Terraform

PROVISIONERS

❖ PROVISIONERS:

- Resources are the most important element in the Terraform language.
- You can use provisioners to model specific actions on the local machine or on a remote machine in order to prepare servers or other infrastructure objects for service.

PASSING DATA INTO VIRTUAL MACHINES AND OTHER COMPUTE RESOURCES:

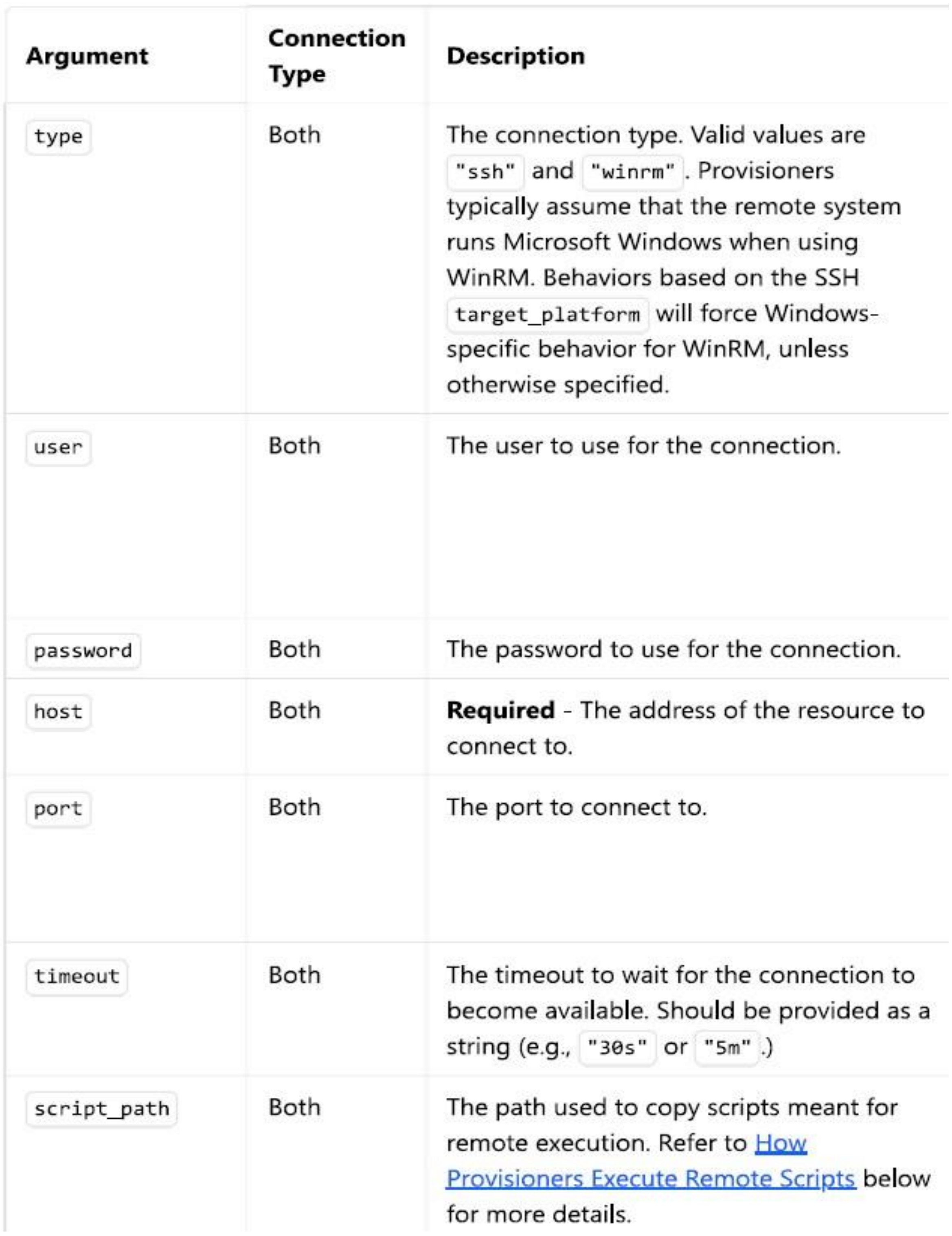
- When deploying virtual machines or other similar compute resources, we often need to pass in data about other related infrastructure that the software on that server will need to do its job.
- The various provisioners that interact with remote servers over **SSH** or **WinRM** can potentially be used to pass such data by logging in to the server and providing it directly, but most cloud computing platforms provide mechanisms to pass data to instances at the time of their creation such that the data is immediately available on system boot. For example:
  - **Amazon EC2:** user_data or user_data_base64 on aws_instance, aws_launch_template, and aws_launch_configuration.
  - **Amazon Lightsail:** user_data on aws_lightsail_instance.
- Provisioners are used to execute scripts on a local or remote machine as part of resource creation or destruction. Provisioners can be used to bootstrap a resource, cleanup before destroy, run configuration management, etc.

