Load Balancer bir IT terimi, AWS ye has bir şey değil. Gelen trafiği dağıtır.

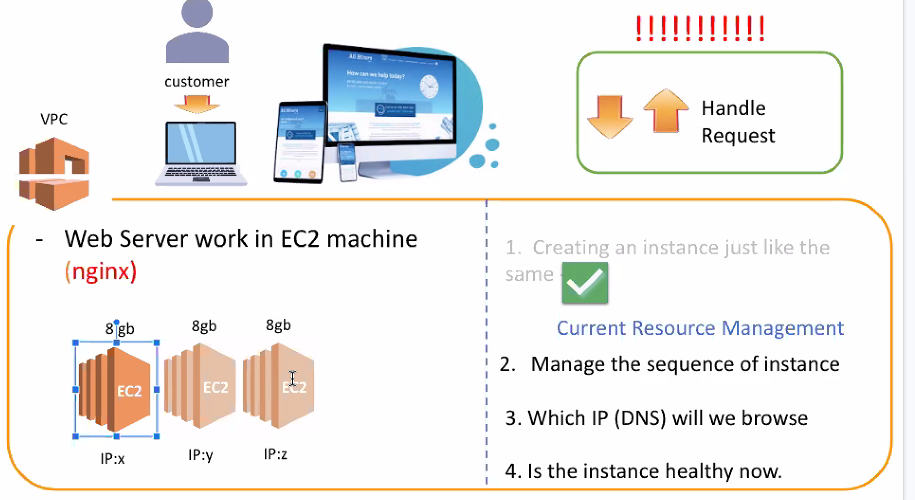
Bir trafik polisi gibi.

Bir çikolata firmamız var ve web sitesini yapıyoruz. Bir tane instance ayağa kaldırdık. Bayram arefesi geldi ve yoğunluk başladı. Bütün requestler o instance a gelecek. Bu instance ın içindeki web sitesinin aynısından bir tane daha lazım ki insanlar rahatça alışveriş yapabilsin.

Snapshot ve Template ile bu sorunu çözmüştük. Aynı siteyi bu yöntemle üç farklı instance ile çalıştırdık. Bu üç instance ın hangisine request yapılacağını müşteri ya da biz seçemeyiz. IP lerin tek bir çatı altında toplanması gerekiyor.

Bir de ayağa kaldırdığımız instance lar sağlıklı mı değil mi? Instance lara yük dağıtalacak ancak instance ların sağlıklı olup olmadığını bilmemiz lazım.

Load Balancer IP leri tek çatı altında toplayarak yükü dağıtır ve healty instance ı tespit ederek trafiği ona yönlendirir. Bir de disaster rcovery ya da failover da fayda sağlar. Diyelim ki AZ miz patladı. Bütün instance lar aynı kesedeydi hepsi gitti. Load balancer da instance ları farlı AZ lere de kurmamızı sağlar.

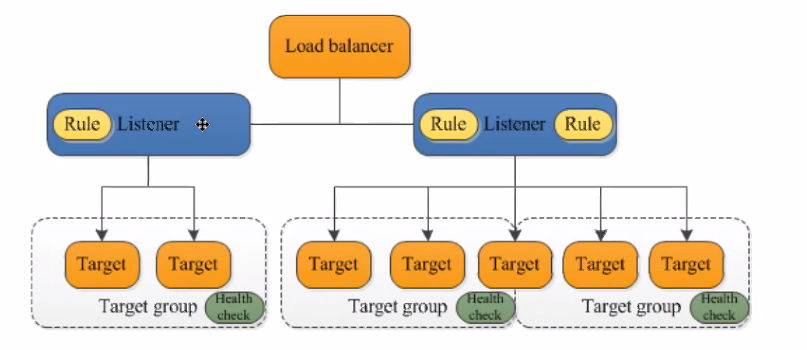


Componentleri var Listener, Configuration ve Target Group

Listener trafiği dinler. Portlardan gelen trafği dinler. Application Load Balancer 80 porttan gelen trafiğe bakar örneğin.

