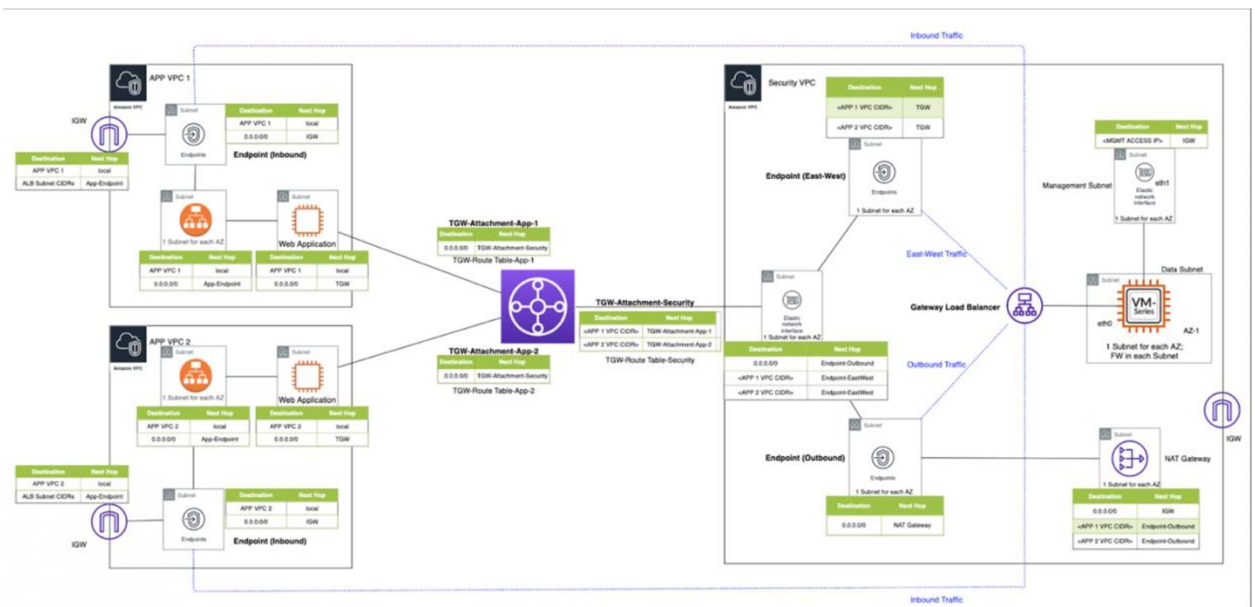
Introduction

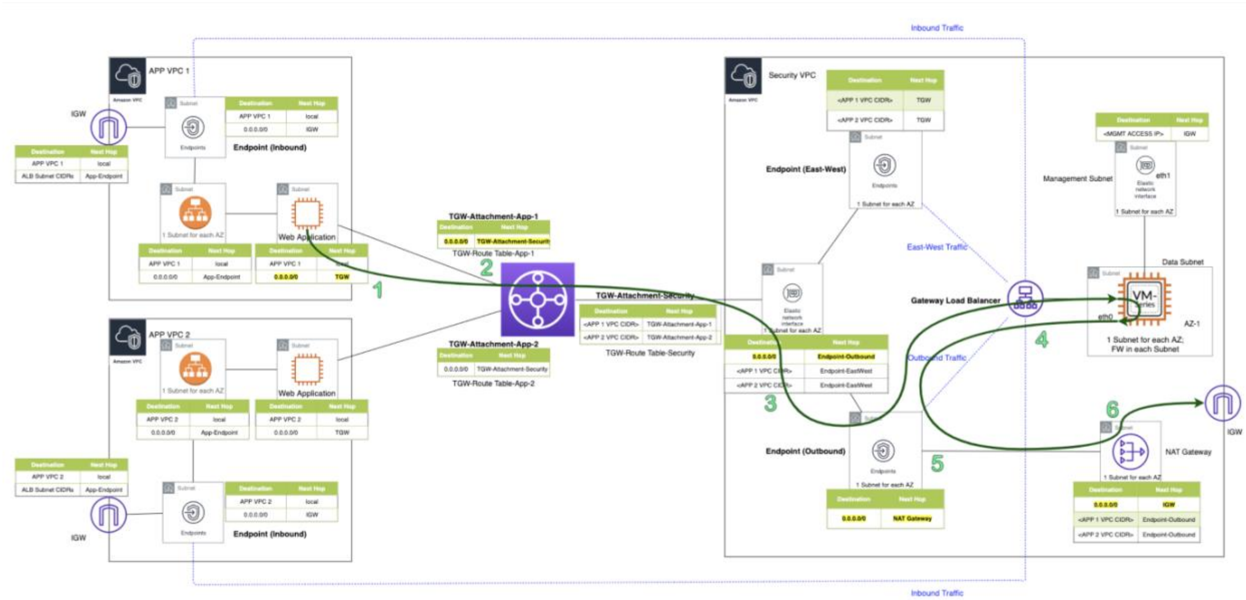
With the introduction of the Gateway Load Balancer (GWLB) in mid-November 2020, AWS provided its customers with any port, load-balancing router . Prior to that, Azure and GCP were the only public clouds that had such a construct. Customers use these to provide a security layer that is scalable, resilient, and adaptable. In the AWS implementation, endpoints are an integral part of the solution but are not a new concept in AWS. They connect elastic network interfaces (ENIs) to targets (e.g. GWLB) via "worm holes" in the fabric and and have been used with network load balancers (NLBs) for some time. These worm holes in the fabric bypass the usual routing constructs and can perforce result in some difficulty when troubleshooting. In this blog post, we will trace the flow of a request originating from a client in one VPC (network 10.101.0.0/16) going out to the internet. The infrastructure was deployed using the following TerraForm template:

https://github.com/wwce/terraform/tree/master/aws/GWLB-Demo

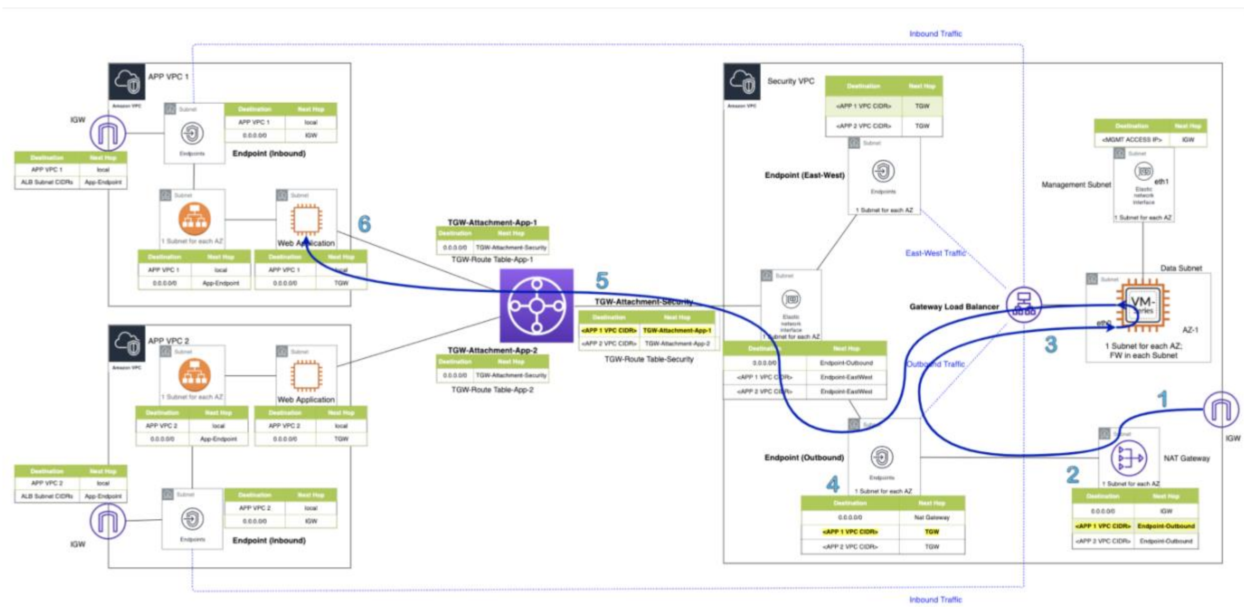
and follows current best practices regarding architecture:



This architecture also supports east-west and outbound traffic flows, which are treated separately in other blog posts. Today, we will focus on the following request flow:



And the corresponding response flow:

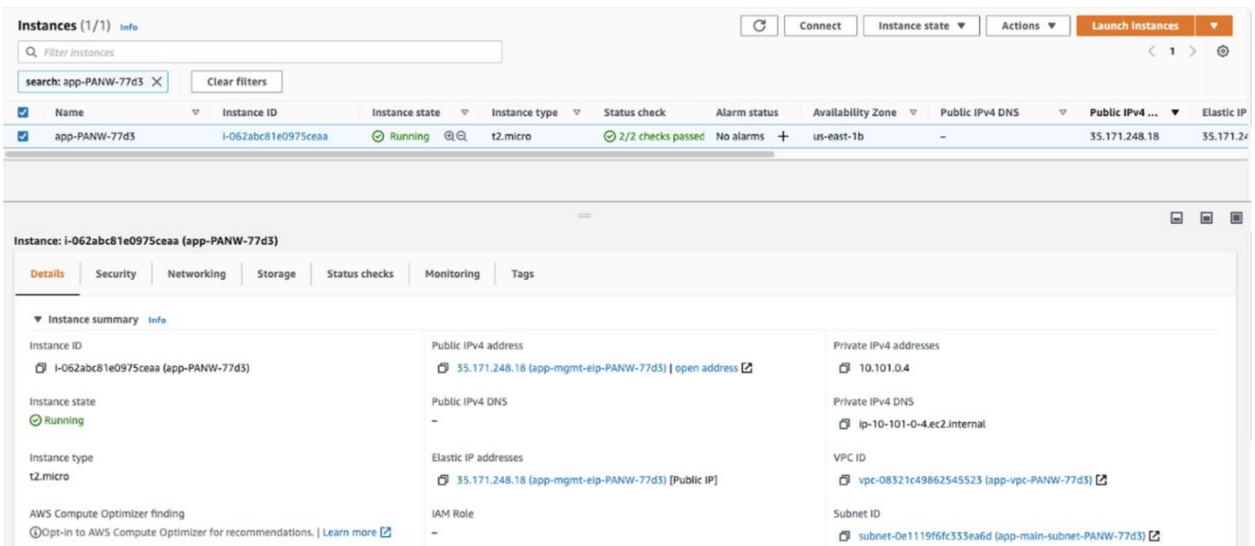


Note that AWS assigns unique resource identifiers to each resource in the environment. Examples include tgw-attach-0b86ac38ab82dfff9 or subnet-0e1119f6fc333ea6d. Every resource created is assigned one of these unique identifiers. This means that although the template creates the environment using identical resources, the individual resource identifiers will be different.

