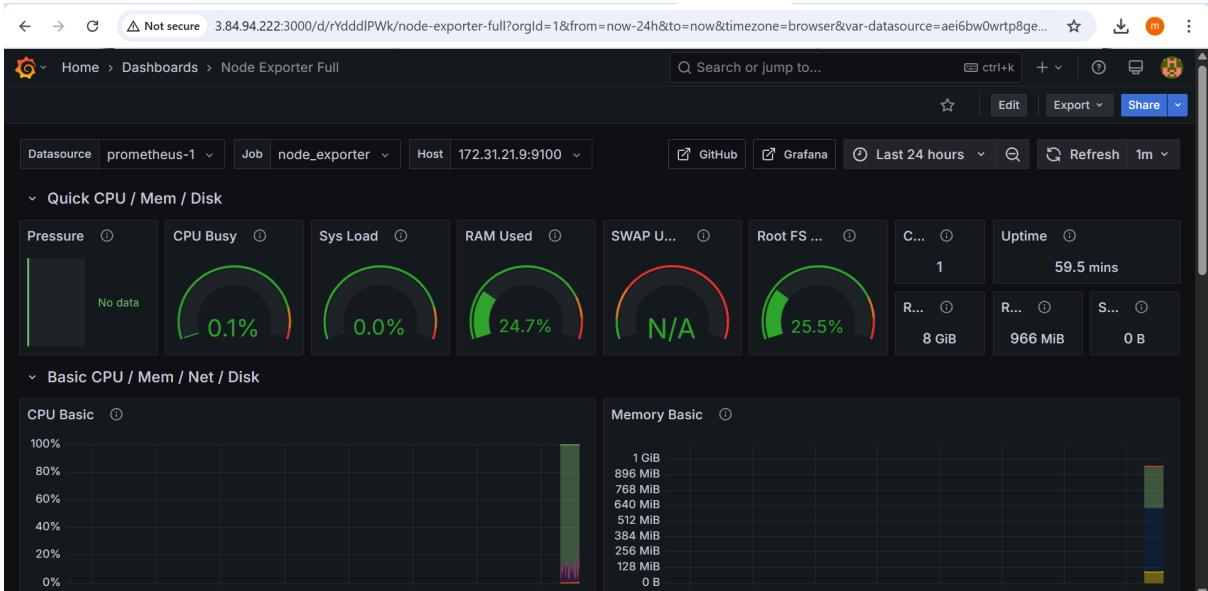
The screenshot shows the 'Edit data source' page for 'prometheus-1'. The URL is 3.84.94.222:3000/connections/datasources/edit/aei6bw0wrtp8ge. The top navigation bar is identical to the previous screenshot. The main content is divided into sections:

- Connection**: Contains a field 'Prometheus server URL *' with the value 'http://localhost:9090'.
- Authentication**: Contains a dropdown 'Authentication method' set to 'No Authentication'. A note below says 'Choose an authentication method to access the data source'.
- TLS settings**: Contains a note 'Additional security measures that can be applied on top of authentication'.

CREATION OF THE DASHBOARD



The screenshot shows the 'Import dashboard' page in Grafana. It features a large input field for uploading a JSON file, a search bar with the value '1860', and a code editor displaying a JSON snippet for a dashboard titled 'Example - Repeating Dictionary variables'.



The screenshot shows the 'Node Exporter Full' dashboard. It includes a navigation bar with 'Datasource: prometheus-1', 'Job: node_exporter', and 'Host: 172.31.21.9:9100'. Below the navigation are several gauge charts and a table for system metrics like CPU, RAM, and disk usage.

DASHBOARD OUTPUT

2. Jenkins Installation and Configuration

Jenkins Ansible Playbook Explained

```
---  
- name: Playbook to Install Jenkins and Amazon Corretto JDK  
hosts: all  
become: yes  
tasks:  
  - name: Install required dependencies
```

```
yum:
  name:
    - fontconfig
    - java-17-amazon-corretto
  state: present
```

Explanation:

- The playbook header specifies it will run on all targeted hosts with elevated privileges (`become: yes`).
- The first task installs required dependencies:
 - `fontconfig`: Required by Jenkins for UI rendering
 - `java-17-amazon-corretto`: Amazon's distribution of OpenJDK, required to run Jenkins

```
- name: Add Jenkins repository
  get_url:
    url: https://pkg.jenkins.io/redhat-stable/jenkins.repo
    dest: /etc/yum.repos.d/jenkins.repo

- name: Import Jenkins repository key
  rpm_key:
    state: present
    key: https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key

- name: Add Amazon Corretto repository
  get_url:
    url: https://yum.corretto.aws/corretto.repo
    dest: /etc/yum.repos.d/corretto.repo
```

