path.root

Returns the filesystem path of the root module.

path.cwd

Returns the filesystem path of the current working directory. In normal use of Terraform, this is the same as `path.root`, but some advanced uses of Terraform run it from a directory other than the root module directory, causing these paths to be different.

For the User Data script, you need a path relative to the module itself, so you should use `path.module` when calling the `templatefile` function in *modules/services/webserver-cluster/main.tf*:

```
  user_data = templatefile("${path.module}/user-data.sh", {
    server_port = var.server_port
    db_address  = data.terraform_remote_state.db.outputs.address
    db_port     = data.terraform_remote_state.db.outputs.port
  })
```

Inline Blocks

The configuration for some Terraform resources can be defined either as inline blocks or as separate resources. An *inline block* is an argument you set within a resource of the format:

```
resource "xxx" "yyy" {
  <NAME> {
    [CONFIG...]
  }
}
```

where `NAME` is the name of the inline block (e.g., `ingress`) and `CONFIG` consists of one or more arguments that are specific to that inline block (e.g., `from_port` and `to_port`). For example, with the `aws_security_group_resource`, you can define ingress and egress rules using either inline blocks (e.g., `ingress { … }`) or separate `aws_security_group_rule` resources.

If you try to use a mix of *both* inline blocks and separate resources, due to how Terraform is designed, you will get errors where the configurations conflict and overwrite one another. Therefore, you must use one or the other. Here's my advice: when creating a module, you should always prefer using separate resources.

The advantage of using separate resources is that they can be added anywhere, whereas an inline block can only be added within the module that creates a resource. So using solely separate resources makes your module more flexible and configurable.

For example, in the `webserver-cluster` module (*modules/services/webserver-cluster/main.tf*), you used inline blocks to define ingress and egress rules:

```
resource "aws_security_group" "alb" {
  name = "${var.cluster_name}-alb"

  ingress {
    from_port   = local.http_port
    to_port     = local.http_port
    protocol    = local.tcp_protocol
    cidr_blocks = local.all_ips
  }

  egress {
    from_port   = local.any_port
    to_port     = local.any_port
    protocol    = local.any_protocol
    cidr_blocks = local.all_ips
  }
}
```

With these inline blocks, a user of this module has no way to add additional ingress or egress rules from outside the module. To make your module more flexible, you should change it to define the exact same ingress and egress rules by using separate `aws_security_group_rule` resources (make sure to do this for both security groups in the module):

```
resource "aws_security_group" "alb" {
  name = "${var.cluster_name}-alb"
}

resource "aws_security_group_rule" "allow_http_inbound" {
  type              = "ingress"
  security_group_id = aws_security_group.alb.id

  from_port   = local.http_port
  to_port     = local.http_port
  protocol    = local.tcp_protocol
  cidr_blocks = local.all_ips
}

resource "aws_security_group_rule" "allow_all_outbound" {
  type              = "egress"
  security_group_id = aws_security_group.alb.id

  from_port   = local.any_port
  to_port     = local.any_port
  protocol    = local.any_protocol
  cidr_blocks = local.all_ips
}
```

You should also export the ID of the `aws_security_group` as an output variable in *modules/services/webserver-cluster/outputs.tf*:

```
output "alb_security_group_id" {
  value       = aws_security_group.alb.id
  description = "The ID of the Security Group attached to the
load balancer"
}
```

Now, if you needed to expose an extra port in just the staging environment (e.g., for testing), you can do this by adding an `aws_security_group_rule` resource to *stage/services/webserver-cluster/main.tf*:

```
module "webserver_cluster" {
  source = "../../../modules/services/webserver-cluster"

  # (parameters hidden for clarity)
}

resource "aws_security_group_rule" "allow_testing_inbound" {
  type              = "ingress"
  security_group_id =
module.webserver_cluster.alb_security_group_id

  from_port   = 12345
  to_port     = 12345
  protocol    = "tcp"
  cidr_blocks = ["0.0.0.0/0"]
}
```

Had you defined even a single ingress or egress rule as an inline block, this code would not work. Note that this same type of problem affects a number of Terraform resources, such as the following:

- `aws_security_group` and `aws_security_group_rule`
- `aws_route_table` and `aws_route`
- `aws_network_acl` and `aws_network_acl_rule`

At this point, you are finally ready to deploy your web server cluster in both staging and production. Run `terraform apply` as usual, and enjoy using two separate copies of your infrastructure.