**MEGHA CHITTAPUR**

**4th Eval**

**1.Create a GitHub repository with name ABC-20092025**

**1: Create via GitHub Website**

1. Go to [GitHub Repositories](https://github.com/new).
2. Enter repository name: ABC-20092025.
3. Choose:
   * Public or Private (your choice).
   * Optionally add a README.
4. Click **Create repository**.

**2. Create an Application gateway using Terraform on a network called agw**

// Terraform configuration to create an Azure Application Gateway placed in a virtual network named `agw`.

// Notes:

// - Application Gateway requires its own dedicated subnet. This example creates a VNet named `agw` and a subnet `appgw-subnet`.

// - Adjust variables below as needed (location, address spaces, SKU, instance\_count).

// - Run: terraform init && terraform plan && terraform apply

terraform {

required\_version = ">= 1.3.0"

required\_providers {

azurerm = {

source = "hashicorp/azurerm"

version = ">= 3.0"

}

}

}

provider "azurerm" {

features = {}

}

// ----------------

// Variables (simple, inline defaults)

// ----------------

variable "location" {

type = string

default = "East US"

}

variable "resource\_group\_name" {

type = string

default = "rg-agw"

}

variable "vnet\_name" {

type = string

default = "agw"

}

variable "vnet\_address\_space" {

type = list(string)

default = ["10.0.0.0/16"]

}

variable "appgw\_subnet\_prefix" {

type = string

default = "10.0.0.0/24"

}

variable "appgw\_subnet\_name" {

type = string

default = "appgw-subnet"

}

variable "public\_ip\_name" {

type = string

default = "agw-pip"

}

variable "application\_gateway\_name" {

type = string

default = "appgw-agw"

}

variable "sku\_name" {

type = string

default = "Standard\_v2"

}

variable "instance\_count" {

type = number

default = 1

}

// ----------------

// Resource Group

// ----------------

resource "azurerm\_resource\_group" "rg" {

name = var.resource\_group\_name

location = var.location

}

// ----------------

// Virtual Network and Subnet (vnet named `agw`)

// ----------------

resource "azurerm\_virtual\_network" "vnet" {

name = var.vnet\_name

address\_space = var.vnet\_address\_space

location = azurerm\_resource\_group.rg.location

resource\_group\_name = azurerm\_resource\_group.rg.name

}

resource "azurerm\_subnet" "appgw\_subnet" {

name = var.appgw\_subnet\_name

resource\_group\_name = azurerm\_resource\_group.rg.name

virtual\_network\_name = azurerm\_virtual\_network.vnet.name

address\_prefixes = [var.appgw\_subnet\_prefix]

// Application Gateway requires delegation? Not necessary; just ensure subnet is dedicated.

}

// ----------------

// Public IP for Application Gateway

// ----------------

resource "azurerm\_public\_ip" "agw\_pip" {

name = var.public\_ip\_name

location = azurerm\_resource\_group.rg.location

resource\_group\_name = azurerm\_resource\_group.rg.name

allocation\_method = "Static"

sku = "Standard"

}

// ----------------

// Application Gateway

// ----------------

resource "azurerm\_application\_gateway" "appgw" {

name = var.application\_gateway\_name

location = azurerm\_resource\_group.rg.location

resource\_group\_name = azurerm\_resource\_group.rg.name

sku {

name = var.sku\_name

tier = var.sku\_name

capacity = var.instance\_count

}

gateway\_ip\_configuration {

name = "appgw-gateway-ip-config"

subnet\_id = azurerm\_subnet.appgw\_subnet.id

}

frontend\_port {

name = "http-port"

port = 80

}

frontend\_port {

name = "https-port"

port = 443

}

frontend\_ip\_configuration {

name = "appgw-frontend-ip"

public\_ip\_address\_id = azurerm\_public\_ip.agw\_pip.id

}

backend\_address\_pool {

name = "default-backend-pool"

// Add backend addresses here or use backend\_http\_settings for IPs/FQDNs

// example:

// backend\_addresses {

// ip\_address = "10.0.1.4"

// }

}

backend\_http\_settings {

name = "http-settings"

cookie\_based\_affinity = "Disabled"

port = 80

protocol = "Http"

request\_timeout = 30

}

http\_listener {

name = "http-listener"

frontend\_ip\_configuration\_name = "appgw-frontend-ip"

frontend\_port\_name = "http-port"

protocol = "Http"

