Architecting Highly Available Applications on AWS

Lab 4

Table of Contents

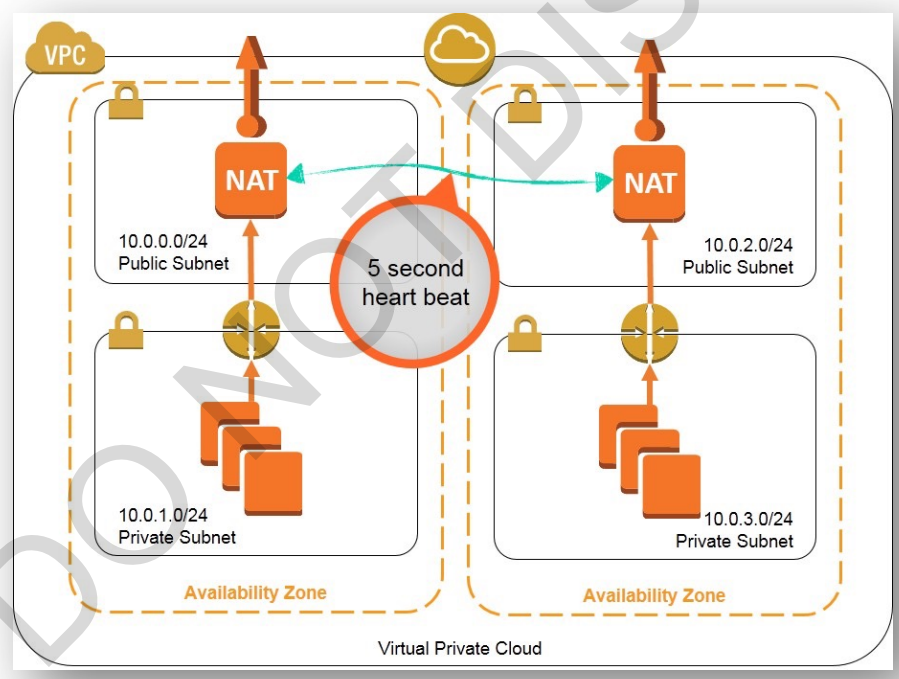
Introduction

Over the course of this bootcamp, you have learned how to better architect your applications for high availability, primarily by providing redundant services across availability zones (AZs) within a region and distributing *inbound* traffic across those services at various application tiers. In this lab, we will look at how to make *outbound* traffic originating from application tiers in VPC highly available, using NAT instances that span multiple AZs.

Before you begin this lab, let's review some important elements of VPC. Subnets within VPCs are either public or private. Public subnets have routes to an Internet gateway that is attached to the VPC, allowing for inbound and outbound public Internet traffic to and from those subnets. Private subnets, do not have routes to the Internet gateway and therefore cannot directly handle inbound or outbound traffic on the public Internet and need to rely on NAT-type devices that will handle outbound Internet traffic on their behalf.

Create a Multi-AZ VPC with Highly-Available NATs

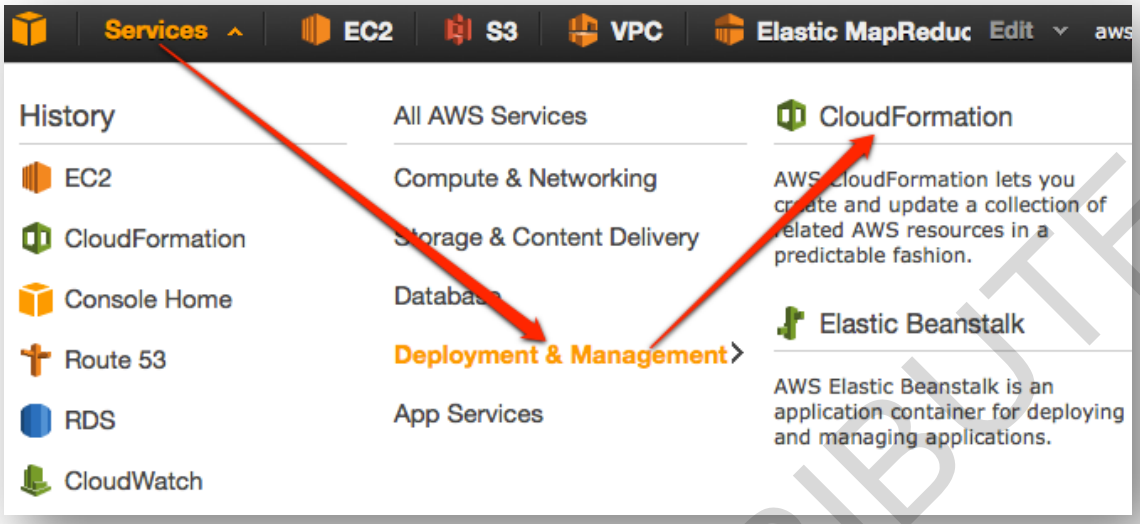
This lab will create a VPC that spans two AZs, with a public and private subnet in each AZ. A NAT instance based on Amazon Linux will be launched in each public subnet, and a non-NAT Amazon Linux instance will be created in each private subnet. For the public subnets, route rules will exist to send all outbound traffic to an Internet gateway. For the private subnets, route rules will exist to send all outbound traffic to the NAT instances in the availability zone of the private subnet. This configuration looks something like this (the IP blocks used in the illustration are different than the ones used in this lab).



