**Scalability & High Availability**

* Scalability means that an application / system can handle greater loads by adapting.
* There are two kinds of scalability:
  + Vertical Scalability
  + Horizontal Scalability (= elasticity)
* Scalability is linked but different to High Availability

**Vertical Scalability**

* Vertical scalability means increasing the size of the instance
* For example, your application runs on a t2.micro
* Scaling that application vertically means running it on a t2.large
* Vertical scalability is very common for non distributed systems, such as a database.
* RDS, ElastiCache are services that can scale vertically.
* There’s usually a limit to how much you can vertically scale (hardware limit)

**Horizontal Scalability**

* Horizontal Scalability means increasing the number of instances / systems for your application
* Horizontal scaling implies distributed systems.
* This is very common for web applications / modern applications
* It’s easy to horizontally scale thanks the cloud offerings such as Amazon EC2

**High Availability**

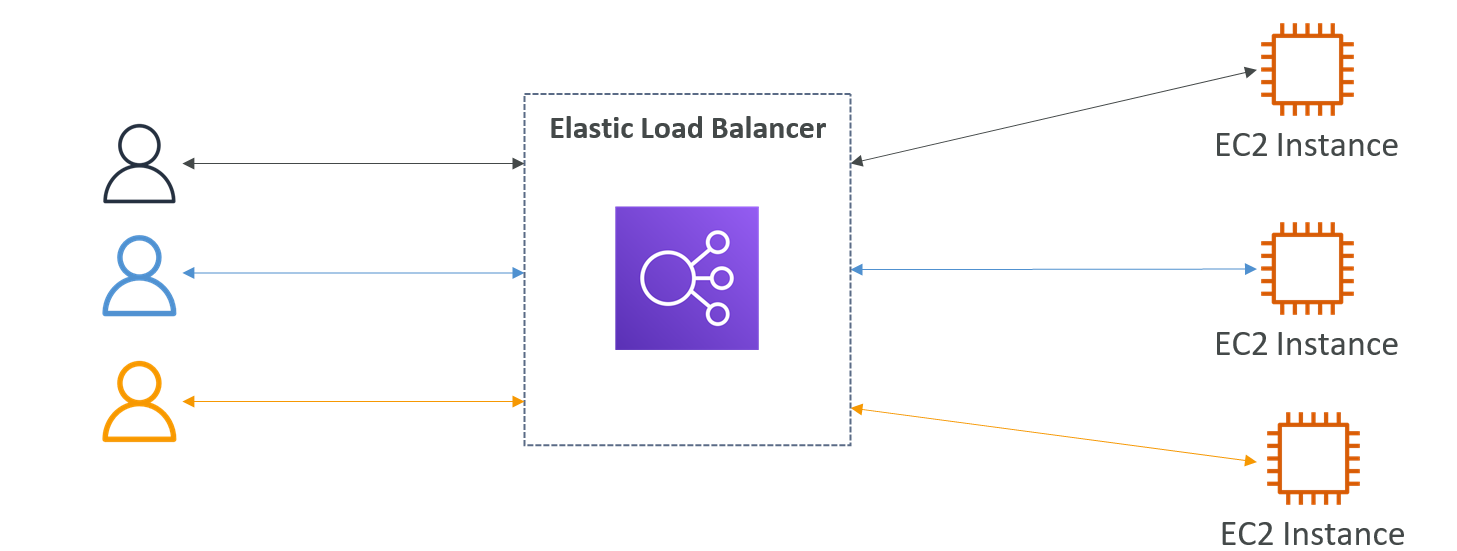
* High Availability usually goes hand in hand with horizontal scaling
* High availability means running your application / system in at least 2 data centers (== Availability Zones)
* The goal of high availability is to survive a data center loss
* The high availability can be passive (for RDS Multi AZ for example)
* The high availability can be active (for horizontal scaling)

**High Availability & Scalability For EC2**

* Vertical Scaling: Increase instance size (= scale up / down)
  + From: t2.nano - 0.5G of RAM, 1 vCPU
  + To: u-12tb1.metal – 12.3 TB of RAM, 448 vCPUs
* Horizontal Scaling: Increase number of instances (= scale out / in)
  + Auto Scaling Group
  + Load Balancer
* High Availability: Run instances for the same application across multi AZ
  + Auto Scaling Group multi AZ
  + Load Balancer multi AZ

