**Terraform Code (Desired State)**

# 1. Define a Security Group allowing SSH (22) and HTTP (80) inbound

resource "aws\_security\_group" "web\_sg" {

name = "web-sg"

description = "Allow SSH and HTTP"

vpc\_id = "vpc-0a1b2c3d4e5f6g7h" # replace with your VPC ID

ingress {

description = "SSH from anywhere"

from\_port = 22

to\_port = 22

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

ingress {

description = "HTTP from anywhere"

from\_port = 80

to\_port = 80

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

egress {

description = "Allow all outbound traffic"

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

}

tags = {

Name = "web-sg"

Env = "prod"

}

}

# 2. Create an EC2 instance and attach the Security Group above

resource "aws\_instance" "web" {

ami = "ami-0abcdef1234567890" # replace with a valid AMI

instance\_type = "t3.micro"

subnet\_id = "subnet-0123456789abcdef0" # replace with your subnet ID

# Attach the web\_sg defined above

vpc\_security\_group\_ids = [aws\_security\_group.web\_sg.id]

tags = {

Name = "example-web-instance"

Env = "prod"

}

}

* **Desired State** (in Terraform code above):
  1. A Security Group (aws\_security\_group.web\_sg) exists with exactly two inbound rules:
     + TCP 22 from 0.0.0.0/0
     + TCP 80 from 0.0.0.0/0
     + All outbound traffic allowed.
  2. An EC2 instance (aws\_instance.web) exists that is launched in the specified subnet and has its vpc\_security\_group\_ids set to exactly ["${aws\_security\_group.web\_sg.id}"].

When you run terraform apply with this configuration:

* Terraform will create the security group with those rules, then launch an EC2 instance and attach that SG.

**2. How Terraform Tracks “Current State”**

1. **State File**
   * After the first terraform apply, Terraform writes a terraform.tfstate entry for both:
     + aws\_security\_group.web\_sg (it records its ID, AMI attributes, rules, etc.)
     + aws\_instance.web (it records the instance ID—e.g., i-0a1b2c3d4e5f6g7h8—and the fact that vpc\_security\_group\_ids = ["sg-0h7g6f5e4d3c2b1a0"]).
2. **Live AWS Query** (during subsequent plans)
   * When you run terraform plan (or apply again), Terraform does a “refresh”:
     + It looks at the state file entry for aws\_security\_group.web\_sg (say, sg-0h7g6f5e4d3c2b1a0) and calls AWS’s DescribeSecurityGroups(sg-0h7g6f5e4d3c2b1a0).
     + It looks at the state file entry for aws\_instance.web (say, i-0a1b2c3d4e5f6g7h8) and calls AWS’s DescribeInstances(i-0a1b2c3d4e5f6g7h8).
   * Now Terraform knows, “Here’s what I *last recorded* in my state file, and here’s what AWS actually reports today.”

**3. Example Scenarios: Detecting Drift & Reconciling**

Below are two scenarios where the **current state** in AWS might drift away from your **desired state** in Terraform, and how Terraform detects and corrects it.

**Scenario A: Someone Manually Removes the SSH Rule**

1. **Initial “Desired State”** (in code):
   * The SG (web-sg) should allow inbound TCP 22 and TCP 80 from 0.0.0.0/0.
2. **After Apply**:
   * In AWS: sg-0h7g6f5e4d3c2b1a0 has inbound ports 22 & 80.
3. **Manual Change in AWS Console**:
   * An administrator deletes the SSH (port 22) rule. Now the live SG only has port 80 open.
4. **Run terraform plan**:
   * Terraform reads state: last-known SG inbound rules = [22, 80].
   * **Refresh**: AWS reports inbound rules = [only port 80].
   * **Diff**: Desired (22, 80) vs. Live (only 80) → Terraform sees “Inbound rule for port 22 is missing.”
   * **Plan Output**:

~ aws\_security\_group.web\_sg

ingress.0.from\_port: 22 -> 22 # it notices the rule that should be there isn’t.

ingress.0.to\_port: 22 -> 22

… (Terraform will show that it needs to re-create or re-add the SSH ingress block)

* + If you run terraform apply, Terraform issues an AuthorizeSecurityGroupIngress API call to add back the port 22 rule. Now AWS matches your desired state again.

**Scenario B: Someone Detaches the Security Group from the EC2 Instance**

1. **Initial “Desired State”** (in code):
   * EC2 instance (i-0a1b2c3d4e5f6g7h8) should have vpc\_security\_group\_ids = ["sg-0h7g6f5e4d3c2b1a0"].
2. **After Apply**:
   * AWS: instance is running with exactly that SG attached.
3. **Manual Change in AWS Console**:
   * An administrator edits the instance’s network settings and replaces web-sg with some other SG (e.g., sg-0x1y2z3a4b5c6d7e8f). Now the live EC2 has only sg-0x1y2z3a4b5c6d7e8f.
4. **Run terraform plan**:
   * Terraform state: i-0a1b2c3d4e5f6g7h8 is attached to sg-0h7g6f5e4d3c2b1a0.
   * **Refresh**: AWS API says: i-0a1b2c3d4e5f6g7h8 is actually attached to sg-0x1y2z3a4b5c6d7e8f.
   * **Diff**:
     + Desired: SG = ["sg-0h7g6f5e4d3c2b1a0"]
     + Live: SG = ["sg-0x1y2z3a4b5c6d7e8f"]
     + Terraform will plan to detach sg-0x1y2z3a4b5c6d7e8f and re-attach sg-0h7g6f5e4d3c2b1a0.
   * **Plan Output** might show something like:

~ aws\_instance.web

vpc\_security\_group\_ids.0: "sg-0x1y2z3a4b5c6d7e8f" -> "sg-0h7g6f5e4d3c2b1a0"

* + On terraform apply, Terraform calls ModifyInstanceAttribute to replace the SG. After it finishes, the instance is back with the correct security group.

**4. Putting It All Together**

* **Desired State** = “What I tell Terraform in my .tf files.”
  + In our example:
    1. Security Group named web-sg with exactly two ingress rules (SSH and HTTP).
    2. EC2 instance launched in a subnet with web-sg attached.
* **Current State** = “What Terraform thinks exists (state file) + what AWS actually has right now (live query).”

Whenever you run:

terraform plan

Terraform does:

1. **Reads the terraform.tfstate**:
   * Knows “I created a security group with ID sg-0h7g6f5e4d3c2b1a0, ingress = [22, 80], egress = [all].”
   * Knows “I created an EC2 instance with ID i-0a1b2c3d4e5f6g7h8 that had vpc\_security\_group\_ids = ['sg-0h7g6f5e4d3c2b1a0'].”
2. **Calls AWS APIs** (DescribeSecurityGroups and DescribeInstances):
   * Fetches the *actual* rule set for sg-0h7g6f5e4d3c2b1a0 and the *actual* SG list attached to i-0a1b2c3d4e5f6g7h8.
3. **Compares**:
   * Desired (.tf) vs. Live (AWS query).
   * Any mismatch shows up as a “difference” in the plan.
4. **Outputs a Plan**:
   * If there’s no mismatch, it says “No changes. Infrastructure is up-to-date.”
   * If there is drift (manual modification), it shows exactly which block (ingress/egress or vpc\_security\_group\_ids) must be updated to restore your desired state.
5. **On Apply** (if you confirm), Terraform issues the necessary AWS calls to reconcile AWS → Desired.