VPC PEERING CONNECTIONS

Mike Macdonald

CONTENTS

1. BRIEF

The brief was to complete a proof of concept of the non-transitive nature of VPC Peering Connections. Create four separate VPCs each with an EC2 instance in it. Include in the first VPC a public subnet with a Bastion Host. SSH into each instance from the Bastion Host and then systematically ping each of the instances to prove the use of the peering connections.

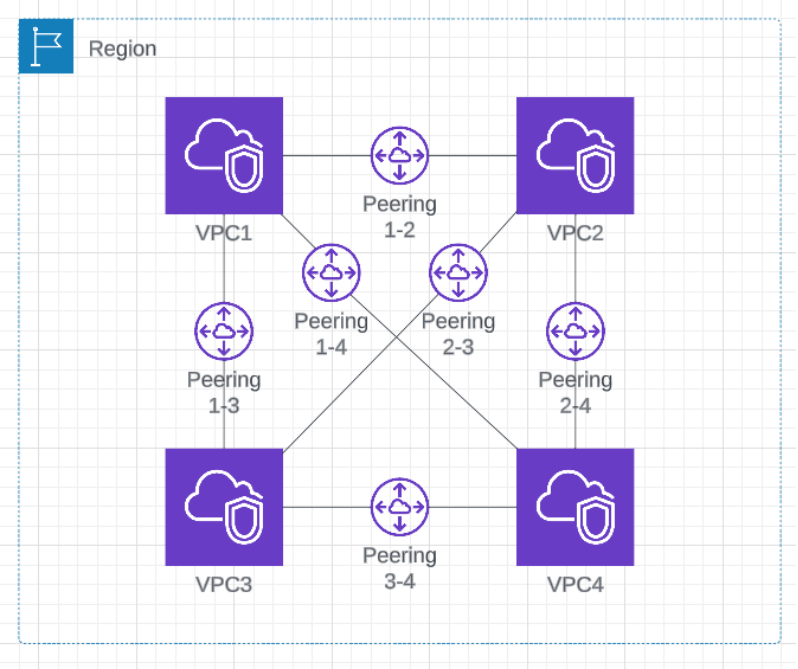
I then chose to extend the brief in two ways – firstly deploy the infrastructure as code (IaC) using AWS CloudFormation. Secondly to write a python script to automatically SSH into each machine and then ping all of the others, and print if it had been successful or not.

2. EXPLANATION

VPCs are unable to communicate with one another. As such there are two main methods to connect VPCs together. You can use either a transit gateway or VPC peering connections. This project uses Peering Connections to connect the VPCs together.

It's critical when planning an architecture that will have multiple VPCs to ensure that their CIDR ranges don't overlap. If they do, cross VPC communication will not be possible. As peering connections are non-transitive each VPC needs its own peering connection with each of the other VPCs. The number of peering connections required is given by the formula $number\ of\ connections\ = \frac{n(n-1)}{2}$ (where n is the number of VPCs). Therefore for 4 VPCs, 6 peering connections are required.

Number of VPCs	No of Peering Connections Required
3	3
4	6
5	10
6	15
7	21
8	28
9	36



Not only do the peering connections need to be created but so do the appropriate security groups of each instance and routes in each route tables to allow pinging from each of the other instances.

3. AWS CLOUDFORMATION

AWS CloudFormation allows the deployment of AWS infrastructure using code in either YAML or JSON format.

3.1 CREATING FOUR VPCS

Four separate VPCs needed creating each with different CIDR blocks to allow for cross VPC communication.

VPC1 – 10.1.0.0/16
VPC2 – 10.2.0.0/16
VPC3 – 10.3.0.0/16
VPC4 – 10.4.0.0/16

```
Vpc1:
  Type: 'AWS::EC2::VPC'
  Properties:
    CidrBlock: 10.1.0.0/16
    EnableDnsSupport: true
    EnableDnsHostnames: true
    Tags:
      - Key: Name
        Value: Vpc1-Production

Vpc2:
  Type: 'AWS::EC2::VPC'
  Properties:
    CidrBlock: 10.2.0.0/16
    EnableDnsSupport: true
    EnableDnsHostnames: true
    Tags:
      - Key: Name
        Value: Vpc2-Development
Vpc3:
  Type: 'AWS::EC2::VPC'
  Properties:
    CidrBlock: 10.3.0.0/16
    EnableDnsSupport: true
    EnableDnsHostnames: true
    Tags:
      - Key: Name
        Value: Vpc3-Testing
Vpc4:
  Type: 'AWS::EC2::VPC'
  Properties:
    CidrBlock: 10.4.0.0/16
    EnableDnsSupport: true
    EnableDnsHostnames: true
    Tags:
      - Key: Name
        Value: Vpc4-Shared
```

