Q3. DevOps – Infrastructure Provisioning Using Terraform

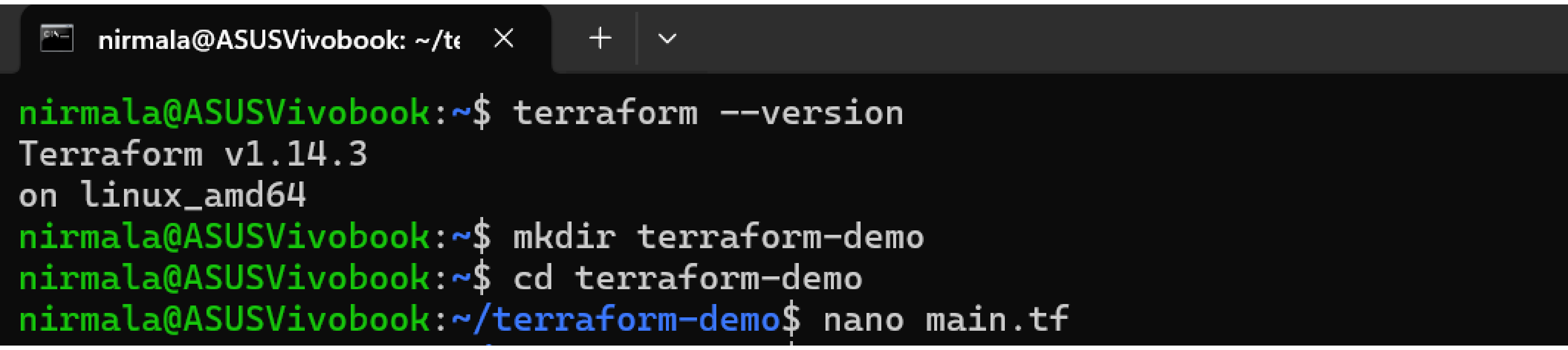
Task:
Provision infrastructure using **Terraform**.

Requirements:

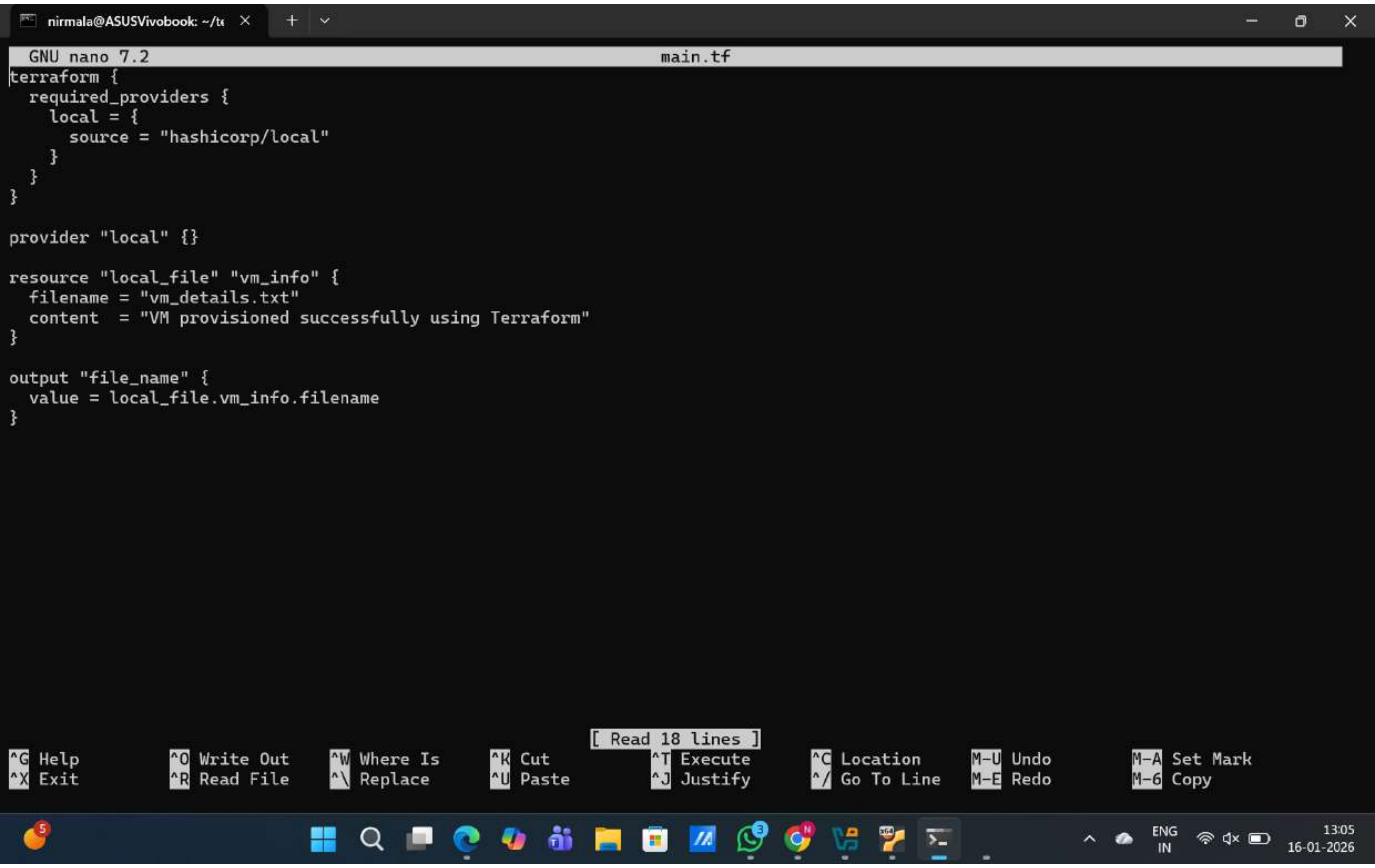
- Create Terraform code to:
  - Provision a VM or cloud instance (or local provider)
  - Output instance details after creation