Explanation:

- These tasks configure the package repositories:
 - First task adds the Jenkins repository configuration
 - Second task imports the Jenkins GPG key for package verification
 - Third task adds the Amazon Corretto repository for Java

```
- name: Upgrade all packages
  yum:
    name: "*"
    state: latest
```

```

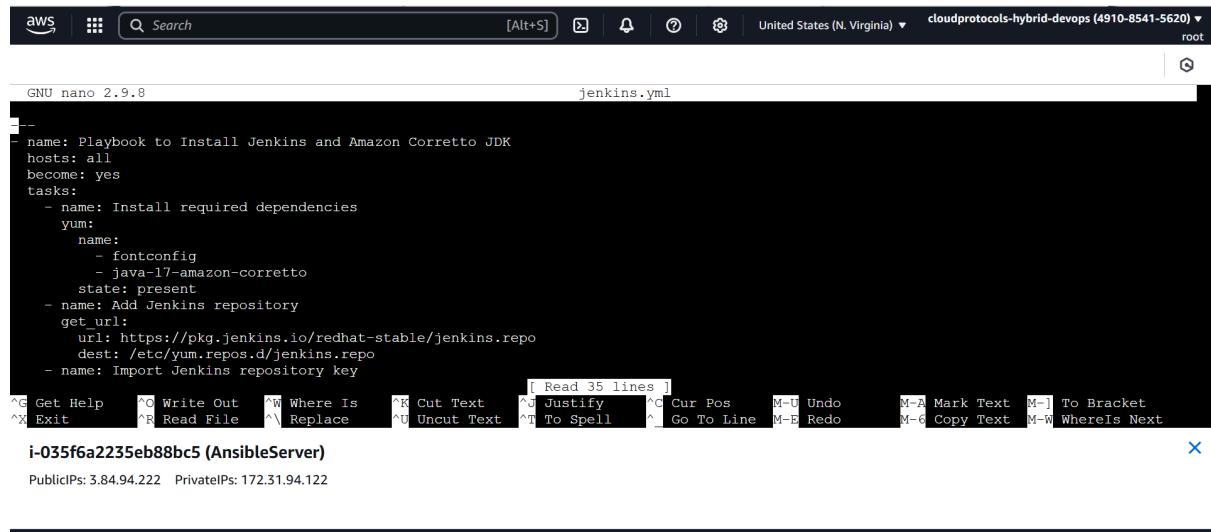
- name: Install Jenkins
  yum:
    name: jenkins
    state: present

- name: Start and enable Jenkins service
  systemd:
    name: jenkins
    state: started
    enabled: yes
    daemon_reload: yes

```

Explanation:

- The first task updates all installed packages to their latest versions.
- The second task installs Jenkins from the configured repository.
- The final task:
 - Starts the Jenkins service
 - Enables it to automatically start on boot
 - Reloads systemd configuration to recognize the new service



The screenshot shows a terminal window titled "jenkins.yml" in the AWS CloudShell interface. The content of the file is as follows:

```

---
- name: Playbook to Install Jenkins and Amazon Corretto JDK
  hosts: all
  become: yes
  tasks:
    - name: Install required dependencies
      yum:
        name:
          - fontconfig
          - java-17-amazon-corretto
        state: present
    - name: Add Jenkins repository
      get_url:
        url: https://pkg.jenkins.io/redhat-stable/jenkins.repo
        dest: /etc/yum.repos.d/jenkins.repo
    - name: Import Jenkins repository key

```

The terminal also displays the nano editor's status bar at the bottom, showing key bindings like **[Read 35 lines]**, **^G Get Help**, **^O Write Out**, etc.

ANSIBLE YML FILE

← → ⌂ 491085415620-6tnyhdjs.us-east-1.console.aws.amazon.com/ec2-instance-connect/ssh/home?region=us-east-1&connType=standard&instanceId=i-035f6... ☆ ⌂ m ⌂