}

request\_routing\_rule {

name = "rule-http"

rule\_type = "Basic"

http\_listener\_name = "http-listener"

backend\_address\_pool\_name = "default-backend-pool"

backend\_http\_settings\_name = "http-settings"

}

identity {

type = "None"

}

tags = {

environment = "terraform"

created\_by = "terraform"

}

}

// ----------------

// Outputs

// ----------------

output "application\_gateway\_id" {

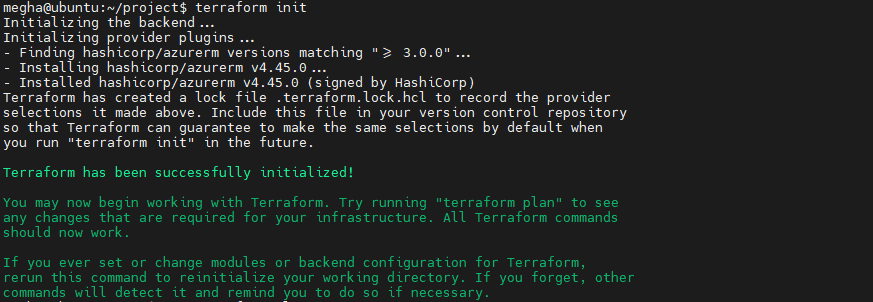
value = azurerm\_application\_gateway.appgw.id

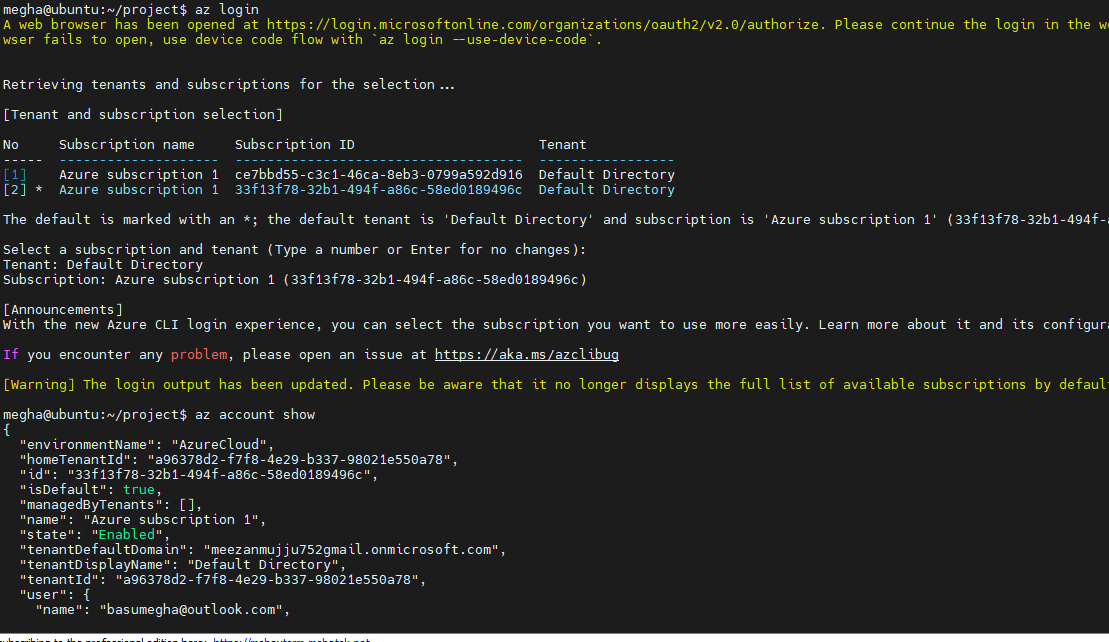
}

output "application\_gateway\_frontend\_ip" {

value = azurerm\_public\_ip.agw\_pip.ip\_address

}



****

**3. Deploy an tomcat application into a VM created in the network**

// ----------------

// Variables

// ----------------

variable "vm\_name" {

default = "tomcat-vm"

}

variable "admin\_username" {

default = "azureuser"

