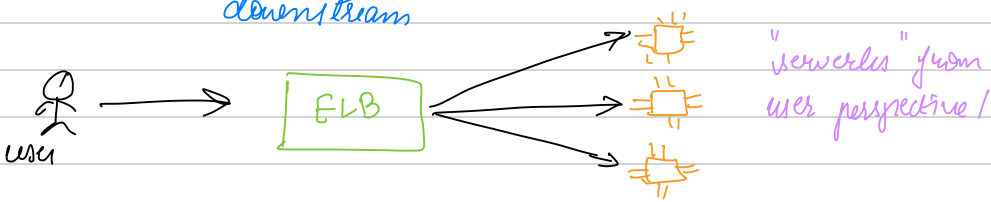
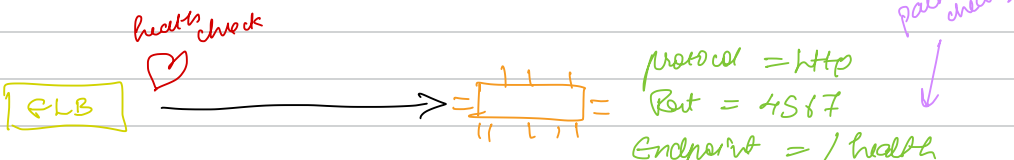
Elastic Load Balancers

→ forward traffic to multiple instances downstream



uses
- → static DNS for application
- → expose a single point of access to application
- → do health checks
- → provide SSL termination
- → high availability
- → separate public from private traffic

* ELB is AWS managed & has ton of integration with other AWS services.



If the response message is not OK (status code = 200) then marked as unhealthy & this will be load balanced to another instance.

* 4 types of load balancer — can be used for private as well as public

1. CLB (Classic) (everything (1st gen) deprecated
2. ALB (Application) (HTTP, HTTPS, Web Socket)
3. NLB (Network) (TCP, TLS, UDP)
4. GLB (Gateway) (packet level, IP protocol) (GENEVE)

(GWLB)

imp

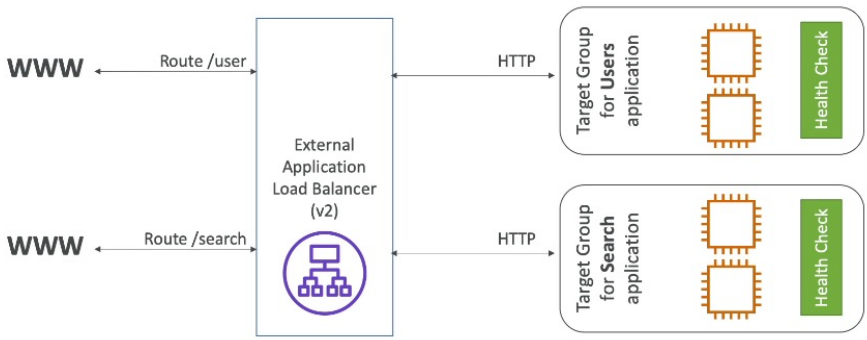
→ instead we can attach security group of EC2 instance to ELB and restrict EC2 instance to allow request from the ELB & nowhere else. hence tightening network sec.

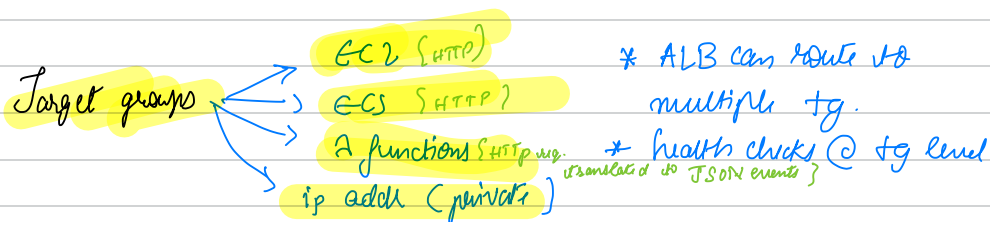
load balancer security group: [HTTP — 80 — 0.0.0.0/0 — Allow http traffic
HTTPS — 443 — — " — — " — https

