Lab: Terraform State Refresh

The `terraform refresh` command reads the current settings from all managed remote objects found in Terraform state and updates the Terraform state to match.

Refreshing state is the first step of the `terraform plan` process: read the current state of any already-existing remote objects to make sure that the Terraform state is up-to-date. If you wish to only perform the first part of the `terraform plan` process you can execute the plan with a `-refresh-only`.

- Task 1: `terraform refresh` command
- Task 2: Show state before refresh
- Task 3: Introduce drift
- Task 4: Refresh state to detect drift
- Task 5: Show state after refresh
- Task 6: Controlling state refresh

Task 1: `terraform refresh` command

The `terraform refresh` command won't modify your real remote objects, but it will modify the the Terraform state. You shouldn't typically need to use this command, because Terraform automatically performs the same refreshing actions as a part of creating a plan in both the terraform plan and terraform apply commands. Once your infrastructure is deployed you can run a `terraform refresh` to read the current settings of objects that terraform is managing.

Deploy your infrastructure and run a `terraform refresh`

```
terraform init
terraform apply
terraform refresh
```

Task 2: Show State before Refresh

Use the `terraform state` commands to show the state of an EC2 instance before a state refresh is applied.

```
terraform state list

data.aws_ami.ubuntu
data.aws_availability_zones.available
data.aws_region.current
```


```
aws_eip.nat_gateway_eip
aws_instance.ubuntu_server
aws_instance.web_server
aws_instance.web_server_2
aws_internet_gateway.internet_gateway
aws_key_pair.generated
aws_nat_gateway.nat_gateway
aws_route_table.private_route_table
aws_route_table.public_route_table
aws_route_table_association.private["private_subnet_1"]
aws_route_table_association.private["private_subnet_2"]
aws_route_table_association.private["private_subnet_3"]
aws_route_table_association.public["public_subnet_1"]
aws_route_table_association.public["public_subnet_2"]
aws_route_table_association.public["public_subnet_3"]
aws_security_group.ingress-ssh
aws_security_group.vpc-ping
aws_security_group.vpc-web
aws_subnet.private_subnets["private_subnet_1"]
aws_subnet.private_subnets["private_subnet_2"]
aws_subnet.private_subnets["private_subnet_3"]
aws_subnet.public_subnets["public_subnet_1"]
aws_subnet.public_subnets["public_subnet_2"]
aws_subnet.public_subnets["public_subnet_3"]
aws_vpc.vpc
local_file.private_key_pem
random_string.random
tls_private_key.generated
module.server.aws_instance.web
module.server_subnet_1.aws_instance.web
```

```
terraform state show aws_instance.web_server

resource "aws_instance" "web_server" {
    ami                                  = "ami-04505e74c0741db8d"
    arn                                  = "arn:aws:ec2:us-east
       -1:508140242758:instance/i-0fdab85c96f92f0e9"
    associate_public_ip_address          = true
    availability_zone                    = "us-east-1b"
    cpu_core_count                       = 1
    cpu_threads_per_core                 = 1
    disable_api_termination              = false
    ebs_optimized                        = false
    get_password_data                    = false
    hibernation                          = false
    id                                   = "i-0fdab85c96f92f0e9"
    instance_initiated_shutdown_behavior = "stop"
    instance_state                       = "running"
```

```
    instance_type                        = "t2.micro"
    ipv6_address_count                   = 0
    ipv6_addresses                       = []
    key_name                             = "MyAWSKey"
    monitoring                           = false
    primary_network_interface_id         = "eni-02b1bae8ab8850b6c"
    private_dns                          = "ip-10-0-101-122.ec2.internal"
    private_ip                           = "10.0.101.122"
    public_dns                           = "ec2-3-236-76-73.compute-1.
        amazonaws.com"
    public_ip                            = "3.236.76.73"
    secondary_private_ips                = []
    security_groups                      = [
        "sg-0609dcff9a0ad9607",
        "sg-06b21df8181c04026",
        "sg-0b3632ec4e1686072",
    ]
    source_dest_check                    = true
    subnet_id                            = "subnet-0422128dd82c11546"
    tags                                 = {
        "Name" = "Web EC2 Server"
    }
    tags_all                             = {
        "Name" = "Web EC2 Server"
    }
    tenancy                              = "default"
    vpc_security_group_ids               = [
        "sg-0609dcff9a0ad9607",
        "sg-06b21df8181c04026",
        "sg-0b3632ec4e1686072",
    ]

    capacity_reservation_specification {
        capacity_reservation_preference = "open"
    }

    credit_specification {
        cpu_credits = "standard"
    }

    enclave_options {
        enabled = false
    }

    metadata_options {
        http_endpoint               = "enabled"
        http_put_response_hop_limit = 1
        http_tokens                 = "optional"
    }
```


```
    root_block_device {
        delete_on_termination = true
        device_name           = "/dev/sda1"
        encrypted             = false
        iops                  = 100
        tags                  = {}
        throughput            = 0
        volume_id             = "vol-012dff0c9b3ce206e"
        volume_size           = 8
        volume_type           = "gp2"
    }
}
```