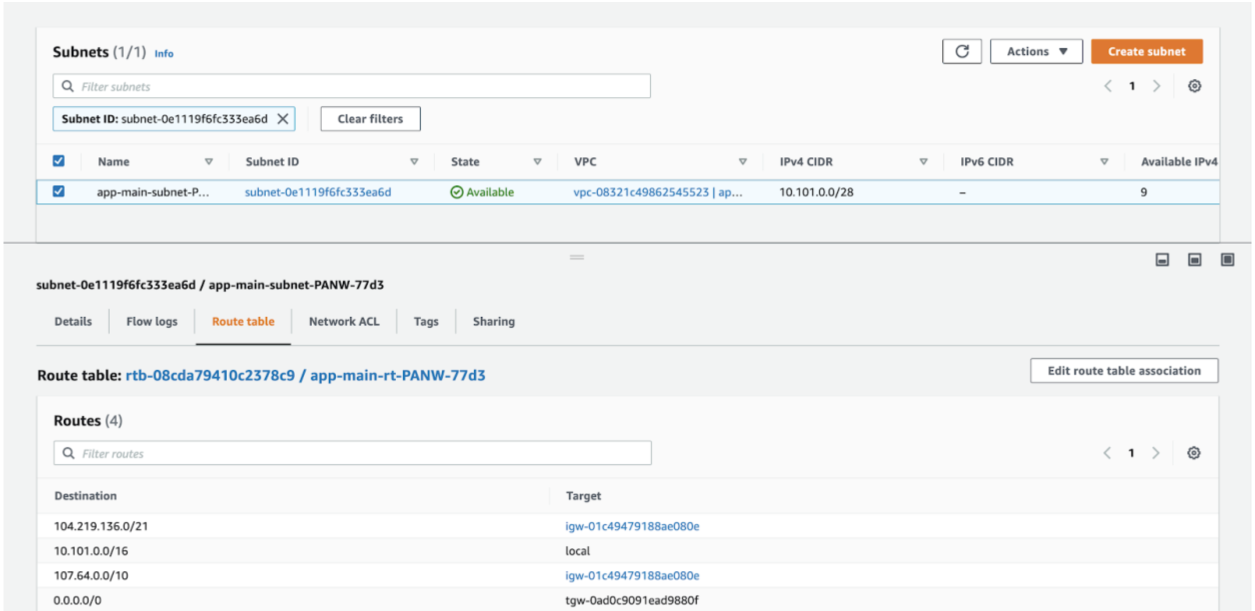
N.B. - Routes to the 104.219.136.0/21 and 107.64.0.0/10 subnets pointing to the internet gateway (IGW) in the APP VPCs are the author's primary/secondary ISP subnets and were added post-deployment to facilitate direct access to the hosts in the VPCs for troubleshooting. They do not exist in the publicly-available templates and can be ignored.

Request Step 1 - Can We Talk?

The process begins when a user/process on a host (IP 10.101.0.4) in APP VPC 1 needs to connect to the internet. Looking at the EC2 instance, we can see the IP address as well as the subnet membership:



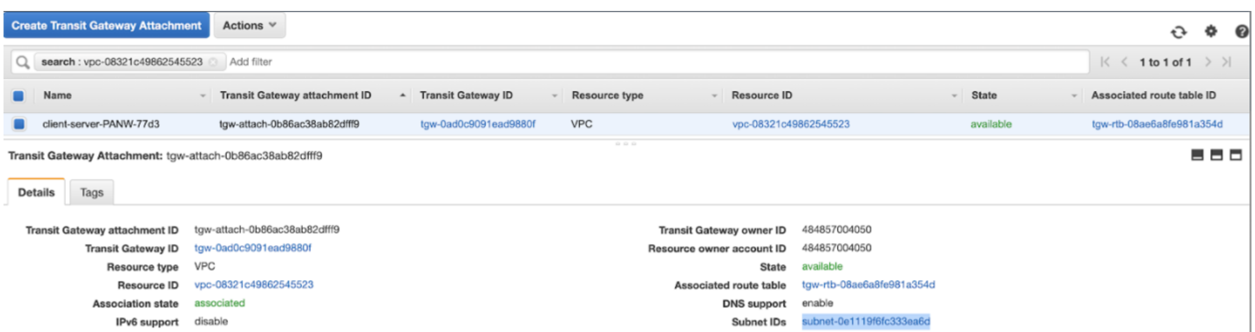
The route table associated with the subnet shows that the default route for the subnet points to the transit gateway (TGW) as the next hop.



The requester does an internal route lookup and puts the packet out on the wire (local subnet) which has a default route via the TGW.

Request Step 2 - Transit Gateway (TGW)

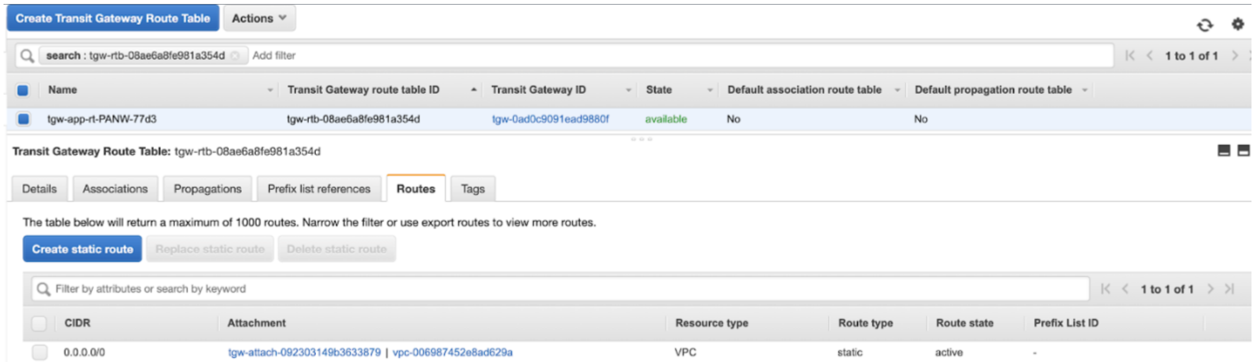
The TGW is connected to the VPC via a Transit Gateway Attachment. To see this association, we navigate to the Transit Gateway Attachment list and filter on the VPC hosting the requester:



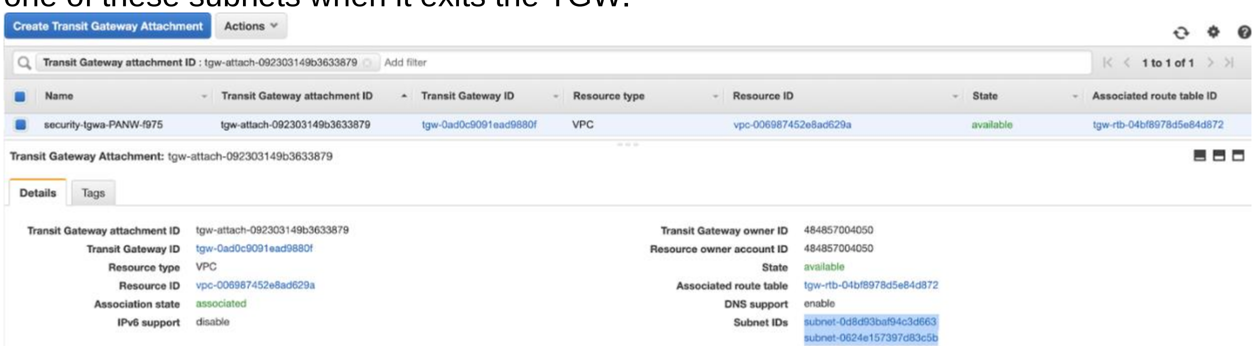
Note that when creating a TGW Attachment, a subnet must be specified and traffic can only be routed to a TGW Attachment in the same availability zone (AZ) as the source. In this case, the TGW Attachment and the host exist on the same subnet (and hence same AZ).

Routing within the TGW is handled via route tables associated with the TGW attachment. In the above picture, we can see that the route table associated with the

TGW Attachment is tgw-rtb-08ae6a8fe981a354d. Clicking on the link to the route table and inspecting the routes, we can see that the default route points to yet another attachment:



Following the rabbit a little further down the hole, we find that the attachment is associated with two different subnets. Traffic from the requester gets dropped off into one of these subnets when it exits the TGW.