Application security grp: [HTTP — 80 — Add security group of load balancer — Only traffic through ALB
{ i.e. inbound from security group of load balancer. }

1. Application Load Balancer (V2)
→ Layer 7 only
→ multiple HTTP application across machines (or on the same machine in containers)
→ Supports redirect (from http to https) सीधा अगर http req आया तो उसको change करदेता है
→ Supports route routing. { routing table }
Based on { URL path { /users } hostname Query string & HTTP headers }
→ great for micro service & container based applica^n
→ has port mapping feature to redirect to a dynamic port in ECS
→ support protocols like HTTP, HTTPS & WebSockets

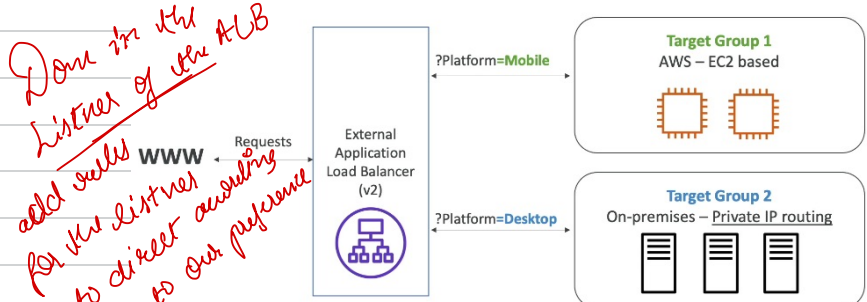
* Same load balancer for different web application!





Application Load Balancer (v2)
Query Strings/Parameters Routing

Done in the Listener of the ALB add rules for the listener to direct routing as per our preference

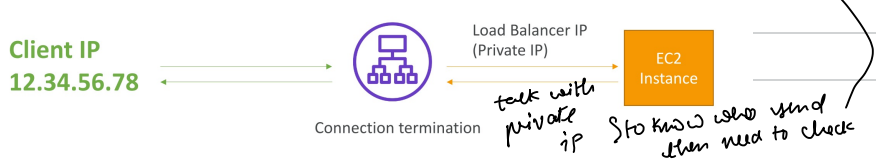


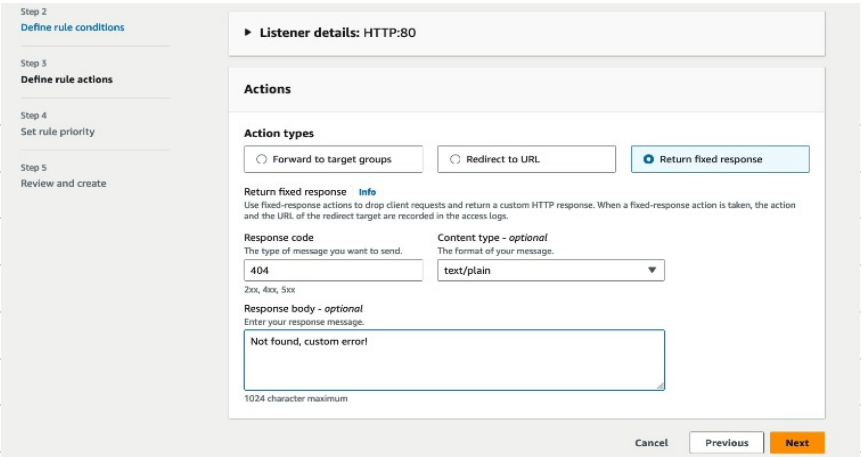
Application Load Balancer (v2)
Good to Know

About the load balancer!

