

Name

Using Terraform and Ansible to provision, Jenkins to Automate Maven packing of WAR file and deployment to Tomcat 9

Table Of Contents

- [Name](#)
- [Table of Contents](#)
- [Description](#)
 - [Overview](#)
 - [Directory Structure](#)
 - [Usage](#)
 - [Summary of Steps Performed](#)
- [Terraform](#)
 - [1. Setup](#)
 - [1. Basic Setup](#)
 - [2. Security Groups](#)
 - [3. S3 bucket](#)
 - [4. IAM role](#)
 - [2. Provisioning and Deployment Server Instances](#)
 - [1. Provisioning Server](#)
 - [2. Deployment Server](#)
 - [3. Inventory file in Ansible Provisioning Directory](#)
 - [4. Zipping Ansible Provisioning Directory](#)
 - [5. Upload to S3 bucket](#)

- 6. Ansible Provisioning Directory Download to Provisioning Server
- 7. Copy SSH Public Key to Deployment Server
- Ansible
 - 1. Tomcat Setup on Deployment Server
 - 1. tomcat_playbook.yml
 - 2. Tomcat Installation
 - 2. Build and Deploy WAR
 - 1. Clone and Build Hello World WAR File
 - 2. Deploy WAR File and Restart Tomcat on Deployment Server
 - 3. Run Ansible Plays in Terraform

Description

Overview

Terraform, an open-source IaC is used to spin up a EC2 server instance, a deployment instance and S3 bucket to upload ansible provisioning playbooks.

Directory Structure

```
Project
├── README.md
├── main.tf
├── aws_terraform.pem
├── variables.tf
├── bucket.tf
├── ansible_provisioning
│   └── ...
├── providers.tf
└── outputs.tf
```

13 directories, 22 files

Usage

1. Create a key pair in AWS, download and replace `aws_terraform.pem` .
2. Copy VPC ID from AWS and place in `variables.tf` as the default value for `vpc_id` .
3. Copy Ubuntu AMI id and place in `variables.tf` as the default value for `ami_id` .
4. Copy AWS access key, secret key, and token and place in `variables.tf` as the default value for `aws_access_key` , `aws_secret_key` , `aws_token` .
5. Run terraform commands in the cloned directory:
 - `terraform init`
 - `terraform plan`
 - `terraform apply -auto-approve`

Summary of Steps Performed

1. [Terraform](#)
 - i. Spun up `S3` bucket
 - ii. Spun up 2 `EC2` instances, provisioning server and deployment, and generated their public and private ips. Attached necessary IAM role to `EC2` instances to access `S3` bucket.
 - iii. Placed Ansible provisioning playbooks and roles, with inventory file for deployment ip address. Zipped using Terraform and uploaded to `S3` bucket.
 - iv. Downloaded ansible playbook using `awscli` commands from `S3` bucket to `EC2` provisioning server. Unzipped. Generated ssh keys to connect with deployment server and uploaded to `S3` bucket.

- v. Downloaded ssh public key from S3 bucket using `awscli` commands to deployment instance and appended to ssh authorized keys.
- vi. Used Terraform to run ansible playbooks.

2. Ansible

- i. Tomcat setup on deployment server.
- ii. Clone and build "Hello World" WAR file using `maven` on provisioning server.
- iii. Deploy WAR file and restart Tomcat on deployment server.

3. Jenkins

- i. Manual setup of pipeline in [previous step](#)
- ii. Testing and validating pipeline.

Terraform

1. Setup

1. Basic Setup

Inside `main.tf`, we ensure that Terraform uses a specific version for Terraform and our required providers (aws).

```
terraform{
  required_version = ">= 1.0" # semver
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "4.55.0"
    }
  }
}
```

Inside `variables.tf`, we setup the requirements for our Virtual Private Cloud and instance settings, as well as the AWS key `aws_terraform_keyname` for creating, managing and accessing our resources.

```
variable "ami_id" {
    default = "ami-00eeedc4036573771"
}
variable "vpc_id"{
    default="vpc-056d622c7a32e4729"
}
variable "aws_terraform_keyname" {
    default = "aws_terraform"
}
variable "aws_access_key" {
    default = "..."
}
variable "aws_secret_key" {
    default = "..."
}
variable "aws_token" {
    default = "..."
}
variable "region" {
    default = "region"
}
```

Inside `main.tf`, we use the variables we specified earlier to

```
locals {
    ami_id = "${var.ami_id}"
    vpc_id = "${var.vpc_id}"
    ssh_user = "ubuntu"
    key_name = "${var.aws_terraform_keyname}"
    private_key_path = "${var.aws_terraform_keyname}.pem"
}
```

2. Security Groups

Both provisioning and deployment servers require a security group. Since both require ports 22 , 80 , 8080 , we can use the same security group for both.

```
resource "aws_security_group" "server_sec_group" {
  name      = "server_sec_group"
  vpc_id    = local.vpc_id

  ingress {
    from_port = 22
    to_port   = 22
    protocol  = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    from_port = 80
    to_port   = 80
    protocol  = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    from_port = 8080
    to_port   = 8080
    protocol  = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  egress {
    from_port = 0
    to_port   = 0
    protocol  = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

3. S3 bucket

In `variables.tf` , the bucket name and acl are defined, with a public-read access control list (ACL).