aws Search [Alt+S] United States (N. Virginia) ▾ cloudprotocols-hybrid-devops (4910-8541-5620) ▾ root

```
[ec2-user@ip-172-31-94-122 ~]$ sudo systemctl status jenkins
● jenkins.service - Jenkins Continuous Integration Server
  Loaded: loaded (/usr/lib/systemd/system/jenkins.service; disabled; vendor preset: disabled)
  Active: active (running) since Mon 2025-04-07 08:39:21 UTC; 37s ago
    Main PID: 18324 (java)
      CGroup: /system.slice/jenkins.service
             └─18324 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=%C/jenkins/war --httpPort=8080

Apr 07 08:39:15 ip-172-31-94-122.ec2.internal jenkins[18324]: 48885afe2ff694b6fb81b41944c5df93
Apr 07 08:39:15 ip-172-31-94-122.ec2.internal jenkins[18324]: This may also be found at: /var/lib/jenkins/secrets/initialAdminPassword
Apr 07 08:39:15 ip-172-31-94-122.ec2.internal jenkins[18324]: ****
Apr 07 08:39:21 ip-172-31-94-122.ec2.internal jenkins[18324]: 2025-04-07 08:39:21.610+0000 [id=32] INFO jenkins.InitRe...ation
Apr 07 08:39:21 ip-172-31-94-122.ec2.internal jenkins[18324]: 2025-04-07 08:39:21.637+0000 [id=24] INFO hudson.lifecyc...ning
Apr 07 08:39:21 ip-172-31-94-122.ec2.internal systemd[1]: Started Jenkins Continuous Integration Server.
Apr 07 08:39:21 ip-172-31-94-122.ec2.internal jenkins[18324]: 2025-04-07 08:39:21.867+0000 [id=47] INFO h.m.DownloadSe...aller
Apr 07 08:39:21 ip-172-31-94-122.ec2.internal jenkins[18324]: 2025-04-07 08:39:21.867+0000 [id=47] INFO hudson.util.Re...pt #1
Hint: Some lines were ellipsized, use -l to show in full.
[ec2-user@ip-172-31-94-122 ~]$
```

i-035f6a2235eb88bc5 (AnsibleServer)

Public IPs: 5.84.94.222 Private IPs: 172.31.94.122

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

← → ⌂ Not secure 3.84.94.222:8080/login?from=%2F

Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

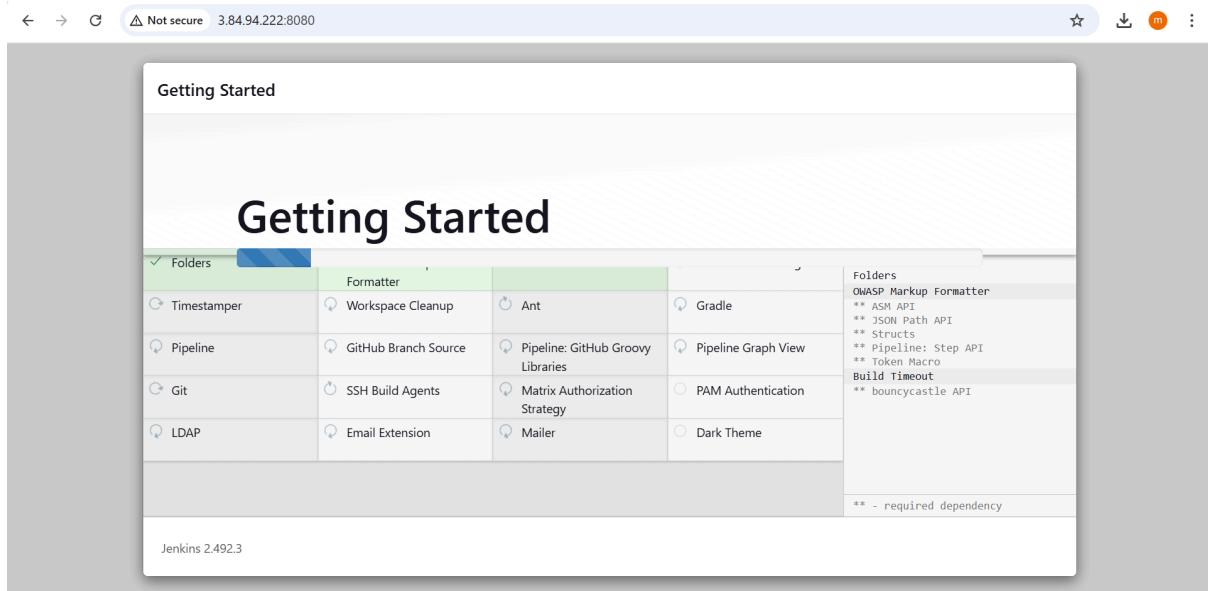
/var/lib/jenkins/secrets/initialAdminPassword

Please copy the password from either location and paste it below.