3.2 INTERNET GATEWAY

To allow SSH into a Bastion Host in a public subnet the subnet must have internet access and therefore the VPC that contains the Bastion Host needs an Internet Gateway attached to it. The Internet Gateway needs creating and then attaching to the chosen VPC.

```
Vpc1Igw:
  Type: AWS::EC2::InternetGateway

Vpc1GatewayAttachment:
  Type: AWS::EC2::VPCGatewayAttachment
  Properties:
    InternetGatewayId: !GetAtt Vpc1Igw.InternetGatewayId
    VpcId: !GetAtt Vpc1.VpcId
```

3.3 PEERING CONNECTIONS

Six peering connections are required to enable communication between all 4 VPCs. For brevity I've only shown the first three here (each from VPC 1 to the other three VPCs) but the other three are identical just connecting 2-3, 2-3 and 3-4.

```
PeeringConnection1to2:
  Type: 'AWS::EC2::VPCPeeringConnection'
  Properties:
    PeerVpcId: !GetAtt Vpc2.VpcId
    VpcId: !GetAtt Vpc1.VpcId
    Tags:
      - Key: Name
        Value: Peering1-2

PeeringConnection1to3:
  Type: 'AWS::EC2::VPCPeeringConnection'
  Properties:
    PeerVpcId: !GetAtt Vpc3.VpcId
    VpcId: !GetAtt Vpc1.VpcId
    Tags:
      - Key: Name
        Value: Peering1-3

PeeringConnection1to4:
  Type: 'AWS::EC2::VPCPeeringConnection'
  Properties:
    PeerVpcId: !GetAtt Vpc4.VpcId
    VpcId: !GetAtt Vpc1.VpcId
    Tags:
      - Key: Name
        Value: Peering1-4
```

3.4 SUBNETS

Each instance needs to be in a subnet within the VPC. VPC1 will contain a public and a private subnet. Only the Bastion Host will be in the public subnet as that needs access to the internet (via the internet gateway) to enable SSH. All the other instances (Production Server, Dev Server, Shared Server and Test Server) will be in private subnets, one in each VPC.

```
PublicSubnet1:
  Type: 'AWS::EC2::Subnet'
  Properties:
    VpcId: !Ref Vpc1
    CidrBlock: 10.1.1.0/24
    AvailabilityZone: eu-west-2a
    MapPublicIpOnLaunch: true
    Tags:
      - Key: Name
        Value: PublicSubnet1

PrivateSubnet1:
  Type: 'AWS::EC2::Subnet'
  Properties:
    VpcId: !Ref Vpc1
    CidrBlock: 10.1.2.0/24
    AvailabilityZone: eu-west-2a
    MapPublicIpOnLaunch: false
    Tags:
      - Key: Name
        Value: PrivateSubnet1

PrivateSubnet2:
  Type: 'AWS::EC2::Subnet'
  Properties:
    VpcId: !Ref Vpc2
    CidrBlock: 10.2.1.0/24
    AvailabilityZone: eu-west-2a
    MapPublicIpOnLaunch: false
    Tags:
      - Key: Name
        Value: PrivateSubnet2

PrivateSubnet3:
  Type: 'AWS::EC2::Subnet'
  Properties:
    VpcId: !Ref Vpc3
    CidrBlock: 10.3.1.0/24
    AvailabilityZone: eu-west-2a
    MapPublicIpOnLaunch: false
    Tags:
      - Key: Name
        Value: PrivateSubnet3

PrivateSubnet4:
  Type: 'AWS::EC2::Subnet'
  Properties:
    VpcId: !Ref Vpc4
    CidrBlock: 10.4.1.0/24
    AvailabilityZone: eu-west-2a
    MapPublicIpOnLaunch: false
    Tags:
      - Key: Name
        Value: PrivateSubnet4
```

Each subnet has its own CIDR range. Both Public Subnet 1 and Private Subnet 1 are in VPC 1 and therefore have CIDR ranges within the VPC1 CIDR range. These are all launch in eu-west-2a (London)

Public Subnet 1 – 10.1.1.0/24
Private Subnet 1 – 10.1.2.0/24
Private Subnet 2 – 10.2.1.0/24
Private Subnet 3 – 10.3.1.0/24
Private Subnet 4 – 10.4.1.0/24