```

variable "bucket_name" {
    default = "project2-reem-ansibledir"
}
variable "acl_value" {
    default = "public-read"
}

```

In `bucket.tf`, the `s3` bucket is defined:

```

resource "aws_s3_bucket" "b1" {
    bucket = "${var.bucket_name}"
    force_destroy=true
    tags = {
        Name          = "${var.bucket_name}"
        Environment = "Dev"
    }
}

resource "aws_s3_bucket_acl" "ansible_provision" {
    bucket = aws_s3_bucket.b1.id
    acl    = "${var.acl_value}"
}

```

4. IAM role

In order to facility access from the provisioning and deployment servers to the `s3` bucket, an Identity and Access Management policy and role need to be defined and applied to the server instances. Otherwise, downloading and uploading objects to and from the servers will not work with `awscli`.

In `main.tf`, a `aws_iam_policy` is defined that can allow for access to all objects under an AWS `s3` bucket.

```

resource "aws_iam_policy" "access-s3-policy" {
    name          = "S3-access-policy"
    description = "Provides permission to access S3"
}

```

```

policy = jsonencode({
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:*",
        "s3-object-lambda:*"
      ],
      "Resource": "*"
    }
  ]
})
}

```

A `aws_iam_role` is created that allows EC2 instances with this policy attached to assume this role.

```

resource "aws_iam_role" "access_bucket_role" {
  name = "access_bucket_role"

  assume_role_policy = jsonencode({
    "Version": "2012-10-17",
    "Statement": [
      {
        "Sid": "",
        "Effect": "Allow",
        "Principal": {
          "Service": [
            "ec2.amazonaws.com"
          ]
        },
        "Action": "sts:AssumeRole"
      }
    ]
  })
}

```

The defined `aws_iam_policy` is attached to the defined `aws_iam_role` through a `aws_iam_policy_attachment`.


```
resource "aws_iam_policy_attachment" "bucket-role-attach" {
  name      = "role-attachment"
  roles     = [aws_iam_role.access_bucket_role.name]
  policy_arn = aws_iam_policy.access-s3-policy.arn
}
```

The defined `aws_iam_role` is passed through an `aws_iam_instance_profile`.

```
resource "aws_iam_instance_profile" "access-s3-profile" {
  name = "access-s3-profile"
  role = aws_iam_role.access_bucket_role.name
}
```

2. Provisioning and Deployment Server Instances

1. Provisioning Server

In `main.tf`, the `aws_instance` for the provisioning server takes in the machine image `ami` for ubuntu, belongs to the same VPC and has the same security group applied, the same AWS key, and `aws_iam_instance_profile` that we created earlier in the [IAM section](#). Connection details are standard. Under `user_data`:

- We perform an `apt update`.
- Install `awscli` to upload and download to the `S3` bucket.
- Install `ansible` to provision the deployment server.
- Install `git` to clone the necessary repository
- Install `maven` to package the WAR file and
- Install `java` and `jenkins` as required.

To overcome interactive prompts that can halt application of the script when starting the instance, the `-y` option is provided.

```

resource "aws_instance" "ansible_provisioning_server" {
  ami = local.ami_id
  instance_type = "t2.micro"
  associate_public_ip_address = "true"
  vpc_security_group_ids = [aws_security_group.server_sec_group.id]
  key_name = local.key_name
  iam_instance_profile = aws_iam_instance_profile.access-s3-profile.name

  tags = {
    Name = "Ansible_Provisioning_Server"
  }

  connection {
    type = "ssh"
    host = self.public_ip
    user = local.ssh_user
    private_key = file(local.private_key_path)
    timeout = "4m"
  }
  user_data = <<EOF
#!/bin/bash
sudo apt-add-repository -y ppa:ansible/ansible
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io.key
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian-stable/ jenkins.io main
sudo apt update
sudo apt install unzip -y
sudo apt install ansible -y
sudo apt install openjdk-11-jdk -y
sudo apt install maven -y
sudo apt install awscli -y
sudo apt install git -y
sudo apt install jenkins -y
sudo systemctl start jenkins.service
EOF
}

```

2. Deployment Server

In `main.tf`, the `aws_instance` for the deployment server takes in the machine image `ami` for ubuntu, belongs to the same VPC and has the same security group applied, the same AWS key, and `aws_iam_instance_profile` that we created earlier in the [IAM section](#). Connection details are standard. Under `user_data`:

- We perform an `apt update`
- Install `acl` to support access control list querying and setting for S3 buckets through CLI.
- Install `awscli` to upload and download to the S3 bucket.

To overcome interactive prompts that can halt application of the script when starting the instance, the `-y` option is provided.