Live demonstration must include:

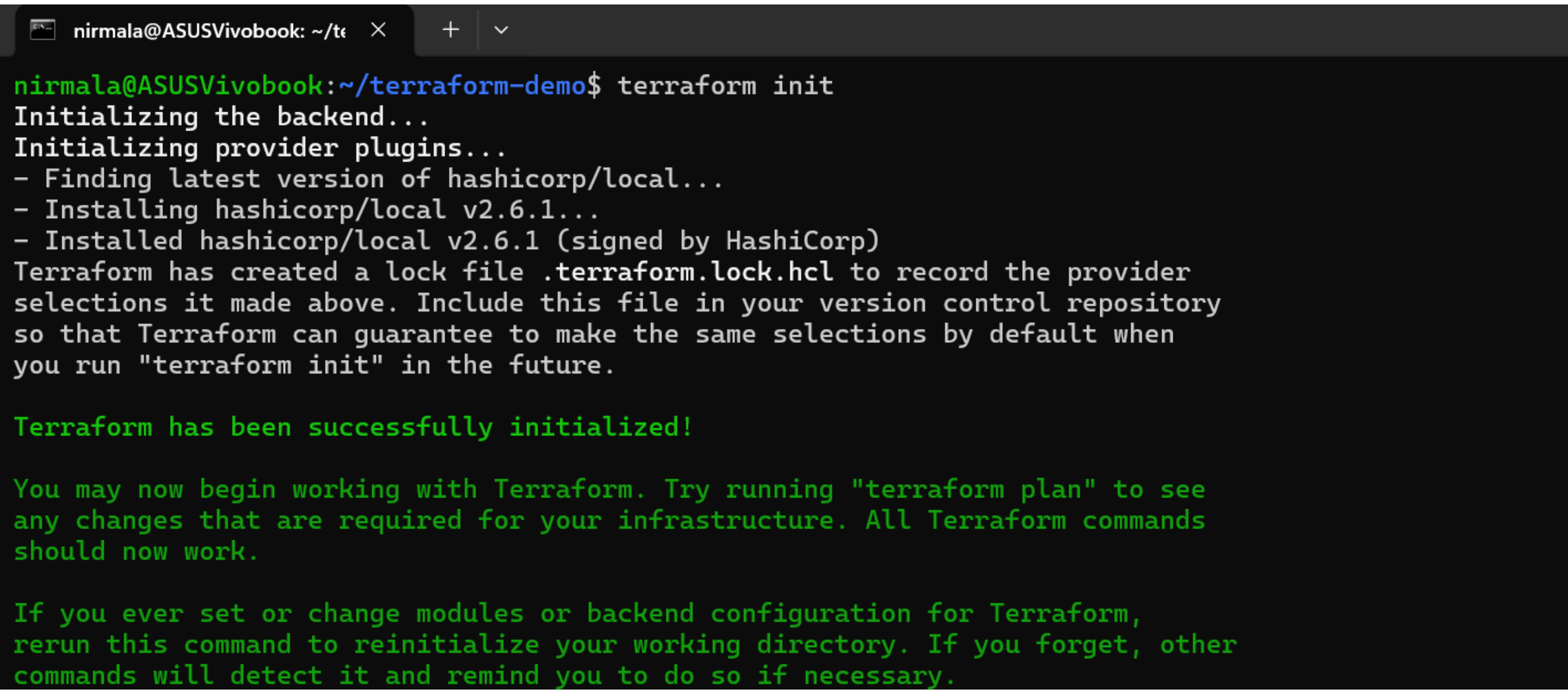
- Terraform files (`.tf`)
- `init`, `plan`, and `apply` workflow
- Resource creation confirmation