3.5 SECURITY GROUPS

When each instance is launched, it needs a security group assigned to it that will allow appropriate data ingress only on the ports desired. The Bastion Host needs SSH permissions (TCP – Port 22) to connect over the internet. This has been allowed from anywhere for the sake of this project but for security the IP addresses should be limited. All of the instances need to be allowed to be pinged (ICMP) from any of the VPCs and SSH just from the Bastion Host. Security groups are not transferrable between VPCs hence needing a security group for the instance in each VPC despite them being identical.

```
BastionSecurityGroupPing1:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupName: Bastion Security GroupPing1
    GroupDescription: Security group to allow ping all VPCs
    VpcId: !GetAtt Vpc1.VpcId
    SecurityGroupIngress:
      - IpProtocol: icmp
        FromPort: -1
        ToPort: -1
        CidrIp: 10.0.0.0/8
      - IpProtocol: TCP
        FromPort: 22
        ToPort: 22
        CidrIp: 0.0.0.0/0
    Tags:
      - Key: Name
        Value: SecurityGroupPing1

SecurityGroupPing2:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupName: Server Security GroupPing2
    GroupDescription: Security group to allow ping all directions
    VpcId: !GetAtt Vpc2.VpcId
    SecurityGroupIngress:
      - IpProtocol: icmp
        FromPort: -1
        ToPort: -1
        CidrIp: 10.0.0.0/8
      - IpProtocol: TCP
        FromPort: 22
        ToPort: 22
        CidrIp: 10.1.1.249/32
    Tags:
      - Key: Name
        Value: SecurityGroupPing2
```

I have only shown two security groups (one in VPC1 and one in VPC2) but there are two more for VPC3 and 4.

3.6 LAUNCHING INSTANCES

Five EC2 t2.micro instances will be launched for this project. One Bastion Host in the public subnet and then four other instances in each of the separate VPCs private subnet. These instances are all launched from a "Launch Template". The Launch Template defines the size of the instances (T2-Micro) and the AMI of a free-tier Linux machine specifically in eu-west-2. These could be done with parameters and mapping to enable this to be easily done in other regions.

```
MyLaunchTemplate:
  Type: 'AWS::EC2::LaunchTemplate'
  Properties:
    LaunchTemplateName: TestLaunchTemplate
    LaunchTemplateData:
      InstanceType: t2.micro
      ImageId: ami-0d76271a8a1525c1a
```

```
BastionHost:
  Type: 'AWS::EC2::Instance'
  Properties:
    AvailabilityZone: eu-west-2a
    LaunchTemplate:
      LaunchTemplateId: !Ref MyLaunchTemplate
      Version: !GetAtt MyLaunchTemplate.LatestVersionNumber
    SubnetId: !GetAtt PublicSubnet1.SubnetId
    PrivateIpAddress: 10.1.1.249
    KeyName: Bastion-Key-Pair
    SecurityGroupIds:
      - !GetAtt BastionSecurityGroupPing1.GroupId
    Tags:
      - Key: Name
        Value: Bastion Host

ProductionServer:
  Type: 'AWS::EC2::Instance'
  Properties:
    AvailabilityZone: eu-west-2a
    LaunchTemplate:
      LaunchTemplateId: !Ref MyLaunchTemplate
      Version: !GetAtt MyLaunchTemplate.LatestVersionNumber
    SubnetId: !GetAtt PrivateSubnet1.SubnetId
    PrivateIpAddress: 10.1.2.249
    KeyName: Servers-Key-Pair
    SecurityGroupIds:
      - !GetAtt SecurityGroupPing1.GroupId
    Tags:
      - Key: Name
        Value: Production Server
```

Shown above is only the Bastion Host and Production Server (in VPC1's Private Subnet) but three other instances were also created in each of the other VPC's private subnets using the same settings (apart from the private IP addresses).

NOTE: It's important to note here that the Bastion Host and the Servers have Key Pairs. This is the only part of the process that has not been automated and was done in advance. I created a Bastion Host Key Pair and a Server Key Pair. The Bastion Host Key Pair is what I use to SSH into the Bastion Host from a local machine over the internet and the Server Key Pair is copied onto the Bastion Host using PowerShell to allow it to SSH into all of the other servers.

3.7 ROUTE TABLES

Each subnet needs a route table that defines where traffic is routed. In CloudFormation this is a three step process.

1. Create the route table
2. Associate the route table with a subnet
3. Create the routes within the route table

Create the Route Tables - PublicSubnet1RouteTable:

```
PublicSubnet1RouteTable:
  Type: 'AWS::EC2::RouteTable'
  Properties:
    VpcId: !GetAtt Vpc1.VpcId
    Tags:
      - Key: Name
        Value: PublicSub1RouteTable
```

Associate the route table with a subnet - PublicSubnet1RouteTableAssociation:

```
PublicSubnet1RouteTableAssociation:
  Type: AWS::EC2::SubnetRouteTableAssociation
  Properties:
    RouteTableId: !GetAtt PublicSubnet1RouteTable.RouteTableId
    SubnetId: !GetAtt PublicSubnet1.SubnetId
```

After these two steps we need to define the routes. This is how the data will around the network.