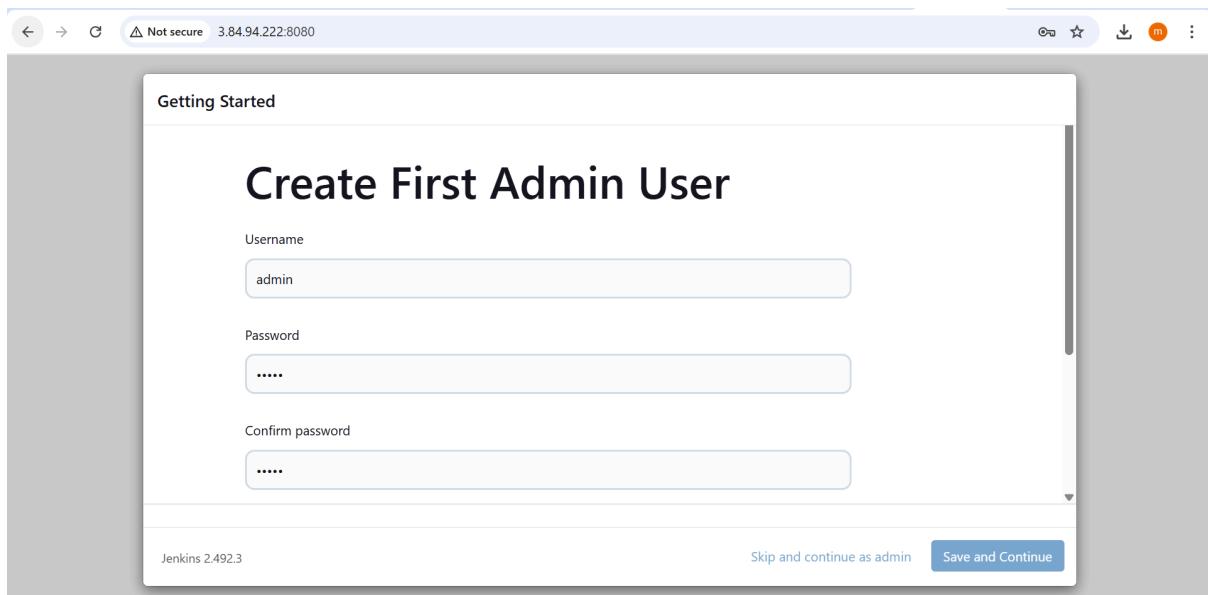
Administrator password

Continue

VERIFY THE JENKINS IN THE BROWSER



INSTALL ALL THE PLUGINS



CREATION OF THE USER

The screenshot shows the Jenkins interface with the URL 3.84.94.222:8080/computer/CICD/log. The left sidebar has links for Status, Delete Agent, Configure, Build History, Load Statistics, and Log, with Log selected. A sub-section titled "Build Executor Status" shows "(0 of 4 executors busy)". The main content area displays a log of an SSH session:

```
SSHLauncher{host='98.84.116.212', port=22, credentialsId='36124185-bc1f-482c-982a-8a4cbda3b7d7', jvmOptions='', javaPath='', prefixStartSlaveCmd='', suffixStartSlaveCmd='', launchTimeoutSeconds=60, maxNumRetries=10, retryWaitTime=15, sshHostKeyVerificationStrategy=hudson.plugins.sshslaves.verifiers.NonVerifyingKeyVerificationStrategy, tcpNoDelay=true, trackCredentials=true}
[04/07/25 15:04:12] [SSH] Opening SSH connection to 98.84.116.212:22.
[04/07/25 15:04:12] [SSH] WARNING: SSH Host Keys are not being verified. Man-in-the-middle attacks may be possible against this connection.
[04/07/25 15:04:12] [SSH] Authentication successful.
[04/07/25 15:04:12] [SSH] The remote user's environment is:
BASH=/usr/bin/bash
BASHOPTS=cmdhist:extquote:force_fignore:hostcomplete:interactive_comments:progcomp:promptvars:sourcepath
BASH_ALIASES=()
BASH_ARGC=()
BASH_ARGV=()
BASH_CMDS=()
BASH_EXECUTION_STRING=set
BASH_LINENO=()
BASH_SOURCE=()
```

CONNECT THE MASTER WITH SLAVE AGENTS USING SSH CONNECTIONS

The screenshot shows the Jenkins interface with the URL 3.84.94.222:8080/computer/CICD/log. The left sidebar has links for Status, Delete Agent, Configure, Build History, Load Statistics, and Log, with Log selected. The main content area displays a log of an SFTP agent connection:

```
[04/07/25 15:04:12] [SSH] Starting sftp client.
[04/07/25 15:04:12] [SSH] Copying latest remoting.jar...
[04/07/25 15:04:12] [SSH] Copied 1,395,562 bytes.
Expanded the channel window size to 4MB
[04/07/25 15:04:12] [SSH] Starting agent process: cd "/home/ec2-user" && java -jar remoting.jar -workDir /home/ec2-user -jar-cache /home/ec2-user/remoting/jarCache
Apr 07, 2025 3:04:12 PM org.jenkinsci.remoting.engine.WorkDirManager initializeWorkDir
INFO: Using /home/ec2-user/remoting as a remoting work directory
Apr 07, 2025 3:04:13 PM org.jenkinsci.remoting.engine.WorkDirManager setupLogging
INFO: Both error and output logs will be printed to /home/ec2-user/remoting
<==[JENKINS REMOTING CAPACITY]==>channel started
Remoting version: 3283.v92c105e0f819
Launcher: SSHLauncher
Communication Protocol: Standard in/out
This is a Unix agent
Agent successfully connected and online
```

REST API Jenkins 2.492.3

Not secure 3.84.94.222:8080/manage/computer/createnode

Dashboard > Manage Jenkins > Nodes >

Use this node as much as possible

Launch method ?