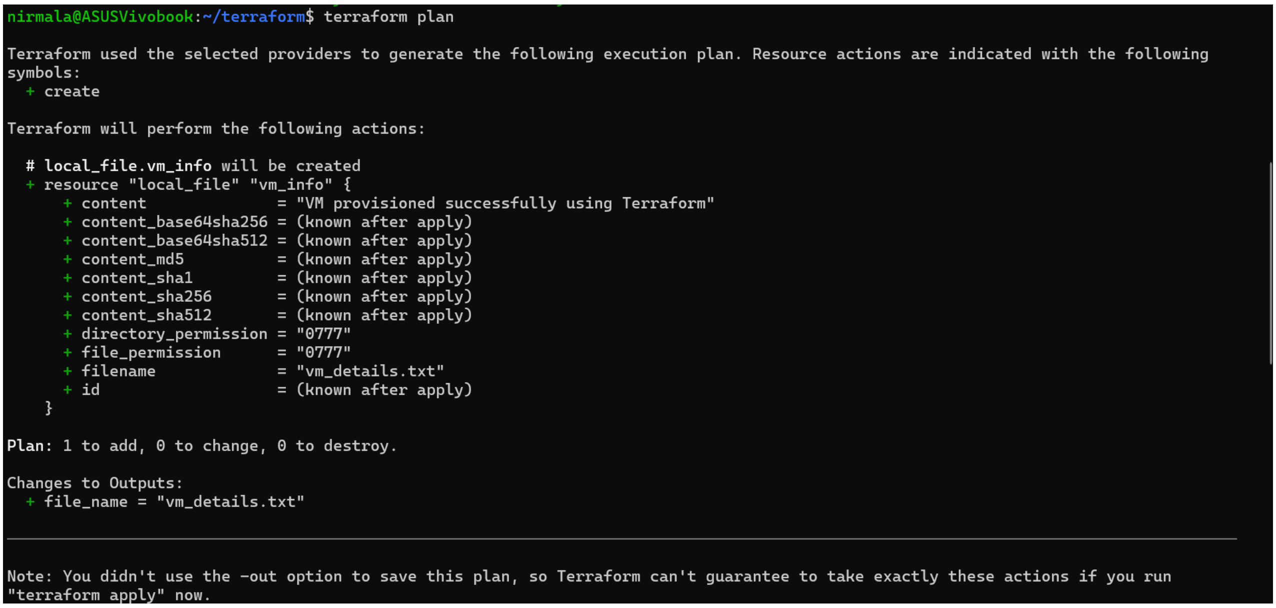
```
nirmala@ASUSVivobook:~$ terraform --version
Terraform v1.14.3
on linux_amd64
nirmala@ASUSVivobook:~$ mkdir terraform-demo
nirmala@ASUSVivobook:~$ cd terraform-demo
nirmala@ASUSVivobook:~/terraform-demo$ nano main.tf
```