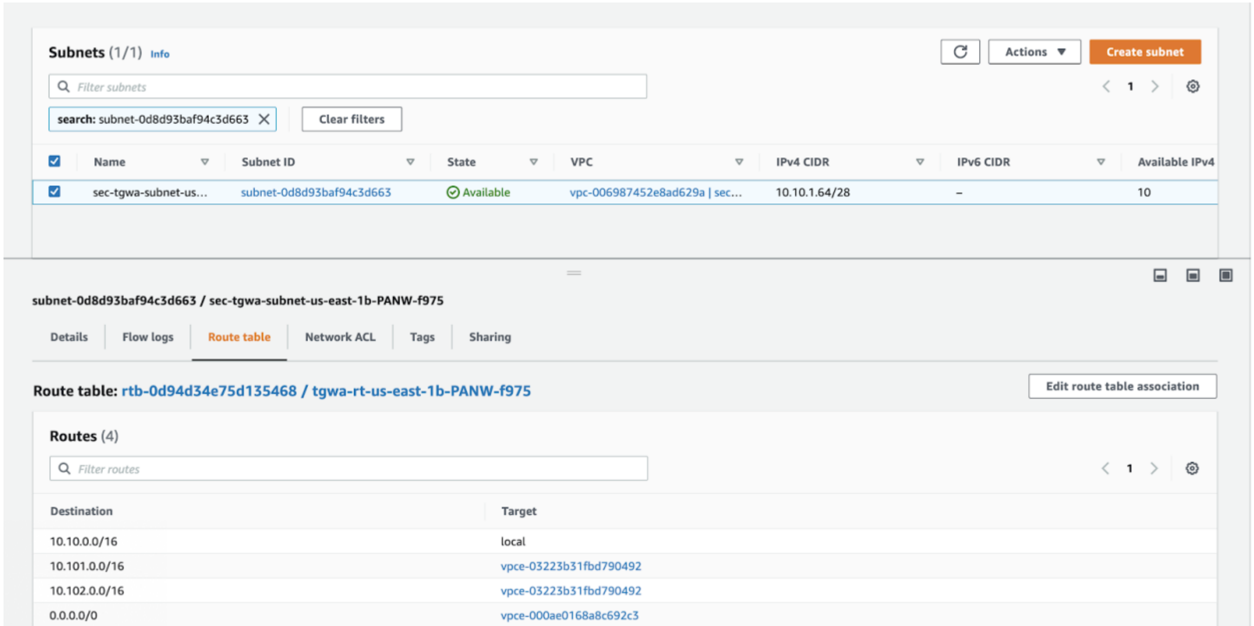
N.B. - The TGW has the ability to load balance across as well as ensure traffic symmetry. More information on traffic symmetry can be found here:

https://docs.aws.amazon.com/vpc/latest/tgw/transit-gateway-appliance-scenario.html

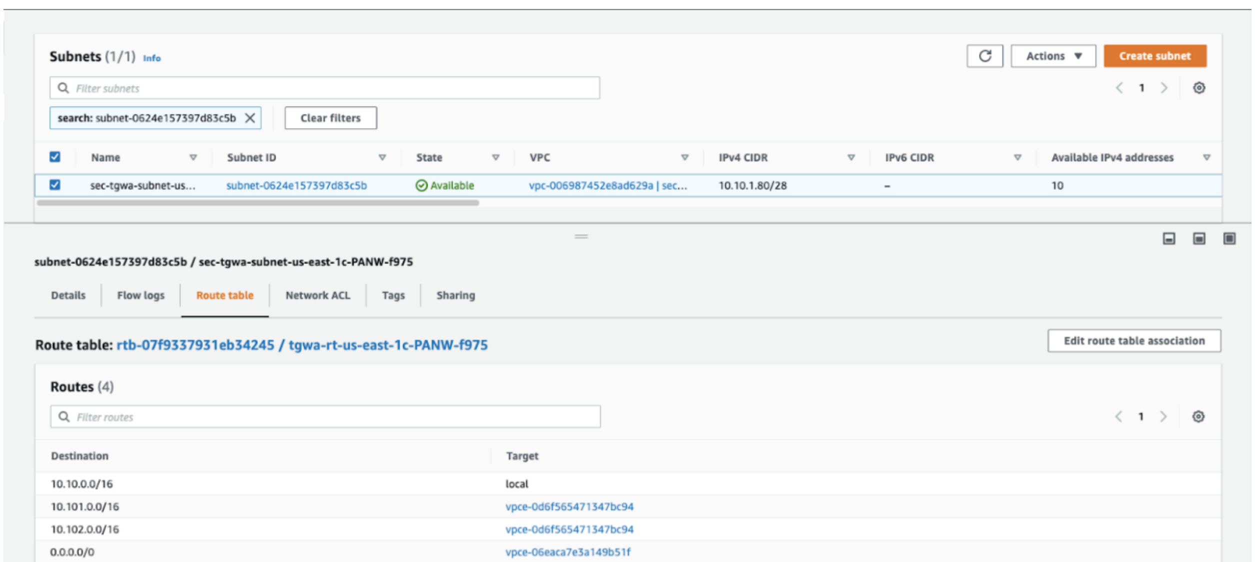
Request Step 3 - The GWLB Endpoint

Recall that endpoints are ENIs that provide direct access to services within the VPC. ENIs are AZ-specific constructs and are instantiated in every AZ where service access is required.

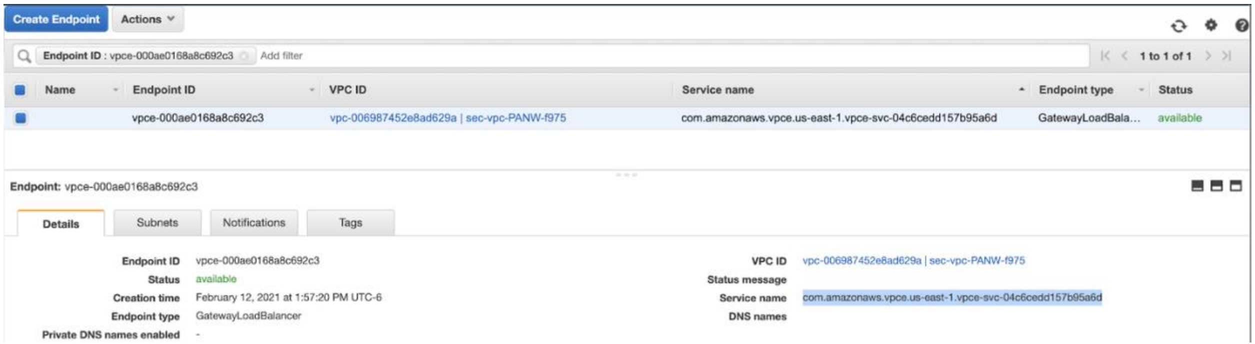
If we look at the route table of one of the subnets, we can see that the traffic is directed to a GWLB endpoint:



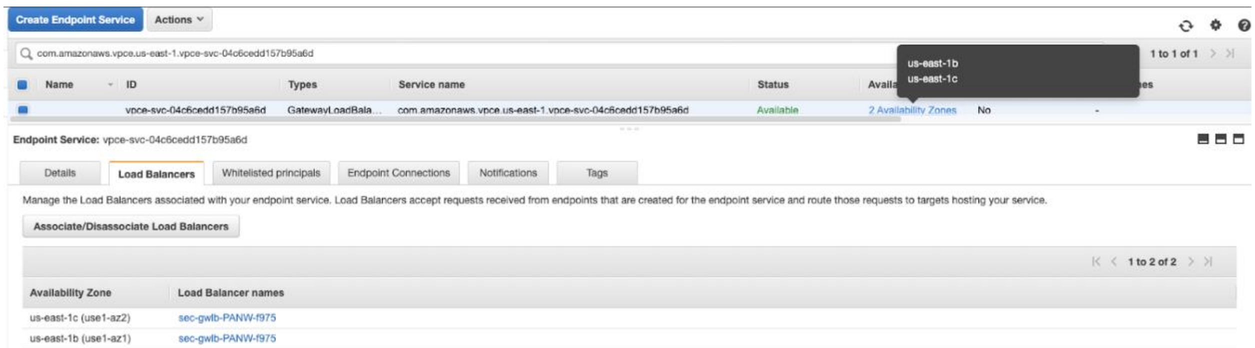
The route table associated with the other subnet looks similar (note that the endpoint ID is different):



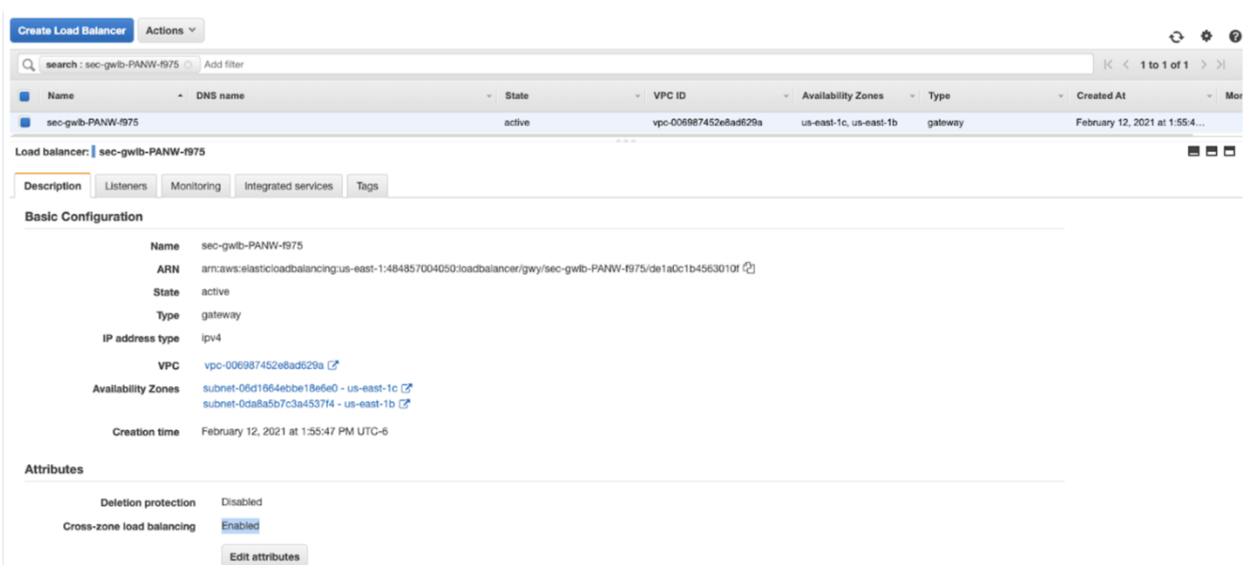
The Endpoint is connected to the GWLB via an Endpoint Service. In this case, the traffic is routed out the Endpoint associated with the default gateway (vpce-000ae0168a8c692c3). To see more information about this connection, click on the target. The subsequent page shows additional information about the Endpoint, including the associated Endpoint Service:



If we then look at Endpoint Services, we can see that this service is associated with a multi-AZ load balancer (also note that the Endpoint Service is associated with multiple AZs):



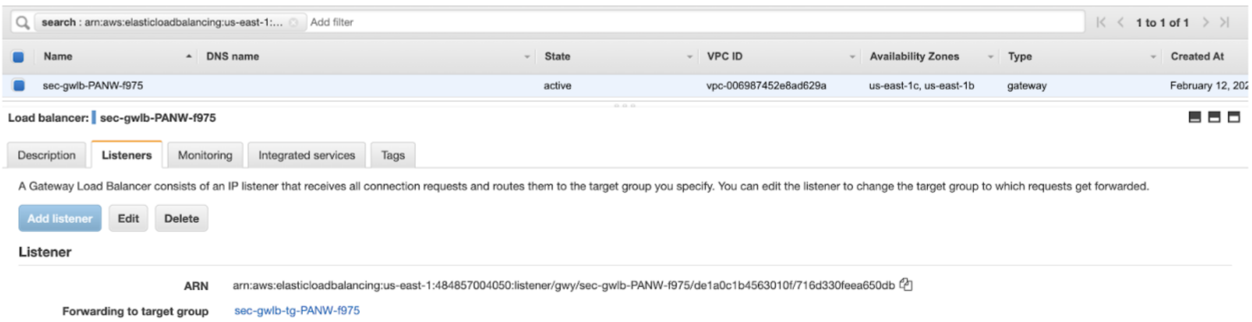
Clicking on the loadbalancer, we see more detailed information:



Pro Tip: If it has not already been done, "Cross-zone load balancing" should be enabled in the attributes. This ensures that the GWLB can use any backend pool member in any availability zone and facilitates resiliency.

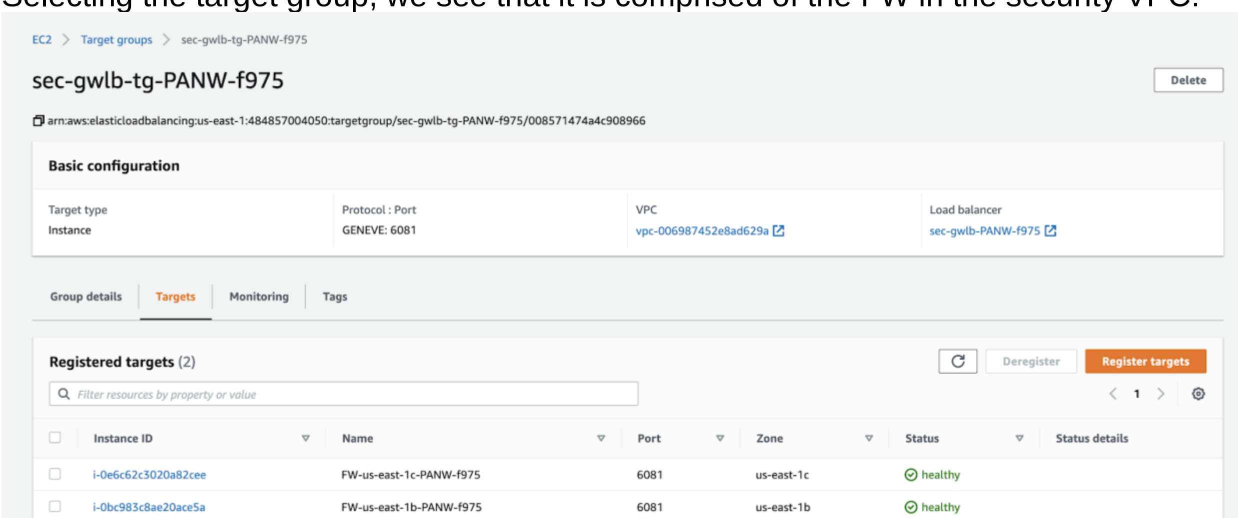
Request Step 4 - The Firewalls

The GWLB uses Generic Network Virtualization Encapsulation (GENEVE) to create an overlay network between the load balancer and the firewalls. At present, this overlay network is not connected to the firewalls virtual router, which improves packet handling efficiency but requires that all traffic ingress/egress the FW via the GENEVE tunnel. Under the hood, the GWLB is a souped-up NLB and the configuration is very similar. Once the traffic reaches the GWLB, it is distributed amongst the available backend pool members. Looking at the listeners for the GWLB, we see one of the first differences between the GWLB and a standard NLB:



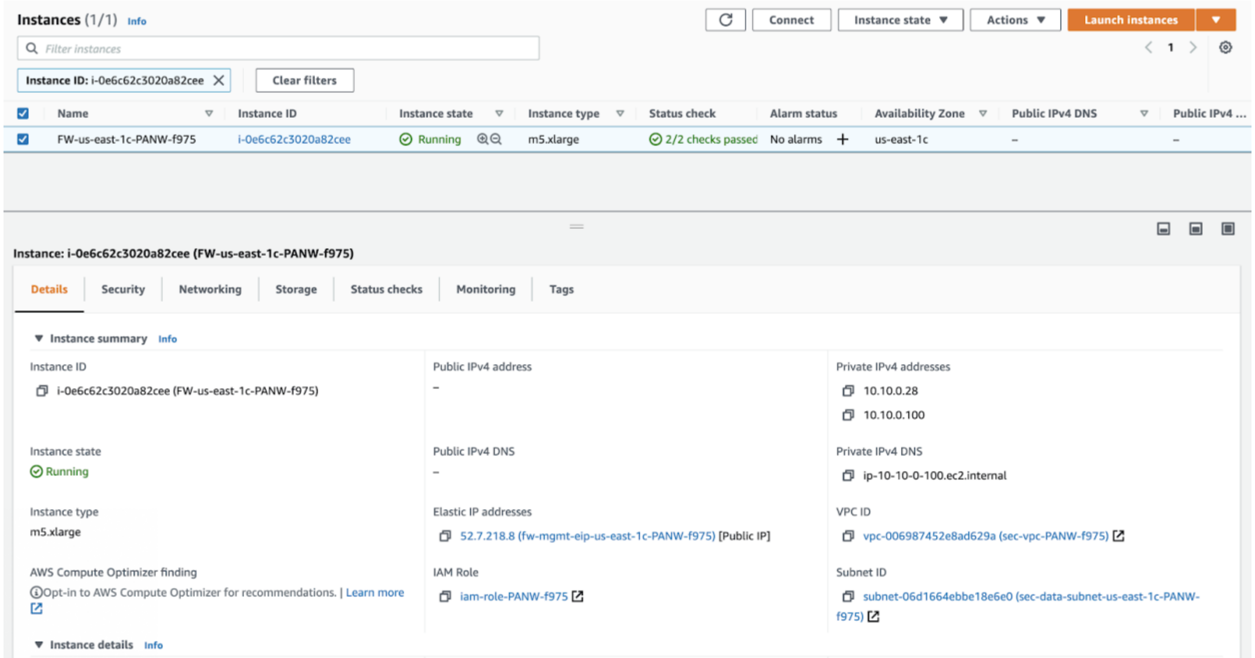
The GWLB is an any port load balancer and consequently no port(s)are specified/required. All TCP/UDP traffic is load balanced to the associated target group.

Selecting the target group, we see that it is comprised of the FW in the security VPC:



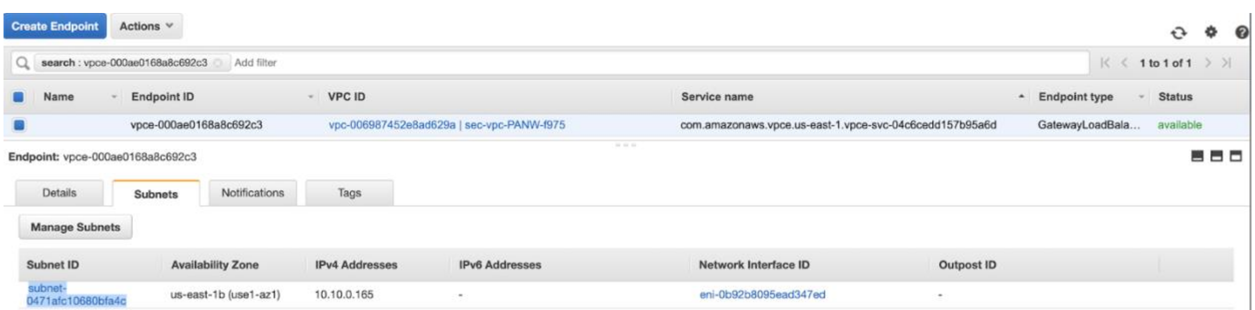
The FW are targeted by instance ID, which ensures source IP preservation but requires that the management and first data plane interface be swapped.

