**AWS Deployment (Automation)**

* **Terraform**

Terraform is an infrastructure as code tool that lets you build, change, and version cloud and on-prem resources safely and efficiently.

For more information, please visit <https://developer.hashicorp.com/terraform/intro>

**Installation** <https://developer.hashicorp.com/terraform/install> terraform --version

**AWS CLI2** <https://docs.aws.amazon.com/cli/latest/userguide/getting-started-version.html> aws -version

We use Terraform CLI <https://developer.hashicorp.com/terraform/cli> to deploy PPRO servers.

terraform -install-autocomplete enables command line autocompletion (*reload config: exec $SHELL*)

Create a new directory for your Terraform workspace (mkdir,cd)

Create configuration files: main.tf, variables.tf and terraform.tfvars (

main.tf in this file you will define resources.

variables.tf is used to declare variables for resources in main.tf

terraform. tfvars is used to set values for the variables.

terraform init to install all necessary plugins

terraform validate check whether the configuration is valid

terraform fmt reformat your configuration in standard style

terraform plan -var-file=terraform.tfvars

terraform apply -var-file=terraform.tfvars

terraform destroy -var-file=terraform.tfvars

To view a list of the commands available in your current Terraform version, run **terraform** with no additional arguments and please read more about this automation tool.

**main.tf**

terraform {

  required\_providers {

*aws =* {

*source*  *=* "hashicorp/aws"

*version* *=* "~> 4.16"

    }

  }

*required\_version =* ">= 1.2.0"

}

provider *"aws"* {

*region = var.region*

}

data *"aws\_vpc"* *"default"* {

*default =* true

}

resource *"aws\_instance"* *"mirror\_server"* {

*ami                    = var.ami*

*instance\_type          = var.instance\_type*

*subnet\_id              = var.subnet\_id*

*vpc\_security\_group\_ids = var.vpc\_security\_group\_ids*

*tags =* {

*Name* *=* "*${var.customer\_name}*-Mirror"

  }

  root\_block\_device {

*volume\_size           = var.root\_volume\_size*

*volume\_type           = var.root\_volume\_type*

*delete\_on\_termination =* true

*tags =* {

*Name* *=* "*${var.customer\_name}*-Mirror-OS"

    }

  }

  ebs\_block\_device {

*device\_name           =* "/dev/sdf"

*volume\_size           = var.ebs\_volume\_size\_ppro*

*volume\_type           = var.ebs\_volume\_type*

*throughput            = var.ebs\_throughput*

*iops                  = var.ebs\_iops*

*delete\_on\_termination =* false

*tags =* {

*Name*              *=* "*${var.customer\_name}*-Mirror-PPro"

*PProMirrorBackups* *=* "Yes"

    }

  }

  ebs\_block\_device {

*device\_name           =* "/dev/sdg"

*volume\_size           = var.ebs\_volume\_size\_images*

*volume\_type           = var.ebs\_volume\_type*

*throughput            = var.ebs\_throughput*

*iops                  = var.ebs\_iops*

*delete\_on\_termination =* false

*tags =* {

*Name*              *=* "*${var.customer\_name}*-Mirror-Images"

*PProMirrorBackups* *=* "Yes"

    }

  }

}

resource *"aws\_eip"* *"main-eip"* {

*vpc =* true

*tags =* {

*Name* *=* "*${var.customer\_name}*-EIP"

  }

}

resource *"aws\_eip\_association"* *"main-eip-association"* {

*instance\_id   =* aws\_instance*.mirror\_server.id*

*allocation\_id =* aws\_eip*.main-eip.id*

}

output *"instance\_name"* {

*description =* "The name of the PPro Server."

*value       =* "*${var.customer\_name}*-Mirror"

}

output *"instance\_id"* {

*description =* "The ID of the created EC2 instance."

*value       =* aws\_instance*.mirror\_server.id*

}

output *"os\_version"* {

*description =* "The operating system version of the EC2 instance."