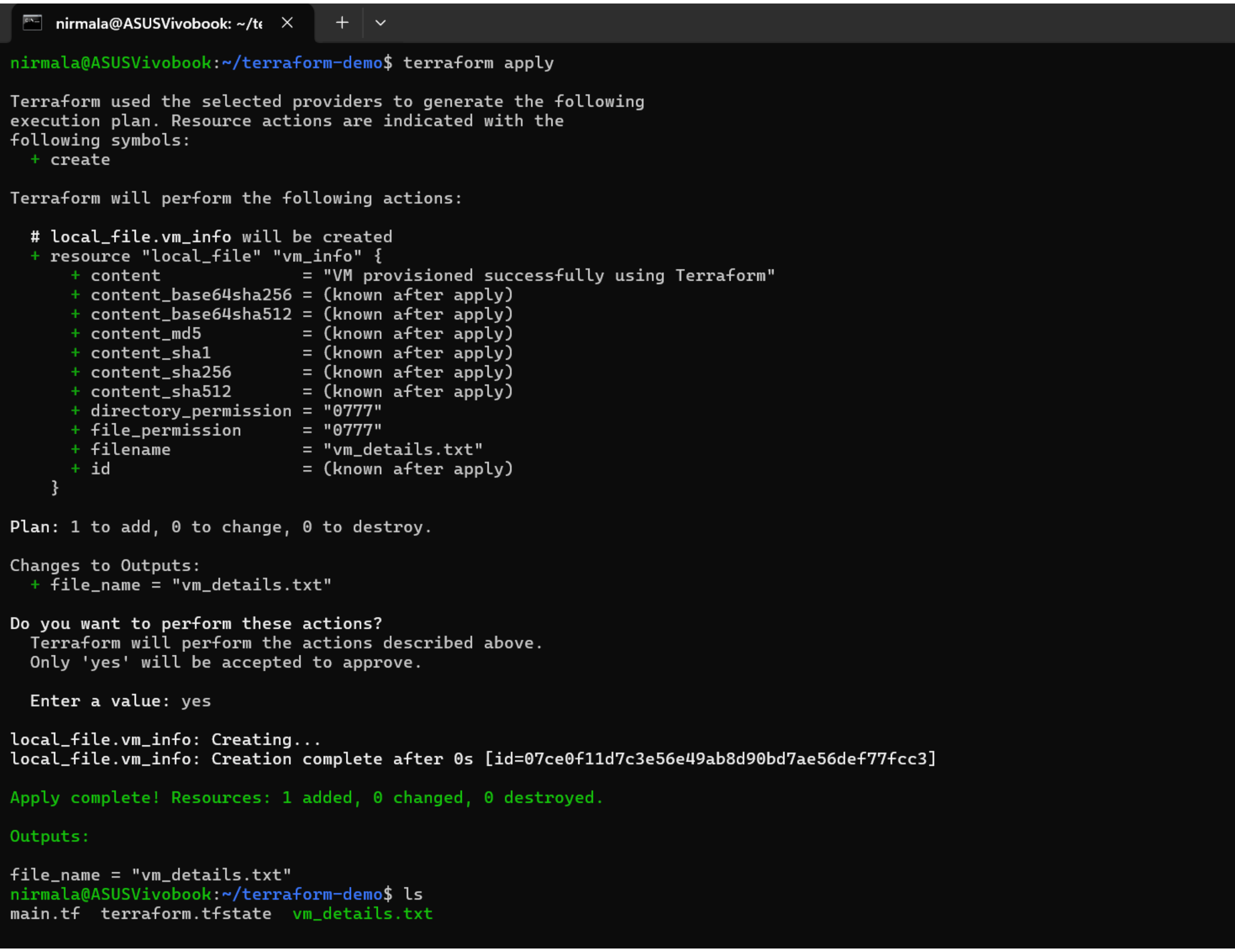
```
  GNU nano 7.2                                   main.tf
terraform {
  required_providers {
    local = {
      source = "hashicorp/local"
    }
  }
}

provider "local" {}

resource "local_file" "vm_info" {
  filename = "vm_details.txt"
  content  = "VM provisioned successfully using Terraform"
}

output "file_name" {
  value = local_file.vm_info.filename
}



                                          [ Read 18 lines ]
^G Help        ^O Write Out   ^W Where Is    ^K Cut         ^T Execute     ^C Location    M-U Undo       M-A Set Mark
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^/ Go To Line  M-E Redo       M-6 Copy
```



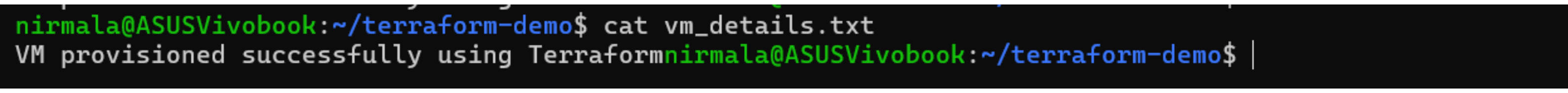
```
nirmala@ASUSVivobook:~/terraform-demo$ terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/local...
- Installing hashicorp/local v2.6.1...
- Installed hashicorp/local v2.6.1 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```



```
nirmala@ASUSVivobook:~/terraform$ terraform plan

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following
symbols:
  + create

Terraform will perform the following actions:

  # local_file.vm_info will be created
  + resource "local_file" "vm_info" {
      + content              = "VM provisioned successfully using Terraform"
      + content_base64sha256 = (known after apply)
      + content_base64sha512 = (known after apply)
      + content_md5          = (known after apply)
      + content_sha1         = (known after apply)
      + content_sha256       = (known after apply)
      + content_sha512       = (known after apply)
      + directory_permission = "0777"
      + file_permission      = "0777"
      + filename             = "vm_details.txt"
      + id                   = (known after apply)
    }

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

Changes to Outputs:
  + file_name = "vm_details.txt"

─────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run
"terraform apply" now.
```



```
nirmala@ASUSVivobook:~/terraform-demo$ terraform apply

Terraform used the selected providers to generate the following
execution plan. Resource actions are indicated with the
following symbols:
  + create

Terraform will perform the following actions:

  # local_file.vm_info will be created
  + resource "local_file" "vm_info" {
      + content              = "VM provisioned successfully using Terraform"
      + content_base64sha256 = (known after apply)
      + content_base64sha512 = (known after apply)
      + content_md5          = (known after apply)
      + content_sha1         = (known after apply)
      + content_sha256       = (known after apply)
      + content_sha512       = (known after apply)
      + directory_permission = "0777"
      + file_permission      = "0777"
      + filename             = "vm_details.txt"
      + id                   = (known after apply)
    }

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

Changes to Outputs:
  + file_name = "vm_details.txt"