**What is load balancing?**

* Load Balances to servers that forward traffic to multiple servers (e.g., EC2 instances) downstream



**Why use a load balancer?**

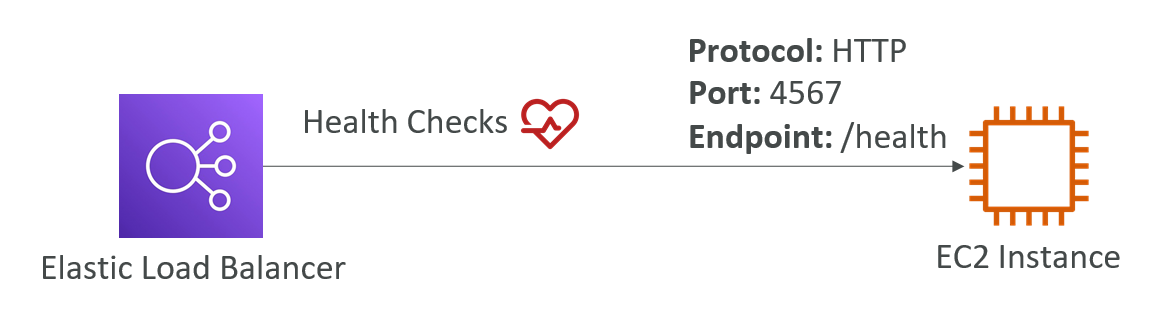
* Spread load across multiple downstream instances
* Expose a single point of access (DNS) to your application
* Seamlessly handle failures of downstream instances
* Do regular health checks to your instances
* Provide SSL termination (HTTPS) for your websites
* Enforce stickiness with cookies
* High availability across zones

**Why use an Elastic Load Balancer?**

* An Elastic Load Balancer is a managed load balancer
  + AWS guarantees that it will be working
  + AWS takes care of upgrades, maintenance, high availability
  + AWS provides only a few configuration knobs
* It costs less to setup your own load balancer but it will be a lot more effort on your end
* It is integrated with many AWS offerings / services
  + EC2, EC2 Auto Scaling Groups, Amazon ECS
  + AWS Certificate Manager (ACM), CloudWatch
  + Route 53, AWS WAF, AWS Global Accelerator

**Health Checks**

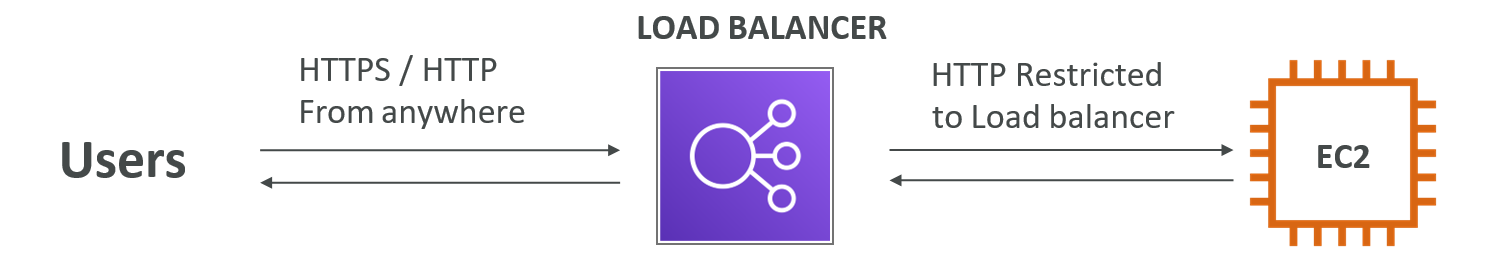
* Health Checks are crucial for Load Balancers
* They enable the load balancer to know if instances it forwards traffic to are available to reply to requests
* The health check is done on a port and a route (/health is common)
* If the response is not 200 (OK), then the instance is unhealthy