CONNECTION BLOCK:

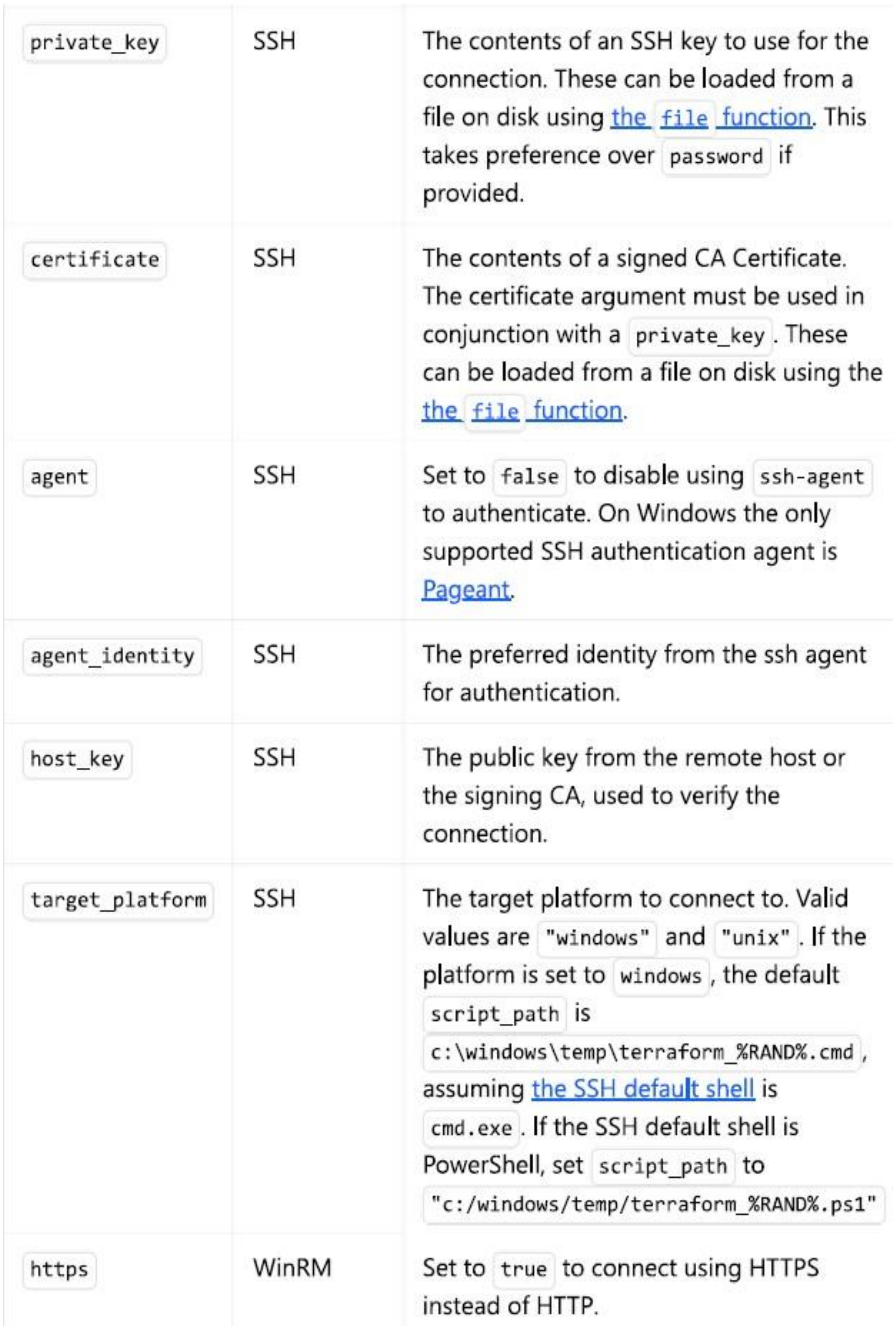
- You can create one or more connection blocks that describe how to access the remote resource.
- Connection blocks don't take a block label and can be nested within either a resource or a provisioner.
  - A **connection** block nested directly within a **resource** affects all of that resource's provisioners.
  - A **connection** block nested in a provisioner block only affects that **provisioner** and overrides any resource-level connection settings.
- The connection block supports the following arguments.