- Fixed hostname (XXX.region.elb.amazonaws.com)
- The application servers don't see the IP of the client directly
  - The true IP of the client is inserted in the header X-Forwarded-For
  - We can also get Port (X-Forwarded-Port) and proto (X-Forwarded-Proto)

protocol





2) Network Load Balancer (v2) (Not free)

↳ Layer 4

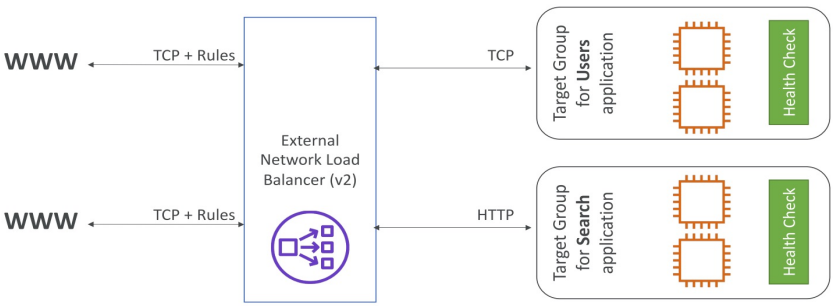
→ forward TCP & UDP traffic, extreme performance

→ less latency (~100ms) [~400ms for ALB]

→ has only 1 static IP per AZ & supports assigning Elastic IP

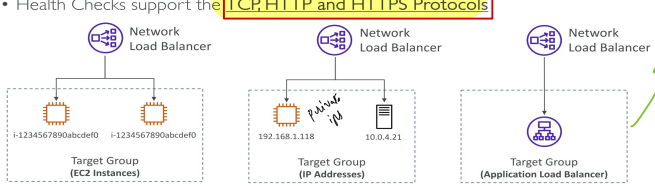
ALBs can't be attached with elastic IPs

Network Load Balancer (v2)
TCP (Layer 4) Based Traffic



Network Load Balancer – Target Groups

- EC2 instances
- IP Addresses – must be private IPs
- Application Load Balancer
- Health Checks support the TCP, HTTP and HTTPS Protocols

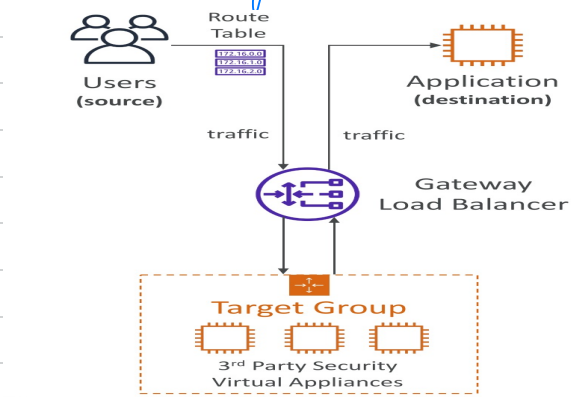


can route to ALB for handling http traffic

③ Gateway Load Balancers

→ deploy, scale & manage fleet of 3rd party network virtual appliances in AWS

Ex: Firewalls, IDPS, Deep packet inspection systems, payload manipulation.



* It analyzes the network layer traffic and then re-routes the traffic

* ✓ Layer 3* (Network layer) – IP pkts

* Uses GENEVE protocol on port 6081

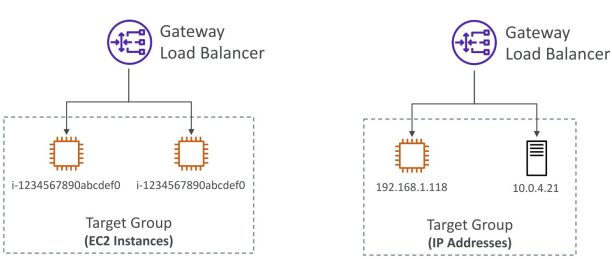
2 kinds

Transparent Network gateway
(single entry/exit for all traffic)

