High Availability and Scalability: ELB & ASG

Scalability

There are two type of scalability, vertical scalability and horizontal scalability.

Vertical scalability/scalable

It means that you are increasing the size of the instance on a more persistent level to meet workload growth.

For example, if you have a call center and your instances are the people taking the calls. We have a junior operator who can take 5 calls per minute, if we want to vertical scale the operator that means we need to hire say a senior operator who can take 10 calls per minute.

Or if we have a chain of piazza place, if we want to vertical scale our business that would mean we need to build more piazza restaurant to adopt to the need of the customer. It is more permanent.

For AWS, it would mean switching from t2.micro to t2.large to meet workload demand.

Vertical scaling is common for distributed systems like database.

Horizontal scalability/elasticity

You increase the number of instances to meet peak of workload demand.

Going back to the example of call center instead of hiring senior operator, you would just increase the number of junior operator to handle the calls.

Horizontal scaling implies distributed systems.

High availability

You run your application or system in at least 2 data center. The goal of high availability is to be able to survive through a data center loss, that you can quickly recover after one of the instances goes down by using the backup instance.

High availability can be passive, or active. Exact replica of the application or a smaller replica of the application that you can scale up after disaster recovery.

High availability & scalability for EC2

You can increase the instance size to have more hardware capability for vertical scaling.

You can increase the number of the same instance for horizontal scaling, this can be done by auto scaling group automatically with load balancer.

Finally you run those instances for same application across multiple availability zones, in case one goes down, the other EC2 instances in other availability zone will be fine, this is for high availability.

ELB

Elastic load balancer. A load balancer is a server that distribute traffic to other servers called downstream servers (EC2 instances to actually handle the request) in order to avoid overloading one instances with too many requests.

You use load balancer to spread load across multiple downstream instances. It will expose one single point of access to your application.

Failures of downstream instances can be handle easily because there is built-in health checks for ELB. it also provide SSL certificates for your websites.

AWS's Elastic load balancer is a managed one so you don't have to worry about upgrading it, making it highly availability, it is all done by AWS. Using it will cost less and you have to do less work than setting up your own load balancer.

Health checks

A way for ELB to verify whether an EC2 instance is working properly. If the instance is not working properly then load balancer will not forward traffic to that dead EC2 instances.

Health check is done on a port and a route to the EC2 instances. EX: on port 5677, endpoint /health, if there is a response that comes back from that endpoint then it is good, otherwise, it is not healthy.

Types of load balancer

There are 4 kinds of managed load balancers.

Classic load balancer

This is the old generation of the load balancer, AWS do not recommend using this load balancer anymore.

Application load balancer (ALB)

Supports HTTP, HTTPS, WebSocket.

Target group: They can be EC2 instances, ECS tasks, Lambda functions, or IP addresses (must be private, your own on premise servers). They designate a set of resources that the ALB can route traffic to.

ALB can route to multiple target groups, it doesn't just have to be one target group, you can have multiple of them. Health checks are done on target group basis.

You can route the traffic to different target groups based on path, host name, or even query strings and headers.

ALB are great for micro services & container-based application.

When you create an ALB you get a fixed hostname. In addition, the server application (the target group instances) do not see the client's IP directly, they are inserted in **X-Forwraded-(For,Port,Proto)** headers. This is because the load balancer is doing the request, not the client. If the application server wishes to see the client's traffic then they need to inspect those extra headers.

When you create an ALB, you will have to define a security group to define what kind of traffics are allowed into the load balancer. Then ALB will then forward the traffic into the EC2 instance, now those EC2 instances in the target group should only be accepting traffics from ALB which means that it will be authorizing the security group the load balancer is part of.

In addition, you can define different routing rules based on hostname, query string, and paths to different target groups.

Network load balancer (NLB)

Supports TCP, TLS, UDP. So instead of only forwarding HTTP/HTTPS you are now allowed to forward TCP/UDP traffic, one layer down the network stack. You are dealing with even lower level network traffics.

Network load balancer is very high performance, handles millions of request per seconds, lower latency compared to ALB.

It only has one static IP per AZ and support assigning elastic IP. This means that if you use network load balancer as your load balancer for your application you will **end up with several IP addresses for each of the availability zone within a region.**

Target group for NLB can be EC2 instances, IP addresses (must be private IPs, on premise servers again), you can also frontend an ALB with a NLB.

Why would you front ALB with NLB? Well you will end up getting a fixed IP address rather than a fixed hostname, then you can route the HTTP traffics using the ALB directly.

Health checks performed on target group for NLB supports TCP, HTTP, and HTTPS protocols.

When you create a network load balancer, you don't define a security group meaning that the traffic is routed directly to the EC2 instance, therefore the security group that the EC2 instance has will be the security group that takes effect. Meaning you should be defining rules that accepts public traffic for NLB.

> Furthermore, Network load balancer provides both a DNS name and static IP per availability zone that you deploy the NLB in. The reason why the static IP is used is because TCP/UDP does not have DNS server to resolve the name thus they must use static IP for communication. The reason why the DNS name is provided like ALB is only for those applications that communicates via HTTP/HTTPS! Otherwise, the DNS name is useless for TCP/UDP protocols.

Gateway load balancer (GWLB)

Operates at layer 3 (network layer). It looks at the IP packets.

How does it works, well you will be setting up 3rd party security virtual appliances, they can be EC2 instances that looks like the request and tell you whether they are malicious or not. Malicious in terms of intrusion detection, deep packet inspection, whether the payload has been manipulated or not.

Then you will configure your routing table to have your traffic to go through a gateway load balancer which forward the traffic into the target group of those security appliances, they will examine the traffic and analyze it whether or not they are good, if they are then the traffic goes back into gateway load balancer which then finally forward it to the actual application.