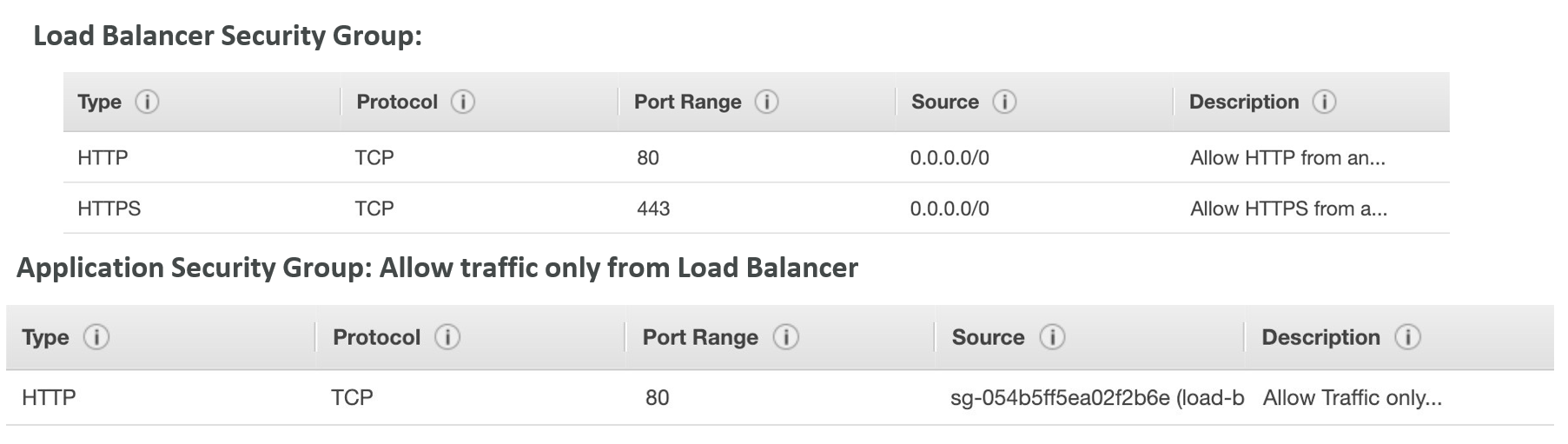


**Types of load balancer on AWS**

* AWS has 4 kinds of managed Load Balancers
  + Classic Load Balancer (v1 - old generation) – 2009 – CLB
  + Application Load Balancer (v2 - new generation) – 2016 – ALB
  + Network Load Balancer (v2 - new generation) – 2017 – NLB
  + Gateway Load Balancer – 2020 – GWLB

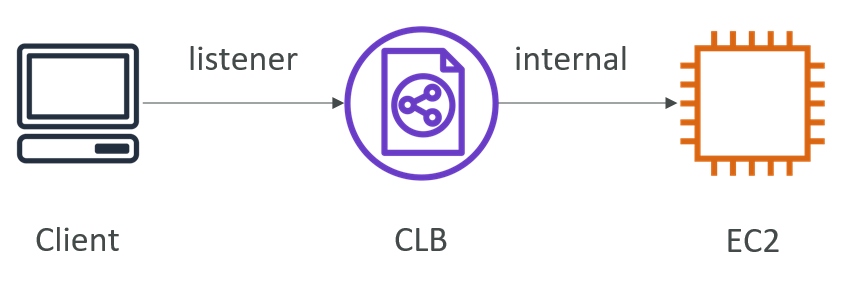
Load Balancer Security Groups





**Classic Load Balancers (v1)**

* Supports TCP (Layer 4), HTTP & HTTPS (Layer 7)
* Health checks are TCP or HTTP based
* Fixed hostname XXX.region.elb.amazonaws.com



