To check if the key pair already exists before creating a new one, you can use a combination of Terraform and AWS CLI commands to achieve this. Unfortunately, Terraform itself does not support conditional resource creation based on the existence of an AWS resource directly within its configuration language. However, you can use external data sources and null resources to work around this.

Here’s a way to check for the existence of the key pair and conditionally create it:

1. **Create a script to check if the key pair exists**.
2. **Use a null resource to run the script**.
3. **Use the output of the script to determine if the key pair should be created**.

**Step-by-Step Configuration**

**1. Create a Script to Check for the Key Pair**

Create a script check\_key\_pair.sh to check if the key pair exists:

sh

Copy code

#!/bin/bash

KEY\_NAME=$1

aws ec2 describe-key-pairs --key-names "$KEY\_NAME" > /dev/null 2>&1

if [ $? -eq 0 ]; then

echo "exists"

else

echo "not\_exists"

fi

Make sure this script has execute permissions:

sh

Copy code

chmod +x check\_key\_pair.sh

**2. Use a null\_resource to Run the Script**

Add the following to your Terraform configuration:

hcl

Copy code

provider "local" {}

data "external" "check\_key\_pair" {

program = ["bash", "${path.module}/check\_key\_pair.sh", "my-key"]

}

resource "null\_resource" "check\_key\_pair" {

triggers = {

key\_pair\_status = data.external.check\_key\_pair.result["key\_pair\_status"]

}

}

**3. Conditionally Create the Key Pair**

Use the result from the script to conditionally create the key pair:

hcl

Copy code

resource "tls\_private\_key" "rsa" {

count = data.external.check\_key\_pair.result["key\_pair\_status"] == "not\_exists" ? 1 : 0

algorithm = "RSA"

rsa\_bits = 4096

}

resource "aws\_key\_pair" "mak" {

count = data.external.check\_key\_pair.result["key\_pair\_status"] == "not\_exists" ? 1 : 0

key\_name = "my-key"

public\_key = tls\_private\_key.rsa[0].public\_key\_openssh

}

resource "local\_file" "mak-key" {

count = data.external.check\_key\_pair.result["key\_pair\_status"] == "not\_exists" ? 1 : 0

content = tls\_private\_key.rsa[0].private\_key\_pem

filename = "my-key.pem"

}

**4. Update Instance Configuration to Use Existing Key**

Finally, update your instance configuration to use the existing key name directly:

hcl

Copy code

resource "aws\_instance" "linux" {

ami = "ami-08a0d1e16fc3f61ea"

instance\_type = "t2.micro"

key\_name = "my-key"

subnet\_id = aws\_subnet.subnet\_1.id

vpc\_security\_group\_ids = [aws\_security\_group.terra1\_sg.id]

associate\_public\_ip\_address = true # Enable auto-assigning public IP

user\_data = <<-EOF

#!/bin/bash

sudo yum -y update

sudo yum install docker -y

sudo systemctl start docker

sudo systemctl enable docker

sudo docker pull maksoft121/makfirstimagenginx:challange

sudo docker container run -itd -p 80:80 maksoft121/makfirstimagenginx:challange

EOF

}

**Full Terraform Configuration**

Here’s the complete Terraform configuration incorporating the above steps:

hcl

Copy code

provider "local" {}

variable "vpc\_cidr" {

default = "10.0.0.0/16"

}

variable "subnet1\_cidr" {

default = "10.0.1.0/24"

}

variable "subnet2\_cidr" {

default = "10.0.2.0/24"

}

# Create VPC

resource "aws\_vpc" "testvpc" {

cidr\_block = var.vpc\_cidr

instance\_tenancy = "default"

tags = {

Name = "test-VPC"

}

}

# Creating Internet Gateway

resource "aws\_internet\_gateway" "testgateway" {

vpc\_id = aws\_vpc.testvpc.id

tags = {

Name = "test-igw"

}

}

# Creating Route Table

resource "aws\_route\_table" "route" {

vpc\_id = aws\_vpc.testvpc.id

route {

cidr\_block = "0.0.0.0/0"

gateway\_id = aws\_internet\_gateway.testgateway.id

}

tags = {

Name = "Route-table1"

}

}

# Check if Key Pair Exists

data "external" "check\_key\_pair" {

program = ["bash", "${path.module}/check\_key\_pair.sh", "my-key"]

}

resource "null\_resource" "check\_key\_pair" {

triggers = {

key\_pair\_status = data.external.check\_key\_pair.result["key\_pair\_status"]

}

}

# Generating RSA Key Pair

resource "tls\_private\_key" "rsa" {

count = data.external.check\_key\_pair.result["key\_pair\_status"] == "not\_exists" ? 1 : 0

algorithm = "RSA"

rsa\_bits = 4096

}

resource "aws\_key\_pair" "mak" {

count = data.external.check\_key\_pair.result["key\_pair\_status"] == "not\_exists" ? 1 : 0

key\_name = "my-key"

public\_key = tls\_private\_key.rsa[0].public\_key\_openssh

}

resource "local\_file" "mak-key" {

count = data.external.check\_key\_pair.result["key\_pair\_status"] == "not\_exists" ? 1 : 0

content = tls\_private\_key.rsa[0].private\_key\_pem

filename = "my-key.pem"

}

# Creating 1st web subnet

resource "aws\_subnet" "subnet\_1" {

vpc\_id = aws\_vpc.testvpc.id

cidr\_block = var.subnet1\_cidr

map\_public\_ip\_on\_launch = true

availability\_zone = "us-east-1a"

tags = {

Name = "Subnet 1"

}

}

resource "aws\_subnet" "subnet\_2" {

vpc\_id = aws\_vpc.testvpc.id

cidr\_block = var.subnet2\_cidr

map\_public\_ip\_on\_launch = true

availability\_zone = "us-east-1b"

tags = {

Name = "Subnet 2"

}

}

# Associating Route Table

resource "aws\_route\_table\_association" "rt1" {

subnet\_id = aws\_subnet.subnet\_1.id

route\_table\_id = aws\_route\_table.route.id

}

# Creating Security Group

resource "aws\_security\_group" "terra1\_sg" {

vpc\_id = aws\_vpc.testvpc.id

egress {

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

}

ingress {

from\_port = 80

to\_port = 80

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

ingress {

from\_port = 22

to\_port = 22

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

}

# Create Linux and install Docker

resource "aws\_instance" "linux" {

ami = "ami-08a0d1e16fc3f61ea"

instance\_type = "t2.micro"

key\_name = "my-key"

subnet\_id = aws\_subnet.subnet\_1.id

vpc\_security\_group\_ids = [aws\_security\_group.terra1\_sg.id]

associate\_public\_ip\_address = true # Enable auto-assigning public IP

user\_data = <<-EOF

#!/bin/bash

sudo yum -y update

sudo yum install docker -y

sudo systemctl start docker

sudo systemctl enable docker

sudo docker pull maksoft121/makfirstimagenginx:challange

sudo docker container run -itd -p 80:80 maksoft121/makfirstimagenginx:challange

EOF

}

This configuration checks if the key pair exists before creating a new one and conditionally creates resources based on the existence of the key pair.