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

local_file.vm_info: Creating...
local_file.vm_info: Creation complete after 0s [id=07ce0f11d7c3e56e49ab8d90bd7ae56def77fcc3]

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

Outputs:

file_name = "vm_details.txt"
nirmala@ASUSVivobook:~/terraform-demo$ ls
main.tf  terraform.tfstate  vm_details.txt
```



```
nirmala@ASUSVivobook:~/terraform-demo$ cat vm_details.txt
VM provisioned successfully using Terraformnirmala@ASUSVivobook:~/terraform-demo$
```

Q1. SDN – OpenFlow-Based Traffic Control

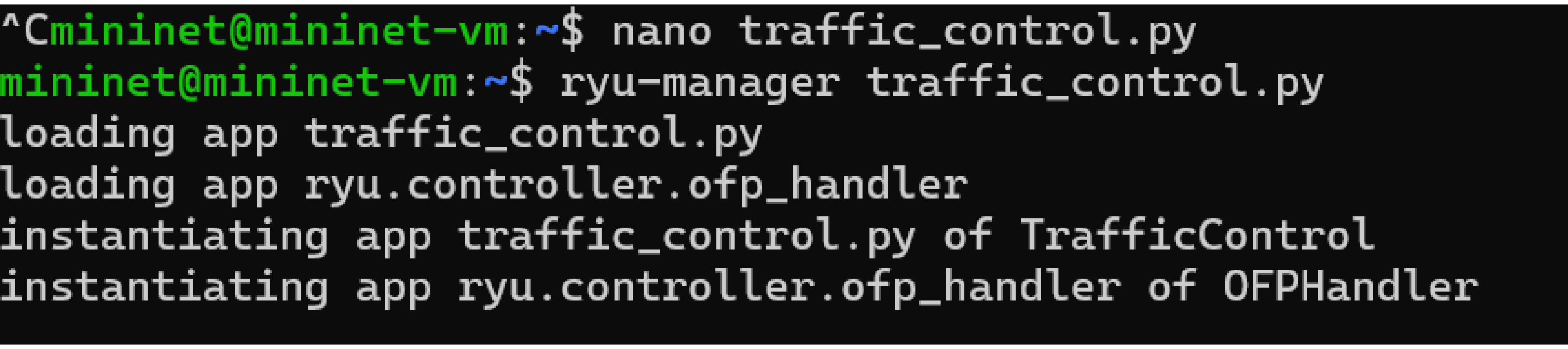
Task:
Build an SDN topology using **Mininet** and an SDN controller (Ryu / OpenDaylight).

Requirements:

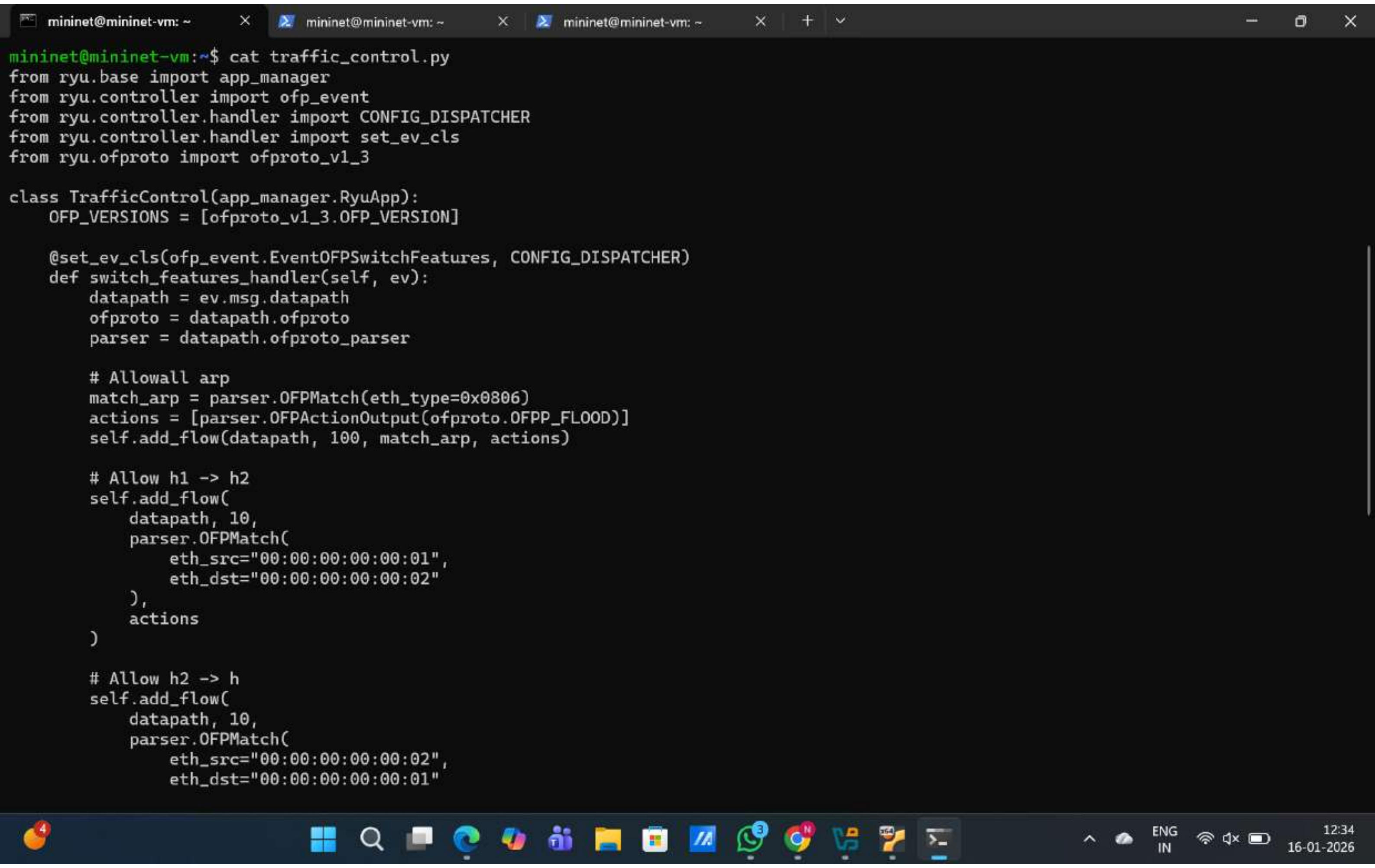
- Create flow rules to:
  - Allow traffic between Host1 and Host2
  - Block traffic between Host1 and Host3
- Flow rules must be installed dynamically from the controller

Live demonstration must include:

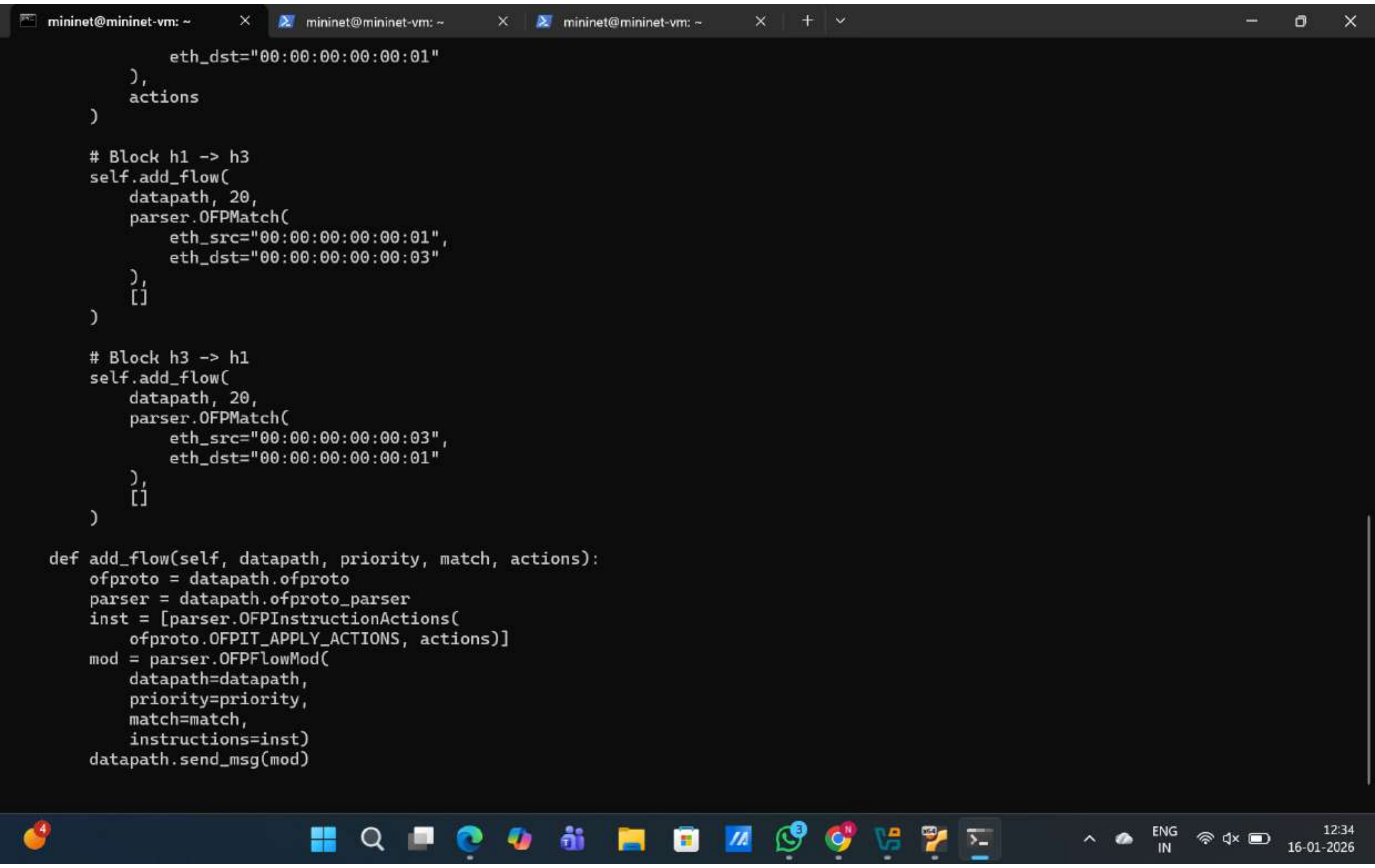
- Mininet topology
- Flow table entries on the switch
- Traffic verification using `ping`