Launch agents via SSH

Host ?

98.81.79.144

Credentials ?

ec2-user (slavemachine)

+ Add

Host Key Verification Strategy ?

Non verifying Verification Strategy

Save

SLAVE1 CONNECTED SUCESSFULLY

CONNECT THE MASTER WITH SLAVE AGENTS USING SSH CONNECTIONS

Not secure 3.84.94.222:8080/computer/CICD2/log

Jenkins

Dashboard > Nodes > CICD2 > Log

- Status
- Delete Agent
- Configure
- Build History
- Load Statistics
- Script Console
- Log**
- System Information
- Disconnect

```
SSHLauncher{host='98.81.79.144', port=22, credentialsId='36124185-bc1f-482c-982a-8a4cbda3b7d7',
jvmOptions='', javaPath='', prefixStartSlaveCmd='', suffixStartSlaveCmd='', launchTimeoutSeconds=60,
maxNumRetries=10, retryWaitTime=15,
sshHostKeyVerificationStrategy=hudson.plugins.sshslaves.verifiers.NonVerifyingKeyVerificationStrategy,
tcpNoDelay=true, trackCredentials=true}
[04/07/25 15:08:13] [SSH] Opening SSH connection to 98.81.79.144:22.
[04/07/25 15:08:13] [SSH] WARNING: SSH Host Keys are not being verified. Man-in-the-middle attacks may be
possible against this connection.
[04/07/25 15:08:14] [SSH] Authentication successful.
[04/07/25 15:08:14] [SSH] The remote user's environment is:
BASH=/usr/bin/bash
BASHEOPTS=cmdhist:extquote:force_fignore:hostcomplete:interactive_comments:progcomp:promptvars:sourcepath
BASH_ALIASES=()
BASH_ARGC=()
BASH_ARGV=()
BASH_CMDS=()
BASH_EXECUTION_STRING=set
BASH_LINENO=()
BASH_SOURCE=()
```

Build Executor Status

(0 of 4 executors busy)

The screenshot shows two Jenkins node management pages. The top page is for the 'CICD2' node, showing its log output. The bottom page is for the 'CICD' node, showing its configuration details like labels and projects tied to it.

CICD2 Node Log:

```

SSHLauncher{host='98.81.79.144', port=22, credentialsId='36124185-bc1f-482c-982a-8a4cbda3b7d7',
jvmOptions='', javaPath='', prefixStartSlaveCmd='', suffixStartSlaveCmd='', launchTimeoutSeconds=60,
maxNumRetries=10, retryWaitTime=15,
sshHostKeyVerificationStrategy=hudson.plugins.sshslaves.verifiers.NonVerifyingKeyVerificationStrategy,
tcpNoDelay=true, trackCredentials=true}
[04/07/25 15:08:13] [SSH] Opening SSH connection to 98.81.79.144:22.
[04/07/25 15:08:13] [SSH] WARNING: SSH Host Keys are not being verified. Man-in-the-middle attacks may be
possible against this connection.
[04/07/25 15:08:14] [SSH] Authentication successful.
[04/07/25 15:08:14] [SSH] The remote user's environment is:
BASH=/usr/bin/bash
BASHOPTS=cmdhist:extquote:force_fignore:hostcomplete:interactive_comments:progcomp:promptvars:sourcepath
BASH_ALIASES=()
BASH_ARGC=()
BASH_ARGV=()
BASH_CMDS=()
BASH_EXECUTION_STRING=set
BASH_lineno=()
BASH_SOURCE=()

```

CICD Node Configuration:

- Labels:** slave1
- Projects tied to CICD:** None

SLAVE2 CONNECTED SUCESSFULLY

3. Multibranch Pipeline Configuration

Create a Multibranch Pipeline Job

1. Go to **Jenkins Dashboard** → **New Item**.
2. Enter a **Job Name** (e.g., CICD).
3. Select **Multibranch Pipeline** → **OK**.