Task 3: Introduce Drift

Change the tag of a given object inside the AWS Console

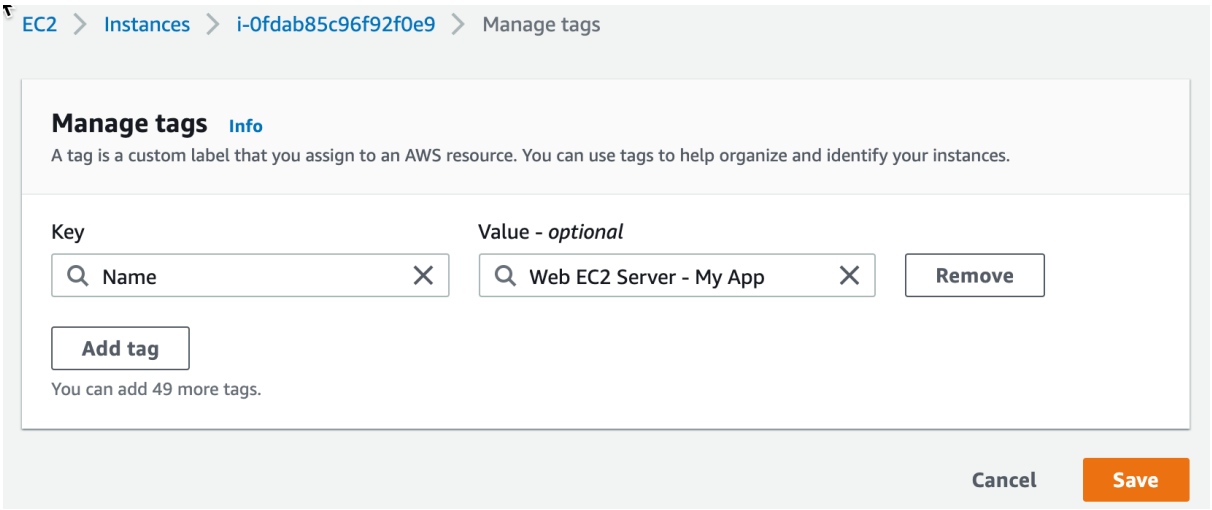

Figure 1: AWS Tag - Drift

Task 4: Refresh State to detect drift

Run a `terraform refresh` command to update Terraform state

```
terraform refresh
```


Task 5: Show State after Refresh

Use the `terraform state` commands to show the state of an EC2 instance after a state refresh is applied.

```
terraform state show aws_instance.web_server

  tags                                   = {
        "Name" = "Web EC2 Server - My App"
    }
```

You will see that object within terraform state has been updated to reflect the drift introduced by applying changes outside of the standard Terraform workflow. If these changes are not updated within the Terraform configuration the drift will be highlighted during the next `terraform plan` and will be reverted the next time a `terraform apply` is performed.

```
terraform plan

Terraform will perform the following actions:

  # aws_instance.web_server will be updated in-place
  ~ resource "aws_instance" "web_server" {
        id                                     = "i-0fdab85c96f92f0e9"
      ~ tags                                   = {
          ~ "Name" = "Web EC2 Server - My App" -> "Web EC2 Server"
        }
      ~ tags_all                               = {
          ~ "Name" = "Web EC2 Server - My App" -> "Web EC2 Server"
        }
        # (28 unchanged attributes hidden)




        # (5 unchanged blocks hidden)
    }

Plan: 0 to add, 1 to change, 0 to destroy.
```

```
terraform apply

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes
```