}

variable "admin\_password" {

default = "P@ssw0rd12345!" // replace with secret / use Key Vault

}

// ----------------

// Subnet for VM (inside agw VNet)

// ----------------

resource "azurerm\_subnet" "vm\_subnet" {

name = "vm-subnet"

resource\_group\_name = azurerm\_resource\_group.rg.name

virtual\_network\_name = azurerm\_virtual\_network.vnet.name

address\_prefixes = ["10.0.1.0/24"]

}

// ----------------

// Network Interface

// ----------------

resource "azurerm\_network\_interface" "vm\_nic" {

name = "${var.vm\_name}-nic"

location = azurerm\_resource\_group.rg.location

resource\_group\_name = azurerm\_resource\_group.rg.name

ip\_configuration {

name = "internal"

subnet\_id = azurerm\_subnet.vm\_subnet.id

private\_ip\_address\_allocation = "Dynamic"

public\_ip\_address\_id = azurerm\_public\_ip.agw\_pip.id

}

}

// ----------------

// Linux VM

// ----------------

resource "azurerm\_linux\_virtual\_machine" "tomcat\_vm" {

name = var.vm\_name

resource\_group\_name = azurerm\_resource\_group.rg.name

location = azurerm\_resource\_group.rg.location

size = "Standard\_B2s"

admin\_username = var.admin\_username

admin\_password = var.admin\_password

network\_interface\_ids = [azurerm\_network\_interface.vm\_nic.id]

os\_disk {

caching = "ReadWrite"

storage\_account\_type = "Standard\_LRS"

}

source\_image\_reference {

publisher = "Canonical"

offer = "0001-com-ubuntu-server-focal"

sku = "20\_04-lts"

version = "latest"

}

computer\_name = var.vm\_name

custom\_data = base64encode(<<-EOT

#!/bin/bash

sudo apt-get update -y

sudo apt-get install -y default-jdk wget

wget https://dlcdn.apache.org/tomcat/tomcat-10/v10.1.30/bin/apache-tomcat-10.1.30.tar.gz -O /tmp/tomcat.tar.gz

sudo mkdir -p /opt/tomcat

sudo tar xzvf /tmp/tomcat.tar.gz -C /opt/tomcat --strip-components=1

sudo sh /opt/tomcat/bin/startup.sh

sudo ufw allow 8080

EOT

)

}

// ----------------

// Add VM to Application Gateway Backend Pool

// ----------------

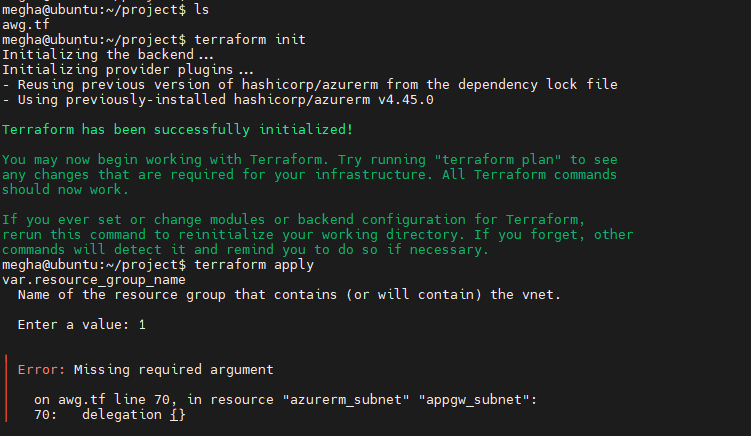
resource "azurerm\_network\_interface\_backend\_address\_pool\_association" "appgw\_backend\_assoc" {

network\_interface\_id = azurerm\_network\_interface.vm\_nic.id

ip\_configuration\_name = "internal"

backend\_address\_pool\_id = azurerm\_application\_gateway.appgw.backend\_address\_pool[0].id

}

****

**4. Run an Tomcat container on another machine with public IP in a different network attach to AGW**

// ----------------

// VNet for second VM

// ----------------

resource "azurerm\_virtual\_network" "vnet2" {

name = "vnet2"

address\_space = ["10.2.0.0/16"]

location = azurerm\_resource\_group.rg.location

resource\_group\_name = azurerm\_resource\_group.rg.name

}

resource "azurerm\_subnet" "vnet2\_subnet" {

name = "vm2-subnet"

resource\_group\_name = azurerm\_resource\_group.rg.name

virtual\_network\_name = azurerm\_virtual\_network.vnet2.name

address\_prefixes = ["10.2.1.0/24"]