*value       = var.ami*

}

output *"availability\_zone"* {

*description =* "The availability zone of the EC2 instance."

*value       =* aws\_instance*.mirror\_server.availability\_zone*

}

output *"public\_ip"* {

*description =* "The public IP address of the Elastic IP."

*value       =* aws\_eip*.main-eip.public\_ip*

}

output *"instance\_type"* {

*description =* "The type of the EC2 instance."

*value       =* aws\_instance*.mirror\_server.instance\_type*

}

**variables.tf**

variable *"region"* {

*description =* "Enter your preferred Mirror Server region: N.Virginia / N.California"

*default     =* "us-east-1"

}

variable *"vpc"* {

*description =* "Please enter preferred VPC Id"

*default     =* "vpc-777"

}

variable *"ami"* {

*description =* "Enter the Linux version (EL7 or EL8)"

*default     =* "ami-123"

}

variable *"instance\_type"* {

*description =* "Enter the instance type: t3.medium / r6i.large / r6i.xlarge / r6i.2xlarge"

*default     =* "t3.medium"

}

variable *"subnet\_id"* {

*description =* "Please choose one of the existing subnets"

*default     =* "abc"

}

variable *"vpc\_security\_group\_ids"* {

*description =* "Please enter sg id"

*default     =* ["xyz"]

}

variable *"customer\_name"* {

*description =* "Enter the customer name"

*type        = string*

*default     =* "economy-mirror"

}

variable *"root\_volume\_type"* {

*description =* "Enter root volume type"

*default     =* "gp3"

}

variable *"root\_volume\_size"* {

*description =* "Enter root volume size"

*default     =* "25"

}

variable *"ebs\_volume\_type"* {

*description =* "Enter ebs volume type"

*default     =* "gp3"

}

variable *"ebs\_volume\_size\_ppro"* {

*description =* "Enter ebs volume size"

*default     =* "200"

}

variable *"ebs\_volume\_size\_images"* {

*description =* "Enter ebs volume size"

*default     =* "200"

}

variable *"ebs\_throughput"* {

*description =* "Enter ebs volume throughput"

*default     =* "250"

}

variable *"ebs\_iops"* {

*description =* "Enter ebs volume iops"

*default     =* "3000"

}

**terraform.tfvars**

*region                 =* "us-west-1"

*vpc                    =* "vpc-xyz"

*ami                    =* "ami-123"

*instance\_type          =* "t3.medium"

*subnet\_id              =* "abc"

*vpc\_security\_group\_ids =* ["abc"]

*customer\_name          =* "Customer"

*root\_volume\_type       =* "gp3"

*root\_volume\_size       =* "25"

*ebs\_volume\_type        =* "gp3"

*ebs\_volume\_size\_ppro   =* "500"

*ebs\_volume\_size\_images =* "500"

*ebs\_throughput         =* "250"

*ebs\_iops               =* "3000"

So, we will make changes according our customers need (disks, size etc)

**STATE file**

Default name is terraform.tfstate <https://developer.hashicorp.com/terraform/language/state> and its created in same directory as your other configuration files.

By default, Terraform uses a local backend, storing tfstate files in local filesystem.

It is also possible to use remote backend (S3).

* **Ansible**

Ansible provides open-source automation that reduces complexity and runs everywhere. Using Ansible lets you automate virtually any task. Here are some common use cases for Ansible:

* Eliminate repetition and simplify workflows
* Manage and maintain system configuration
* Continuously deploy complex software
* Perform zero-downtime rolling updates

Please visit <https://docs.ansible.com/ansible/latest/getting_started/introduction.html> for more information

**Ansible is an agentless automation tool that you install on a single host (referred to as the control node).**

<https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html#pipx-install>

Ansible uses simple, human-readable scripts called playbooks to automate your tasks. You declare the desired state of a local or remote system in your playbook. Ansible ensures that the system remains in that state.