It uses the GENEVE protocol on port 6081.

Target group can be the EC2 instances or IP addresses, again 3rd party security virtual appliance or on premise virtual appliances.

Security group for load balancer

The user can access the load balancer using HTTPS/HTTP from anywhere. This means that the security group attached to the load balancer should be accepting on port 80 and 443 with IP range anywhere.

On the other hand for the EC2 instances, because it is being fronted by the load balancer, it should not be accepting any external traffics except from load balancer itself. Which means that the security group for the EC2 instances should reference the security group that the load balancer have. Remember that if you reference a security group, then any traffic that comes from entity that has the security group will be accepted. In this case, only traffic forwarded from the load balancer will be accepted by the EC2 instance.

Sticky sessions

Sticky sessions allows the client to be redirected by the load balancer to the same instance behind the load balancer. For example if Client A is being redirected by the load balancer to EC2 instance 1, then if it makes another request it will still be redirected to EC2 instance 1 instead of another instance by chance.

You would want to do this in order to persist session data like authentication. However, at the cost of causing imblanace to the load balancer.

Sticky session is implemented by a cookie.

There is two types of cookie that you can generate one is application-based cookies, you generate it yourself to include any custom attributes, or use the one that's generated by the load balancer automatically.

They are all cookies so they all have expiration period that you can set, once that cookie expires then the session will no longer be sticky anymore.

Cross-zone load balancing

This feature is enabled by default for application load balancers, meaning that it will distribute the traffic to all of the target groups evenly regardless of their availability zones.

For example, say you have a target group in availability zone A, and another target group in availability zone B for an ALB. The traffic is going to be distributed as if they are all in one availability zone evenly across all the instances regardless of their availability zone locations.

https://wiki.tamarinne.me/uploads/images/gallery/2023-02/Sxsimage.png

If you turn this feature off, then it will be distributed first by the availability zone then among the instances within the availability zone.

https://wiki.tamarinne.me/uploads/images/gallery/2023-02/Q33image.png

You will not be charged by having this feature turn on by default.

However, for NLB and GWLB if you enable this feature then you will be charged for the data that crosses availability zone.

SSL/TLS certificates

Having SSL certificate allows data between client and load balancer to be encrypted.

TLS is the newer version of SSL but people still refer to it as SSL. SSL certificates are issued by certificate authorities.

How does this come into play for a load balancer? Well user will connect to the load balancer via HTTPS which is encrypted, then load balancer talk to the EC2 instances using HTTP in the private VPC. There is no need for encryption because it is within the cloud.

Server name indication

How do you load mulitple SSL certificates onto one web server? How do you host example.com, foo.com, bar.com all in one server? Back in the day is one hostname per one ip, now days servers allows virtual host meaning you can have many different hostname under one server and they all can be hosted on one machine.

How is this achieved? Via Server name indiction, the client indicate to the server the hostname that they want to reach in the initial SSL handshake, then the web server will fetch the corresponding SSL certificate for the hostname that the client ask.

SNI with ALB allows you to host many SSL certificate so it will be directed to the correct EC2 instances.

Classic load balancer can only support one SSL certificate. ALB and NLB support multiple SSL certificates by using SNI.

Connection draining

Also called deregistration delay: Give time to complete "in-flight requests" (request that's already sent out) while the instance is de-registering or unhealthy, then stop sending new requests to the EC2 instance which is de-registering. Newer request will not be sent to the "draining" EC2 instances.

You can set a time for the draining period. Set to a low value if your requests are short, set to high value if your request is long lived, but tradeoff is that your EC2 instances will take more time to drain.

Auto scaling group

Let you do horizontal scaling, in or out automatically to match workloads. It will increase EC2 instances to match increase load, it will decrease EC2 instances to match decrease load all done automatically by the auto scaling group.

You can set up minimum and maximum number of EC2 instances running. Automatically register those new instances to the load balancer. It will also terminate and recreate those unhealthy EC2 instances for you (ELB does the health check then pass those health check to the auto scaling groups which then terminate those unhealthy ones).

When you create an auto scaling group it will contain similar configuration to an EC2 in addition the load balancer that you are pairing it with. **The instance that you choose to launch can be in multi-AZ, auto scaling group will automatically balance it across different AZ that you choose to launch in.**

You will also pair auto scaling group with CloudWatch Alarm to set up a scale-out event or scale-in event.

Dynamic scaling policies

There is three types:

1. Target tracking scaling: The easiest to set up, you can set it up so that you want the average ASG CPU to stay at around 40%
2. Simple / step scaling: This one requires you to set up your own CloudWatch alarm on a custom metric, like CPU utilization. If it is > 70% then you add X units, if it is < say 30% you remove X unit
3. Scheduled action: This is scaling based on known usage patterns, if you know that on Friday you will get a big spike of requests you can set up this scaling policy to increase the minimum capacity on every Friday.

Predictive scaling

Have a forecast of request load and schedule ahead. This is done by machine learning which is pretty cool!

Metrics

Good metrics to scale on is definitely CPU utilization. If CPU utilization is high then it is highly likely that your instances is doing lots of work.

RequestCountPerTarget: Have a specific number of request that you want each of your EC2 instances to have. Say 3 requests on average.

Average network in/out: You can also scale on the amount of data going in or out from the EC2 instances.

Everytime there is a scaling activity occurs there is a cooldown! Default is 5 minutes. During cooldown ASG will not launch or terminate instances, this is to allow metrics to stabilize.

Revision #8
Created 2023-02-11 18:26:41 UTC by Tamarine
Updated 2023-07-27 01:56:09 UTC by Tamarine