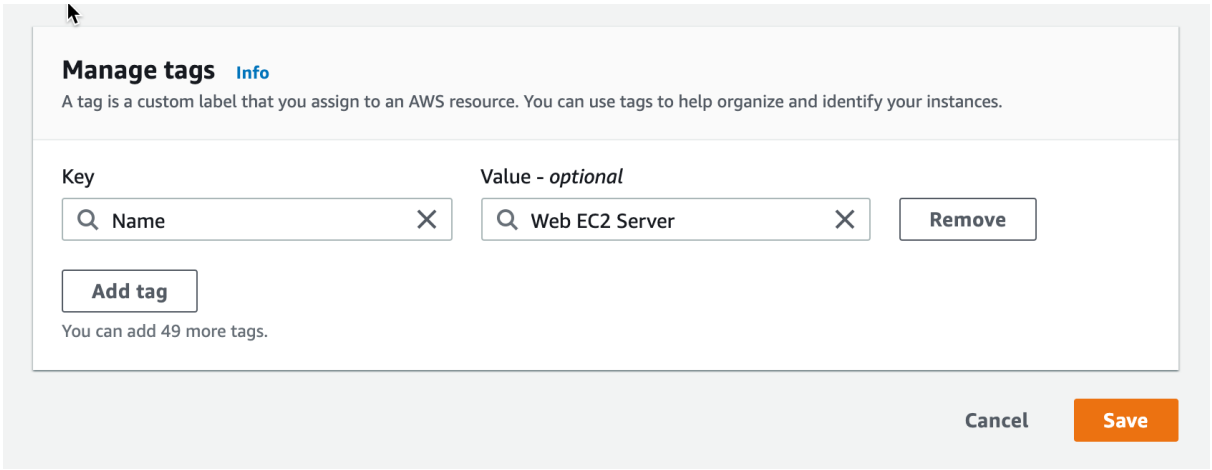

Figure 2: AWS Tag - Revert

You can see the out of band changes were reverted to remediate the drift introduced outside of the standard Terraform workflow. Alternatively, you could keep these changes but would need to update the Terraform configuration appropriately.

Task 6: Controlling state refresh

Automatically applying the effect of a refresh is risky, because if you have misconfigured credentials for one or more providers then the provider may be misled into thinking that all of the managed objects have been deleted, and thus remove all of the tracked objects without any confirmation prompt. This is is why the `terraform refresh` command has been deprecated.

Instead it is recommended to use the `-refresh-only` option to get the same effect as a `terraform refresh` but with the opportunity to review the the changes that Terraform has detected before committing them to the state. The first step of the `terraform plan` process is to read the current state of any already-existing remote objects to make sure that the Terraform state is up-to-date. It will then compare the current configuration to the prior state and note differences. If you wish to only perform the first part of this flow you can execute the plan with a `-refresh-only` option to make the.

Change the tag of a given object inside the AWS Console

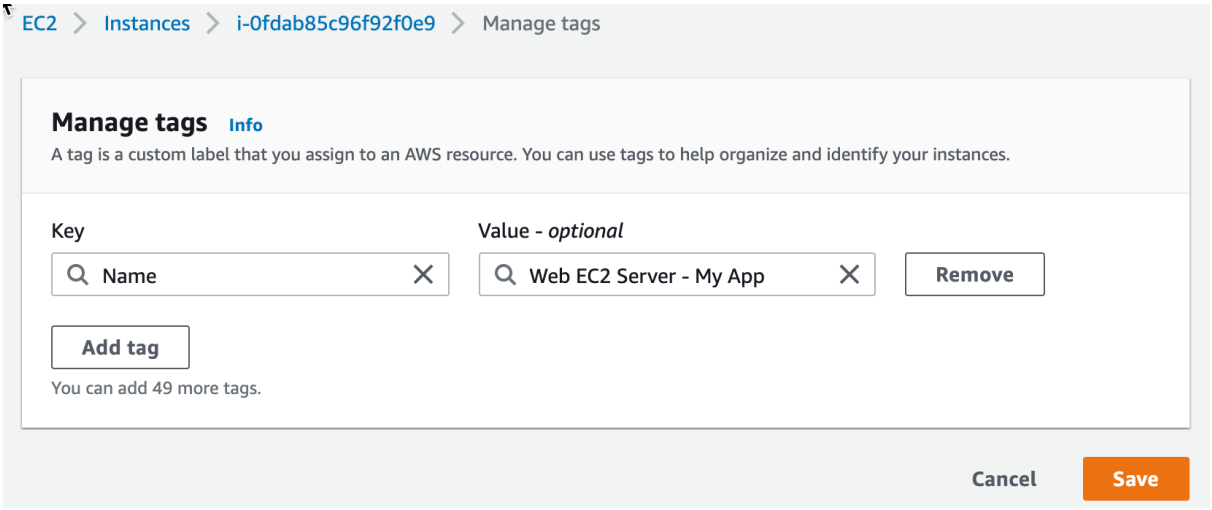

Figure 3: AWS Tag - Drift

```
terraform plan -refresh-only

...

Note: Objects have changed outside of Terraform

Terraform detected the following changes made outside of Terraform since
   the last "terraform apply":

  # aws_instance.web_server has changed
  ~ resource "aws_instance" "web_server" {
        id                                   = "i-0fdab85c96f92f0e9"
      ~ tags                                 = {
          ~ "Name" = "Web EC2 Server" -> "Web EC2 Server - My App"
        }
      ~ tags_all                             = {
          ~ "Name" = "Web EC2 Server" -> "Web EC2 Server - My App"
        }
        # (28 unchanged attributes hidden)




        # (5 unchanged blocks hidden)
    }


This is a refresh-only plan, so Terraform will not take any actions to
```


```
    undo these. If you were expecting these changes then you can apply this
     plan to record the updated values in the Terraform state without
    changing any remote objects.
```

Terraform's Refresh-only mode goal is only to update the Terraform state and any root module output values to match changes made to remote objects outside of Terraform. This can be useful if you've intentionally changed one or more remote objects outside of the usual workflow (e.g. while responding to an incident) and you now need to reconcile Terraform's records with those changes.