**Playbook.yaml** <https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_intro.html#playbook-syntax>

---

- hosts: all

  become: true

  remote\_user: root

  vars:

    Customer\_domain: example.com

    Timezone: New\_York

    Customer: sandbox

    desired\_fstab\_content:

      - "LABEL=root / ext4 defaults 1 1"

      - "LABEL=/ppro /ppro ext4 defaults,noatime,nodiratime 0 0"

      - "tmpfs /ppro/wrk tmpfs defaults,mode=0777,noatime,nodiratime 0 0"

      - "tmpfs /ppro/homedirs tmpfs defaults,mode=0777,noatime,nodiratime 0 0"

      - "tmpfs /tmp tmpfs defaults,mode=0777,noatime,nodiratime 0 0"

  tasks:

    - name: Check if a partition exists on /dev/nvme1n1

      command: parted /dev/nvme1n1 print

      register: partition\_table

      ignore\_errors: yes

    - name: Execute fdisk command to create a new partition

      shell: '(echo n; echo p; echo ""; echo ""; echo ""; echo w) | fdisk /dev/nvme1n1'

      when: "'Partition Table: gpt' not in partition\_table.stdout"

      ignore\_errors: yes

    - name: Check if /ppro disk (nvme1n1p1) is already formatted

      command: blkid -s TYPE -o value /dev/nvme1n1p1

      register: formatted\_check

      ignore\_errors: yes

    - name: Format the /ppro disk (nvme1n1p1) as ext4 if not formatted

      command: mkfs.ext4 /dev/nvme1n1p1

      ignore\_errors: yes

      when: "'ext4' not in formatted\_check.stdout"

    - name: Label the /ppro disk (nvme1n1p1)

      command: e2label /dev/nvme1n1p1 /ppro

      ignore\_errors: yes

    - name: Check if /ppro disk (nvme1n1p1) is already mounted

      command: mountpoint -q /ppro

      register: mounted\_check

      ignore\_errors: yes

    - name: Mount /ppro if it's not already mounted

      mount:

        path: /ppro

        src: LABEL=/ppro

        fstype: ext4

        state: mounted

      when: not mounted\_check.rc | default(1) | bool

    - name: Create /ppro/wrk and /ppro/homedirs directories

      file:

        path: "{{ item }}"

        state: directory

      loop:

        - /ppro/wrk

        - /ppro/homedirs

    - name: Mount all filesystems specified in fstab (except root)

      mount:

        name: "{{ item.split(' ')[1] }}"

        src: "{{ item.split(' ')[0] }}"

        fstype: "{{ item.split(' ')[2] }}"

        state: mounted

      loop: "{{ desired\_fstab\_content }}"

      when: item.split(' ')[0] != '/' and item != "LABEL=root / ext4 defaults 1 1"

    - name: Change permissions for /ppro directory

      file:

        path: /ppro

        mode: "0777"

        recurse: yes

    - name: Ensure /ppro/etc/ssh directory exist

      stat:

        path: /ppro/etc/ssh

      register: ssh\_dir\_check

      ignore\_errors: yes

    - name: Ensure SSH keys exist

      stat:

        path: "{{ item }}"

      register: ssh\_key\_check

      with\_items:

        - /ppro/etc/ssh/ppro-key

        - /ppro/etc/ssh/ppro-upload-key

      ignore\_errors: yes

    - name: Create /ppro/etc/ssh directory if it doesn't exist

      file:

        path: /ppro/etc/ssh

        state: directory

      when: ssh\_dir\_check.stat.isreg is not defined or ssh\_dir\_check.stat.isreg == false

