Article
Eduard Lebedyuk · Nov 4, 2022 9m read

VIP in AWS

If you're running IRIS in a mirrored configuration for HA in AWS, the question of providing a Mirror VIP (Virtual IP) becomes relevant. Virtual IP offers a way for downstream systems to interact with IRIS using one IP address. Even when a failover happens, downstream systems can reconnect to the same IP address and continue working.

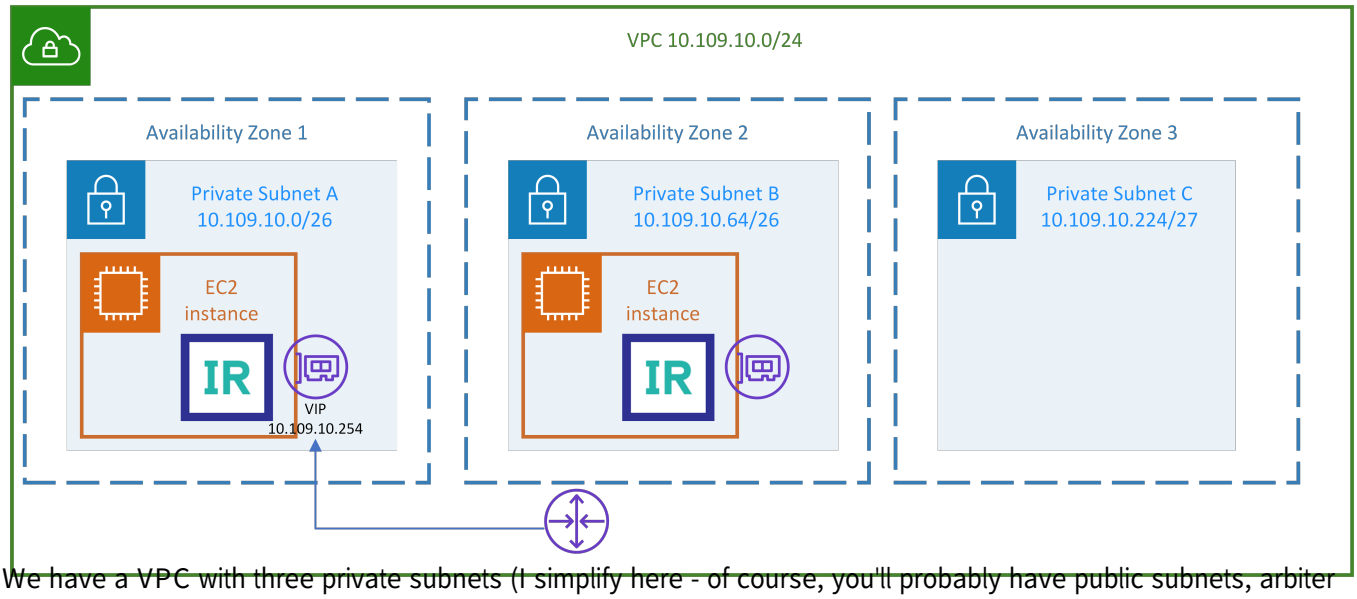
The main issue, when deploying to AWS, is that an IRIS VIP has a requirement of both mirror members being in the same subnet, from the docs:

> To use a mirror VIP, both failover members must be configured in the same subnet, and the VIP must belong to the same subnet as the network interface that is selected on each system

However, to get HA, IRIS mirror members must be deployed to different availability zones, which means different subnets (as subnets can be in only one az). One of the solutions might be load balancers, but they (A) cost money, and (B) if you need to route non-HTTP traffic (think TCP for HL7), you'll have to use Network Load Balancers which have a limit of 50 ports total.

In this article, I would like to provide a way to configure a Mirror VIP without the use of Network Load Balancing suggested in most other AWS reference architectures. In production, we have found limitations that impeded solutions with cost, 50 listener limits, DNS dependencies, and the dynamic nature of the two IP addresses AWS provides across the availability zones.

Architecture



We have a VPC with three private subnets (I simplify here - of course, you'll probably have public subnets, arbiter in another az, and so on, but this is an absolute minimum enough to demonstrate this approach). VPC is allocated IPs: 10.109.10.1 to 10.109.10.254; subnets (in different AZs) are: 10.109.10.1 to 10.109.10.62, 10.109.10.65 to 10.109.10.126, and 10.109.10.224 to 10.109.10.254.

Implementing VIP

1. On each EC2 instance (SourceDestCheck must be set to false), we will allocate the same IP address on the eth0:1 network interface. This IP address is in the VPC CIDR range - in a special VIP AZ. For example, we can use the last IP in a range - 10.109.10.254:

```
cat << EOFVIP >> /etc/sysconfig/network-scripts/ifcfg-eth0:1
          DEVICE=eth0:1
          ONPARENT=on
          IPADDR=10.109.10.254
          PREFIX=27
          EOFVIP
sudo chmod -x /etc/sysconfig/network-scripts/ifcfg-eth0:1
sudo ifconfig eth0:1 up
```

Depending on the os you might need to run:

```
ifconfig eth0:1
systemctl restart network
```

Important Routing Note!