ARGUMENT REFERENCE:

- The connection block supports the following arguments. Some arguments are only supported by either the **SSH** or the **WinRM** connection type.



Argument	Connection Type	Description
`type`	Both	The connection type. Valid values are `"ssh"` and `"winrm"`. Provisioners typically assume that the remote system runs Microsoft Windows when using WinRM. Behaviors based on the SSH `target_platform` will force Windows-specific behavior for WinRM, unless otherwise specified.
`user`	Both	The user to use for the connection.
`password`	Both	The password to use for the connection.
`host`	Both	**Required** - The address of the resource to connect to.
`port`	Both	The port to connect to.
`timeout`	Both	The timeout to wait for the connection to become available. Should be provided as a string (e.g., `"30s"` or `"5m"`.)
`script_path`	Both	The path used to copy scripts meant for remote execution. Refer to How Provisioners Execute Remote Scripts below for more details.



`private_key`	SSH	The contents of an SSH key to use for the connection. These can be loaded from a file on disk using the `file` function. This takes preference over `password` if provided.
`certificate`	SSH	The contents of a signed CA Certificate. The certificate argument must be used in conjunction with a `private_key`. These can be loaded from a file on disk using the the `file` function.
`agent`	SSH	Set to `false` to disable using `ssh-agent` to authenticate. On Windows the only supported SSH authentication agent is Pageant.
`agent_identity`	SSH	The preferred identity from the ssh agent for authentication.
`host_key`	SSH	The public key from the remote host or the signing CA, used to verify the connection.
`target_platform`	SSH	The target platform to connect to. Valid values are `"windows"` and `"unix"`. If the platform is set to `windows`, the default `script_path` is `c:\windows\temp\terraform_%RAND%.cmd`, assuming the SSH default shell is `cmd.exe`. If the SSH default shell is PowerShell, set `script_path` to `"c:/windows/temp/terraform_%RAND%.ps1"`
`https`	WinRM	Set to `true` to connect using HTTPS instead of HTTP.

HOW PROVISIONERS EXECUTE REMOTE SCRIPTS:

- Provisioners which execute commands on a remote system via a protocol such as SSH typically achieve that by uploading a script file to the remote system and then asking the default shell to execute it.
- Most importantly, there must be a suitable location in the remote filesystem where the provisioner can create the script file. By default, Terraform chooses a path containing a random number using the following patterns depending on how target_platform is set:
  - **"unix":** /tmp/terraform_%RAND%.sh
  - **"windows":** C:/windows/temp/terraform_%RAND%.cmd

➢ PROVISIONERS TYPES:

1. FILE PROVISIONER:

- The file provisioner copies files or directories from the machine running Terraform to the newly created resource.
- The file provisioner supports both **ssh** and **winrm** type connections.

ARGUMENT REFERENCE:

SOURCE: The source file or directory. Specify it either relative to the current working directory or as an absolute path. This argument cannot be combined with content.

CONTENT: The direct content to copy on the destination. If destination is a file, the content will be written on that file. In case of a directory, a file named tf-file-content is created inside that directory. We recommend using a file as the destination when using content. This argument cannot be combined with source.

DESTINATION: **(Required)** The destination path to write to on the remote system.

EXAMPLE OF USAGE:

```
resource "aws_security_group" "sg_22_80" {
  name   = "sg_22"
  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

resource "aws_instance" "web" {
  ami           = "ami-079db87dc4c10ac91"
  instance_type = "t2.micro"
  key_name = "ram"
  tags = {
    Name = "AppServerInstance"
  }
connection {
   type        = "ssh"
   user        = "ec2-user"
   private_key = file("~/.ssh/ram.pem")
   host        = self.public_ip
  }

# Copying file script.sh to /home/ec2-user/script.sh
 provisioner "file" {
        source      = "./script.sh"
        destination = "/home/ec2-user/script.sh"
         }
}
```

NOTE: Create **"script.sh"** file in the current configuration location and also keep a **"ram.pem"** file in the user home directory location **.ssh.**

2. LOCAL-EXEC PROVISIONER:

- The local-exec provisioner invokes a local executable after a resource is created. This invokes a process on the machine running Terraform, not on the resource.