The screenshot shows the Jenkins job configuration interface for a 'CICD' job. The left sidebar lists various configuration options: General, Branch Sources (selected), Build Configuration, Scan Multibranch Pipeline Triggers, Orphaned Item Strategy, Appearance, Health metrics, and Properties. The main panel is titled 'Git Project Repository' and contains the URL 'https://github.com/tupakulamanoj/JENKINS.git'. Under 'Credentials', it says '- none -' and has a '+ Add' button. A 'Behaviours' section includes a 'Discover branches' option with an 'Add' button. At the bottom are 'Save' and 'Apply' buttons.

3. Configure the Multibranch Pipeline

Inside your job settings:

- **Branch Sources** → **Add Source** → Choose **GitHub**.

If you don't see GitHub option, you missed GitHub Branch Source plugin.

- **Credentials** → Add credentials if your repo is private (GitHub Username + Token).

This screenshot is identical to the one above, showing the Jenkins job configuration interface for a 'CICD' job. The 'Branch Sources' tab is selected in the sidebar. The main panel shows the 'Git Project Repository' set to 'https://github.com/tupakulamanoj/JENKINS.git', 'Credentials' set to '- none -', and a 'Behaviours' section with a 'Discover branches' option. The 'Save' and 'Apply' buttons are at the bottom.

The screenshot shows the Jenkins CICD dashboard. On the left, there's a sidebar with various options like Status, Configure, Scan Multibranch Pipeline Now, Scan Multibranch Pipeline Log, Multibranch Pipeline Events, Delete Multibranch Pipeline, Build History, Project Relationship, Check File Fingerprint, Rename, and Pipeline Syntax. The main area is titled 'CICD' and shows a table for 'Branches (1)'. The table has columns for S (Status), W (Workload), Name (with a dropdown arrow), Last Success, Last Failure, and Last Duration. There is one entry for 'master' with a status icon (blue circle with a white dot), a workload icon (yellow sun), and the name 'master'. The last success, failure, and duration are all listed as 'N/A'.

4. Scan Repository

- Click **Scan Multibranch Pipeline Now**.
- Jenkins will detect all branches that have a **Jenkinsfile**.

The screenshot shows a GitHub repository page for 'tupakulamanoj / JENKINS'. The 'Code' tab is selected. In the code editor, there is a file named 'jenkinsfile' in the 'master' branch. The code content is as follows:

```
1 pipeline {
2     agent any
3
4     environment {
5         DEPLOY_DIR = "/var/www/html"
6     }
7
8     stages {
9         stage('checkout') {
10             steps {
11                 checkout scm
12             }
13         }
14
15         stage('Deploy to Apache') {
16             steps {
17                 sshagent (credentials: ['e3d7b74b-d50e-42cd-aa18-7b7b49d6f428']) {
18                     sh ***
19                 }
20             }
21         }
22     }
23 }
```

At the bottom of the code editor, there are buttons for 'Cancel changes' and 'Commit changes..'. The commit message says '55% faster with GitHub Copilot'.

5. Add GitHub Webhook

So that Jenkins builds automatically when you push.

- Go to your **GitHub repository** → **Settings** → **Webhooks** → **Add webhook**.
- (Use public IP or domain of Jenkins, and port if needed like **8080**)

- **Content type:** application/json

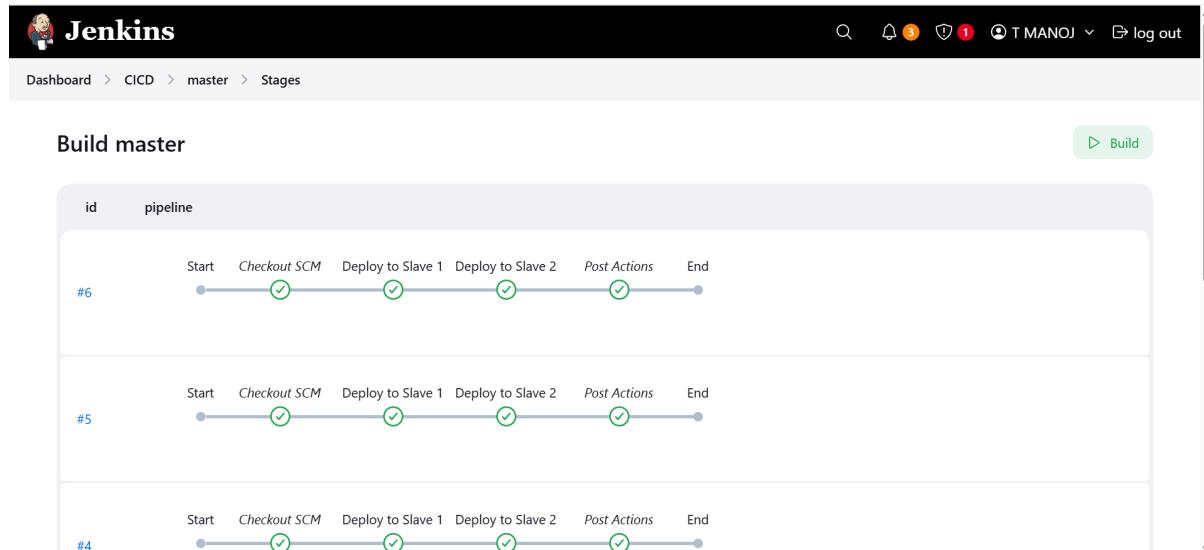
- **Event:** Send "Just the push event".