AWS routing between local subnets is subnet specific - you can route an entire subnet to an eni, but not an individual /32 IP address. For VIP to work, VIP needs to reside in a small subnet, and the entire subnet would be routed to the eni, which is the approach used in this article. That means that only one VIP can be in a subnet, and nothing else can be placed in a VIP subnet. If you need multiple VIPs, each must reside in its own subnet. Alternatively, you can use an external to the VPC IP address (but still private) for the /32 routing to work.

2. On mirror failover event, update route table to point to eni on a new Primary. We'll use a ZMIRROR callback to update the routing table after the current mirror member becomes the primary. This code uses Embedded Python to:
    - Get the IP on eth0:1
    - Get InstanceId and Region from instance metadata
    - Find the main route table for the EC2 VPC
    - Delete old route, if any
    - Add a new route pointing to itself

Here's the code:

```
import os
import urllib.request
import boto3
from botocore.exceptions import ClientError

PRIMARY_INTERFACE = 'eth0'
VIP_INTERFACE = 'eth0:1'

eth0_addresses = [
    line
    for line in os.popen(f'ip -4 addr show dev {PRIMARY_INTERFACE}').read().split('\n
```

```
')
    if line.strip().startswith('inet')
]

VIP = None
for address in eth0_addresses:
    if address.split(' ')[-1] == VIP_INTERFACE:
        VIP = address.split(' ')[5]

if VIP is None:
    raise ValueError('Failed to retrieve a valid VIP!')

# Lookup the current mirror member instance ID
instanceid = (
    urllib.request.urlopen('http://169.254.169.254/latest/meta-data/instance-id')
    .read()
    .decode()
)

region = (
    urllib.request.urlopen('http://169.254.169.254/latest/meta-
data/placement/region')
    .read()
    .decode()
)

session = boto3.Session(region_name=region)
ec2Resource = session.resource('ec2')
ec2Client = session.client('ec2')
instance = ec2Resource.Instance(instanceid)

# Look up the main route table ID for this VPC
vpc = ec2Resource.Vpc(instance.vpc.id)

for route_table in vpc.route_tables.all():
    # Update the main route table to point to this instance
    try:
        ec2Client.delete_route(
            DestinationCidrBlock=VIP, RouteTableId=str(route_table.id)
        )
    except ClientError as exc:
        if exc.response['Error']['Code'] == 'InvalidRoute.NotFound':
            print('Nothing to remove, continue')
        else:
            raise exc
    # Add the new route
    ec2Client.create_route(
        DestinationCidrBlock=VIP,
        NetworkInterfaceId=instance.network_interfaces[0].id,
        RouteTableId=str(route_table.id),
    )
```