Selecting one of the targets, we can see the firewall details:

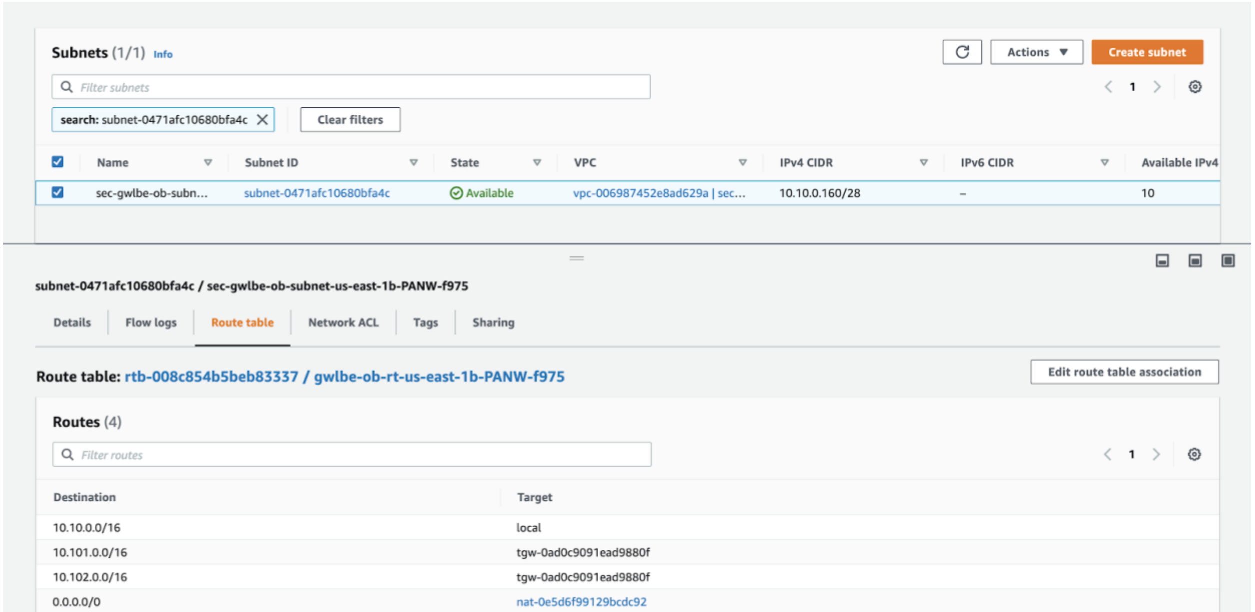


Request Step 5 - Return to the GWLB Endpoint

The permitted request is returned to the GWLB via the GENEVE tunnel and then back to the endpoint. Recall that the ID of the endpoint in step 3 is vpce-000ae0168a8c692c3. If we take a closer look at that endpoint, we can determine the subnet that it resides in:

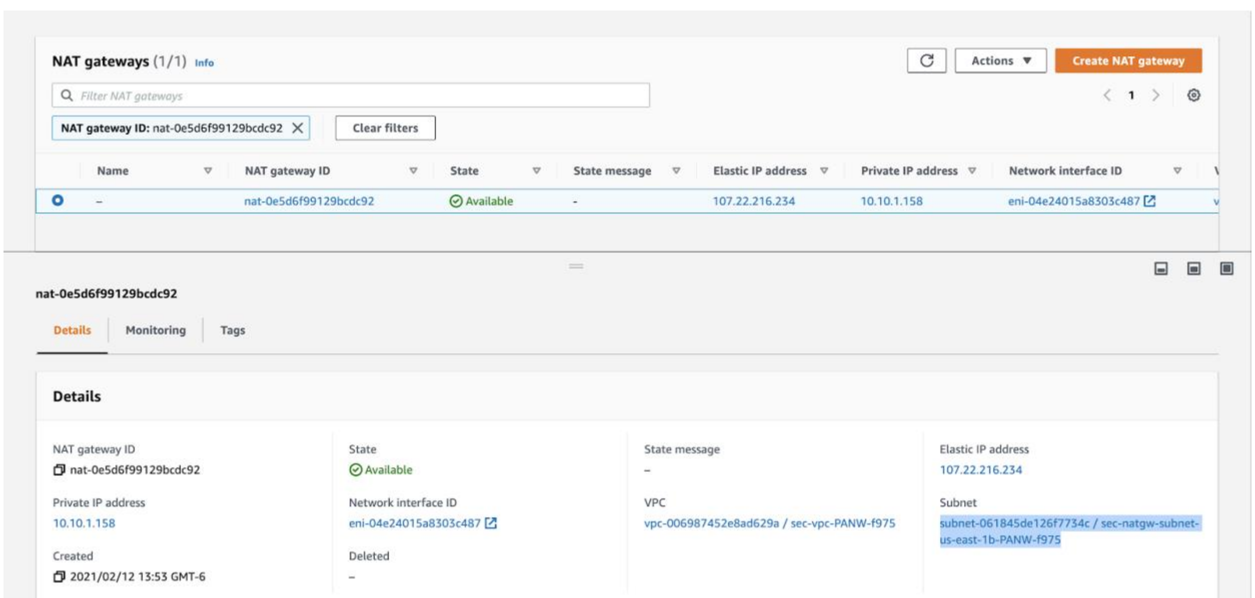


The subnet default route points to the NAT GW as the next hop:

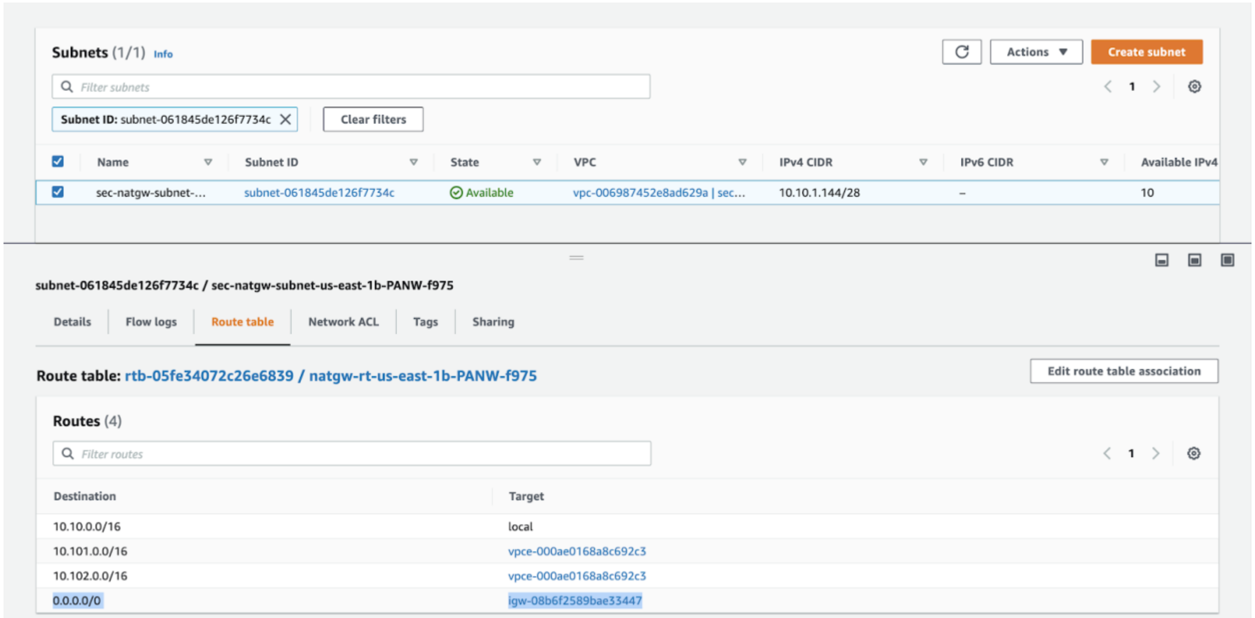


Request Step 6 - Into the Great Wide Open

Clicking on the NAT GW, we can see the subnet it is associated with as well as the associated Elastic IP (EIP):

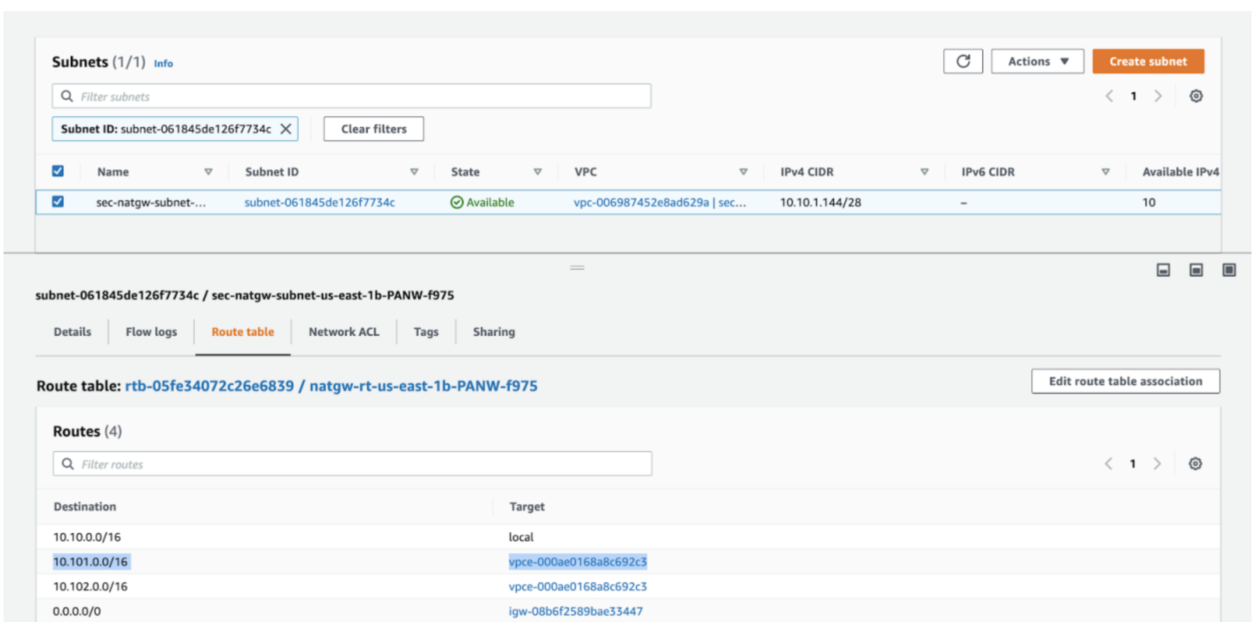


Looking at the routes associated with the subnet, we can see that the traffic is routed out via the IGW