1. Internet access for the public subnet via the internet gateway
2. Connect each VPC using the previously created peering connections

This is all of the routes required for this project.

```
PublicSubnet1IgwRoute:
  Type: AWS::EC2::Route
  Properties:
    RouteTableId: !GetAtt PublicSubnet1RouteTable.RouteTableId
    DestinationCidrBlock: 0.0.0.0/0
    GatewayId: !GetAtt Vpc1Igw.InternetGatewayId


PublicSubnet1RouteVpc1to2:
  Type: AWS::EC2::Route
  Properties:
    RouteTableId: !GetAtt PublicSubnet1RouteTable.RouteTableId
    DestinationCidrBlock: !GetAtt Vpc2.CidrBlock
    VpcPeeringConnectionId: !GetAtt PeeringConnection1to2.Id

PublicSubnet1RouteVpc1to3:
  Type: AWS::EC2::Route
  Properties:
    RouteTableId: !GetAtt PublicSubnet1RouteTable.RouteTableId
    DestinationCidrBlock: !GetAtt Vpc3.CidrBlock
    VpcPeeringConnectionId: !GetAtt PeeringConnection1to3.Id

PublicSubnet1RouteVpc1to4:
  Type: AWS::EC2::Route
  Properties:
    RouteTableId: !GetAtt PublicSubnet1RouteTable.RouteTableId
    DestinationCidrBlock: !GetAtt Vpc4.CidrBlock
    VpcPeeringConnectionId: !GetAtt PeeringConnection1to4.Id
```

This process is then repeated (excluding the IGW route) for VPC-2, 3 and 4 to allow each VPC to connect to the others.

3.8 OUTPUTS

Finally as I need to SSH into the Bastion Host in order to ping the instances I ensured that CloudFormation would output the public IP Address of the Bastion Host to make that process easier.

```
Outputs:
  BastionHostPublicIp:
    Value: !GetAtt BastionHost.PublicIp
```

4. PYTHON CODE

Although it would be easy to SSH manually into each server from the Bastion Host and then manually ping each instance I wanted the extra challenge of writing a script in python that would automate the process.

This involved two separate python files. The first is designed to iterate through the list of private IP addresses (hardcoded in) of each server and do two things. Firstly it secure copies (using SCP) the second python file onto each server and SSH into the machine and run the second python file. If either of those stages fail, the code sets the key of the IP address into a dictionary and then the value of fail. If it succeeds then that is added instead.

When the second python file is run on each server, this iterates through a list of IP address and attempts to ping each one. It the writes the results to a dictionary and prints the dictionary upon completion.

4.1 FIRST FILE THAT INSTALLS AND RUNS SECOND FILE ON EACH SERVER

This is the first python code:

```
#!/bin/env python3
import os

def run_ping_on_all_instances():
    ipaddresses = {
        "10.1.2.249": "Production Server (VPC1)",
        "10.2.1.29":  "Dev Server (VPC2)",
        "10.3.1.129": "Test Server(VPC3)",
        "10.4.1.22":  "Shared Server(VPC4)"
}
    results_dict = {}
    for ipaddress in ipaddresses:
        scp_result = os.system(f"scp -o StrictHostKeyChecking=no -o
ConnectTimeout=5 ~/pythonfiles/pingallinstances.py {ipaddress}:/tmp/ ")
        if scp_result != 0:
            results_dict[ipaddress] = "Fail"
            continue
        ssh_result = os.system(f"ssh -o StrictHostKeyChecking=no -o
ConnectTimeout=5 {ipaddress} python3 /tmp/pingallinstances.py")

        if ssh_result != 0:
            results_dict[ipaddress] = "Fail"
        else:
            results_dict[ipaddress] = "Success"

    print("-----------------------------------------------")
    print("These are the overall results for each machine:")
    for ipaddress in results_dict:
        print(f"{ipaddresses[ipaddress]} ({ipaddress}):
{results_dict[ipaddress]}")
    print("-----------------------------------------------")

run_ping_on_all_instances()
```