and the same code as a ZMIRROR routine:

```
NotifyBecomePrimary() PUBLIC {
  try {
    set dir = $system.Util.ManagerDirectory()_ "python"
```

```
    do ##class(%File).CreateDirectoryChain(dir)

    try {
      set boto3 = $system.Python.Import("boto3")
    } catch {
      set cmd = "pip3"
      set args($i(args)) = "install"
      set args($i(args)) = "--target"
      set args($i(args)) = dir
      set args($i(args)) = "boto3"
      set sc = $ZF(-100,"", cmd, .args)
      // for python before 3.7 also install dataclasses
      set boto3 = $system.Python.Import("boto3")
    }
    kill boto3

    set code =  "import os" _ $c(10) _
                "import urllib.request" _ $c(10) _
                "import boto3" _ $c(10) _
                "from botocore.exceptions import ClientError" _ $c(10) _
                "PRIMARY_INTERFACE = 'eth0'" _ $c(10) _
                "VIP_INTERFACE = 'eth0:1'" _ $c(10) _
                "eth0_addresses = [" _ $c(10) _
                "    line" _ $c(10) _
                "    for line in os.popen(f'ip -4 addr show dev {PRIMARY_INTERFACE}')
.read().split('\n')" _ $c(10) _
                "    if line.strip().startswith('inet')" _ $c(10) _
                "]" _ $c(10) _
                "VIP = None" _ $c(10) _
                "for address in eth0_addresses:" _ $c(10) _
                "    if address.split(' ')[-1] == VIP_INTERFACE:" _ $c(10) _
                "        VIP = address.split(' ')[5]" _ $c(10) _
                "if VIP is None:" _ $c(10) _
                "    raise ValueError('Failed to retrieve a valid VIP!')" _ $c(10) _
                "# Lookup the current mirror member instance ID" _ $c(10) _
                "instanceid = (" _ $c(10) _
                "    urllib.request.urlopen('http://169.254.169.254/latest/meta-
data/instance-id')" _ $c(10) _
                "    .read()" _ $c(10) _
                "    .decode()" _ $c(10) _
                ")" _ $c(10) _
                "region = (" _ $c(10) _
                "    urllib.request.urlopen('http://169.254.169.254/latest/meta-
data/placement/region')" _ $c(10) _
                "    .read()" _ $c(10) _
                "    .decode()" _ $c(10) _
                ")" _ $c(10) _
                "session = boto3.Session(region_name=region)" _ $c(10) _
                "ec2Resource = session.resource('ec2')" _ $c(10) _
                "ec2Client = session.client('ec2')" _ $c(10) _
                "instance = ec2Resource.Instance(instanceid)" _ $c(10) _
                "# Look up the main route table ID for this VPC" _ $c(10) _
                "vpc = ec2Resource.Vpc(instance.vpc.id)" _ $c(10) _
                "for route_table in vpc.route_tables.all():" _ $c(10) _
                "    # Update the main route table to point to this instance" _ $c(10
) _
                "    try:" _ $c(10) _
                "        ec2Client.delete_route(" _ $c(10) _
                "            DestinationCidrBlock=VIP, RouteTableId=str(route_table.i
```

```
d)" _ $c(10) _
                "        )" _ $c(10) _
                "    except ClientError as exc:" _ $c(10) _
                "        if exc.response['Error']['Code'] == 'InvalidRoute.NotFound':
" _ $c(10) _
                "            print('Nothing to remove, continue')" _ $c(10) _
                "        else:" _ $c(10) _
                "            raise exc" _ $c(10) _
                "    # Add the new route" _ $c(10) _
                "    ec2Client.create_route(" _ $c(10) _
                "        DestinationCidrBlock=VIP," _ $c(10) _
                "        NetworkInterfaceId=instance.network_interfaces[0].id," _ $c(
10) _
                "        RouteTableId=str(route_table.id)," _ $c(10) _
                "    )"


    set rc = $system.Python.Run(code)
    set sc = ##class(%SYS.System).WriteToConsoleLog("VIP assignment " _ $case(rc, 0:"
successful", :"error"), , $case(rc, 0:0, :1), "NotifyBecomePrimary:ZMIRROR")

  } catch ex {
    #dim ex As %Exception.General
    do ex.Log()
    set sc = ##class(%SYS.System).WriteToConsoleLog("Caught exception during VIP assi
gnment: " _ ex.DisplayString(), , 1, "NotifyBecomePrimary:ZMIRROR")
  }
  quit 1
}
```

Initial start

NotifyBecomePrimary is also called automatically on system start (after mirror reconnection), but if you want your non-mirrored environments to acquire VIP the same way use ZSTART routine:

```
SYSTEM() PUBLIC {
  if '$SYSTEM.Mirror.IsMember() {
    do NotifyBecomePrimary^ZMIRROR()
  }
  quit 1
}
```

Deletion

If you use automated provisioning tools, such as CloudFormation, this route must be deleted before the subnet can be deleted. You can add the deletion code to ^%ZSTOP, just don't forget to check for $SYSTEM.Mirror.IsPrimary() because when mirror primary shuts down, during ^%ZSTOP it's still primary. Overall I'd recommend external route deletion as a part of a provisioning tools script.

Permissions

Instance profile for a EC2 needs the following permissions:

1. Modify routes on specific route tables:

```
{
      "Action": [
            "ec2:CreateRoute",
            "ec2:ReplaceRoute",
            "ec2:DeleteRoute"
      ],
      "Resource": [
            "arn:aws:ec2:eu-west-2:123456789:route-table/rtb-1234567890",
            "arn:aws:ec2:eu-west-2:123456789:route-table/rtb-0987654321"
      ],
      "Effect": "Allow"
}
```

2. Discover routes (can be limited by region but not by a specific resource.

```
{
      "Condition": {
            "StringEquals": {
                  "ec2:Region": "eu-west-2"
            }
      },
      "Action": [
            "ec2:DescribeAddresses",
            "ec2:DescribeNetworkInterfaces",
            "ec2:DescribeRouteTables"
      ],
      "Resource": "*",
      "Effect": "Allow"
}
```

3. Allow moving Elastic IPs (can ' t be limited by resource, so we limit by Tag value):

```
{
      "Condition": {
            "StringLike": {
                  "ec2:ResourceTag/deploymentid": "your_value"
            }
      },
      "Action": "ec2:AssociateAddress",
      "Resource": "*",
      "Effect": "Allow"
}
```

Conclusion

And that's it! In the route table, we get a new route pointing to a current mirror Primary when the NotifyBecomePrimary event happens.

The author would like to thank Jared Trog and @Ron Sweeney for creating this approach.

The author would like to thank @Tomohiro Iwamoto for testing this approach and figuring out all the requirements for it to work.

#AWS #Cloud #Mirroring #InterSystems IRIS