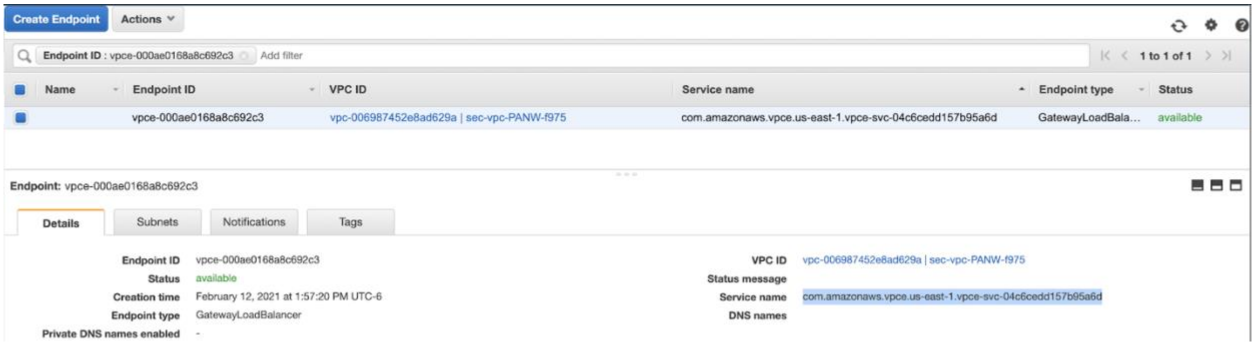
Response Step 1 - The Destination Deigns to Respond

Recall that the requesting host resides on the 10.101.0.0/16 subnet. When the response from the server returns to the IGW, it is sent to the NAT GW, which performs a reverse translation and puts the packet out on the wire where it is handled by the local subnet routing table. Looking at those routes, we see that the route to the destination subnet is via an Endpoint:

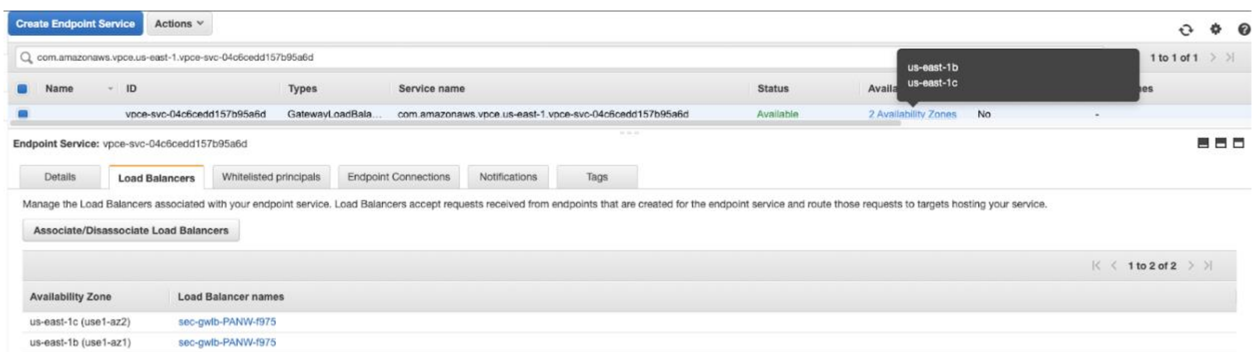


Response Step 2 - The GWLB Endpoint

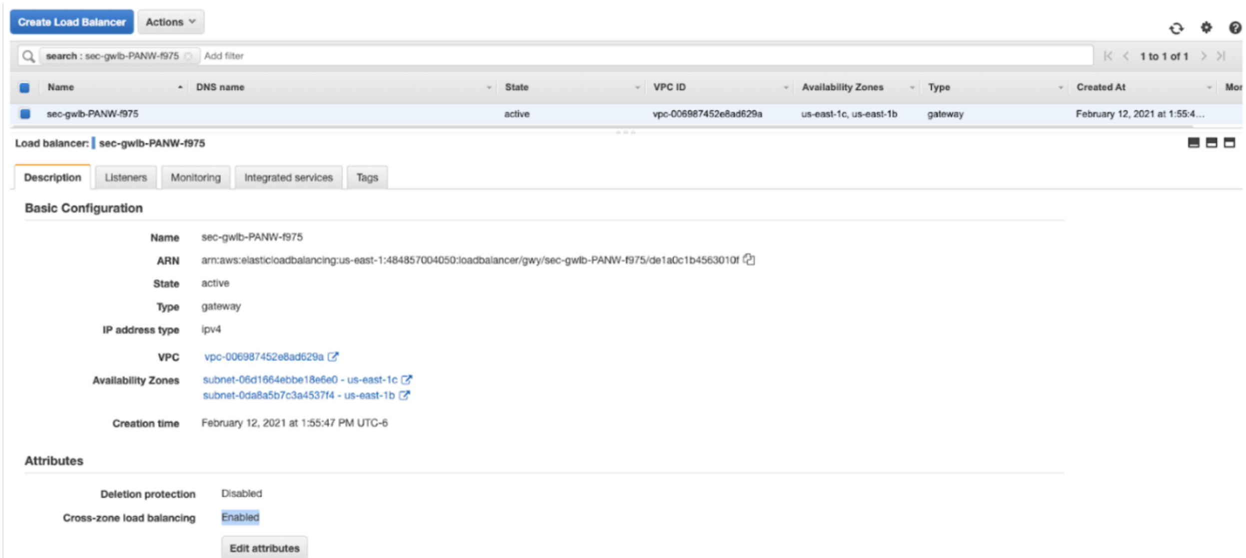
The Endpoint is connected to the GWLB via an Endpoint Service. In this case, the traffic is routed out the Endpoint associated with the 10.101.0.0/16 subnet (vpce-000ae0168a8c692c3). To see more information about this connection, click on the target. The subsequent page shows additional information about the Endpoint, including the associated Endpoint Service:



If we then look at Endpoint Services, we can see that this service is associated with a multi-AZ load balancer (also note that the Endpoint Service is associated with multiple AZs):



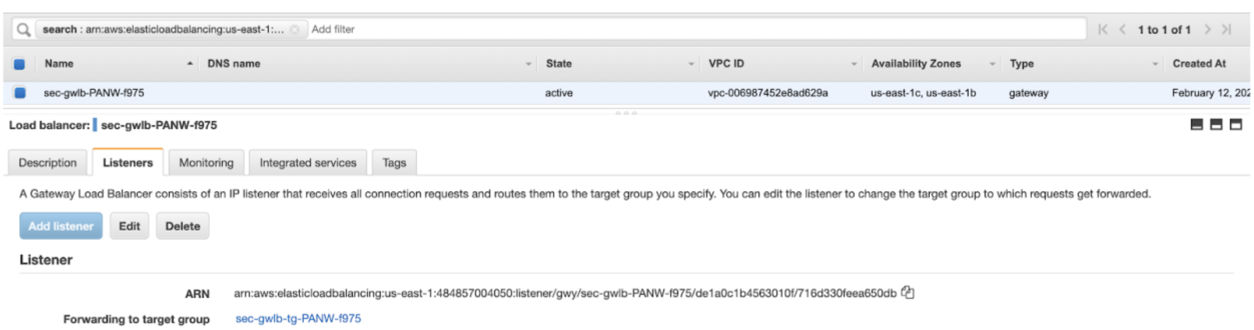
Clicking on the loadbalancer, we can see more detailed information:



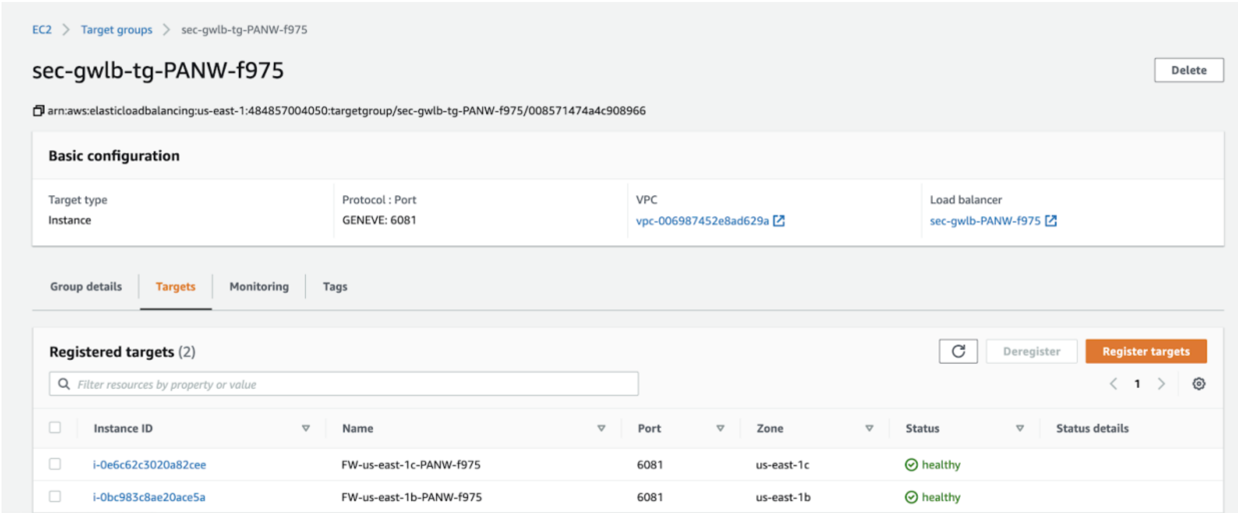
Pro Tip: If it has not already been done, "Cross-zone load balancing" should be enabled in the attributes. This ensures that the GWLB can use any backend pool member in any availability zone and facilitates resiliency.