Note: **Certain details such as instance names, database names, EIP addresses, security groups, CloudFormation stack names, etc., will differ from what's captured in the screen shots.**

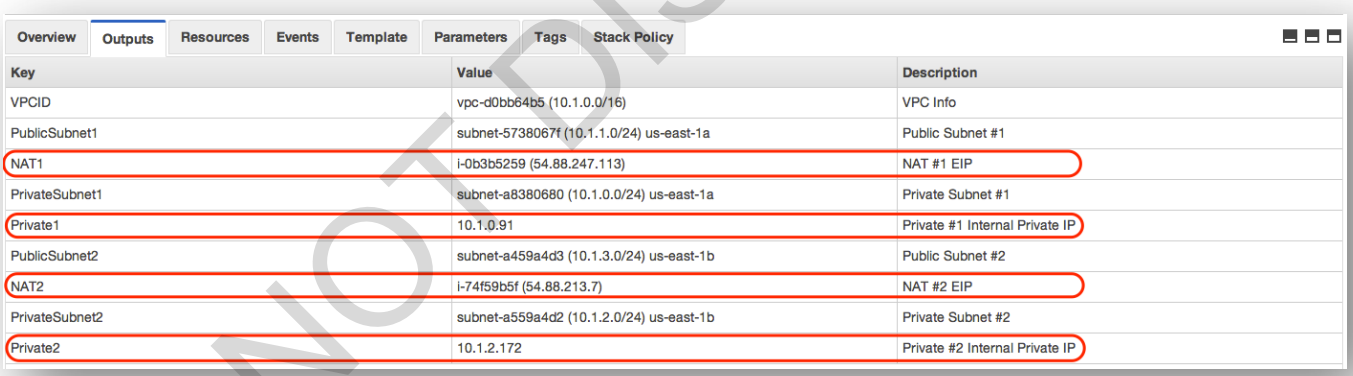
1. Log into the AWS Console with the user name and password that QwikLab provided for Lab 4.

2. From the AWS Console page, near the top left corner, click **Services**, and then **Deployment & Management** and **CloudFormation**.



3. On the CloudFormation page, you will see a stack for Lab 4. The status will be either **CREATE_IN_PROGRESS** or **CREATE_COMPLETE**. This stack will take upwards of 5-10 minutes to launch so it's best to pre-launch this lab.

4. Select the stack. At the bottom of the page, click the **Outputs** tab to view the EIPs of the newly formed NAT instances (NAT1 and NAT2). You will refer to this information later in this lab.



Key	Value	Description
VPCID	vpc-d0bb64b5 (10.1.0.0/16)	VPC Info
PublicSubnet1	subnet-5738067f (10.1.1.0/24) us-east-1a	Public Subnet #1
NAT1	i-0b3b5259 (54.88.247.113)	NAT #1 EIP
PrivateSubnet1	subnet-a8380680 (10.1.0.0/24) us-east-1a	Private Subnet #1
Private1	10.1.0.91	Private #1 Internal Private IP
PublicSubnet2	subnet-a459a4d3 (10.1.3.0/24) us-east-1b	Public Subnet #2
NAT2	i-74f59b5f (54.88.213.7)	NAT #2 EIP
PrivateSubnet2	subnet-a559a4d2 (10.1.2.0/24) us-east-1b	Private Subnet #2
Private2	10.1.2.172	Private #2 Internal Private IP