4.2 SECOND FILE THAT PINGS EACH SERVER

```
#!/bin/env python3
import os
import socket
import sys


def pingallinstances():
    hostname = socket.gethostname()
    host_ipaddress = str(socket.gethostbyname(hostname))

    ipaddresses = {"10.1.2.249":"Production Server (VPC1)",
"10.2.1.29":"Dev Server (VPC2)", "10.3.1.129":"Test Server(VPC3)",
"10.4.1.22":"Shared Server(VPC4)"}
    passfaildict = {}
    success = True
    print()
    print()
    print("-------------------------------------------------------")
    print(f"Starting to ping all ip addresses from: {host_ipaddress}
({ipaddresses[host_ipaddress]})")
    print("-------------------------------------------------------")
    print()
    print()
    sys.stdout.flush()
    for ipaddress in ipaddresses:
        print(f"Pinging: {ipaddresses[ipaddress]}")
        sys.stdout.flush()
        individual_result = os.system(f"ping -c 2 -W 1 {ipaddress}")
        print()
        print("----------------------")
        print()
        sys.stdout.flush()
        if individual_result == 0:
            passfaildict[ipaddress] = "Success"
        else:
            passfaildict[ipaddress] = "Fail"
            success = False
    for i in [4]:
        print()
    print("////////////////////////////////////////////////////////////////")
    if success:
        print(f"{ipaddresses[host_ipaddress]} has succeeded in pinging all
other machines.")
    else:
        print(f"Unfortunately the {ipaddresses[host_ipaddress]}
({host_ipaddress}) has failed to ping all other machines.")
    for ipaddress in passfaildict:
        print(f"{ipaddresses[ipaddress]} ({ipaddress}):
{passfaildict[ipaddress]}")
    # print(passfaildict)
    print("////////////////////////////////////////////////////////////////")
    for i in [4]:
        print()
    if not success:
        return 1

    return 0


exit(pingallinstances())
```

5. POWERSHELL COMMANDS

After CloudFormation has completed building the infrastructure it is possible to SSH into the Bastion Host from a local machine using the Bastion Host Key Pair as its identity (see note in 3.6 regarding the key pairs).

1. Set a variable in PowerShell for the public IP address of the Bastion Host (found in the output of the CloudFormation stack).

   ${bastion-host-ipaddress} = "Public-ip-Output-Here"

2. Set a variable for the location of the Bastion Host Key Pair.

   ${bastion-host-keypair} = ".\Downloads\Bastion-Key-Pair.pem"

3. Create a folder on the Bastion Host for the python files to be stored in.

   ssh -i ${bastion-host-keypair} ec2-user@${bastion-host-ipaddress} mkdir ./pythonfiles

4. Secure copy (SCP) local python files into newly created pythonfiles folder on Bastion Host.

   scp -i ${bastion-host-keypair} .\local_path_of_two_python_files*.py" ec2-user@${bastion-host-ipaddress}:./pythonfiles/

5. Secure copy Server Key Pair from local host onto Bastion Host so that it will be able to access each of the servers. This copies the private key into a file called id_rsa which is the default file used for authentication.

   scp -i ${bastion-host-keypair} .\Downloads\Servers-Key-Pair.pem ec2-user@${bastion-host-ipaddress}:~/.ssh/id_rsa

6. Change the permissions of the id_rsa file so that only the user can read and write and that the group and other can no read, write or execute. This is required to use id_rsa as the identity.

   ssh -i ${bastion-host-keypair} ec2-user@${bastion-host-ipaddress} chmod 600 ~/.ssh/id_rsa

7. Finally SSH onto the Bastion Host and using python3 run the “Launch_Ping_All_Instances.py” file. This then begins the pinging process.

   ssh -i ${bastion-host-keypair} ec2-user@${bastion-host-ipaddress} python3 ~/pythonfiles/Launch_Ping_All_Instances.py