ARGUMENT REFERENCE:

COMMAND: (Required) This is the command to execute.

WORKING_DIR: (Optional) If provided, specifies the working directory where command will be executed. The directory must exist.

INTERPRETER: (Optional) If provided, this is a list of interpreter arguments used to execute the command. The first argument is the interpreter itself.

ENVIRONMENT: (Optional) block of key value pairs representing the environment of the executed command. Inherits the current process environment.

WHEN: (Optional) If provided, specifies when Terraform will execute the command. For example, **when = destroy** specifies that the provisioner will run when the associated resource is destroyed.

QUIET: (Optional) If set to true, Terraform will not print the command to be executed to stdout, and will instead print **"Suppressed by quiet=true".** Note that the output of the command will still be printed in any case.

EXAMPLE OF USAGE:

```
resource "aws_instance" "web" {
  ami           = "ami-079db87dc4c10ac91"
  instance_type = "t2.micro"
  key_name = "ram"
  tags = {
    Name = "AppServerInstance"
  }

  provisioner "local-exec" {
    command = "echo The server IP address is
${self.private_ip}"
  }
}
```

MULTIPLE PROVISIONERS:

```
resource "aws_instance" "web" {
  ami           = "ami-079db87dc4c10ac91"
  instance_type = "t2.micro"
  key_name      = "ram"
  tags = {
    Name = "AppServerInstance"
  }

  provisioner "local-exec" {
    command = "echo The server PublicIP address is
${self.private_ip}"
  }

  provisioner "local-exec" {
    command = "echo The server PrivateIP address is
${self.public_ip}"
  }
}
```

THE SELF OBJECT:

- Expressions in provisioner blocks cannot refer to their parent resource by name. Instead, they can use the special self object.
- The self object represents the provisioner's parent resource, and has all of that resource's attributes.

  For example, use **self.public_ip** to reference an **aws_instance's public_ip** attribute.

3. REMOTE-EXEC PROVISIONER:

- The remote-exec provisioner invokes a script on a remote resource after it is created. This can be used to run a configuration management tool, bootstrap into a cluster, etc.
- To invoke a local process, see the local-exec provisioner instead. The remote-exec provisioner requires a connection and supports both **ssh** and **winrm.**

ARGUMENT REFERENCE:

INLINE: This is a list of command strings. The provisioner uses a default shell unless you specify a shell as the first command (eg., #!/bin/bash). You cannot provide this with script or scripts.

SCRIPT: This is a path (relative or absolute) to a local script that will be copied to the remote resource and then executed. This cannot be provided with inline or scripts.

SCRIPTS: This is a list of paths (relative or absolute) to local scripts that will be copied to the remote resource and then executed. They are executed in the order they are provided. This cannot be provided with inline or script.

SCRIPT ARGUMENTS:

- You cannot pass any arguments to scripts using the script or scripts arguments to this provisioner.
- If you want to specify arguments, upload the script with the file provisioner and then use inline to call it.

EXAMPLE OF USAGE:

```
resource "aws_security_group" "sg_22_80" {
  name   = "sg_22"
  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
```

```
 }
 ingress {
   from_port   = 80
   to_port     = 80
   protocol    = "tcp"
   cidr_blocks = ["0.0.0.0/0"]
 }
 egress {
   from_port   = 0
   to_port     = 0
   protocol    = "-1"
   cidr_blocks = ["0.0.0.0/0"]
 }
}
resource "aws_instance" "web" {
 ami              = "ami-079db87dc4c10ac91"
 instance_type   = "t2.micro"
 key_name        = "ram"
 tags = {
   Name = "WEB-SERVER"
 }
 connection {
   type        = "ssh"
   user        = "ec2-user"
   private_key = file("~/.ssh/ram.pem")
```

```
    host        = self.public_ip
  }
provisioner "remote-exec" {
    inline = [
      "sudo yum update -y",
      "sudo yum install nginx -y",
      "sudo systemctl start nginx",
    ]
  }
}
```

```
$terraform fmt
$terraform validate
$terraform plan
$terraform apply
$terraform destroy
```