- Explanation of control vs data plane



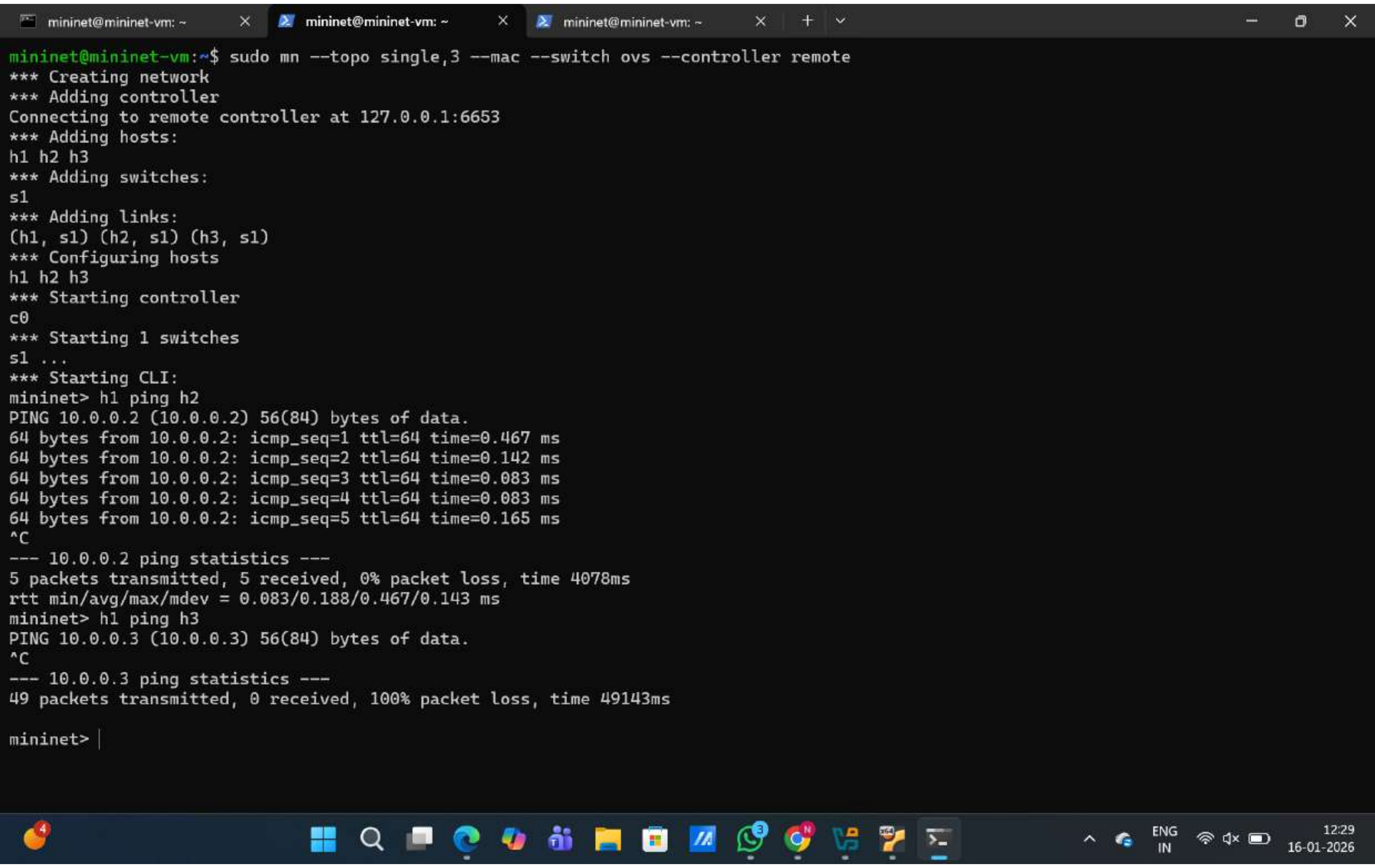
```
^Cmininet@mininet-vm:~$ nano traffic_control.py
mininet@mininet-vm:~$ ryu-manager traffic_control.py
loading app traffic_control.py
loading app ryu.controller.ofp_handler
instantiating app traffic_control.py of TrafficControl
instantiating app ryu.controller.ofp_handler of OFPHandler
```



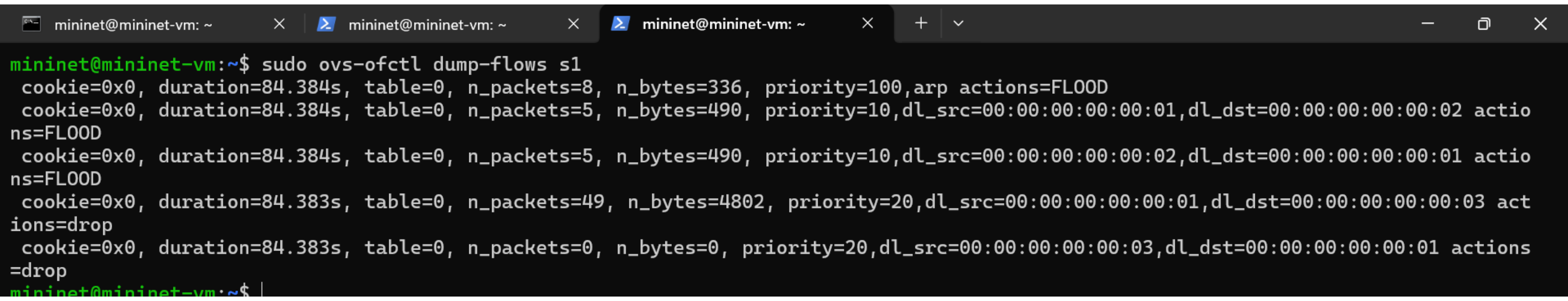
```
mininet@mininet-vm:~$ cat traffic_control.py
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3