6. OUTPUT

6.1 EVERYTHING WORKING AS EXPECTED

Below is the output that will be seen in PowerShell as everything is working as expected and all instances can ping all others via each peering connection.

```
--------------------------------------------------
Starting to ping all ip addresses from: 10.1.2.249 (Production Server (VPC1))
--------------------------------------------------


Pinging: Production Server (VPC1)
PING 10.1.2.249 (10.1.2.249) 56(84) bytes of data.
64 bytes from 10.1.2.249: icmp_seq=1 ttl=127 time=0.019 ms
64 bytes from 10.1.2.249: icmp_seq=2 ttl=127 time=0.032 ms

--- 10.1.2.249 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1050ms
rtt min/avg/max/mdev = 0.019/0.025/0.032/0.006 ms

--------------------

Pinging: Dev Server (VPC2)
PING 10.2.1.29 (10.2.1.29) 56(84) bytes of data.
64 bytes from 10.2.1.29: icmp_seq=1 ttl=127 time=0.909 ms
64 bytes from 10.2.1.29: icmp_seq=2 ttl=127 time=0.490 ms

--- 10.2.1.29 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1037ms
rtt min/avg/max/mdev = 0.490/0.699/0.909/0.209 ms

--------------------

Pinging: Test Server(VPC3)
PING 10.3.1.129 (10.3.1.129) 56(84) bytes of data.
64 bytes from 10.3.1.129: icmp_seq=1 ttl=127 time=0.884 ms
64 bytes from 10.3.1.129: icmp_seq=2 ttl=127 time=0.417 ms

--- 10.3.1.129 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1036ms
rtt min/avg/max/mdev = 0.417/0.650/0.884/0.233 ms

--------------------

Pinging: Shared Server(VPC4)
PING 10.4.1.22 (10.4.1.22) 56(84) bytes of data.
64 bytes from 10.4.1.22: icmp_seq=1 ttl=127 time=0.799 ms
64 bytes from 10.4.1.22: icmp_seq=2 ttl=127 time=0.420 ms
```

```
--- 10.4.1.22 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1036ms
rtt min/avg/max/mdev = 0.420/0.609/0.799/0.189 ms

--------------------


//////////////////////////////////////////////////////////
Production Server (VPC1) has succeeded in pinging all other machines.
Production Server (VPC1) (10.1.2.249): Success
Dev Server (VPC2) (10.2.1.29): Success
Test Server(VPC3) (10.3.1.129): Success
Shared Server(VPC4) (10.4.1.22): Success
//////////////////////////////////////////////////////////

Warning: Permanently added '10.2.1.29' (ED25519) to the list of known
hosts.


--------------------------------------------------
Starting to ping all ip addresses from: 10.2.1.29 (Dev Server (VPC2))
--------------------------------------------------


Pinging: Production Server (VPC1)
PING 10.1.2.249 (10.1.2.249) 56(84) bytes of data.
64 bytes from 10.1.2.249: icmp_seq=1 ttl=127 time=0.759 ms
64 bytes from 10.1.2.249: icmp_seq=2 ttl=127 time=0.508 ms

--- 10.1.2.249 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1078ms
rtt min/avg/max/mdev = 0.508/0.633/0.759/0.125 ms

--------------------

Pinging: Dev Server (VPC2)
PING 10.2.1.29 (10.2.1.29) 56(84) bytes of data.
64 bytes from 10.2.1.29: icmp_seq=1 ttl=127 time=0.018 ms
64 bytes from 10.2.1.29: icmp_seq=2 ttl=127 time=0.033 ms

--- 10.2.1.29 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1036ms
rtt min/avg/max/mdev = 0.018/0.025/0.033/0.007 ms

--------------------

Pinging: Test Server(VPC3)
PING 10.3.1.129 (10.3.1.129) 56(84) bytes of data.
64 bytes from 10.3.1.129: icmp_seq=1 ttl=127 time=0.673 ms
64 bytes from 10.3.1.129: icmp_seq=2 ttl=127 time=0.412 ms
```

```
--- 10.3.1.129 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1037ms
rtt min/avg/max/mdev = 0.412/0.542/0.673/0.130 ms

--------------------

Pinging: Shared Server(VPC4)
PING 10.4.1.22 (10.4.1.22) 56(84) bytes of data.
64 bytes from 10.4.1.22: icmp_seq=1 ttl=127 time=0.840 ms
64 bytes from 10.4.1.22: icmp_seq=2 ttl=127 time=0.433 ms

--- 10.4.1.22 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1036ms
rtt min/avg/max/mdev = 0.433/0.636/0.840/0.203 ms

--------------------


//////////////////////////////////////////////////////////
Dev Server (VPC2) has succeeded in pinging all other machines.
Production Server (VPC1) (10.1.2.249): Success
Dev Server (VPC2) (10.2.1.29): Success
Test Server(VPC3) (10.3.1.129): Success
Shared Server(VPC4) (10.4.1.22): Success
//////////////////////////////////////////////////////////

Warning: Permanently added '10.3.1.129' (ED25519) to the list of known
hosts.