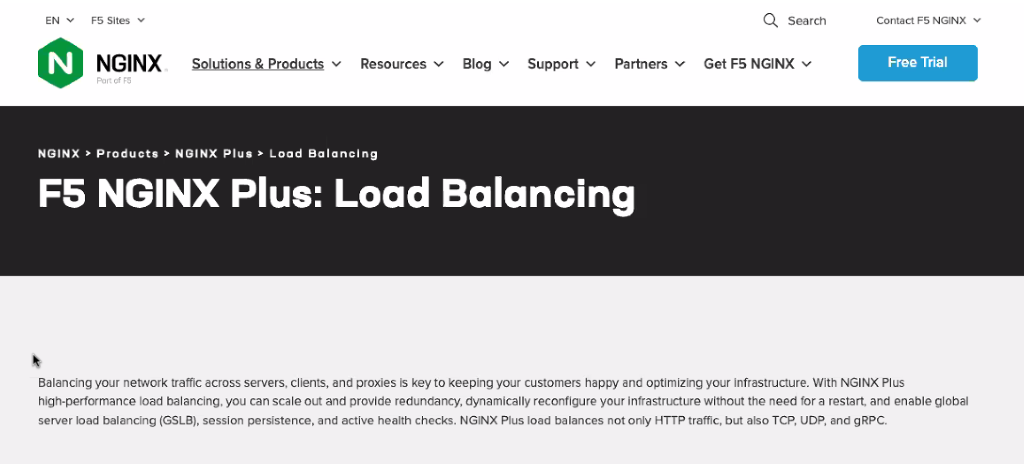
Configuration da kurallardır.

Target Group da hedef instance lardır. Havuzda kaç instance ımız var ve bunlar hangileridir. (EC2 pool) DevOps ta bunlar instance değil container olacak.

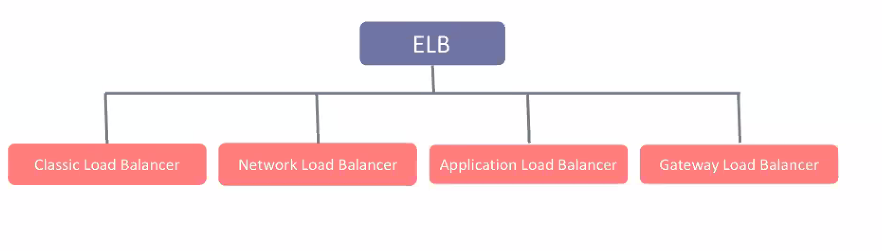


Biz bir kural (şu ip ile gelenleri şu instance a gönder gibi ) belirliyoruz listener trafiği dinliyor ve uygun target a route ediyor. Birden fazla Load Balancing metodu tek projede de kullanılır. Bir instance birden fazla kurala tabi olabilir. Bir sıralaması vardır.

Load Balancer AWS ye has bir şey değil. Örneğin NGINX in Load Balancer ı var. AWS de buradan almıştır.



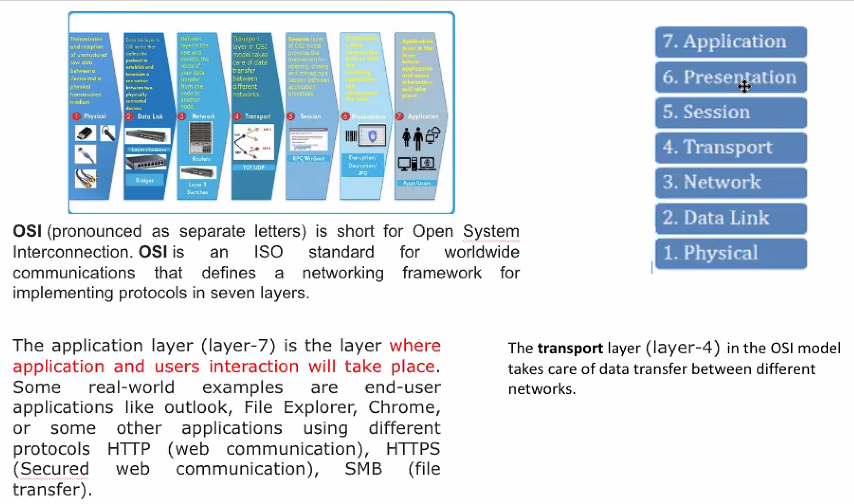
<https://www.nginx.com/products/nginx/load-balancing/>



DevOps ta Network Load Balancer kullanacağız. Application LB Layer 7 de, Network LB Layer 3-4 te çalışır.

