Alright — I'll prepare a **detailed YouTube-friendly tutorial script** on Terraform **Provisioners** so you can directly save it as a PDF later.
I'll include **intro, use cases, syntax, examples, best practices**, and a **step-by-step demo**.

Terraform Provisioners — Complete Guide

1. Introduction

Provisioners in Terraform are used to execute scripts or commands on a resource after it is created or before it is destroyed.
They help in bootstrapping instances, configuring software, or performing tasks not directly handled by Terraform providers.

⚠ **Important:** HashiCorp recommends using provisioners only when necessary — prefer native provider features instead.

---

2. Types of Provisioners

Terraform supports several provisioners:

Provisioner	Purpose
local-exec	Executes commands on the machine running Terraform.
remote-exec	Executes commands on a remote machine via SSH or WinRM.
file	Copies files or directories to a remote machine.
chef / puppet	Runs Chef/Puppet configuration management. *(Less common now)*

---

3. Syntax Overview

local-exec

```
resource "aws_instance" "example" {
  ami           = "ami-12345678"
  instance_type = "t2.micro"

  provisioner "local-exec" {
    command = "echo Instance created: ${self.public_ip}"
```

```
  }
}
```

Runs locally on your machine.

---

remote-exec

```
resource "aws_instance" "example" {
  ami           = "ami-12345678"
  instance_type = "t2.micro"
  key_name      = "mykey"

  provisioner "remote-exec" {
    inline = [
      "sudo apt-get update -y",
      "sudo apt-get install -y nginx"
    ]
  }

  connection {
    type        = "ssh"
    user        = "ubuntu"
    private_key = file("mykey.pem")
    host        = self.public_ip
  }
}
```

Runs commands **on the remote EC2 instance**.

---

file

```
provisioner "file" {
```

```
  source      = "local_script.sh"
  destination = "/home/ubuntu/remote_script.sh"

  connection {
    type        = "ssh"
    user        = "ubuntu"
    private_key = file("mykey.pem")
    host        = self.public_ip
  }
}
```

Copies local files to remote instances.

---

4. Lifecycle Hooks

You can specify when provisioners run:

- create *(default)* — runs after resource creation.
- destroy — runs before resource destruction.

Example:

```
provisioner "local-exec" {
  when    = destroy
  command = "echo Destroying instance ${self.id}"
}
```

---

5. Real-World Example — EC2 with Apache

```
resource "aws_instance" "web" {
  ami           = "ami-12345678"
  instance_type = "t2.micro"
  key_name      = "mykey"
```

```
  provisioner "remote-exec" {
    inline = [
      "sudo apt update -y",
      "sudo apt install -y apache2",
      "sudo systemctl start apache2",
      "sudo systemctl enable apache2"
    ]
  }

  connection {
    type        = "ssh"
    user        = "ubuntu"
    private_key = file("mykey.pem")
    host        = self.public_ip
  }
}
```

---

6. Best Practices

✅ Use provisioners **only if** you can't achieve the task with native Terraform providers.
✅ For software installation, prefer **cloud-init**, **user data**, or configuration management tools.
✅ Keep provisioners **idempotent** (running them multiple times should not break things).
✅ Secure your SSH keys — don't hardcode them in .tf files.
✅ Always test commands locally before using them in Terraform.

---

7. Common Errors

Error	Cause	Solution
Connection refused	SSH not yet ready	Add provisioner "remote-exec" with timeouts or use depends_on to wait for networking.

Error	Cause	Solution
Permission denied (publickey)	Wrong key or username	Verify private_key and user in connection block.
Timeout	Firewall blocking SSH	Ensure port 22 is open in Security Group.

---

8. Summary

Provisioners bridge the gap between resource creation and configuration in Terraform. They are **powerful but should be a last resort** — whenever possible, use declarative infrastructure and native provider features.