```
resource "aws_instance" "ansible_deployment" {
  ami = local.ami_id
  instance_type = "t2.micro"
  associate_public_ip_address = "true"
  vpc_security_group_ids = [aws_security_group.server_sec_group.id]
  key_name = local.key_name
  iam_instance_profile = aws_iam_instance_profile.access-s3-profile

  tags = {
    Name = "Ansible_Provisioning_Deployment"
  }

  connection {
    type = "ssh"
    host = self.public_ip
    user = local.ssh_user
    private_key = file(local.private_key_path)
    timeout = "4m"
  }

  user_data = <<EOF
#!/bin/bash
sudo apt update
sudo apt install acl -y
sudo apt install awscli -y
EOF
}
```

Creation of AWS Instances:

Instances (2) Info		Refresh	Connect	Instance state ▼	Actions ▼	Launch instances	▼
Find instance by attribute or tag (case-sensitive)							
Instance state = running X		Clear filters					
<input type="checkbox"/>	Name ▼	Instance ID	Instance state ▼	Instance type ▼	Status check	Alarm status	Av
<input type="checkbox"/>	Ansible_Provisioning_Deployment	i-0f07bb0914f1d86bc	Running	t2.micro	2/2 checks passed	No alarms +	us-
<input type="checkbox"/>	Ansible_Provisioning_Server	i-04d47ef85fe70f0a0	Running	t2.micro	2/2 checks passed	No alarms +	us-

Creation of AWS resources mentioned

```
aws_iam_instance_profile.access-s3-profile: Creating...
aws_s3_bucket.b1: Creating...
aws_security_group.server_sec_group: Creating...
aws_iam_instance_profile.access-s3-profile: Creation complete after 1s [id=access-s3-profile]
aws_security_group.server_sec_group: Creation complete after 4s [id=sg-0c9dd975ab5516c23]
aws_instance.ansible_deployment: Creating...
aws_instance.ansible_provisioning_server: Creating...
aws_s3_bucket.b1: Creation complete after 5s [id=project2-reem-ansibledir]
aws_s3_bucket_acl.ansible_provision: Creating...
aws_s3_bucket_acl.ansible_provision: Creation complete after 0s [id=project2-reem-ansibledir,public-read]
aws_instance.ansible_provisioning_server: Still creating... [10s elapsed]
aws_instance.ansible_deployment: Still creating... [10s elapsed]
aws_instance.ansible_deployment: Still creating... [20s elapsed]
aws_instance.ansible_provisioning_server: Still creating... [20s elapsed]
aws_instance.ansible_deployment: Creation complete after 28s [id=i-0f07bb0914f1d86bc]
local_file.tf_ansible_inventory: Creating...
local_file.tf_ansible_inventory: Creation complete after 0s [id=b6afb51af9b1124055df6f6ebe20fe18a8165db6]
data.archive_file.data_backup: Reading...
data.archive_file.data_backup: Read complete after 0s [id=b93e60c03969a5b5a4fac752034bef730534ecae]
aws_s3_object.upload_ansible: Creating...
aws_s3_object.upload_ansible: Creation complete after 0s [id=ansible.zip]
aws_instance.ansible_provisioning_server: Creation complete after 28s [id=i-04d47ef85fe70f0a0]
```

3. Inventory file in Ansible Provisioning Directory

Once the AWS EC2 instances are spun up, we would like to save the deployment server's private IP address in a `inventory` file under the directory `ansible_provisioning`.

Should we end up performing `terraform destroy`, I would like to remove the `ansible.zip` file that is generated [further on](#).

To that end, we add the following `local_file` resource, which writes out the private ip address under the group `[webserver]`. In order to avoid the interactive prompt when performing `ssh` to add the new host to the host group, we add the `StrictHostKeyChecking=no` argument to the variables of the host group.

```
resource "local_file" "tf_ansible_inventory" {
  content = <<-DOC
```

```

# Generated by Terraform mgmt configuration.

[webservers]
${aws_instance.ansible_deployment.private_ip}

[webservers:vars]
ansible_ssh_common_args="-o StrictHostKeyChecking=no"
DOC
filename = "./ansible_provisioning/inventory"

provisioner "local-exec" {
  when      = destroy
  command   = "rm ansible.zip"
}
}

```

4. Zipping Ansible Provisioning Directory

In order to upload the directory to the S3 bucket, the simplest way is to compress it prior to the upload. This can be done via the `archive_file` data source, with the source directory specified as `ansible_provisioning`, which contains all ansible playbooks, roles, and the newly-generated `inventory` file. I also add the `depends_on` meta-argument to allow the compression only to be performed when the `inventory` file has been created in the `ansible_provisioning` directory.