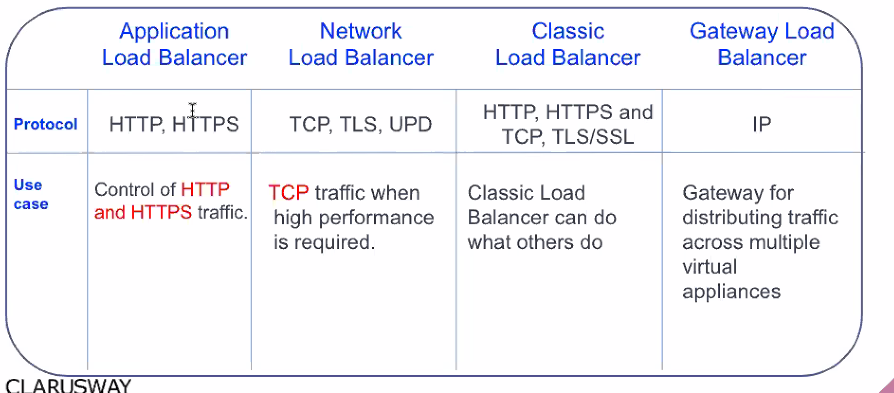
Gateway LB ayrı bir subnet te bulunan firework lardaki çıkışı dağıtan LB lerdir. Gelen requestlerin güvenli olup olmadığını denetledikten sonra dağıtılsın istediğimizde kullanılır.

Application Layer TV gibi Layer 3-4 Radyo gibi. Yani daha hızlıdır Bu nedenle Network LB bize hız sağlar.



Network LB static IP alır ancak Application LB static IP almaz. Network LB daha çok back end operasonlar içindir. Arka tarafta dönen dahili operasyonlarda kullanılıyor.

Application LB dışarıdan gelen trafikte yoğun olarak kullanılıyor.





Launch Config Launch Template ile çok benziyor ancak birkaç fark var. En temel farkı Launch Config de versionlama yok.

Hands ona geçiyoruz.

Best practice: Bir public subnetimiz olacak. Oraya LB konur.

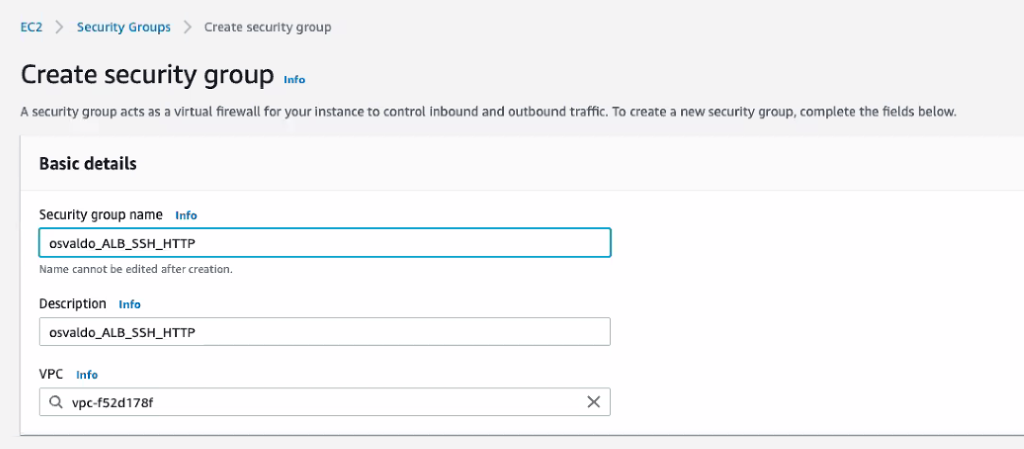
Instance lar private olur. Dışarıdan direk instance lara ulaşılmaz. Önce LB ye ulaşılır o trafiği instance lara yönlendirir.

LB ye bir Security Group konur. Bu dışarıdan gelen bütün 22 ve 80 leri dinler ve hepsini alır. Instance lara da security group belirlenir. Bunlar ise sadece LB nin Security Group ubundan geçenleri alır.