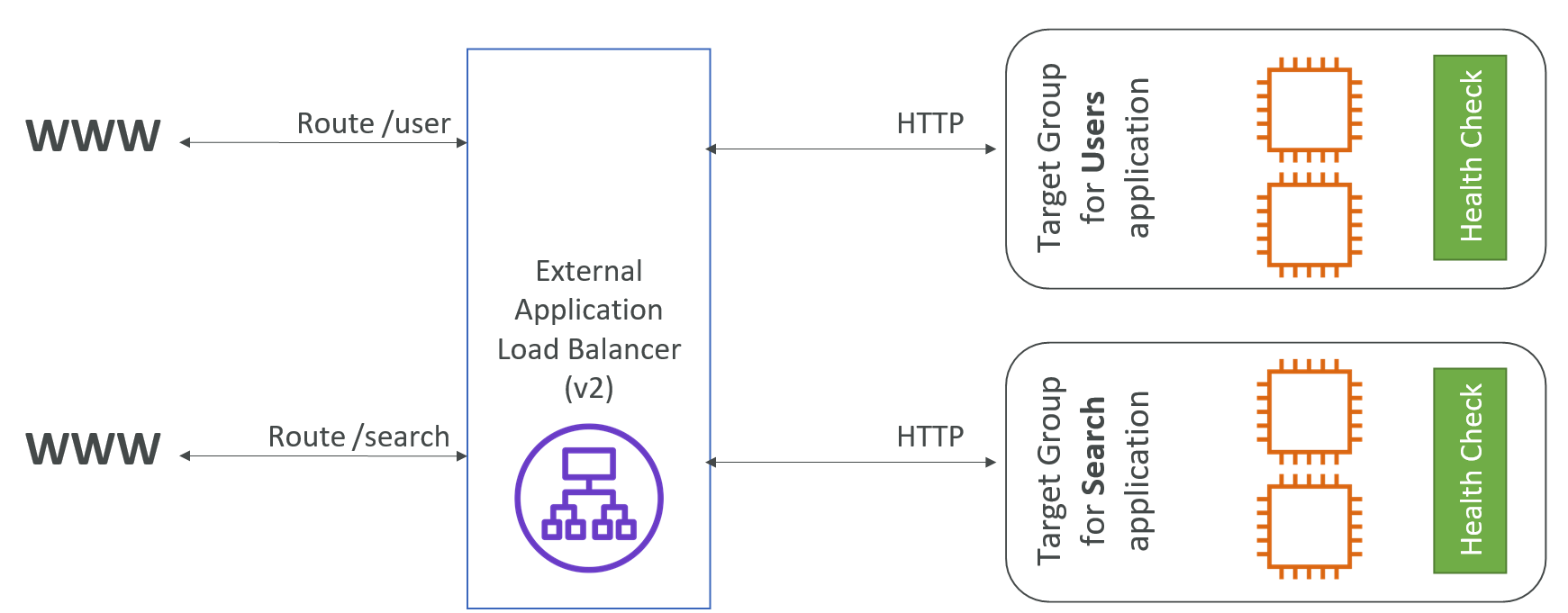
**Application Load Balancer (v2)**

* Application load balancers is Layer 7 (HTTP)
* Load balancing to multiple HTTP applications across machines (target groups)
* Load balancing to multiple applications on the same machine (ex: containers)
* Support for HTTP/2 and WebSocket
* Support redirects (from HTTP to HTTPS for example)

**Application Load Balancer (v2)**

* Routing tables to different target groups:
  + Routing based on path in URL (example.com/users & example.com/posts)
  + Routing based on hostname in URL (one.example.com & other.example.com)
  + Routing based on Query String, Headers (example.com/users?id=123&order=false)
* ALB are a great fit for micro services & container-based application (example: Docker & Amazon ECS)
* Has a port mapping feature to redirect to a dynamic port in ECS
* In comparison, we’d need multiple Classic Load Balancer per application