```

data "archive_file" "data_backup" {
  type      = "zip"
  source_dir = "./ansible_provisioning"
  output_path = "${var.ansible_zip}"
  depends_on = [
    local_file.tf_ansible_inventory
  ]
}

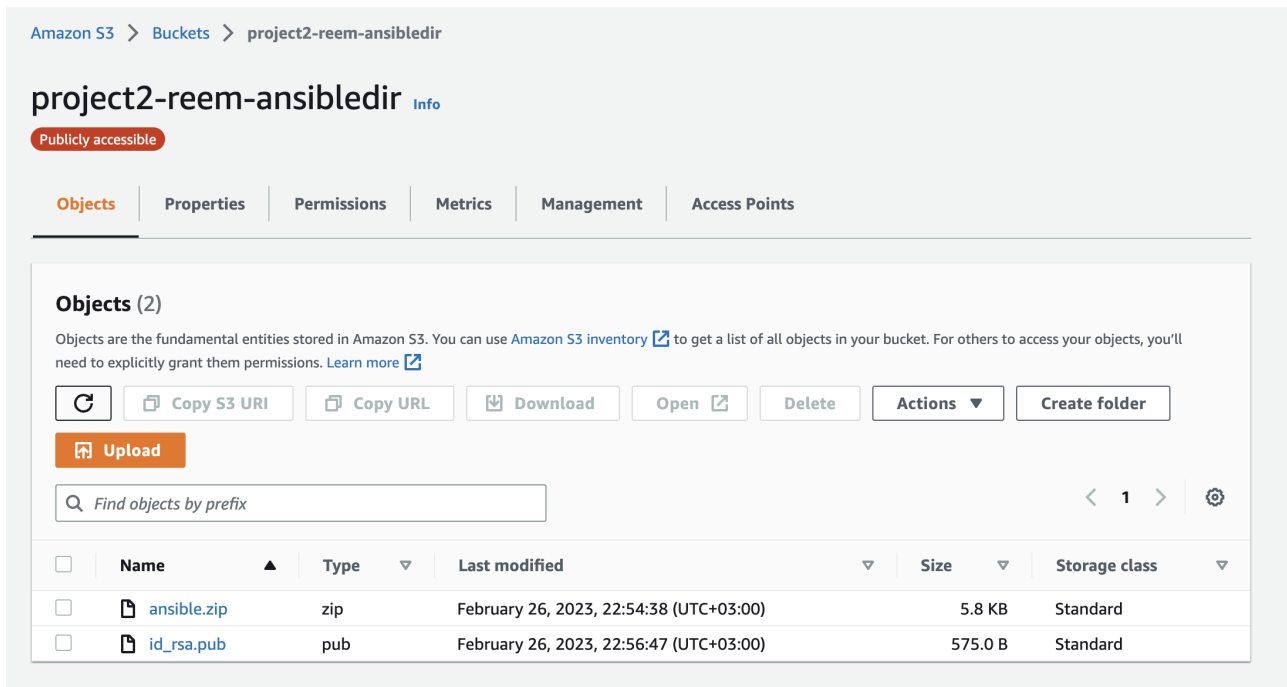
```

5. Upload to S3 bucket

To upload the zip file that was created in the [previous step](#), we use the `aws_s3_object` resource and specify the source to be the newly-created zip file. I also set the `depends_on` meta-argument to depend on the [previous step](#).

```
# Upload an object
resource "aws_s3_object" "upload_ansible" {
  bucket = aws_s3_bucket.b1.id
  key     = "${var.ansible_zip}"
  source  = "${var.ansible_zip}"
  #etag = filemd5("${var.ansible_zip}")
  depends_on = [
    data.archive_file.data_backup
  ]
}
```

Created S3 bucket on AWS:



Amazon S3 > Buckets > project2-reem-ansibledir

project2-reem-ansibledir [Info](#)

Publicly accessible

Objects | Properties | Permissions | Metrics | Management | Access Points

Objects (2)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

[Refresh](#) [Copy S3 URI](#) [Copy URL](#) [Download](#) [Open](#) [Delete](#) [Actions](#) [Create folder](#)

[Upload](#)

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	ansible.zip	zip	February 26, 2023, 22:54:38 (UTC+03:00)	5.8 KB	Standard
<input type="checkbox"/>	id_rsa.pub	pub	February 26, 2023, 22:56:47 (UTC+03:00)	575.0 B	Standard

6. Ansible Provisioning Directory Download to Provisioning Server

To start setting up the provisioning server and download the `ansible.zip` from our `S3` bucket, we will need to use a `null_resource` which connects to the aws instance `ansible_provisioning_server` that has just been created.

First, we generate the `ssh rsa` key-pair. To avoid interactive prompts, we first make sure we are using the shell `#!/bin/bash`, and pass `<<< y` to ensure yes is passed to the `ssh-keygen` prompt.

Next, we want to make sure that the `unzip` and `awscli` packages have been installed before proceeding, so we perform a while loop that sleeps 10 seconds if the desired command does not succeed.

```
sudo aws s3 cp /home/ubuntu/.ssh/id_rsa.pub s3://{aws_s3_bucket
```

This copies the newly-generated `ssh` public key to the `S3` bucket. This will be downloaded to the deployment server later.

```
sudo aws s3 cp s3://{aws_s3_bucket.b1.id}/ansible.zip .
```

This will download the `zip` file from our `S3` bucket to the current (`home`) directory in the provisioning server.

