Hello and welcome back to the Terraform and Ansible for AWS course on Linux Academy!   
My name is Derek, and in this lesson, we’re going to really get our hands dirty and start creating our dev instance that will be used as our golden image for our launch configuration that the autoscaling group will use to deploy production instances. This image will be our golden image and also serve as our development instance. This instance will be a part of the public security group and will have SSH access enabled for the IP address specified by the variable localip. This instance will also be accessible via HTTP in order to perform the wordpress setup and verify it is working.

Let’s get started!

First, we will create a new aws\_instance resource:

resource “aws\_instance” “dev” {

instance\_type = “${var.dev\_instance\_type}”

ami = “${var.dev\_ami}”

tags {

Name = “dev”

}

key\_name = “${aws\_key\_pair.auth.id}”

vpc\_security\_group\_ids = [“${aws\_security\_group.public.id}”]

iam\_instance\_profile = “${aws\_iam\_instance\_profile.s3\_access.id}”

subnet\_id = “${aws\_subnet.public.id}”

So, let’s go through this:

We created the resource using aws\_resource, then we gave it an ID of “dev”. We then set the instance type to the variable “dev\_instance\_type” which you should define in your variables.tf and tfvars files. You can also use lookup values to do this, but we are going to keep this one simple. I will include documentation that shows how to do this with this course. Following this, we use the var.dev\_ami variable to define what ami to use and tag it with a name of dev. After that, we specify the key\_name, the vpc\_security\_group\_ids, and the iam\_instance\_profile to give it access to S3 buckets. Then we choose the public subnet to retrieve IP addresses from and give it access to the internet.

Ok, we have created the instance, but ansible still need to know what instance to connect to during the script! So, we will run a local provisioner in order to create the ansible hosts file that will contain the proper IP address and code bucket name:

provisioner “local-exec” {

command = “cat <<EOF > aws\_hosts

[dev]

${aws\_instance.dev.public\_ip}

[dev:vars]

s3code=${aws\_s3\_bucket.code.bucket}

EOF”

}

This piece of code will call the local-exec provisioner to export the public IP address of the new dev instance as well as the name of the code bucket you set to a file called aws\_hosts. This can, of course, be changed to whatever name you would like for the file. I have kept this file within the terransible directory to keep everything easy to manage. If you decide you want this file somewhere else, you can do so, just remember to use the proper path.

The ansible hosts file is now created, next up is running the actual playbook!

provisioner "local-exec" {

command = "sleep 6m && ansible-playbook -i aws\_hosts wordpress.yml"

}

}

As you can see, this is pretty straightforward. We simply use the local-exec provisioner to run the ansible playbook we created earlier using the aws\_hosts to define the target. This aws\_hosts file is the file that the previous provisioner added the proper IP address and code bucket to. There is one thing that you may notice is a little different. The Sleep condition. This condition was added because, at the time of this video, there wasn’t a very easy way to ensure the instance was up and running before ansible tried to connect. Ansible has plenty of wait conditions, but only if it can connect to the instance. You could potentially wrap the entire script in an ansible playbook, but it was easier in this case just to tell it to sleep for 6 minutes while waiting on the instance to become fully operational. You can experiment with shorter times, but this one worked 100% of the time for me.

That about does it for the dev instance! You can actually terraform apply at this point and you should have a completely working instance, but only accessible via its IP address. Please join me on the next lessons as we add the production instances and the DNS records required to make this infrastructure fully operational!