Load balancers
(distribute target to virtual appliances)

Gateway Load Balancer – Target Groups

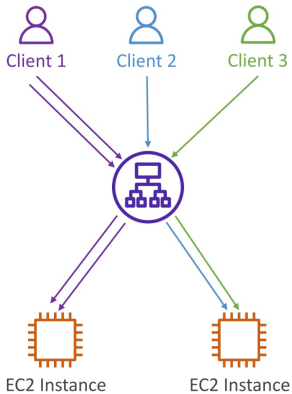
- EC2 instances
- IP Addresses – must be private IPs



Stick session (session affinity)

- It is possible to implement stickiness so that the same client is always redirected to the same instance behind a load balancer
- This works for **Classic Load Balancer, Application Load Balancer, and Network Load Balancer**
- For both CLB & ALB, the "cookie" used for stickiness has an expiration date you control
- Use case: make sure the user doesn't lose his session data
- Enabling stickiness may bring imbalance to the load over the backend EC2 instances

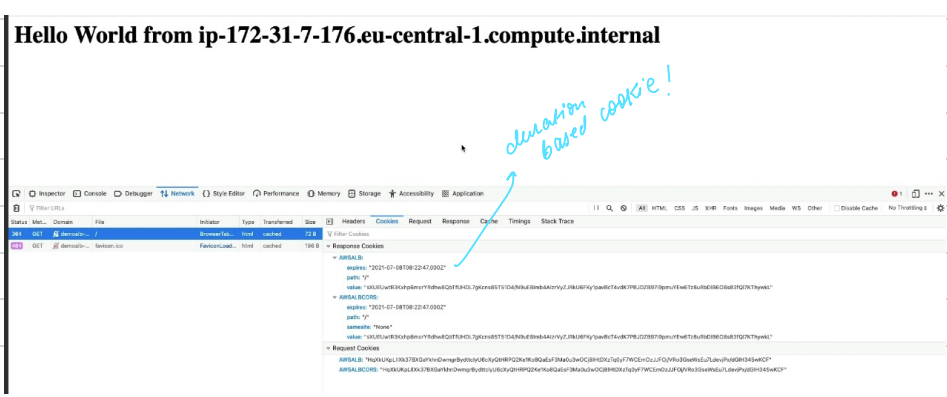
* NLB doesn't use cookie!



Cookie → Application based ⇄ custom application
↓
duration based

* can introduce stickiness during configuration.

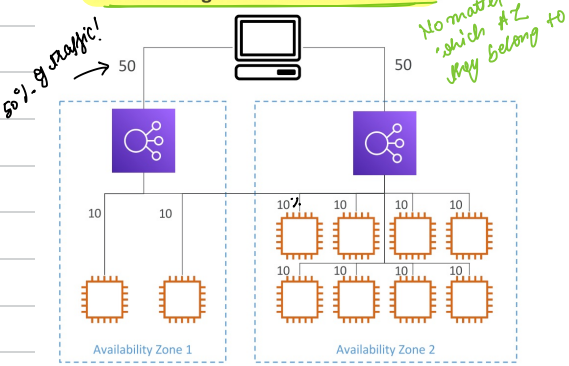
- Application-based Cookies
  - Custom cookie
    - Generated by the target
    - Can include any custom attributes required by the application
    - Cookie name must be specified individually for each target group
    - Don't use **AWSALB**, **AWSALBAPP**, or **AWSALBTG** (reserved for use by the ELB)
  - Application cookie
    - Generated by the load balancer
    - Cookie name is **AWSALBAPP**
- Duration-based Cookies
  - Cookie generated by the load balancer
  - Cookie name is **AWSALB** for ALB, **AWSELB** for CLB