Response Step 3 - The Firewalls

As mentioned earlier, there is no port associated with the listener on the GWLB. All TCP/UDP traffic is load balanced to the associated target group:

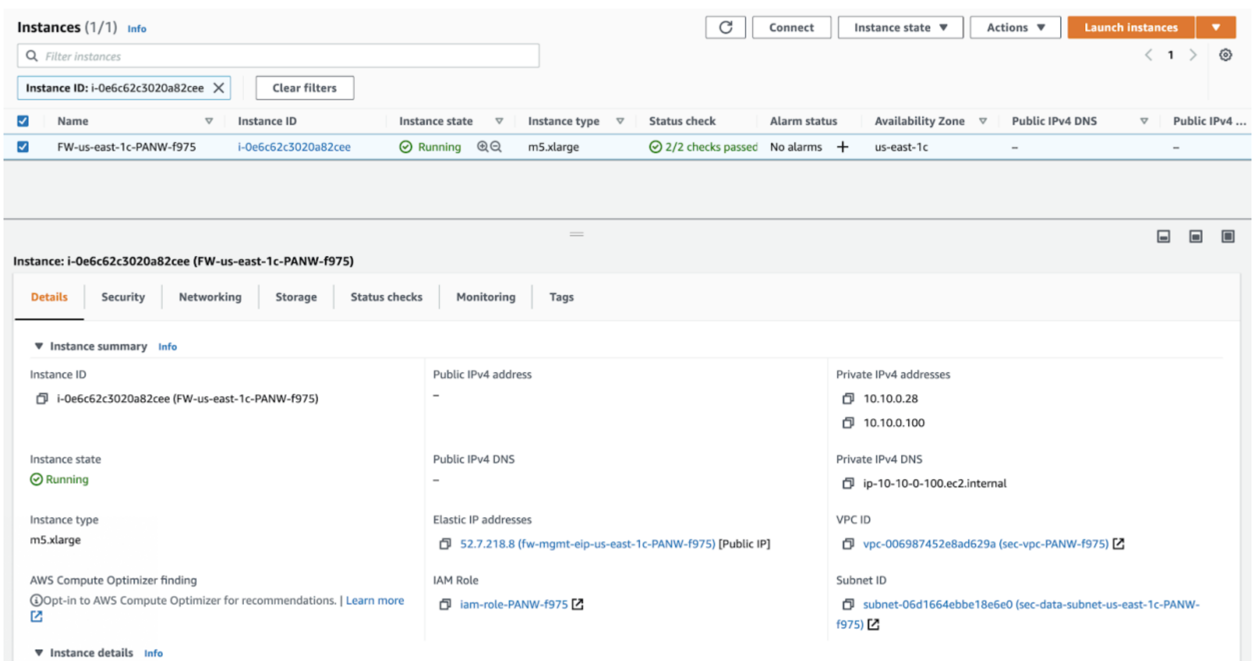


Selecting the target group, we see that it is comprised of the FW in the security VPC:



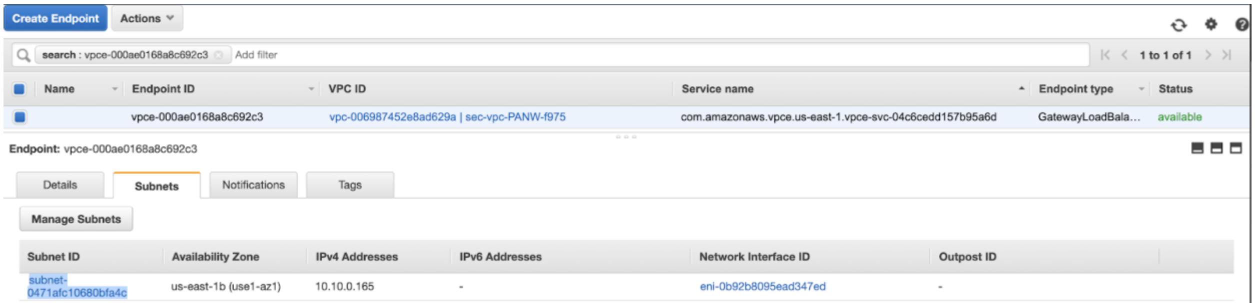
The FW are targeted by instance ID, which ensures source IP preservation but requires that the management and first data plane interface be swapped.

Selecting one of the targets, we can see the firewall details:

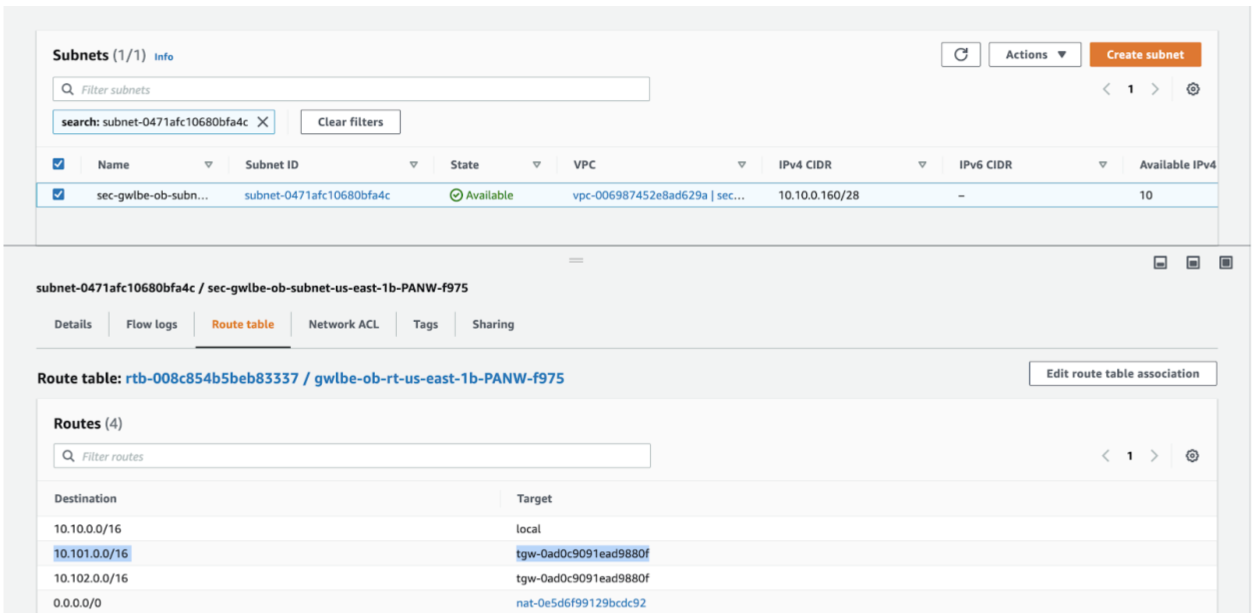


Response Step 4 - Return to the GWLB Endpoint

The permitted request is returned to the GWLB via the GENEVE tunnel and then back to the endpoint. Recall that the ID of the Endpoint is vpce-000ae0168a8c692c3. If we take a closer look at that endpoint, we can determine the subnet that it resides in:

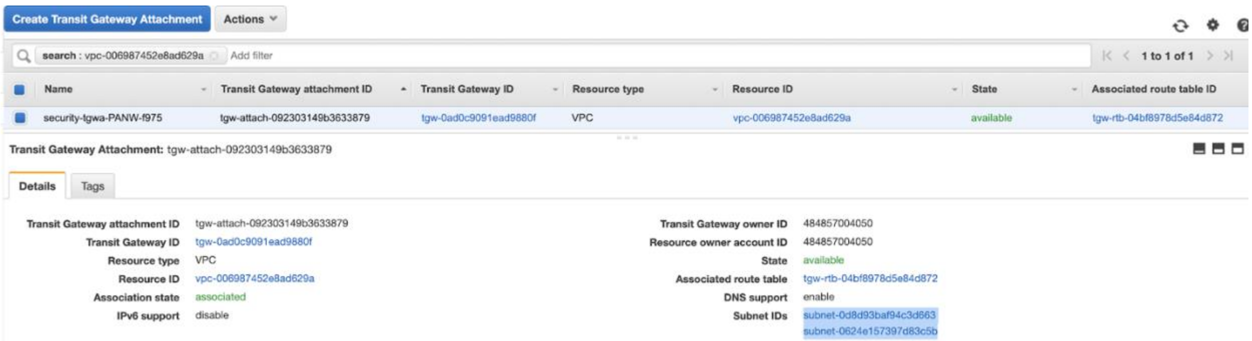


The subnet route table points has the next hop to the destination as the TGW:



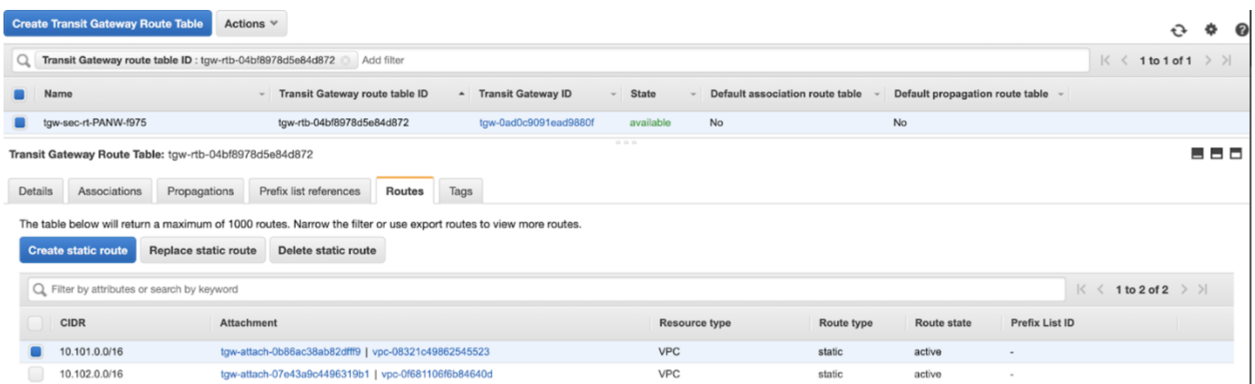
Response Step 5 - Return to the TGW

Recall that the TGW is connected to the VPC at the subnet level via a Transit Gateway Attachment. To see this association, we navigate to the Transit Gateway Attachment list in the VPC section of the GUI and filter on the security VPC (vpc-006987452e8ad629a in this example):

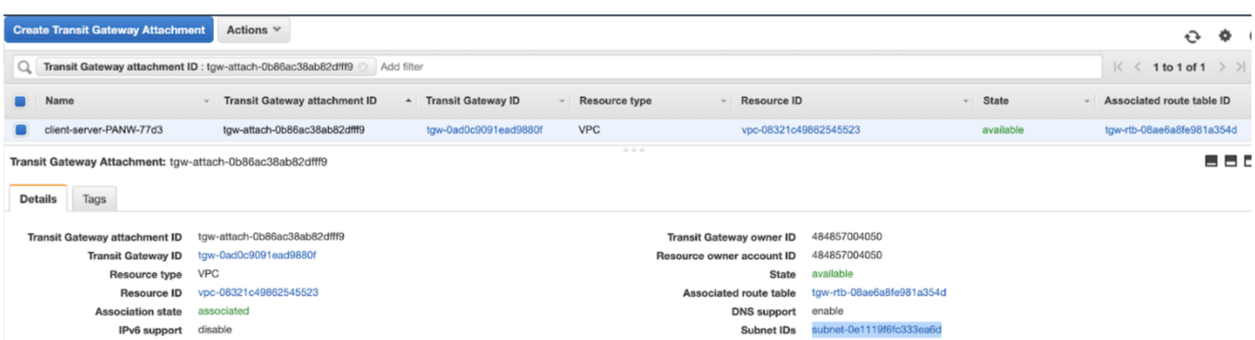


As long as the Endpoint and TGW Attachment are in the same AZ, subnet routing will ensure that the packet is delivered to the TGW.

