Lab: Local Values

A local value assigns a name to an expression, so you can use it multiple times within a configuration without repeating it. The expressions in local values are not limited to literal constants; they can also reference other values in the configuration in order to transform or combine them, including variables, resource attributes, or other local values.

You can use local values to simplify your Terraform configuration and avoid repetition. Local values (locals) can also help you write a more readable configuration by using meaningful names rather than hard-coding values. If overused they can also make a configuration hard to read by future maintainers by hiding the actual values used.

Use local values only in moderation, in situations where a single value or result is used in many places and that value is likely to be changed in future. The ability to easily change the value in a central place is the key advantage of local values.

- Task 1: Create local values in a configuration block
- Task 2: Interpolate local values
- Task 3: Using locals with variable expressions
- Task 4: Using locals with terraform expressions and operators

Task 1: Create local values in a configuration block

Add local values to your `main.tf` module directory:

```
locals {
  service_name = "Automation"
  app_team     = "Cloud Team"
  createdby    = "terraform"
}
```

Task 2: Interpolate local values into your existing code

Update the `aws_instance` block inside your `main.tf` to add new tags to the `web_server` instance using interpolation.

```
...

 tags = {
    "Service"    = local.service_name
```


```
    "AppTeam"    = local.app_team
    "CreatedBy" = local.createdby
  }

...
```

After making these changes, rerun `terraform plan`. You should see that there will be some tagging updates to your server instances. Execute a `terraform apply` to make these changes.

Task 3: Using locals with variable expressions

Expressions in local values are not limited to literal constants; they can also reference other values in the module in order to transform or combine them, including variables, resource attributes, or other local values.

Add another local values block to your `main.tf` module configuration that references the local values set in the previous portion of the lab.

```
locals {
  # Common tags to be assigned to all resources
  common_tags = {
    Name       = var.server_name
    Owner      = local.team
    App        = local.application
    Service    = local.service_name
    AppTeam    = local.app_team
    CreatedBy = local.createdby
 }
}
```

Update the `aws_instance` tags block inside your `main.tf` to reference the `local.common_tags` value.

```
resource "aws_instance" "web_server" {
  ami                         = data.aws_ami.ubuntu.id
  instance_type               = "t2.micro"
  subnet_id                   = aws_subnet.public_subnets["public_subnet_1
     "].id
  ...
  tags = local.common_tags
}
```

After making these changes, rerun `terraform plan`. You should see that there are no changes to apply, which is correct, since the values contain the same values we had previously hard-coded, but now we are grabbing those values through the use of locals variables.

_Note: You may see that the output mentions that `local_file.private_key_pem` will be created on each run, including this one but you can ignore that. Scroll up and note that there are no additional changes to the configuration.

```
...

No changes. Infrastructure is up-to-date.

This means that Terraform did not detect any differences between your
configuration and real physical resources that exist. As a result, no
actions need to be performed.
```