```
unzip ansible.zip
```

This will unzip the file that was just downloaded.

```
resource "null_resource" "InitialSetup" {  
  
  connection {  
    type = "ssh"  
    host = aws_instance.ansible_provisioning_server.public_ip  
    user = local.ssh_user  
    private_key = file(local.private_key_path)  
    timeout = "4m"  
  }  
}
```

```

provisioner "remote-exec" {
  inline = [
    "#!/bin/bash",
    "ssh-keygen -q -t rsa -N '' -f ~/.ssh/id_rsa <<<y >/dev/n",
    "cat ~/.ssh/id_rsa.pub",
    "while ! which unzip; do sleep 10; echo \"Sleeping for a l",
    "while ! which aws; do sleep 10; echo \"Sleeping for a bi",
    "sudo aws s3 cp /home/ubuntu/.ssh/id_rsa.pub s3://${aws_s",
    "sudo aws s3 cp s3://${aws_s3_bucket.b1.id}/ansible.zip ."
    "unzip ansible.zip",
  ]
}
depends_on = [
  aws_s3_object.upload_ansible,
  aws_instance.ansible_provisioning_server
]
}

```

Terraform execution of `null_resource.Initial_Setup` - ssh connection and generation of rsa key pair:

```

null_resource.InitialSetup (remote-exec): Connecting to remote host via SSH...
null_resource.InitialSetup (remote-exec): Host: 3.134.91.195
null_resource.InitialSetup (remote-exec): User: ubuntu
null_resource.InitialSetup (remote-exec): Password: false
null_resource.InitialSetup (remote-exec): Private key: true
null_resource.InitialSetup (remote-exec): Certificate: false
null_resource.InitialSetup (remote-exec): SSH Agent: true
null_resource.InitialSetup (remote-exec): Checking Host Key: false
null_resource.InitialSetup (remote-exec): Target Platform: unix
null_resource.InitialSetup: Still creating... [10s elapsed]
null_resource.InitialSetup (remote-exec): Connected!
null_resource.InitialSetup (remote-exec): ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQg0C10mLqtmboYt+yfuIk9dsH7gl/rrzdd4RqxNa+p4j28gCA9f4FIIm7620UWti0gIAu3Tgwsbwyd28
/DBGeElSVmcbnuNenHFs6ykalM9MGyG4IbCuqo2razyTXXDuvkILNMTbJYNcr1LEmXH+yELn7B01uIHskZCE1fs86K0u/w75qXrda9/wvrL6YsWjmq2+ft8215gnDsQajZmDnr0w9LazVJz0DYpuaKLI0If
driBli7w8MXnGK70ziiaeCDb9Hs1owcQX0F5TsbikqpZ4X1FW2EN29EzzTXeUYpkn0hAaFGnExKyhSviLnT0R59Gvm01XTFGAA2CLQ6EuH08thf0ZkIGrT0Y58L+sRtiG7IP4LQF8UWdoyqMhH7tMXfin0J
K/w1J26KN9FT0KMgxxYHxW4y2kWA5773huaeYbo+HYLp80ZkL226qmhRJzxyi/wgVqIBdHLSb0S93hFpvtEWUd10K6+za0fvI8wY7RpxmaEtrmJuAY2LsRHcy6vE= ubuntu@ip-172-31-37-9

```

Terraform execution of `null_resource.Initial_Setup` - upload public key, download zip and perform unzip:

```

null_resource.InitialSetup: Still creating... [2m0s elapsed]
null_resource.InitialSetup (remote-exec): Sleeping for a bit to make sure awscli is installed.
null_resource.InitialSetup (remote-exec): /usr/bin/aws
null_resource.InitialSetup (remote-exec): Completed 575 Bytes/575 Bytes (1.5 KiB/s) with 1 file(s) remaining
null_resource.InitialSetup (remote-exec): upload: .ssh/id_rsa.pub to s3://project2-reem-ansibledir/id_rsa.pub
null_resource.InitialSetup: Still creating... [2m10s elapsed]
null_resource.InitialSetup (remote-exec): Completed 5.8 KiB/5.8 KiB (17.7 KiB/s) with 1 file(s) remaining
null_resource.InitialSetup (remote-exec): download: s3://project2-reem-ansibledir/ansible.zip to ./ansible.zip
null_resource.InitialSetup (remote-exec): Archive: ansible.zip
null_resource.InitialSetup (remote-exec): inflating: deploy_war.yml
null_resource.InitialSetup (remote-exec): inflating: deploy_war_role/tasks/main.yml
null_resource.InitialSetup (remote-exec): inflating: deploy_war_role/tests/inventory
null_resource.InitialSetup (remote-exec): inflating: deploy_war_role/tests/test.yml
null_resource.InitialSetup (remote-exec): inflating: inventory
null_resource.InitialSetup (remote-exec): inflating: maven_build_role/tasks/main.yml
null_resource.InitialSetup (remote-exec): inflating: maven_build_role/tests/inventory
null_resource.InitialSetup (remote-exec): inflating: maven_build_role/tests/test.yml

```

Ansible provisioning server contents after performing unzip:


```

ubuntu@ip-172-31-35-76:~$ ls -l
total 36
-rw-r--r-- 1 root    root    5690 Feb 26 20:11 ansible.zip
-rw-r--r-- 1 ubuntu  ubuntu  260 Jan  1  2049 deploy_war.yml
drwxrwxr-x 4 ubuntu  ubuntu 4096 Feb 26 20:13 deploy_war_role
-rwxr-xr-x 1 ubuntu  ubuntu  214 Jan  1  2049 inventory
drwxrwxr-x 4 ubuntu  ubuntu 4096 Feb 26 20:13 maven_build_role
drwxr-xr-x 5 root    root    4096 Feb 26 20:14 sparkjava-war-example
drwxrwxr-x 5 ubuntu  ubuntu 4096 Feb 26 20:13 tomcat_installation_role
-rw-r--r-- 1 ubuntu  ubuntu  221 Jan  1  2049 tomcat_playbook.yml
ubuntu@ip-172-31-35-76:~$

```

i-067981a913e48681f (Ansible_Provisioning_Server)

PublicIPs: 52.14.74.102 PrivateIPs: 172.31.35.76

7. Copy SSH Public Key to Deployment Server

In order to allow ansible to connect to the deployment server from the provisioning server, we need to copy the public key from the s3 bucket to the deployment server.

```

resource "null_resource" "CopyPubTodeployment" {

  connection {
    type = "ssh"
    host = aws_instance.ansible_deployment.public_ip
    user = local.ssh_user
    private_key = file(local.private_key_path)
    timeout = "4m"
  }

  provisioner "remote-exec" {
    inline = [
      "#!/bin/bash",
      "while ! which aws; do sleep 10; echo \"Sleeping for a bit\";",
      "sudo aws s3 cp s3://${aws_s3_bucket.bucket}/id_rsa.pub .",
      "cat id_rsa.pub >> ~/.ssh/authorized_keys",
    ]
  }

  depends_on = [
    null_resource.InitialSetup
  ]
}

```

```
]
}
```

Terraform execution of `null_resource.CopyPubTodeployment` - ssh connection and download of public key:

```
null_resource.CopyPubTodeployment: Creating...
null_resource.CopyPubTodeployment: Provisioning with 'remote-exec'...
null_resource.CopyPubTodeployment (remote-exec): Connecting to remote host via SSH...
null_resource.CopyPubTodeployment (remote-exec): Host: 18.116.201.213
null_resource.CopyPubTodeployment (remote-exec): User: ubuntu
null_resource.CopyPubTodeployment (remote-exec): Password: false
null_resource.CopyPubTodeployment (remote-exec): Private key: true
null_resource.CopyPubTodeployment (remote-exec): Certificate: false
null_resource.CopyPubTodeployment (remote-exec): SSH Agent: true
null_resource.CopyPubTodeployment (remote-exec): Checking Host Key: false
null_resource.CopyPubTodeployment (remote-exec): Target Platform: unix
null_resource.CopyPubTodeployment (remote-exec): Connected!
null_resource.CopyPubTodeployment (remote-exec): /usr/bin/aws
null_resource.CopyPubTodeployment (remote-exec): Completed 575 Bytes/575 Bytes (2.2 KiB/s) with 1 file(s) remaining
null_resource.CopyPubTodeployment (remote-exec): download: s3://project2-reem-ansible/dir/id_rsa.pub to ./id_rsa.pub
null_resource.CopyPubTodeployment: Creation complete after 7s [id=2465459167498615139]
```

Above, we use a `null resource` to connect to the deployment server, check that `awscli` is installed, then copy the public key into `authorize_keys`.

```
sudo aws s3 cp s3://{aws_s3_bucket.b1.id}/id_rsa.pub .
cat id_rsa.pub >> ~/.ssh/authorized_keys
```

We will halt discussing our Terraform setup for now and jump to [ansible](#), where we run `ansible-playbook` commands from within Terraform using playbooks.

Ansible

1. Tomcat Setup on Deployment Server

Here is the folder structure for `ansible_provisioning` :

```
ansible_provisioning
├── tomcat_playbook.yml
├── deploy_war.yml
├── inventory
└── tomcat_installation_role
```

```

|   |— ...
|—  deploy_war_role
|   |— ...
|—  maven_build_role
|   |— ...

```

The two YAML files `tomcat_playbook.yml` and `deploy_war.yml` perform Tomcat installation and WAR deployment on the deployment server from the ansible provisioning server. They use the `inventory` file with our host groups and the roles defined the the `*_role` directories.

We use `tomcat_playbook.yml` and directory `tomcat_installation_role` . The YAML instructions here are a slightly-modified version of the ones from devopstricks.in.

1. `tomcat_playbook.yml`