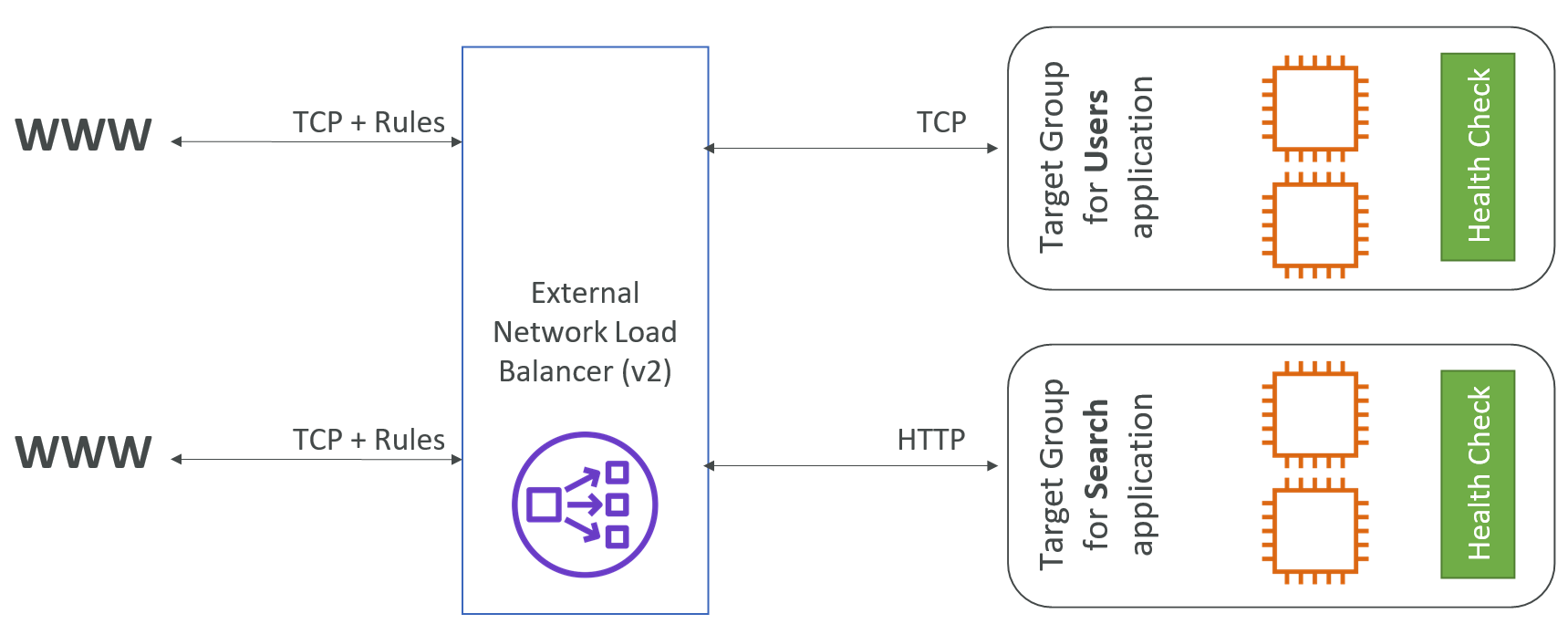
**HTTP Based Traffic**



**Network Load Balancer (v2)**

* Network load balancers (Layer 4) allow to:
  + Forward TCP & UDP traffic to your instances
  + Handle millions of request per seconds
  + Less latency ~100 ms (vs 400 ms for ALB)
* NLB has one static IP per AZ, and supports assigning Elastic IP
* (helpful for whitelisting specific IP)
* NLB are used for extreme performance,TCP or UDP traffic
* Not included in the AWS free tier

**TCP (Layer 4) Based Traffic**

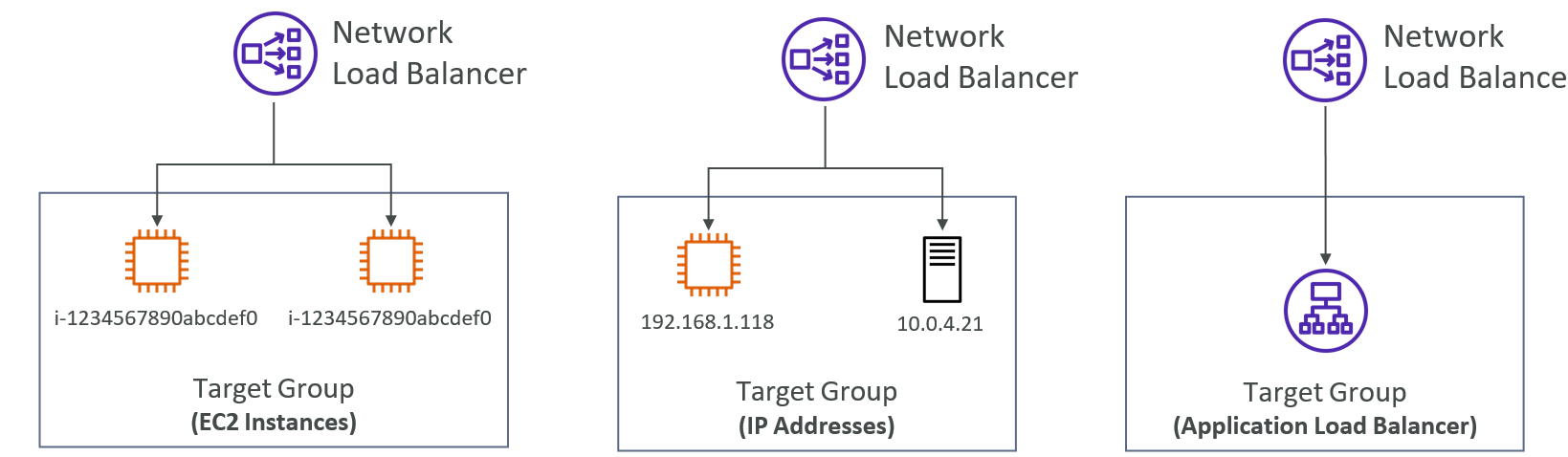


**Target Groups**

* EC2 instances (can be managed by an Auto Scaling Group) – HTTP
* ECS tasks (managed by ECS itself) – HTTP
* Lambda functions – HTTP request is translated into a JSON event
* IP Addresses – must be private IPs
* ALB can route to multiple target groups
* Health checks are at the target group level