Within the TGW is handled via route tables associated with the TGW attachment. In the above picture, we can see that the route table associated with the TGW attachment is tgw-rtb-04bf8978d5e84d872. Clicking on the link to the route table and inspecting the routes, we can see that the route to the requester subnet (10.101.0.0/16) points to another attachment (tgw-attach-0b86ac38ab82dfff9):

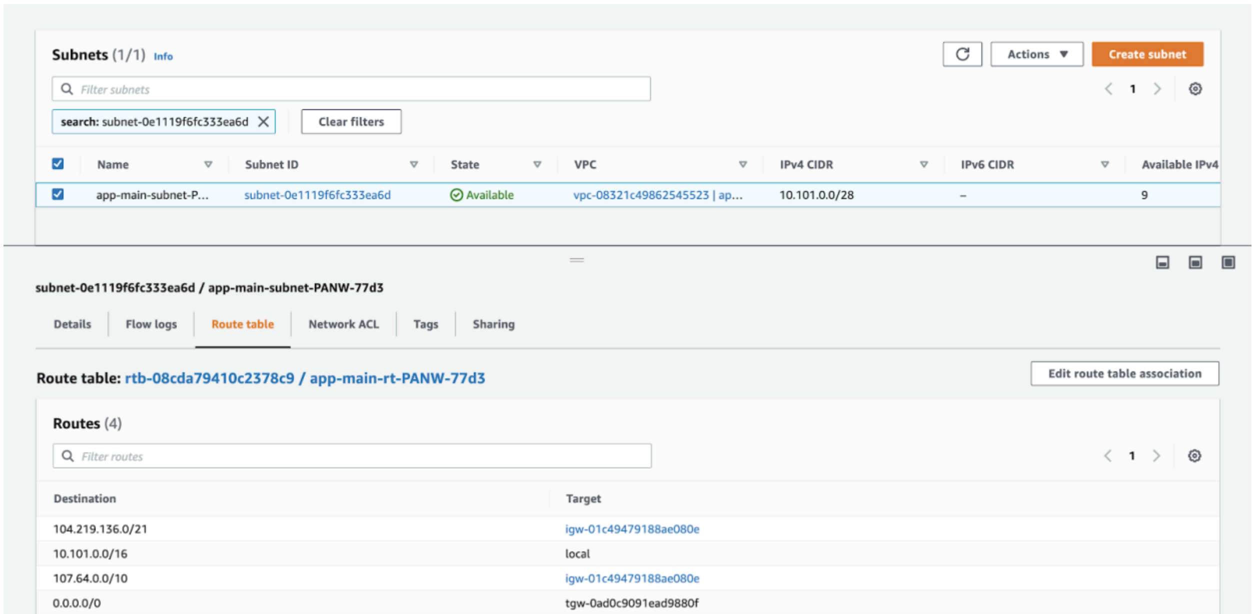


Following this rabbit a little further down the hole, we find that the attachment is associated with a single subnet. Traffic exiting the TGW gets dropped off into this subnet.

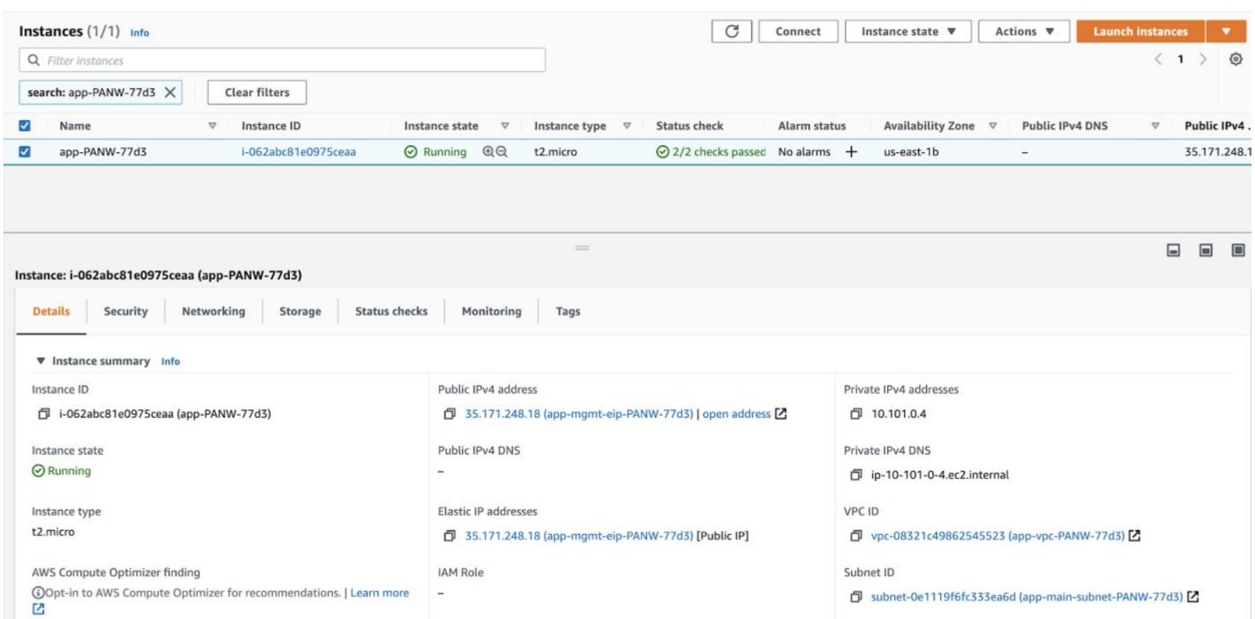


Response Step 6 - At Last

Inspection of the subnet route table reveals that any traffic destined for the VPC network is delivered directly to the target:



Inspection of the target host reveals that it resides on the destination network. This tells us that the traffic exiting the TGW is delivered directly to the target.



Et violà:

ubuntu@admin-appliance: ~ (com.docker.cli) ⌘1 | ubuntu@ip-10-101-0-4: ~ (ssh) ⌘2

```
ubuntu@ip-10-101-0-4:~$ curl -I https://www.keycdn.com
HTTP/2 200
server: keycdn-engine
date: Wed, 17 Feb 2021 14:22:49 GMT
content-type: text/html
last-modified: Tue, 16 Feb 2021 17:40:33 GMT
vary: Accept-Encoding
etag: W/"602c0391-10111"
expires: Wed, 24 Feb 2021 14:22:49 GMT
cache-control: max-age=604800
strict-transport-security: max-age=31536000; includeSubdomains; preload
content-security-policy: default-src 'self' 'unsafe-inline' 'unsafe-eval' https: data:
x-frame-options: SAMEORIGIN
x-xss-protection: 1; mode=block
x-content-type-options: nosniff
referrer-policy: no-referrer-when-downgrade
x-cache: HIT
x-edge-location: usda
access-control-allow-origin: *

ubuntu@ip-10-101-0-4:~$
```

Looking at the FW logs, we can see both original source and original destination:

(addr.src in 10.101.0.4) and (addr.dst in 68.70.205.0/24)

	GENERATE TIME	TYPE	FROM ZONE	TO ZONE	SOURCE	DESTINATI...	SOURCE USER	NAT APPLIED	NAT SOURCE IP	NAT DEST IP	TO PORT	APPLICATI...	ACTION	RULE	SESSION END REASON	BYTES
	02/17 14:23:03	end	Trust	Trust	10.101.0.4	68.70.205.2		no			443	ssl	allow	Allowed-traffic	tcp-fin	7.4k
	02/17 14:22:58	end	Trust	Trust	10.101.0.4	68.70.205.3		no			443	ssl	allow	Allowed-traffic	tcp-fin	7.4k
	02/17 14:22:48	start	Trust	Trust	10.101.0.4	68.70.205.2		no			443	ssl	allow	Allowed-traffic	n/a	797
	02/17 14:22:43	start	Trust	Trust	10.101.0.4	68.70.205.3		no			443	ssl	allow	Allowed-traffic	n/a	797
	02/17 14:21:04	end	Trust	Trust	10.101.0.4	68.70.205.3		no			443	ssl	allow	Allowed-traffic	tcp-fin	7.5k
	02/17 14:21:03	end	Trust	Trust	10.101.0.4	68.70.205.3		no			443	ssl	allow	Allowed-traffic	tcp-fin	7.3k
	02/17 14:20:49	start	Trust	Trust	10.101.0.4	68.70.205.3		no			443	ssl	allow	Allowed-traffic	n/a	797
	02/17 14:20:48	start	Trust	Trust	10.101.0.4	68.70.205.3		no			443	ssl	allow	Allowed-traffic	n/a	797