}

// ----------------

// Public IP for VM2

// ----------------

resource "azurerm\_public\_ip" "vm2\_pip" {

name = "vm2-pip"

location = azurerm\_resource\_group.rg.location

resource\_group\_name = azurerm\_resource\_group.rg.name

allocation\_method = "Static"

sku = "Standard"

}

// ----------------

// NIC for VM2

// ----------------

resource "azurerm\_network\_interface" "vm2\_nic" {

name = "vm2-nic"

location = azurerm\_resource\_group.rg.location

resource\_group\_name = azurerm\_resource\_group.rg.name

ip\_configuration {

name = "internal"

subnet\_id = azurerm\_subnet.vnet2\_subnet.id

private\_ip\_address\_allocation = "Dynamic"

public\_ip\_address\_id = azurerm\_public\_ip.vm2\_pip.id

}

}

// ----------------

// VM2 running Docker + Tomcat Container

// ----------------

resource "azurerm\_linux\_virtual\_machine" "vm2" {

name = "vm2-docker-tomcat"

resource\_group\_name = azurerm\_resource\_group.rg.name

location = azurerm\_resource\_group.rg.location

size = "Standard\_B2s"

admin\_username = "azureuser"

admin\_password = "P@ssw0rd12345!" // replace with Key Vault

network\_interface\_ids = [azurerm\_network\_interface.vm2\_nic.id]

os\_disk {

caching = "ReadWrite"

storage\_account\_type = "Standard\_LRS"

}

source\_image\_reference {

publisher = "Canonical"

offer = "0001-com-ubuntu-server-focal"

sku = "20\_04-lts"

version = "latest"

}

computer\_name = "vm2"

custom\_data = base64encode(<<-EOT

#!/bin/bash

sudo apt-get update -y

sudo apt-get install -y docker.io

sudo systemctl enable docker

sudo systemctl start docker

sudo docker run -d -p 8080:8080 tomcat:10.1-jdk17

sudo ufw allow 8080

EOT

)

}

// ----------------

// Add VM2 Public IP to AGW backend

// ----------------

resource "azurerm\_application\_gateway\_backend\_address\_pool" "vm2\_backend" {

name = "vm2-backend-pool"

resource\_group\_name = azurerm\_resource\_group.rg.name

application\_gateway\_name = azurerm\_application\_gateway.appgw.name

backend\_addresses {

fqdn = azurerm\_public\_ip.vm2\_pip.fqdn

ip\_address = azurerm\_public\_ip.vm2\_pip.ip\_address

}

}

// ----------------

// Routing Rule for VM2

// ----------------

resource "azurerm\_application\_gateway\_request\_routing\_rule" "vm2\_rule" {

name = "rule-vm2"

rule\_type = "Basic"

resource\_group\_name = azurerm\_resource\_group.rg.name

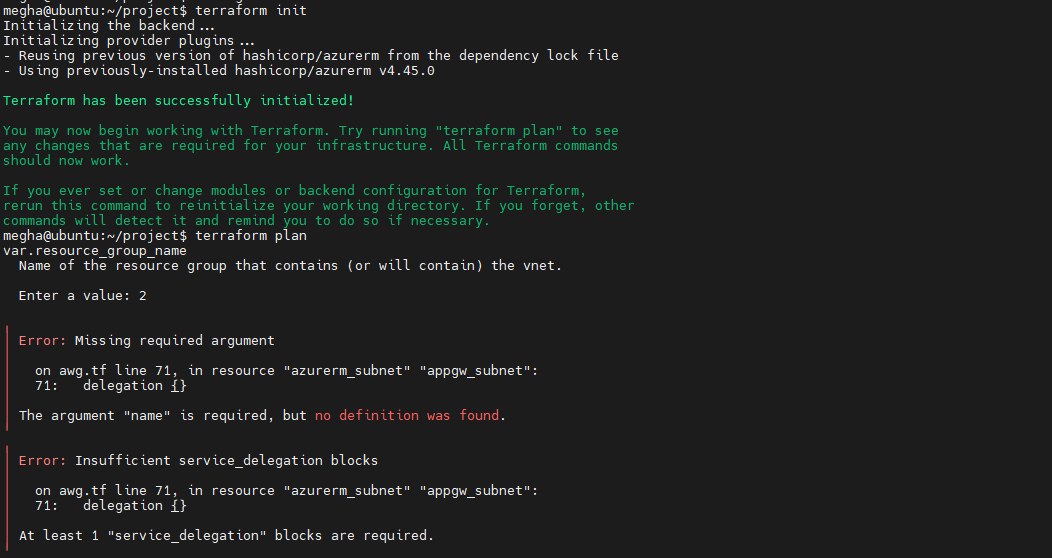
application\_gateway\_name = azurerm\_application\_gateway.appgw.name