**Network Load Balancer – Target Groups**

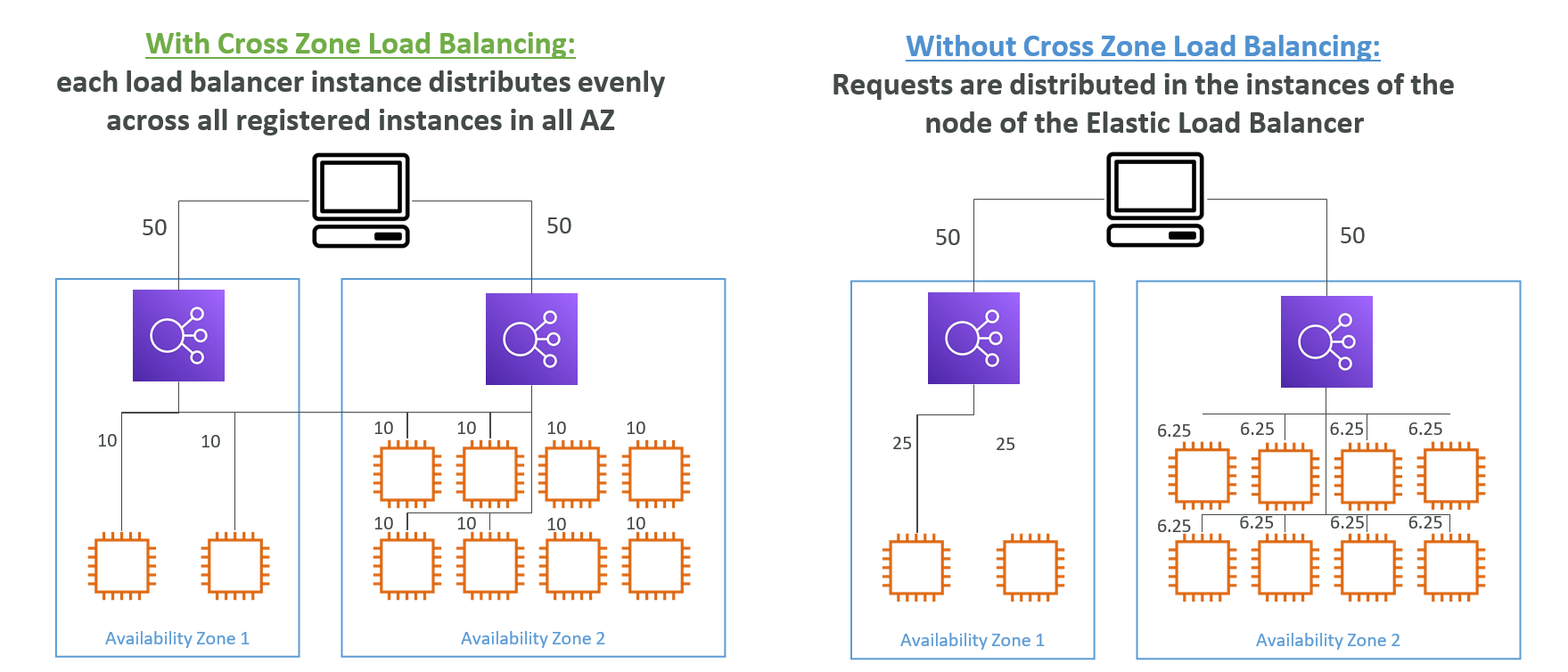
* EC2 instances
* IP Addresses – must be private IPs
* Application Load Balancer



**Sticky Sessions (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 Balancers & Application Load Balancers
* 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