class TrafficControl(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        # Allowall arp
        match_arp = parser.OFPMatch(eth_type=0x0806)
        actions = [parser.OFPActionOutput(ofproto.OFPP_FLOOD)]
        self.add_flow(datapath, 100, match_arp, actions)

        # Allow h1 -> h2
        self.add_flow(
            datapath, 10,
            parser.OFPMatch(
                eth_src="00:00:00:00:00:01",
                eth_dst="00:00:00:00:00:02"
            ),
            actions
        )

        # Allow h2 -> h
        self.add_flow(
            datapath, 10,
            parser.OFPMatch(
                eth_src="00:00:00:00:00:02",
                eth_dst="00:00:00:00:00:01"
```



```
                eth_dst="00:00:00:00:00:01"
            ),
            actions
        )

        # Block h1 -> h3
        self.add_flow(
            datapath, 20,
            parser.OFPMatch(
                eth_src="00:00:00:00:00:01",
                eth_dst="00:00:00:00:00:03"
            ),
            []
        )

        # Block h3 -> h1
        self.add_flow(
            datapath, 20,
            parser.OFPMatch(
                eth_src="00:00:00:00:00:03",
                eth_dst="00:00:00:00:00:01"
            ),
            []
        )

    def add_flow(self, datapath, priority, match, actions):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        inst = [parser.OFPInstructionActions(
            ofproto.OFPIT_APPLY_ACTIONS, actions)]
        mod = parser.OFPFlowMod(
            datapath=datapath,
            priority=priority,
            match=match,
            instructions=inst)
        datapath.send_msg(mod)
```



```
mininet@mininet-vm:~$ sudo mn --topo single,3 --mac --switch ovs --controller remote
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.467 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.142 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.083 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.083 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.165 ms
^C
--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4078ms
rtt min/avg/max/mdev = 0.083/0.188/0.467/0.143 ms
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
^C
--- 10.0.0.3 ping statistics ---
49 packets transmitted, 0 received, 100% packet loss, time 49143ms

mininet>
```



```
mininet@mininet-vm:~$ sudo ovs-ofctl dump-flows s1
 cookie=0x0, duration=84.384s, table=0, n_packets=8, n_bytes=336, priority=100,arp actions=FLOOD
 cookie=0x0, duration=84.384s, table=0, n_packets=5, n_bytes=490, priority=10,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02 actio
ns=FLOOD
 cookie=0x0, duration=84.384s, table=0, n_packets=5, n_bytes=490, priority=10,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01 actio
ns=FLOOD
 cookie=0x0, duration=84.383s, table=0, n_packets=49, n_bytes=4802, priority=20,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03 act
ions=drop
 cookie=0x0, duration=84.383s, table=0, n_packets=0, n_bytes=0, priority=20,dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01 actions
=drop
mininet@mininet-vm:~$
```