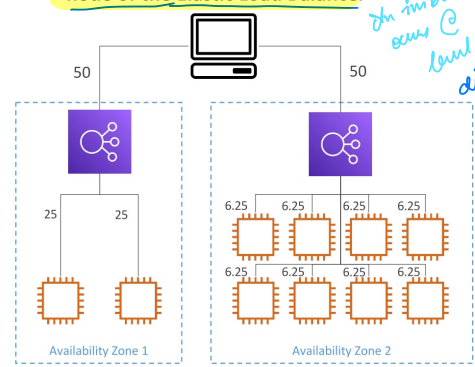
Cross-Zone Load Balancing

Cross-Zone Load Balancing

With Cross Zone Load Balancing:
each load balancer instance distributes evenly across all registered instances in all AZ



Without Cross Zone Load Balancing:
Requests are distributed in the instances of the node of the Elastic Load Balancer



Cross-Zone Load Balancing

- Application Load Balancer
  - Enabled by default (can be disabled at the Target Group level)
  - No charges for inter AZ data
- Network Load Balancer & Gateway Load Balancer
  - Disabled by default
  - You pay charges ($) for inter AZ data if enabled
- Classic Load Balancer
  - Disabled by default
  - No charges for inter AZ data if enabled

LB > Attributes > C Z LB [disabled] > edit > enable

by adding listners to the ELB

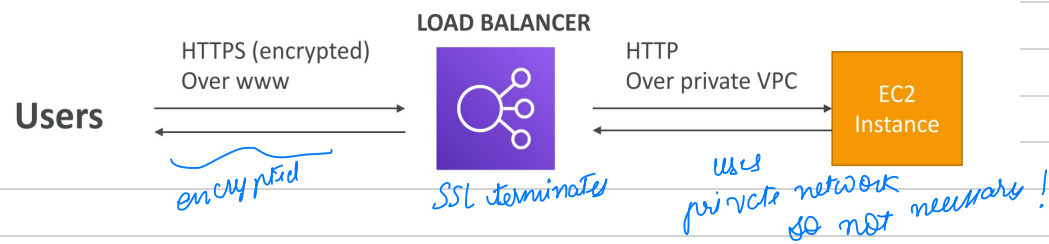
SSL / TLS

SSL allows inflight encryption b/w client & LB

Secure Socket Layer

TLS → newer version } Transport Layer Security }

public SSL certificates issued by Certificate Authorities (CA)
they need to be renewed from time to time because they have an expiration date



- The load balancer uses an X.509 certificate (SSL/TLS server certificate)
- You can manage certificates using ACM (AWS Certificate Manager)
- You can create upload your own certificates alternatively
- HTTPS listener:
  - You must specify a default certificate
  - You can add an optional list of certs to support multiple domains
  - Clients can use SNI (Server Name Indication) to specify the hostname they reach
  - Ability to specify a security policy to support older versions of SSL / TLS (legacy clients)

SNI { server name indication }

→ Allows to expose multiple HTTPS application each with its own SSL certificate on the same listner.

- SNI solves the problem of loading **multiple SSL certificates onto one web server** (to serve multiple websites)
- It's a "newer" protocol, and requires the client to **indicate** the hostname of the target server in the initial SSL handshake
- The server will then find the correct certificate, or return the default one

Note:

- Only works for ALB & NLB (newer generation), CloudFront
- Does not work for CLB (older gen)

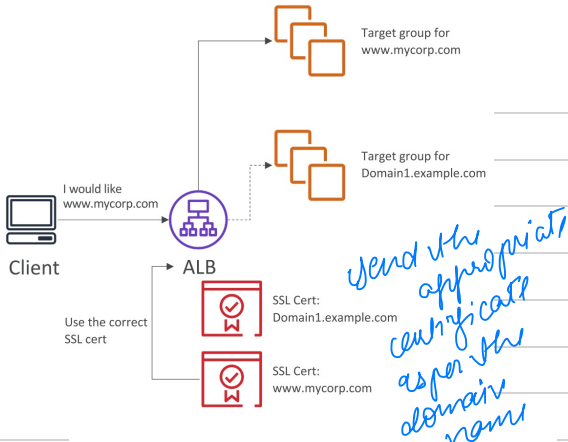
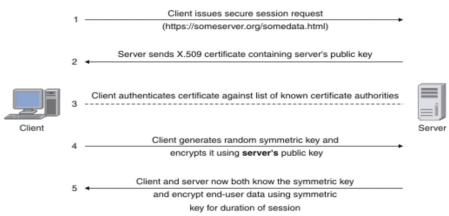