```

- name: Install Tomcat
  hosts: webservers
  become: true
  vars:
    - TOMCAT9_URL: "https://dlcdn.apache.org/tomcat/tomcat-9/v9

  roles:
    - role: tomcat_installation_role

```

This specifies that the task is applied to the host group `webservers` and requires privilege escalation to `root` . It also defines the variable `TOMCAT9_URL` with the URL for tomcat version 9, and calls on the role defined in the `tomcat_installation_role` directory.

2. Tomcat Installation

```

tomcat_installation_role
|— files
|   |— localhost
|       |— host-manager.xml
|       |— manager.xml
|   |— tomcat-users.xml

```

```

├─ tasks
│   └─ main.yml
└─ tests
    ├── inventory
    └─ test.yml

```

1. Files:

As a quick-workaround to configuring gui-manager admin settings (not recommended):

- i. We modify and overwrite the built-in `tomcat-users.xml` configuration file as suggested with username: `admin` , password `cu1984` .
- ii. To allow external access to the manager, we need to modify `host-manager.xml` and `manager.xml` and place them under the localhost directory under `conf/Catalina` . You can see this in the next section.

2. Tasks:

Under `tomcat_installation_role/tasks/main.yml`

```

---
# tasks file for tomcat_installation_role
- name: Update apt-get repo and cache
  apt: update_cache=yes force_apt_get=yes cache_valid_time=3600

- name: Install Java
  apt:
    name: openjdk-11-jdk
    state: present

- name: Download Tomcat
  get_url:
    url: "{{ TOMCAT9_URL }}"
    dest: /tmp/
    validate_certs: no

- name: Creating Apache Tomcat home directory.

```

```

    command: mkdir /opt/tomcat

- name: Extract Tomcat
  shell: tar -xzvf /tmp/apache-tomcat-*tar.gz -C /opt/tomcat

- name: overwrite localhost in conf/Catalina
  copy:
    src: localhost/
    dest: /opt/tomcat/conf/Catalina/localhost

- name: overwrite tomcat-users.xml in conf
  copy:
    src: tomcat-users.xml
    dest: /opt/tomcat/conf/tomcat-users.xml

- name: Start Tomcat
  shell: /opt/tomcat/bin/startup.sh
  ignore_errors: true

- name: Wait for Tomcat to start
  wait_for:
    host: localhost
    port: 8080
    state: started

- name: Connect to Tomcat server on port 8080 and check status
  tags: test
  uri:
    url: http://localhost:8080
    register: result
    until: "result.status == 200"
    retries: 5
    delay: 10

```

This role is clear, but can be explained further in the link referenced above.

2. Build and Deploy WAR

We use `deploy_war.yml`, and directories `maven_build_role`, and `deploy_war_role`

```

---
- name: clone repo from github and build package with maven
  hosts: localhost
  become: true
  roles:
    - role: maven_build_role
- name: deploy war file to tomcat in deployment
  hosts: webserver
  become: true
  roles:
    - role: deploy_war_role

```

1. Clone and Build Hello World WAR File

```

maven_build_role
├── tasks
│   └── main.yml
└── tests
    ├── inventory
    └── test.yml

```

maven_build_role\tasks\main.yml

```

---
# tasks file for maven_build_role
- name: clone repo
  git:
    repo: https://github.com/kliakos/sparkjava-war-example.git
    dest: /home/ubuntu/sparkjava-war-example
    clone: yes
- name: run maven command mvn package
  shell:
    chdir: sparkjava-war-example
    cmd: mvn package

```

2. Deploy WAR File and Restart Tomcat on Deployment Server

```
deploy_war_role
├── tasks
│   └── main.yml
└── tests
    ├── inventory
    └── test.yml
```

deploy_war_role\tasks\main.yml

```
---
# tasks file for deploy_war_role
- name: clone repo
  copy:
    src: /home/ubuntu/sparkjava-war-example/target/sparkjava-he
    dest: /opt/tomcat/webapps/sparkjava-hello-world-1.0.war

- name: Shutdown Tomcat
  shell: /opt/tomcat/bin/shutdown.sh
  ignore_errors: true

- name: Start Tomcat
  shell: /opt/tomcat/bin/startup.sh
  ignore_errors: true

- name: Wait for Tomcat to start
  wait_for:
    host: localhost
    port: 8080
    state: started

- name: Connect to Tomcat server on port 8080 and check status :
  tags: test
  uri:
    url: http://localhost:8080
  register: result
  until: "result.status == 200"
  retries: 5
  delay: 10
```

3. Run Ansible Plays in Terraform

```
resource "null_resource" "FinalSetup" {

  connection {
    type = "ssh"
    host = aws_instance.ansible_provisioning_server.public_ip
    user = local.ssh_user
    private_key = file(local.private_key_path)
    timeout = "4m"
  }

  provisioner "remote-exec" {
    inline = [
      "#!/bin/bash",
      "ansible-playbook -i ./inventory tomcat_playbook.yml",
      "ansible-playbook -i inventory deploy_war.yml",
      "echo \"URL: http://${aws_instance.ansible_deployment.pub",
      "sudo cat /var/lib/jenkins/secrets/initialAdminPassword"
    ]
  }
  depends_on = [
    null_resource.CopyPubTodeployment
  ]
}
```

Terraform execution of `null_resource.FinalSetup` - running ansible playbooks and printing deployment server URL Hello World webapp:

```
null_resource.FinalSetup: Creating...
null_resource.FinalSetup: Provisioning with 'remote-exec'...
null_resource.FinalSetup (remote-exec): Connecting to remote host via SSH...
null_resource.FinalSetup (remote-exec): Host: 3.134.91.195
null_resource.FinalSetup (remote-exec): User: ubuntu
null_resource.FinalSetup (remote-exec): Password: false
null_resource.FinalSetup (remote-exec): Private key: true
null_resource.FinalSetup (remote-exec): Certificate: false
null_resource.FinalSetup (remote-exec): SSH Agent: true
null_resource.FinalSetup (remote-exec): Checking Host Key: false
null_resource.FinalSetup (remote-exec): Target Platform: unix
null_resource.FinalSetup (remote-exec): Connected!

null_resource.FinalSetup (remote-exec): PLAY [Install Tomcat] *****

null_resource.FinalSetup (remote-exec): TASK [Gathering Facts] *****
null_resource.FinalSetup (remote-exec): ok: [172.31.41.46]

null_resource.FinalSetup (remote-exec): TASK [tomcat_installation_role : Update apt-get repo and cache] *****
null_resource.FinalSetup (remote-exec): ok: [172.31.41.46]

null_resource.FinalSetup (remote-exec): TASK [tomcat_installation_role : Install Java] *****
null_resource.FinalSetup: Still creating... [10s elapsed]
null_resource.FinalSetup: Still creating... [20s elapsed]
null_resource.FinalSetup: Still creating... [30s elapsed]
```


Contents of Tomcat deployment server /opt/tomcat directory:

```
root@ip-172-31-41-46:/# ls -laF /opt/tomcat/
total 156
drwxr-xr-x 9 root root 4096 Feb 26 19:57 ./
drwxr-xr-x 3 root root 4096 Feb 26 19:57 ../
-rw-r----- 1 root root 19992 Feb 18 09:25 BUILDING.txt
-rw-r----- 1 root root 6210 Feb 18 09:25 CONTRIBUTING.md
-rw-r----- 1 root root 57092 Feb 18 09:25 LICENSE
-rw-r----- 1 root root 2333 Feb 18 09:25 NOTICE
-rw-r----- 1 root root 3398 Feb 18 09:25 README.md
-rw-r----- 1 root root 6901 Feb 18 09:25 RELEASE-NOTES
-rw-r----- 1 root root 16505 Feb 18 09:25 RUNNING.txt
drwxr-x--- 2 root root 4096 Feb 26 19:57 bin/
drwx----- 3 root root 4096 Feb 26 19:57 conf/
drwxr-x--- 2 root root 4096 Feb 26 19:57 lib/
drwxr-x--- 2 root root 4096 Feb 26 19:57 logs/
drwxr-x--- 2 root root 4096 Feb 26 19:57 temp/
drwxr-x--- 8 root root 4096 Feb 26 19:58 webapps/
drwxr-x--- 3 root root 4096 Feb 26 19:57 work/
root@ip-172-31-41-46:/# ls -laF /opt/tomcat/webapps/
total 2408
drwxr-x--- 8 root root 4096 Feb 26 19:58 ./
drwxr-xr-x 9 root root 4096 Feb 26 19:57 ../
drwxr-x--- 3 root root 4096 Feb 26 19:57 ROOT/
drwxr-x--- 16 root root 4096 Feb 26 19:57 docs/
drwxr-x--- 7 root root 4096 Feb 26 19:57 examples/
drwxr-x--- 6 root root 4096 Feb 26 19:57 host-manager/
drwxr-x--- 6 root root 4096 Feb 26 19:57 manager/
drwxr-x--- 4 root root 4096 Feb 26 19:58 sparkjava-hello-world-1.0/
-rw-r--r-- 1 root root 2427996 Feb 26 19:58 sparkjava-hello-world-1.0.war
root@ip-172-31-41-46:/#
```

i-0f07bb0914f1d86bc (Ansible_Provisioning_Deployment)

PublicIPs: 18.116.201.213 PrivateIPs: 172.31.41.46

Terraform execution of null_resource.FinalSetup - printing deployment server URL Hello World webapp:*

```
null_resource.FinalSetup (remote-exec): Hello World URL: http://18.222.151.122:8080/sparkjava-hello-world-1.0/hello
null_resource.FinalSetup (remote-exec): Jenkins Initial Password: c8ce9537401a44eda9006a2878a9f50a
null_resource.FinalSetup: Still creating... [1m40s elapsed]
null_resource.FinalSetup: Creation complete after 1m41s [id=6349886931661416439]

Apply complete! Resources: 10 added, 0 changed, 0 destroyed.

Outputs:
ip_deployment = "172.31.38.213"
ip_server = "172.31.35.76"
```