http\_listener\_name = "http-listener"

backend\_address\_pool\_name = azurerm\_application\_gateway\_backend\_address\_pool.vm2\_backend.name

backend\_http\_settings\_name = "http-settings"

}

****

**5. Create a infra pipeline using 2 automate completely with AZD**

**Azure.yml**

name: agw-infra

metadata:

template: custom-terraform

infra:

provider: terraform

path: infra

main.tf

terraform {

backend "azurerm" {}

}

Infra.yml

name: AZD Infra Pipeline

on:

push:

branches: [ main ]

jobs:

build-and-deploy:

runs-on: ubuntu-latest

steps:

- name: Checkout

uses: actions/checkout@v3

- name: Setup AZD

uses: Azure/setup-azd@v1

with:

version: latest

- name: Login with Service Principal

uses: azure/login@v1

with:

creds: ${{ secrets.AZURE\_CREDENTIALS }}

- name: Provision Infra

run: azd provision --no-prompt --environment prod

- name: Deploy App (if applicable)

run: azd deploy --no-prompt --environment prod

pipeline.yml

trigger: none

pool:

name:pool2

steps:

- checkout: self

- task: UseAzureCLI@2

inputs:

azureSubscription: 'spn-connection'

scriptType: bash

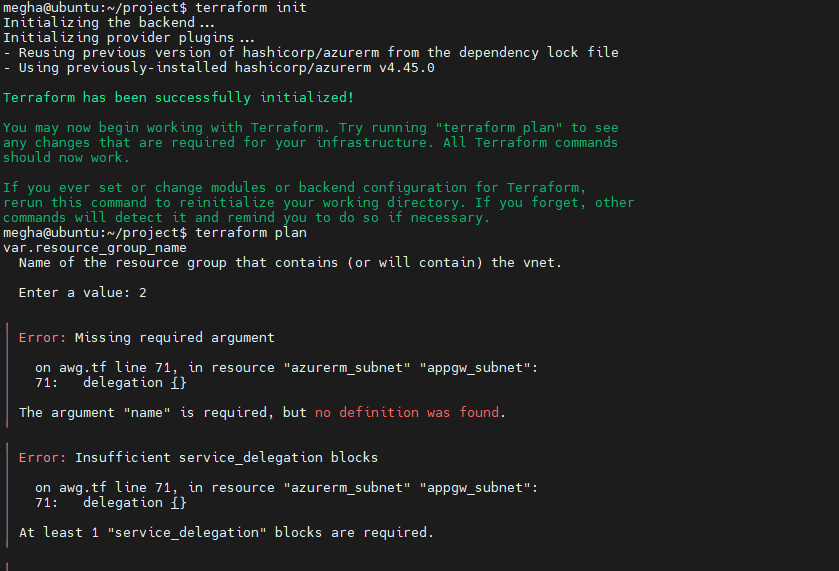
scriptLocation: inlineScript

inlineScript: |

curl -fsSL https://aka.ms/install-azd.sh | bash

azd provision --no-prompt --environment prod

azd deploy --no-prompt --environment prod



**6. Validate the GW routing with a Shell script**

#!/usr/bin/env bash

# validate-gw-routing.sh

# Usage:

# ./validate-gw-routing.sh --gateway HOST\_OR\_IP --host HEADER\_HOST --path /health --backends backend1:expected\_string,backend2:expected\_string

