Lc\_asg

Hello! And welcome back to the Terraform and Ansible for AWS course on linuxacademy!

Now we are going to create the Launch configuration.

Before we can do that, we need to create the AMI from the dev instance we created. First, we want to create a random number that can be used to name the AMI to remove any chance of duplicates.

resource "random\_id" "ami" {

byte\_length = 8

}

Ok, that was easy enough. This just creates a random id of 8 characters.

Now we need to create the ami:

resource "aws\_ami\_from\_instance" "golden" {

name = "ami-${random\_id.ami.b64}"

source\_instance\_id = "${aws\_instance.dev.id}"

provisioner "local-exec" {

command = "cat <<EOF > userdata

#!/bin/bash

/usr/bin/aws s3 sync s3://${aws\_s3\_bucket.code.bucket} /var/www/html/

/bin/touch /var/spool/cron/root

sudo /bin/echo '\*/5 \* \* \* \* aws s3 sync s3://${aws\_s3\_bucket.code.bucket} /var/www/html/' >> /var/spool/cron/root

EOF"

}

}

I copied and pasted this to avoid any typos as it can be very easy to introduce a typo in these EOF scripts. First, we created the aws\_ami\_from\_instance resource and ggave it an id of “golden”. We then gave it a name ami-and the random integer we created above. This will prevent it from crashing if it tries to create a duplicate. We then specify the source\_instance\_id which is the dev instance.

Then for the fun part! We call the local-exec provisioner and have it create our userdata. This userdata script will be used by the launch configuration to automatically upload the cronjob we need to tell the production instances to automatically update their code from the code bucket every 5 minutes. As you can see, it runs an initial sync from the s3 bucket, then creates a cronjob that syncs the s3 code bucket every 5 minutes. So, if you make an update to the dev instance, this will automatically update the production instances every 5 minutes. If you wish to tweak this to a longer or shorter time, feel free to do so.

Now we create our Launch configuration:

resource "aws\_launch\_configuration" "lc" {

name\_prefix = "lc-"

image\_id = "${aws\_ami\_from\_instance.golden.id}"

instance\_type = "${var.lc\_instance\_type}"

security\_groups = ["${aws\_security\_group.private.id}"]

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

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

user\_data = "${file("userdata")}"

lifecycle {

create\_before\_destroy = true

}

}

First, we create the resource aws\_launch\_configuraiton and give it an ID of lc, then we give it a name prefix of lc-. This creates a random launch configuration name and prefixes it with “lc”, this prevents any duplicate names that can crash our script. We then specify the image\_id of the ami created from the dev instance. After that, we set the instance type, which is a variable we create, we add the security groups, the IAM instance profile to give it access to S3, and the key name. Next is the user\_data. To do this, we have used “file” to extract the data from the userdata file we created when we created the golden image and use it as userdata for the production instances. This is similar to what we did to create the key pair in an earlier lesson.

After this, we set a lifecycle policy of “create\_before\_destroy” which instructs terraform to create a new resource before we destroy the old one if you change something in your script.

And finally, we will create our autoscaling group. To do this, we first need to create another random\_id:

resource "random\_id" "asg" {

byte\_length = 8

}

Now we create the autoscaling group resource:

resource "aws\_autoscaling\_group" "asg" {

availability\_zones = ["${var.aws\_region}a", "${var.aws\_region}c"]

name = "asg-${aws\_launch\_configuration.lc.id}"

max\_size = "${var.asg\_max}"

min\_size = "${var.asg\_min}"

health\_check\_grace\_period = "${var.asg\_grace}"

health\_check\_type = "${var.asg\_hct}"

desired\_capacity = "${var.asg\_cap}"

force\_delete = true

load\_balancers = ["${aws\_elb.prod.id}"]

vpc\_zone\_identifier = ["${aws\_subnet.private1.id}", "${aws\_subnet.private2.id}"]

launch\_configuration = "${aws\_launch\_configuration.lc.name}"

tag {

key = "Name"

value = "asg-instance"

propagate\_at\_launch = true

}

lifecycle {

create\_before\_destroy = true

}

}

First, we create the aws\_autoscaling\_group resource with ID of asg. We use our two availability zones by specifying the region variable followed by “a” and “c” in this case, but this could vary based on your region. You can also create variables for all of this, but to save time, we have done it this way. We then specify the launch configuration, the max/min size, health checks, desired capacity, force\_delete…which allows it to be deleted whether it has instances running or not...and the load balancer we created earlier.

Next up is the vpc\_zone\_identifier. The name sounds confusing, but it is really just the subnets in which the new instances will be deployed. In this case, we are deploying them into the private1 and private2 subnets.

Then we specify the launch configuration we specified earlier, a Name tag of asg-instance for the instances. This will name all new instances “asg-instance” so we know they are the production instances and “propagate\_at\_launch which names them when they are launched.

Finally, we specify a lifecycle of “create\_before\_destroy = true” to create a new autoscaling group before this one is destroyed.

Wow, we have come a long way now. We are almost ready to deploy our infrastructure and get our site up and running! Join me in the next lesson as we create more of our infrastructure!