Figure 1. SSL handshake



1. The client sends a request to the server for a secure session. The server responds by sending its X.509 digital certificate to the client.
2. The client receives the server's X.509 digital certificate.
3. The client authenticates the server, using a list of known certificate authorities.
4. The client generates a random symmetric key and encrypts it using server's public key.
5. The client and server now both know the symmetric key and can use the SSL encryption process to encrypt and decrypt the information contained in the client request and the server response.

how is SSL connection established!

Elastic Load Balancers – SSL Certificates

- Classic Load Balancer (v1)
  - Support only one SSL certificate
  - Must use multiple CLB for multiple hostname with multiple SSL certificates
- Application Load Balancer (v2)
  - Supports multiple listeners with multiple SSL certificates
  - Uses Server Name Indication (SNI) to make it work
- Network Load Balancer (v2)
  - Supports multiple listeners with multiple SSL certificates
  - Uses Server Name Indication (SNI) to make it work

ELB > Listner > Add listner > {HTTPS (443) > Forward to tg > ELB security policy > ACM (client) or import it}

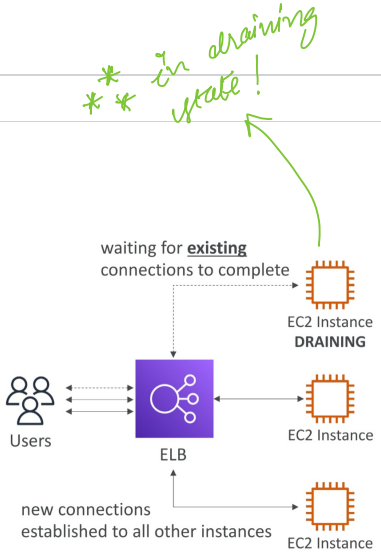
Connection Draining

Retaining the connection while the instance is de-registering itself or going unhealthy.

existing req. के पूरा होने तक पूरा जाएगी! उसके बाद ही वो जाएगा unhealthy!

Connection Draining

- Feature naming
  - Connection Draining – for CLB
  - Deregistration Delay – for ALB & NLB
- Time to complete "in-flight requests" while the instance is de-registering or unhealthy
- Stops sending new requests to the EC2 instance which is de-registering
- Between 1 to 3600 seconds (default: 300 seconds)
- Can be disabled (set value to 0)
- Set to a low value if your requests are short



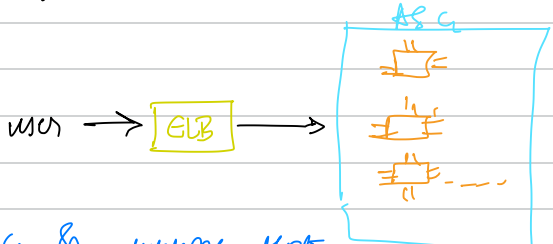
Auto scaling group

get rid of servers very quickly!

Scale out { increase the no. of instances } or
Scale in { decrease the no. of instances }

* ASG is free { but have to pay for the instance created }

minimum capacity
desired — " —
maximum — " —

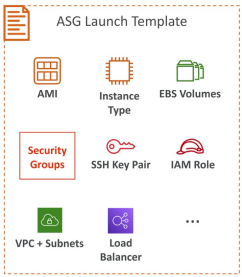


→ When ALB frontend, a ASG & suppose that one of the EC2 instance fails the health check, then the ASG will automatically terminate that instance & relaunch a new EC2 instance.