# Example:

# ./validate-gw-routing.sh --gateway agw.example.com --host app.example.com --path / --backends "10.1.0.5:APP1\_UP,10.1.0.6:APP2\_OK"

set -euo pipefail

IFS=$'\n\t'

# Defaults

TIMEOUT=5

CURL\_OPTS="--silent --show-error --max-time ${TIMEOUT} --location --write-out '\n%{http\_code}' -i"

usage() {

cat <<EOF

Validate GW routing

Options:

--gateway HOST\_OR\_IP Gateway hostname or IP to contact (required)

--host HOST\_HEADER Host header to send in HTTP requests (optional)

--path PATH HTTP path to validate (default: /)

--backends LIST Comma-separated list backendIP:expect\_pattern (optional)

--http-port PORT HTTP port to use (default: 80)

--https-port PORT HTTPS port to use (default: 443)

--protocol http|https Protocol to test (default: http)

--timeout SECS Network timeout in seconds (default: 5)

-h, --help Show this help

EOF

exit 1

}

# Parse args (simple)

GATEWAY=""

HOST\_HEADER=""

PATH\_TO\_CHECK="/"

BACKENDS=""

HTTP\_PORT=80

HTTPS\_PORT=443

PROTO="http"

while [[ $# -gt 0 ]]; do

case "$1" in

--gateway) GATEWAY="$2"; shift 2;;

--host) HOST\_HEADER="$2"; shift 2;;

--path) PATH\_TO\_CHECK="$2"; shift 2;;

--backends) BACKENDS="$2"; shift 2;;

--http-port) HTTP\_PORT="$2"; shift 2;;

--https-port) HTTPS\_PORT="$2"; shift 2;;

--protocol) PROTO="$2"; shift 2;;

--timeout) TIMEOUT="$2"; CURL\_OPTS="--silent --show-error --max-time ${TIMEOUT} --location --write-out '\n%{http\_code}' -i"; shift 2;;

-h|--help) usage;;

\*) echo "Unknown option: $1"; usage;;

esac

done

if [[ -z "$GATEWAY" ]]; then

echo "ERROR: --gateway is required"

usage

fi

# Tool checks

for cmd in curl dig traceroute nc ip route ss; do

command -v "$cmd" >/dev/null 2>&1 || true

done

echo "=== GW routing validation started ==="

echo "Gateway: $GATEWAY"

[[ -n "$HOST\_HEADER" ]] && echo "Host header: $HOST\_HEADER"

echo "Path: $PATH\_TO\_CHECK"

echo "Protocol: $PROTO"

echo "Timeout: ${TIMEOUT}s"

echo

# 1) DNS resolution (if hostname)

if [[ "$GATEWAY" =~ [A-Za-z] ]]; then

echo "-- DNS resolution for gateway:"

if command -v dig >/dev/null 2>&1; then

dig +short "$GATEWAY" || echo "(dig failed)"

else

nslookup "$GATEWAY" 2>/dev/null | sed -n '1,4p' || echo "(no dig/nslookup)"

fi

echo

fi

# 2) Local route table / default route

echo "-- Local routing table / default gateway:"

if command -v ip >/dev/null 2>&1; then

ip route show

else

route -n || true

fi

echo

# 3) Ping / basic connectivity

echo "-- Connectivity check (ping 3 packets):"

if ping -c 3 -W 2 "$GATEWAY" >/dev/null 2>&1; then

ping -c 3 -W 2 "$GATEWAY" | sed -n '1,4p'

else

echo "Ping failed (may be ICMP blocked). Attempting TCP connect test..."

fi

echo

# 4) Traceroute to gateway (may require sudo on some systems)

if command -v traceroute >/dev/null 2>&1; then

echo "-- Traceroute to gateway (first 10 hops):"

traceroute -m 10 "$GATEWAY" || echo "(traceroute failed or needs privileges)"

else

echo "(traceroute not available)"

fi

echo

# 5) TCP port checks against gateway

echo "-- TCP port reachability to gateway:"

if [[ "$PROTO" == "https" ]]; then

PORT="$HTTPS\_PORT"

else

PORT="$HTTP\_PORT"

fi

if command -v nc >/dev/null 2>&1; then

if nc -z -w ${TIMEOUT} "$GATEWAY" "$PORT" >/dev/null 2>&1; then

echo "TCP ${PORT} - reachable"

else

echo "TCP ${PORT} - NOT reachable (connection refused or filtered)"

fi

else

echo "(nc not installed; skipping TCP connect test)"

fi

echo

# 6) Issue HTTP request to gateway and show useful headers + body snippet

echo "-- HTTP request via gateway:"