--------------------------------------------------
Starting to ping all ip addresses from: 10.3.1.129 (Test Server(VPC3))
--------------------------------------------------


Pinging: Production Server (VPC1)
PING 10.1.2.249 (10.1.2.249) 56(84) bytes of data.
64 bytes from 10.1.2.249: icmp_seq=1 ttl=127 time=0.383 ms
64 bytes from 10.1.2.249: icmp_seq=2 ttl=127 time=0.469 ms

--- 10.1.2.249 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1065ms
rtt min/avg/max/mdev = 0.383/0.426/0.469/0.043 ms

--------------------

Pinging: Dev Server (VPC2)
PING 10.2.1.29 (10.2.1.29) 56(84) bytes of data.
64 bytes from 10.2.1.29: icmp_seq=1 ttl=127 time=0.469 ms
64 bytes from 10.2.1.29: icmp_seq=2 ttl=127 time=0.357 ms
```

```
--- 10.2.1.29 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1036ms
rtt min/avg/max/mdev = 0.357/0.413/0.469/0.056 ms

--------------------

Pinging: Test Server(VPC3)
PING 10.3.1.129 (10.3.1.129) 56(84) bytes of data.
64 bytes from 10.3.1.129: icmp_seq=1 ttl=127 time=0.016 ms
64 bytes from 10.3.1.129: icmp_seq=2 ttl=127 time=0.032 ms

--- 10.3.1.129 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1036ms
rtt min/avg/max/mdev = 0.016/0.024/0.032/0.008 ms

--------------------

Pinging: Shared Server(VPC4)
PING 10.4.1.22 (10.4.1.22) 56(84) bytes of data.
64 bytes from 10.4.1.22: icmp_seq=1 ttl=127 time=0.729 ms
64 bytes from 10.4.1.22: icmp_seq=2 ttl=127 time=0.524 ms

--- 10.4.1.22 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1037ms
rtt min/avg/max/mdev = 0.524/0.626/0.729/0.102 ms

--------------------


//////////////////////////////////////////////////////////
Test Server(VPC3) has succeeded in pinging all other machines.
Production Server (VPC1) (10.1.2.249): Success
Dev Server (VPC2) (10.2.1.29): Success
Test Server(VPC3) (10.3.1.129): Success
Shared Server(VPC4) (10.4.1.22): Success
//////////////////////////////////////////////////////////

Warning: Permanently added '10.4.1.22' (ED25519) to the list of known
hosts.


---------------------------------------------------
Starting to ping all ip addresses from: 10.4.1.22 (Shared Server(VPC4))
---------------------------------------------------


Pinging: Production Server (VPC1)
PING 10.1.2.249 (10.1.2.249) 56(84) bytes of data.
64 bytes from 10.1.2.249: icmp_seq=1 ttl=127 time=0.457 ms
64 bytes from 10.1.2.249: icmp_seq=2 ttl=127 time=0.578 ms
```

```
--- 10.1.2.249 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1017ms
rtt min/avg/max/mdev = 0.457/0.517/0.578/0.060 ms

--------------------

Pinging: Dev Server (VPC2)
PING 10.2.1.29 (10.2.1.29) 56(84) bytes of data.
64 bytes from 10.2.1.29: icmp_seq=1 ttl=127 time=0.425 ms
64 bytes from 10.2.1.29: icmp_seq=2 ttl=127 time=0.473 ms

--- 10.2.1.29 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1036ms
rtt min/avg/max/mdev = 0.425/0.449/0.473/0.024 ms

--------------------

Pinging: Test Server(VPC3)
PING 10.3.1.129 (10.3.1.129) 56(84) bytes of data.
64 bytes from 10.3.1.129: icmp_seq=1 ttl=127 time=0.649 ms
64 bytes from 10.3.1.129: icmp_seq=2 ttl=127 time=0.385 ms

--- 10.3.1.129 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1036ms
rtt min/avg/max/mdev = 0.385/0.517/0.649/0.132 ms

--------------------

Pinging: Shared Server(VPC4)
PING 10.4.1.22 (10.4.1.22) 56(84) bytes of data.
64 bytes from 10.4.1.22: icmp_seq=1 ttl=127 time=0.017 ms
64 bytes from 10.4.1.22: icmp_seq=2 ttl=127 time=0.033 ms

--- 10.4.1.22 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1036ms
rtt min/avg/max/mdev = 0.017/0.025/0.033/0.008 ms

--------------------


//////////////////////////////////////////////////////////
Shared Server(VPC4) has succeeded in pinging all other machines.
Production Server (VPC1) (10.1.2.249): Success
Dev Server (VPC2) (10.2.1.29): Success
Test Server(VPC3) (10.3.1.129): Success
Shared Server(VPC4) (10.4.1.22): Success
//////////////////////////////////////////////////////////

------------------------------------------
These are the overall results for each machine:
```

```
Production Server (VPC1) (10.1.2.249): Success
Dev Server (VPC2) (10.2.1.29): Success
Test Server(VPC3) (10.3.1.129): Success
Shared Server(VPC4) (10.4.1.22): Success
-----------------------------------------
```