    - name: Copy ppro-ssh-singleuserkeys script to EC2

      copy:

        src: /usr/bin/ppro-ssh-singleuserkeys

        dest: /usr/bin/ppro-ssh-singleuserkeys

        mode: 0755

      become: yes

    - name: Generate SSH keys if the SSH directory or keys do not exist

      command: /usr/bin/ppro-ssh-singleuserkeys --generate

      args:

        chdir: /ppro/etc/ssh

      when: not (ssh\_dir\_check.stat.exists | default(false)) | bool or

            (ssh\_key\_check.results | map(attribute='stat.exists') | max < 1)

      become: yes

    - name: Install SSH keys if they were generated

      command: /usr/bin/ppro-ssh-singleuserkeys --install

      when: not (ssh\_dir\_check.stat.exists | default(false)) | bool or

            (ssh\_key\_check.results | map(attribute='stat.exists') | max < 1)

      become: yes

    - name:  Check if /images directory exists

      stat:

        path: /images

      register: images\_dir\_check

      ignore\_errors: yes  *# Ignore errors if /images doesn't exist*

    - name: Create a symbolic link from /ppro/docimages to /images

      command: ln -s /ppro/docimages /images

      become: yes

      when: not images\_dir\_check.stat.exists

    - name: Set the hostname

      hostname:

        name: "ppro-main.{{ Customer\_domain }}"

    - name: Set the timezone

      command: "timedatectl set-timezone America/{{ Timezone }}"

    - name: Update the ppro-public-repo

      yum:

        name: ppro-public-repo

        state: latest

    - name: Clean YUM cache

      command: yum clean all

    - name: Enable ppro-supplemental repository

      command: yum config-manager --enable ppro-supplemental

    - name: Install ppro-install-managed package

      yum:

        name: ppro-install-managed

        state: latest

    - name: Update installed packages

      yum:

        name: '\*'

        state: latest

    - name: Check if hostanme ID laready exist

      command: ppro-zabbix --gethostidname

      register: hostname\_id\_result

      changed\_when: false

    - name: Set Zabbix host ID name for Customer

      command: ppro-zabbix --sethostidname "{{ Customer }}-Main"

      when: "'{{ Customer }}-Main' not in hostname\_id\_result.stdout"

      register: ppro\_zabbix\_output

      changed\_when: false

    - name: Display ppro-zabbix command output

      debug:

        var: ppro\_zabbix\_output.stdout\_lines

    - name: Display hostanme hostname\_id\_result

      debug:

        var: hostname\_id\_result.stdout

    - name: Reboot

      reboot:

      become: yes

**hosts.ini** file is an inventory file with defined IP addresses of destination, target servers, where ansible will run its tasks.

<https://docs.ansible.com/ansible/latest/inventory_guide/intro_inventory.html>

[all]

123.123.123.123

123.123.123.123

123.123.123.123

123.123.123.123

123.123.123.123

123.123.123.123

Command to run your ansible playbook: ansible-playbook -I hosts.ini playbook.yaml

Ensure that your public SSH key is present in the authorized\_keys file on each of the servers.

For PPRO environment I used athena (jump box/bastion host) to run ansible playbooks ( because athena has its ssh key pre-defined in all existing servers).

**To get IP addresses of existing servers in AWS:**

aws configure sso ( validate with credentials which aws account you want to use)

*aws ec2 describe-instances --region us-east-1 --filters "Name=instance-state-name,Values=running" --query "Reservations[].Instances[].PublicIpAddress" --output text > us-east-1-ips.txt*

Output will be smth like below, just grap IP addresses or modify your command.

---------------------------------------------------------------------------------------------

| DescribeInstances |

+----------------------+---------------+----------------------------------------------------+

| i-0347f7d9648077e76 | 52.8.219.230 | ec2-52-8-219-230.us-west-1.compute.amazonaws.com |

| i-050f5f528911dc5be | 52.8.245.70 | ec2-52-8-245-70.us-west-1.compute.amazonaws.com |

| i-01879f43ad7cd7b5a | 54.193.4.73 | ec2-54-193-4-73.us-west-1.compute.amazonaws.com |

|  |  |  |
| --- | --- | --- |
|  |  |  |