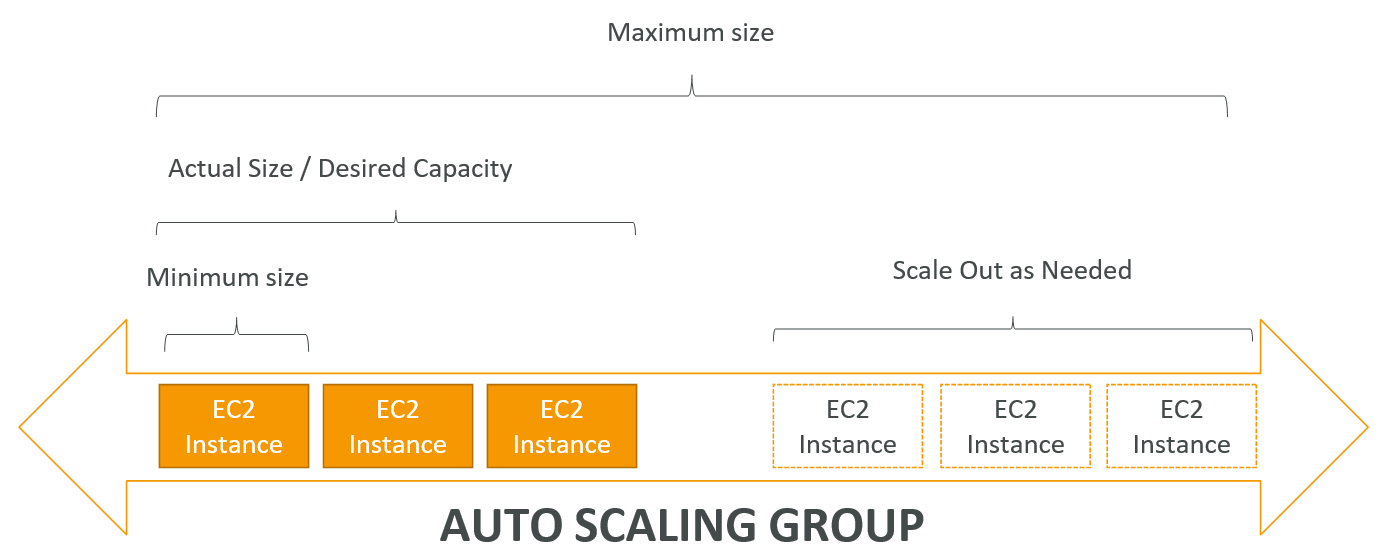
**Cross-Zone Load Balancing**



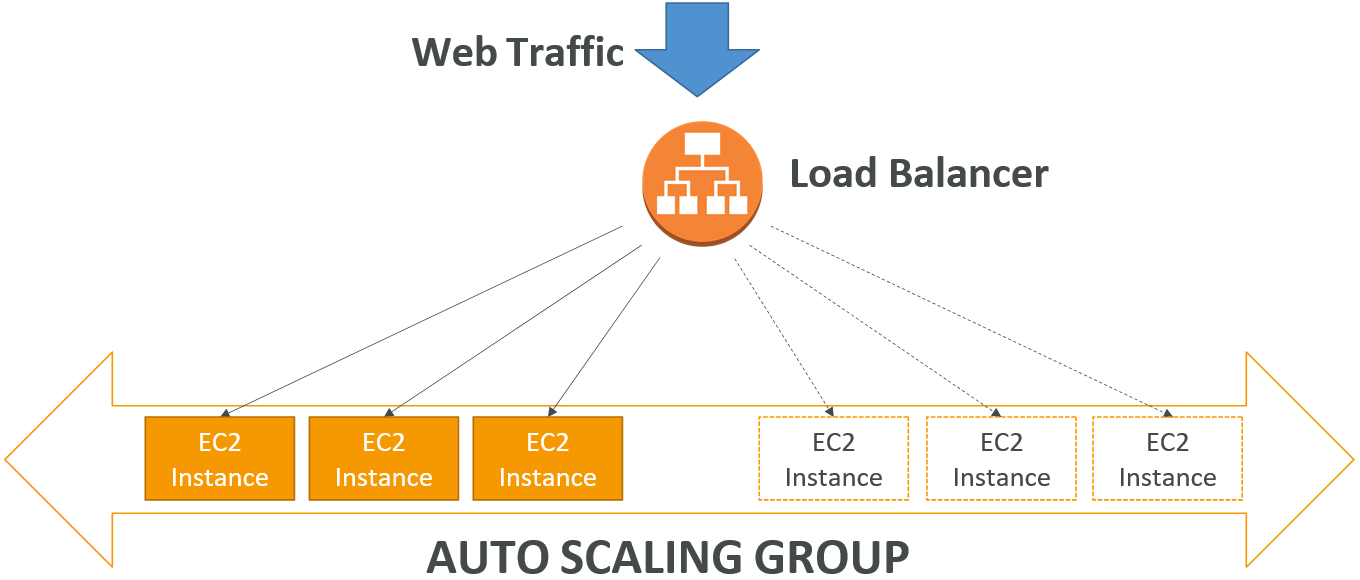
**What’s an Auto Scaling Group?**

* In real-life, the load on your websites and application can change
* In the cloud, you can create and get rid of servers very quickly
* The goal of an Auto Scaling Group (ASG) is to:
  + Scale out (add EC2 instances) to match an increased load
  + Scale in (remove EC2 instances) to match a decreased load
  + Ensure we have a minimum and a maximum number of machines running
  + Automatically Register new instances to a load balancer

**Auto Scaling Group in AWS**



**Auto Scaling Group in AWS With Load Balancer**

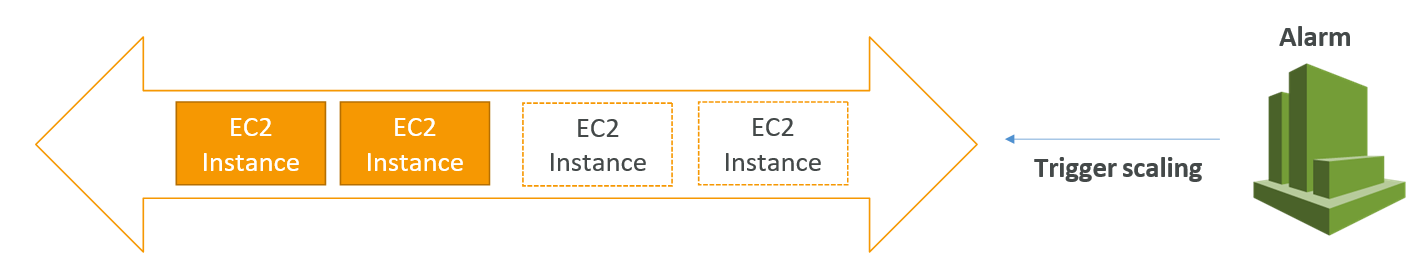


**ASGs have the following attributes**

* A launch configuration/Launch Template
  + AMI + Instance Type
  + EC2 User Data
  + EBS Volumes
  + Security Groups
  + SSH Key Pair
* Min Size / Max Size / Initial Capacity
* Network + Subnets Information
* Load Balancer Information
* Scaling Policies

**Auto Scaling Alarms**

* It is possible to scale an ASG based on CloudWatch alarms
* An Alarm monitors a metric (such as Average CPU)
* Metrics 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)



**Auto Scaling New Rules**

* It is now possible to define ”better” auto scaling rules that are directly managed by EC2
  + Target Average CPU Usage
  + Number of requests on the ELB per instance