CURL\_URL="${PROTO}://${GATEWAY}${PATH\_TO\_CHECK}"

CURL\_CMD=(curl ${CURL\_OPTS})

# If a Host header was provided, add it via -H or --resolve for direct IP

if [[ -n "$HOST\_HEADER" ]]; then

# If gateway is an IP and host header is set, add Host header

if [[ "$GATEWAY" =~ ^[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+$ ]]; then

CURL\_CMD+=(--header "Host: ${HOST\_HEADER}")

else

# If gateway is hostname but different Host header is requested, still send header

CURL\_CMD+=(--header "Host: ${HOST\_HEADER}")

fi

fi

# Print the curl command (safe)

echo "curl -> ${CURL\_URL} (sending Host: ${HOST\_HEADER:-<none>})"

echo

# Execute curl and capture output

HTTP\_RAW="$("${CURL\_CMD[@]}" "${CURL\_URL}" 2>&1 || true)"

echo "---- response start ----"

# Show up to first 120 lines with markers to avoid huge dumps

echo "${HTTP\_RAW}" | sed -n '1,120p'

echo "---- response end ----"

echo

# 7) Backend-specific validation (if provided)

if [[ -n "$BACKENDS" ]]; then

echo "-- Backend routing verification (probing via gateway):"

# backends format: comma separated list of ip:expected\_string

IFS=',' read -r -a BE\_ARRAY <<< "$BACKENDS"

for item in "${BE\_ARRAY[@]}"; do

ip\_expected="${item%%:\*}"

expected="${item#\*:}"

# craft a Host header that requests the backend IP via the gateway if needed,

# many gateways route by Host header or path. We'll send Host header if provided,

# and add a query param `\_\_backend\_check=IP` to help the backend reflect identity if supported.

test\_url="${CURL\_URL}"

# append query param intelligently

if [[ "$test\_url" == \*\?\* ]]; then

test\_url="${test\_url}&\_\_backend\_check=${ip\_expected}"

else

test\_url="${test\_url}?\_\_backend\_check=${ip\_expected}"

fi

echo "Probing via gateway for backend ${ip\_expected}, expecting '${expected}'..."

# prefer Host header if set, otherwise try to use --resolve to force gateway IP with Host header.

if [[ -n "$HOST\_HEADER" && "$GATEWAY" =~ ^[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+$ ]]; then

# gateway is IP: use Host header to attempt route-by-host

resp="$("${CURL\_CMD[@]}" --header "Host: ${HOST\_HEADER}" "${test\_url}" 2>&1 || true)"

else

# normal request

resp="$("${CURL\_CMD[@]}" "${test\_url}" 2>&1 || true)"

fi

# Look for expected pattern in response headers or body

if echo "$resp" | grep -q -- "$expected"; then

echo " OK: Found expected pattern '${expected}'."

else

# Also test if Server header or X-Backend header contains the IP

if echo "$resp" | grep -i -E 'server:|x-backend|x-origin' >/dev/null 2>&1; then

echo " Note: response headers present, but expected pattern not found."

echo "$resp" | sed -n '1,20p'

else

echo " FAIL: expected '${expected}' not found in response."

fi

fi

echo

done

fi

# 8) Helpful hints / next steps

cat <<EOF

Done.

Suggested next steps if anything failed:

- If DNS failed: check DNS zone and record for the gateway.

- If ICMP/traceroute failed but TCP works: ICMP may be blocked, focus on TCP/gateway logs.

- If TCP port is closed: check firewall / NSG / security group or gateway listener.

- If HTTP returns unexpected backend: check gateway routing rules, host-based rules, path-based rules, and backend health in your GW control plane.

- To capture traffic and inspect headers from your machine, run 'sudo tcpdump -i any -A host YOUR\_GATEWAY\_IP and port ${PORT}' (requires root).

EOF