- **Save.**

The screenshot shows the GitHub 'Webhooks / Manage webhook' page. On the left, there's a sidebar with 'General' selected under 'Access' and 'Webhooks' selected under 'Actions'. The main area has tabs for 'Settings' and 'Recent Deliveries', with 'Settings' active. It contains fields for 'Payload URL *' (set to `http://3.84.94.222:8080/multibranch-webhook-trigger/invoke?token=[token]`), 'Content type *' (set to `application/json`), and 'Secret'. Under 'SSL verification', it says 'By default, we verify SSL certificates when delivering payloads.' with options to 'Enable SSL verification' or 'Disable (not recommended)'.

6.BUILD AND DEPLOY STAGES :

The screenshot shows the Jenkins 'Stage View' for the 'master' branch. The top navigation bar shows 'Not secure' and the URL `3.84.94.222:8080/job/CICD/job/master/`. The left sidebar includes links for 'Status', 'Changes', 'Build Now', 'View Configuration', 'Full Stage View', 'Stages', and 'Pipeline Syntax'. Below this is a 'Builds' section with a table of recent builds. The main area displays a grid of stages for three builds (">#10, #9, #8). Each stage row has four columns: 'Declarative: Checkout SCM' (363ms), 'Deploy to Slave 1' (1s), 'Deploy to Slave 2' (2s), and 'Declarative: Post Actions' (61ms). The first stage of build #10 is highlighted in blue, showing a duration of 240ms. The second stage of build #9 is also highlighted in blue, showing a duration of 844ms.



DEPLOYED IN BOTH SLAVES

← → ⌂ ⚡ Not secure 98.81.79.144

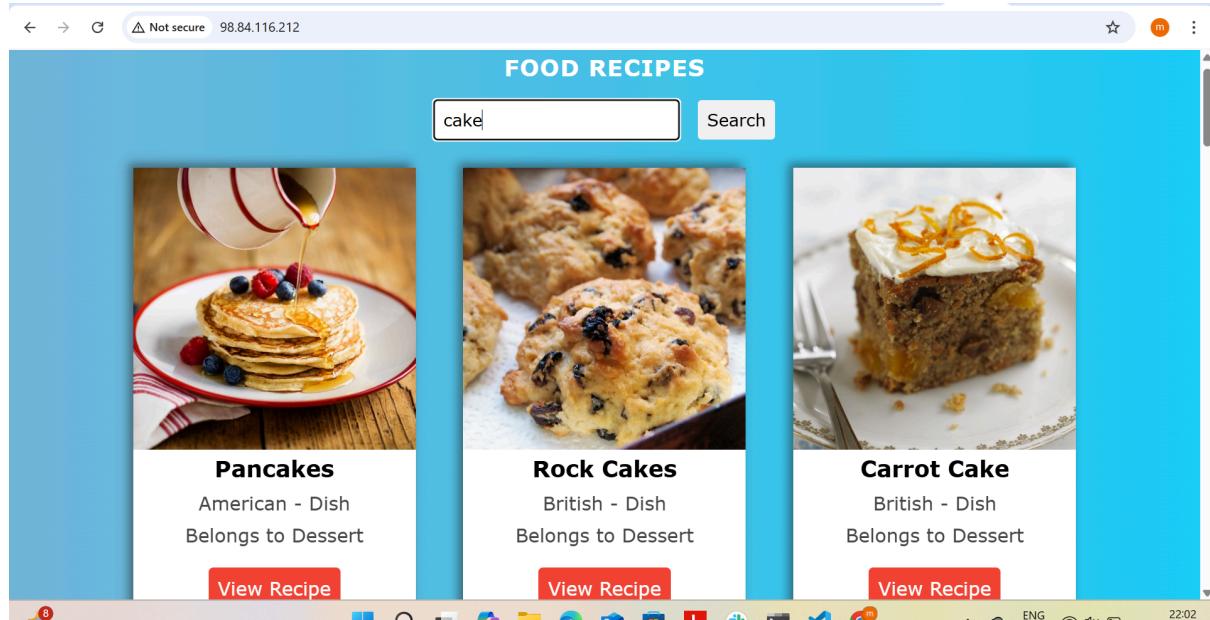
☆ ⚡ :

FOOD RECIPES

Find your recipes

Search Your Favorite recipe
Are you wondering what to cook with the ingredients you have? Our Recipe Generator makes meal planning effortless! Just enter the ingredients you have in your kitchen, and let our smart tool suggest tasty recipes tailored to your preferences.