Auto Scaling Group Attributes

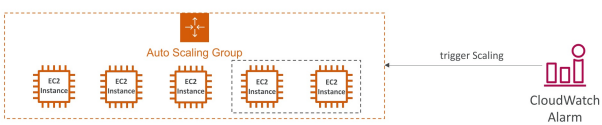
information about the instances to be launched!

- A Launch Template (older "Launch Configurations" are deprecated)
  - AMI + Instance Type
  - EC2 User Data
  - EBS Volumes
  - Security Groups
  - SSH Key Pair
  - IAM Roles for your EC2 Instances
  - Network + Subnets Information
  - Load Balancer Information
- Min Size / Max Size / Initial Capacity
- Scaling Policies



Auto Scaling - CloudWatch Alarms & Scaling

- It is possible to scale an ASG based on CloudWatch alarms
- An alarm monitors a metric (such as Average CPU, or a custom metric)
- Metrics such as Average CPU are computed for the overall ASG instances
- Based on the alarm:
  - We can create scale-out policies (increase the number of instances)
  - We can create scale-in policies (decrease the number of instances)

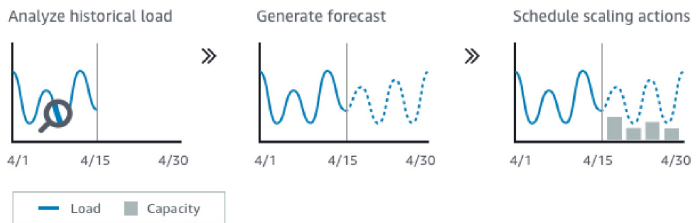


Scaling policies for ASG!

1) Dynamic Scaling
   - Target Tracking scaling (simpler)
     { avg CPU utilization to 70% (example) }
   - Simple / Step scaling
     { if CPU > 70% then add 2 units
     if CPU < 30% then remove 1 unit }

2) Scheduled Scaling
   { Anticipate a scaling based on known usage pattern
   ex time fix होतो उदा रात्री 10 AM to 6 PM (1) min capacity }

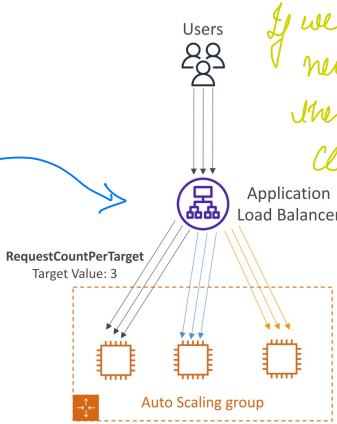
3) Predictive scaling { continuously forecast load & schedule scaling ahead }



Good metrics to scale on

- **CPUUtilization**: Average CPU utilization across your instances
- **RequestCountPerTarget**: to make sure the number of requests per EC2 instances is stable
- **Average Network In / Out** (if you're application is network bound)
- **Any custom metric** (that you push using CloudWatch)

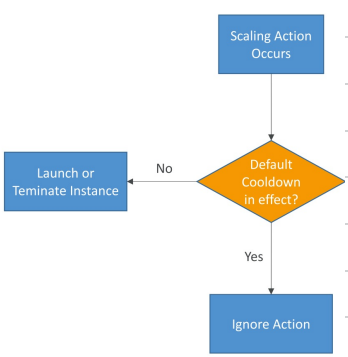
Already present just choose these metrics.



If we need a new metric then create a CloudWatch metric
Create an alarm on this metric to scale the ASG.

- After a scaling activity happens, you are in the cooldown period (default 300 seconds)
- During the cooldown period, the ASG will not launch or terminate additional instances (to allow for metrics to stabilize)

- Advice: Use a ready-to-use AMI to reduce configuration time in order to be serving request fasters and reduce the cooldown period



ASG > attributes { Automatic scaling } > add scaling policies