You can see here how each instance tries to ping all the instances (including itself). This reports back on the success or failure of this. A summary is given at the end for each instance.

6.2 CAUSING ISSUES INTENTIONALLY TO SHOW PROOF OF CONCEPT

There are a number of ways to force failed results. These include, but are not limited to, stopping instances, changing security group permissions, deleting peering connections and changing Network ACL permissions. To prove that the python script is pinging and returns the correct output I undertook two tests.

6.2.1 STOPPING AN INSTANCE

Using the AWS console I stopped one of the servers (Dev Server – VPC2) and the re-ran the python script to show how it will timeout and the results.

(I have only shown a brief part of the output to show what is different)

Each of the instances tried to ping the Dev Server but due to it being offline they all failed and so each machine had an overall fail.

```
--------------------

Pinging: Dev Server (VPC2)
PING 10.2.1.29 (10.2.1.29) 56(84) bytes of data.

--- 10.2.1.29 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1036ms



--------------------


////////////////////////////////////////////////////////
Unfortunately the Production Server (VPC1) (10.1.2.249) has failed to ping all other
machines.
Production Server (VPC1) (10.1.2.249): Success
Dev Server (VPC2) (10.2.1.29): Fail
Test Server(VPC3) (10.3.1.129): Success
```

```
Shared Server(VPC4) (10.4.1.22): Success
////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////
Unfortunately the Test Server(VPC3) (10.3.1.129) has failed to ping all other machines.
Production Server (VPC1) (10.1.2.249): Success
Dev Server (VPC2) (10.2.1.29): Fail
Test Server(VPC3) (10.3.1.129): Success
Shared Server(VPC4) (10.4.1.22): Success
////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////
Unfortunately the Shared Server(VPC4) (10.4.1.22) has failed to ping all other machines.
Production Server (VPC1) (10.1.2.249): Success
Dev Server (VPC2) (10.2.1.29): Fail
Test Server(VPC3) (10.3.1.129): Success
Shared Server(VPC4) (10.4.1.22): Success
////////////////////////////////////////////////////////////


-------------------------------------------
These are the overall results for each machine:
Production Server (VPC1) (10.1.2.249): Fail
Dev Server (VPC2) (10.2.1.29): Fail
Test Server(VPC3) (10.3.1.129): Fail
Shared Server(VPC4) (10.4.1.22): Fail
-------------------------------------------
```

6.2.2 DELETING A PEERING CONNECTION

After having restarted the Dev Server I then deleted one of the peering connections that connects VPC3 and VPC4 as well as the routes in the route table associated with those peering connections. This means that the instances in those VPCs can not communicate with eachother, however all of the instances can communicate with them. We therefore expect to see a different failure output.

Production Server should fully succeed.
Dev Server should fully succeed.
Test Server (VPC3) can ping itself, Production and Dev but not the Shared Server in VPC4.
Shared Server (VPC4) can ping itself, Production and Dev but not the Test Server in VPC3.

The following is the output from that test:

```
//////////////////////////////////////////////////////////
Production Server (VPC1) has succeeded in pinging all other machines.
Production Server (VPC1) (10.1.2.249): Success
Dev Server (VPC2) (10.2.1.29): Success
Test Server(VPC3) (10.3.1.129): Success
Shared Server(VPC4) (10.4.1.22): Success
//////////////////////////////////////////////////////////


//////////////////////////////////////////////////////////
Dev Server (VPC2) has succeeded in pinging all other machines.
Production Server (VPC1) (10.1.2.249): Success
Dev Server (VPC2) (10.2.1.29): Success
Test Server(VPC3) (10.3.1.129): Success
Shared Server(VPC4) (10.4.1.22): Success
//////////////////////////////////////////////////////////


//////////////////////////////////////////////////////////
Unfortunately the Test Server(VPC3) (10.3.1.129) has failed to ping all other machines.
Production Server (VPC1) (10.1.2.249): Success
Dev Server (VPC2) (10.2.1.29): Success
Test Server(VPC3) (10.3.1.129): Success
Shared Server(VPC4) (10.4.1.22): Fail
//////////////////////////////////////////////////////////


//////////////////////////////////////////////////////////
Unfortunately the Shared Server(VPC4) (10.4.1.22) has failed to ping all other
machines.
Production Server (VPC1) (10.1.2.249): Success
Dev Server (VPC2) (10.2.1.29): Success
Test Server(VPC3) (10.3.1.129): Fail
Shared Server(VPC4) (10.4.1.22): Success
//////////////////////////////////////////////////////////

---------------------------------------------
These are the overall results for each machine:
Production Server (VPC1) (10.1.2.249): Success
Dev Server (VPC2) (10.2.1.29): Success
Test Server(VPC3) (10.3.1.129): Fail
Shared Server(VPC4) (10.4.1.22): Fail
```

It can be seen from these that there isn't anything wrong with the servers as the Production Server and Dev Server successfully pinged all of the other machine. However there was an issue with the Test Server and Shared Server. As both of these successfully pinged the others servers but failed to ping each other this implies an issue with the peering connection and that should be looked into as a the issue.

7. CONCLUSION

Overall this exercise successfully shows how peering connections are able to connect separate VPCs as long as the CIDR ranges of each VPC don't overlap. This could also be done using a Transit Gateway. Transit Gateways are significantly easier to set up especially if you are connecting more then 4 VPCS. The number of peering connections based on the number of VPCs being connected is shown below.

However transit connections are often more expensive in their monthly costs. Therefore a trade-off is required between hours and complexity of setting up and maintenance vs. recurring monthly costs.

After having completed the test I deleted the stack using CloudFormation. This deletes everything that was created automatically including the instances, VPCs, Peering Connections, Security Groups etc. This prevents the risk of accidentally incurring any costs.