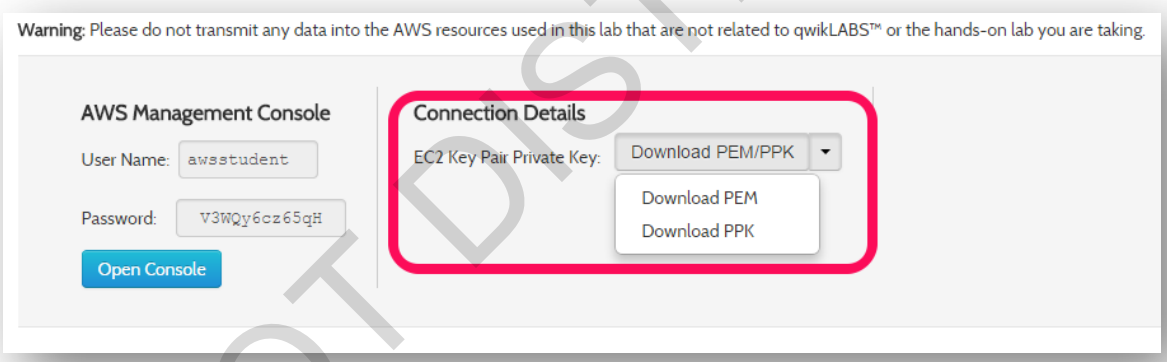
Exploring the Architecture

In this lab, we have a VPC with 4 subnets total – 2 in one AZ (1 public, 1 private), and 2 in another AZ (1 public, 1 private). The route tables for both public subnets map all outbound traffic to an Internet gateway that has been created and associated to this VPC. For the private subnets, routes exist for all outbound traffic to the NAT instance in the public subnet in the respective AZ (e.g. private subnet 1 0.0.0.0/0 -> NAT instance in public subnet 1). Internal routes exist for all subnets within the VPC that allow for unrestricted inter-subnet traffic. Feel free to explore this using the AWS console.

Testing Outbound Internet Access

With the current implementation in place, instances in private subnet 1 can originate outbound traffic through the NAT in public subnet 1, and instances in private subnet 2 can originate outbound traffic through the NAT in public subnet 2. Using the SSH keys provided for this lab, let's test this. Let's SSH into the private instance in subnet 1. In order to do that, we have to connect to one of the NAT servers and then connect to the private instance. In this case, connect to NAT2 and then to Private1. In order to connect to any instance, a SSH key is required so we will need to bring our SSH key with us by copying it and creating a key file on NAT2 so we can SSH into Private1.

1. If you haven't done so, download your EC2 SSH key from the Qwiklab Lab 4 start page. Download the PEM file if you use Mac or Linux. If you use the Windows PuTTY SSH client, download the PPK and PEM file. Here are instructions to connect to an instance using PuTTY (follow the instructions in the "**Starting a PuTTY Session**" section). Mac and Linux users can simply use the native ssh client found on their systems.



PuTTY users should download both the PEM and PPK file. The PPK file will be used to SSH from Windows to NAT2. The PEM file will be required to SSH from NAT2 to Private1.

2. View your SSH key and copy its entire contents, including the "BEGIN/END RSA PRIVATE KEY" headers.

3. SSH into NAT2 using the SSH key and NAT EIP information you recorded earlier. In this example, the EIP for NAT2 is **`54.208.63.207`**.

    a. `ssh -i /path/to/your/key.pem ec2-user@54.208.63.207` **[Instructions will be different for other SSH clients.]**

        i. If you get the error "`Permissions 0644 for '/path/you/your/key.pem' are too open`" then you may (on Mac/Linux) need to issue the command `chmod 600 /path/you/your/key.pem`

4. Using an editor such as vi, paste the SSH key information from Step 1 into a new file and save it with .pem extension (e.g key.pem). Change permissions on the key accordingly.

a. `vi key.pem`

b. `<paste and save> the key that you copied in the Step 2.`

c. `chmod 600 key.pem`

5. Log into the instance Private1 using this SSH key and the IP address. In this example, the IP address for Private1 is **`10.1.0.154.`**

   a. `ssh -i key.pem ec2-user@10.1.0.154`

You are now logged in to the instance on private subnet 1. With the NAT in place on public subnet 1, we will be able to reach the public Internet. Let's test this using the `curl` utility.

6. `curl http://ipecho.net/plain`

   NOTE – as an interesting side note, we decided to have you curl a public site that will reflect your public IP. What do you expect this to be? Did you expect it to be this?

7. This demonstrates that an instance with a non routable address is able to access the Internet through the NAT instance. Stay logged in to Private1. We'll need to use this in the next section.