CLEANING OF RESOURCES

```
manoj@IND-140:~/new$ terraform destroy --auto-approve
data.aws_vpc.default: Reading...
data.aws_vpc.default: Read complete after 3s [id=vpc-0c33a120e93d325f9]
aws_security_group.app_security_group: Refreshing state... [ids=sg-0c374e4bdcc63192]
aws_instance.slave1: Refreshing state... [id=i-0ccf76f0120a072ed]
aws_instance.slave2: Refreshing state... [id=i-019565c249fff915f]
aws_instance.ansible_server: Refreshing state... [id=i-035f6a2235eb88bc5]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the
following symbols:
- destroy

Terraform will perform the following actions:

# aws_instance.ansible_server will be destroyed
- resource "aws_instance" "ansible_server" {
    - ami = "ami-0c02fb55956c7d316" -> null
    - arn = "arn:aws:ec2:us-east-1:491085415620:instance/i-035f6a2235eb88bc5" -> null
    - associate_public_ip_address = true -> null
    - availability_zone = "us-east-1a" -> null
    - cpu_core_count = 1 -> null
    - cpu_threads_per_core = 1 -> null
    - disable_api_stop = false -> null
    - disable_api_termination = false -> null
    - ebs_optimized = false -> null
    - get_password_data = false -> null
    - hibernation = false -> null
    - id = "i-035f6a2235eb88bc5" -> null
}
```

```
manoj@IND-140: ~/new      X + √
Plan: 0 to add, 0 to change, 4 to destroy.

Changes to Outputs:
- ansible_server_public_ip = "3.84.94.222" -> null
- grafana_url              = "http://3.84.94.222:3000" -> null
- prometheus_url            = "http://3.84.94.222:9090" -> null
- slave1_public_ip          = "98.81.79.144" -> null
- slave2_public_ip          = "98.84.116.212" -> null
aws_instance.ansible_server: Destroying... [id=i-035f6a2235eb88bc5]
aws_instance.ansible_server: Still destroying... [id=i-035f6a2235eb88bc5, 10s elapsed]
aws_instance.ansible_server: Still destroying... [id=i-035f6a2235eb88bc5, 21s elapsed]
aws_instance.ansible_server: Still destroying... [id=i-035f6a2235eb88bc5, 31s elapsed]
aws_instance.ansible_server: Still destroying... [id=i-035f6a2235eb88bc5, 41s elapsed]
aws_instance.ansible_server: Still destroying... [id=i-035f6a2235eb88bc5, 52s elapsed]
aws_instance.ansible_server: Still destroying... [id=i-035f6a2235eb88bc5, 1m2s elapsed]
aws_instance.ansible_server: Destruction complete after 1m6s
aws_instance.slave2: Destroying... [id=i-019565c249fff915f]
aws_instance.slave1: Destroying... [id=i-0ccf76f0120a072ed]
aws_instance.slave2: Still destroying... [id=i-019565c249fff915f, 10s elapsed]
aws_instance.slave1: Still destroying... [id=i-0ccf76f0120a072ed, 10s elapsed]
aws_instance.slave1: Still destroying... [id=i-0ccf76f0120a072ed, 21s elapsed]
aws_instance.slave2: Still destroying... [id=i-019565c249fff915f, 21s elapsed]
aws_instance.slave1: Destruction complete after 23s
aws_instance.slave2: Still destroying... [id=i-019565c249fff915f, 31s elapsed]
aws_instance.slave2: Destruction complete after 34s
aws_security_group.app_security_group: Destroying... [id=sg-0c374e4bbdcc63192]
aws_security_group.app_security_group: Destruction complete after 1s

Destroy complete! Resources: 4 destroyed.
manoj@IND-140: ~/new$
```

Conclusion

This DevOps infrastructure provides a solid foundation for modern software development practices. By automating infrastructure provisioning with Terraform, standardizing configurations with Ansible, implementing CI/CD with Jenkins, and ensuring comprehensive monitoring with Prometheus and Grafana, organizations can achieve greater agility, reliability, and visibility. The modular nature of this architecture allows teams to adapt and extend it according to their specific needs, whether scaling to handle larger workloads, incorporating additional tools, or optimizing for particular use cases. With proper implementation and security considerations addressed, this infrastructure can support everything from small team projects to enterprise-scale applications. By embracing these DevOps practices and tools, organizations position themselves to respond more effectively to changing requirements, deliver features faster, and maintain high-quality standards throughout their development process.