To update Terraform's state with these items that were modified outside of the usual workflow run a `terraform apply -refresh-only`

```
terraform apply -refresh-only


Note: Objects have changed outside of Terraform

Terraform detected the following changes made outside of Terraform since
    the last "terraform apply":

  # aws_instance.web_server has changed
  ~ resource "aws_instance" "web_server" {
        id                                   = "i-0fdab85c96f92f0e9"
      ~ tags                                 = {
          ~ "Name" = "Web EC2 Server" -> "Web EC2 Server - My App"
        }
      ~ tags_all                             = {
          ~ "Name" = "Web EC2 Server" -> "Web EC2 Server - My App"
        }
        # (28 unchanged attributes hidden)




        # (5 unchanged blocks hidden)
    }


This is a refresh-only plan, so Terraform will not take any actions to
    undo these. If you were expecting these
changes then you can apply this plan to record the updated values in the
    Terraform state without changing any remote
objects.

Would you like to update the Terraform state to reflect these detected
    changes?
```

```
  Terraform will write these changes to the state without modifying any
     real infrastructure.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes
```

Use the `terraform state` commands to show the state of an EC2 instance after a state refresh is applied.

```
terraform state show aws_instance.web_server


  tags                                    = {
        "Name" = "Web EC2 Server - My App"
    }
```

If this is permanent change then you would need to update the object within Terraform's configuration to reflect this change otherwise it will be reverted back during the next apply. The `terraform refresh` and `-refresh-only` are useful for being able to help detect and handle drift within an environment.

Run a `terraform apply` to eliminate the drift within the environment.

```
terraform apply

Terraform used the selected providers to generate the following execution
    plan. Resource actions are indicated with
the following symbols:
  ~ update in-place

Terraform will perform the following actions:

  # aws_instance.web_server will be updated in-place
  ~ resource "aws_instance" "web_server" {
        id                                   = "i-0fdab85c96f92f0e9"
      ~ tags                                 = {
          ~ "Name" = "Web EC2 Server - My App" -> "Web EC2 Server"
        }
      ~ tags_all                             = {
          ~ "Name" = "Web EC2 Server - My App" -> "Web EC2 Server"
        }
        # (28 unchanged attributes hidden)
```


```
        # (5 unchanged blocks hidden)
    }

Plan: 0 to add, 1 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_instance.web_server: Modifying... [id=i-0fdab85c96f92f0e9]
aws_instance.web_server: Modifications complete after 2s [id=i-0
   fdab85c96f92f0e9]

Apply complete! Resources: 0 added, 1 changed, 0 destroyed.
```

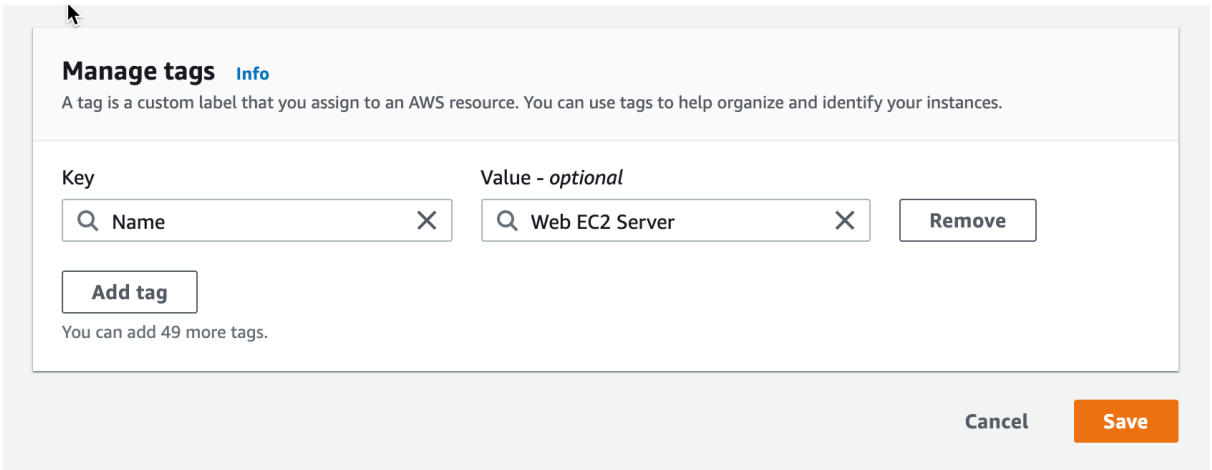

Figure 4: AWS Tag - Eliminate Drift