Considerations for Availability

While the design of this implementation offers outbound connectivity for private subnet instances, certain availability concerns exist. Can you name a few and the effects they might have on outbound traffic from the private subnets? For example:

- Single point of failure: NAT instance fails, so instances are not able to send outbound traffic from private subnet.
- Inappropriately sized: NAT instance size is underpowered and cannot handle increased amounts of traffic, ability to send outbound traffic is non-deterministic, requires instance type upgrade

In the event that the NAT instance fails, traffic originating from the private subnet will not reach the public Internet. This single point of failure creates an availability issue within the VPC. Let's verify this behavior.

1. From the AWS EC2 console, stop the NAT instance in public subnet 1, NAT1. **Quickly switch back to SSH session for Private1.**

   a. NOTE – in the previous section, we established an SSH connection to NAT2 and then to Private1. We did this to support this exercise. Had we gone through NAT1 to Private1, we would have lost all of our SSH connections when we stopped the NAT1 in this exercise.

2. From the Private1, test outbound connectivity once again using the `curl` utility

   a. `curl http://ipecho.net/plain`

Do you get a response? Did it take a long time? Was the response different from the previous time?

3. Stay logged into the current Private1. We will need this connection for the next session.

4. Note that if you did not do the last 2 steps quickly enough, you will have received a response from the curl command rapidly. ☺ You'll see why in the next section.

Enabling Multi-AZ NAT HA

The downside of our original solution is that the NAT design appliance is not fault-tolerant without some other considerations. When a NAT instance fails, all outbound traffic to that NAT stops. For some environments, this may be tolerable. Operating system updates from instances in private subnets may not be as critical as outbound calls from application servers in the private subnets to critical external systems such as Salesforce, PayPal, etc. Regardless of the criticality, the current design needs to be fixed.

One way the limitations of the current design can be solved is to introduce a "watchdog" monitoring element into the environment that can monitor the up/down status of a NAT, and re-route outbound traffic by updating route tables in the event that a NAT instance goes down. To do this, we have a script running in `cron` on each NAT instance. It runs every *X*-number of minutes, pings the other NAT instance to see if it is available and if it doesn't get a response over the span of *Y*-number of pings, considers it dead. The monitoring script will then modify the route table of the private subnet that the dead NAT serviced and update the outbound traffic route (0.0.0.0/0) to map to the NAT instance that is still alive. Doing so will allow the private subnet that was impacted to continue sending outbound traffic through the NAT it was redirected to. Behind the scenes, the monitoring script continues to ping the dead NAT and if it determines it is back up, will re-map the routes for the impacted private subnet back to the original NAT.

The curl operation that failed in the previous section should now execute in this section.

1. Return to the SSH session for Private1 and execute the following curl operation

   a. `curl http://ipecho.net/plain`

   Did you get a response? If so, what is the IP address? Did you expect this?

2. Using the AWS console, which NAT instance is private subnet 1 referencing? You may want to refer to the Outputs section for the CloudFormation stack.

Let's examine the monitoring script.

3. Exit the SSH connection on the instance in Private1. This will bring you back to the shell in NAT2.

4. The monitoring script is located at `/root/nat_monitor.sh`. You can view it by

   a. `sudo more /root/nat_monitor.sh`

5. The script outlines the steps required to for one NAT to monitor the other and in case the other fails and update the target route table appropriately. The script demonstrates how the programmable infrastructure of AWS can simplify complicated tasks.

6. `cron` will launch the script at boot. This was configured in the CloudFormation template:

   a. `echo '@reboot /root/nat_monitor.sh >> /tmp/nat_monitor.log' | crontab`

   b. `./nat_monitor.sh >> /tmp/nat_monitor.log &`

7. Viewing the log (`cat /tmp/nat_monitor.log`) shows the detection of the failed NAT and the route table takeover events.

Summary

Instances in private subnets can reach the Internet two ways: through a VPN tunnel and out to the Internet via the customer's network or directly to the Internet from VPC. The latter option requires a NAT service, therefore you must consider how NATs – and the overall availability of those NATs – plays into your design and implementation. In this lab, we used dual NATs that monitor each other and take over in case one fails.

This lab demonstrates *one method* of implementing a highly available NAT solution. Other solutions include Squid proxies and commercial software appliances with advanced features. Each has tradeoffs in terms of cost, features, configurability, etc., so you should evaluate these options to determine the best fit for your infrastructure.

References

- High Availability for Amazon VPC NAT Instances: An Example
- Using Squid Proxy Instances for Web Service Access in Amazon VPC: An Example
- From One to Many: Evolving VPC Design
- Slides: NAT discussion begins on slide #26.
- Video