  + Average Network In
  + Average Network Out
* These rules are easier to set up and can make more sense

**ASG Brain Dump**

* Scaling policies can be on CPU, Network… and can even be on custom metrics or based on a schedule (if you know your visitors patterns)
* ASGs use Launch configurations or Launch Templates (newer)
* To update an ASG, you must provide a new launch configuration / launch template
* IAM roles attached to an ASG will get assigned to EC2 instances
* ASG are free.You pay for the underlying resources being launched
* Having instances under an ASG means that if they get terminated for whatever reason, the ASG will automatically create new ones as a replacement. Extra safety!
* ASG can terminate instances marked as unhealthy by an LB (and hence replace them)

**Auto Scaling Groups – Dynamic Scaling Policies**

* Target Tracking Scaling
  + Most simple and easy to set-up
  + Example: I want the average ASG CPU to stay at around 40%
* Simple / Step Scaling
  + When a CloudWatch alarm is triggered (example CPU > 70%), then add 2 units
  + When a CloudWatch alarm is triggered (example CPU < 30%), then remove 1
* Scheduled Actions
  + Anticipate a scaling based on known usage patterns
  + Example: increase the min capacity to 10 at 5 pm on Fridays

Auto Scaling Groups – Predictive Scaling

* Predictive scaling: continuously forecast load and schedule scaling ahead