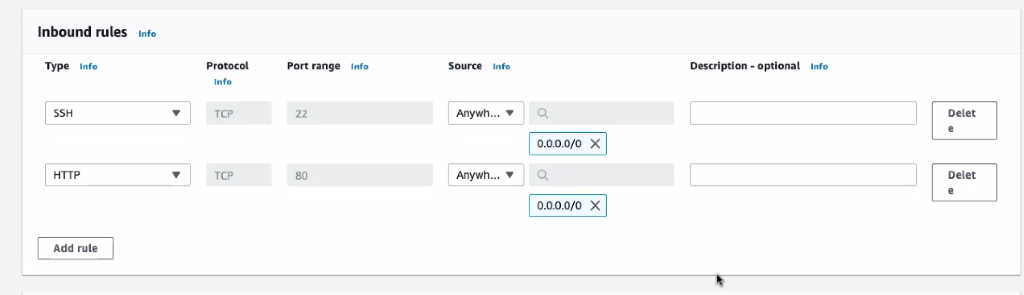
Biz bugün VPC görmediğimiz için böyle yapmayacaz. Bugün Instance lar ve LB public subnet te olacak ve aynı SG ye sahip olacaklar.

EC2 konsola geçip Sol taraftaki menüden Security Groups a geliyoruz.

Create diyoruz, isim ve description belirliyoruz:



Rule ları belirliyoruz:

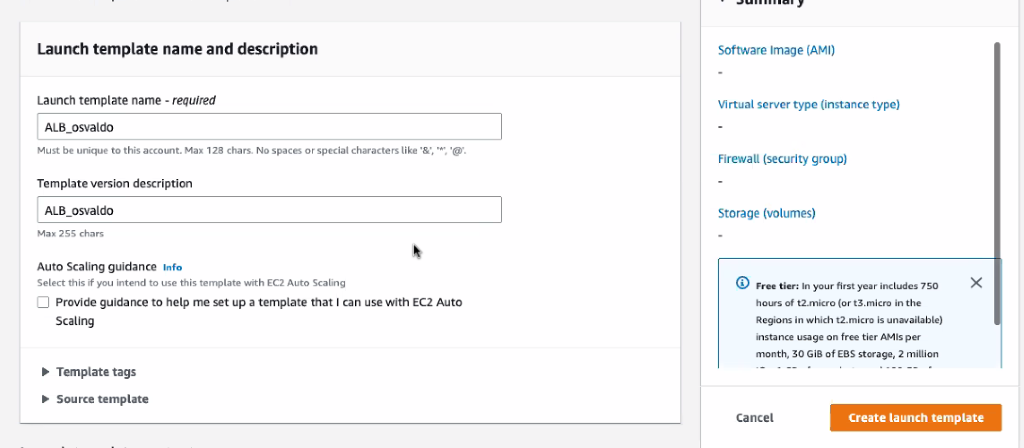


Create diyerek SG yi oluşturduk.

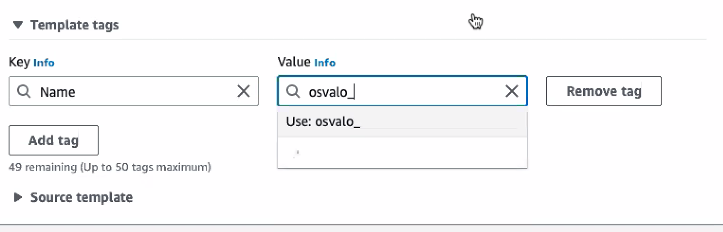


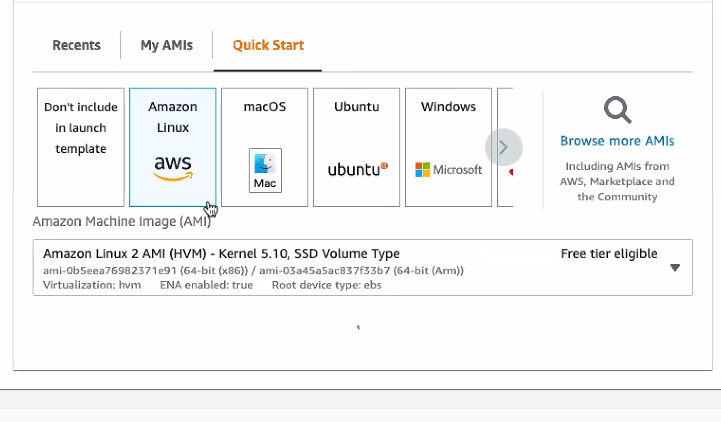
Buradan sol menüden Launch Template e gidiyoruz ve create diyoruz:

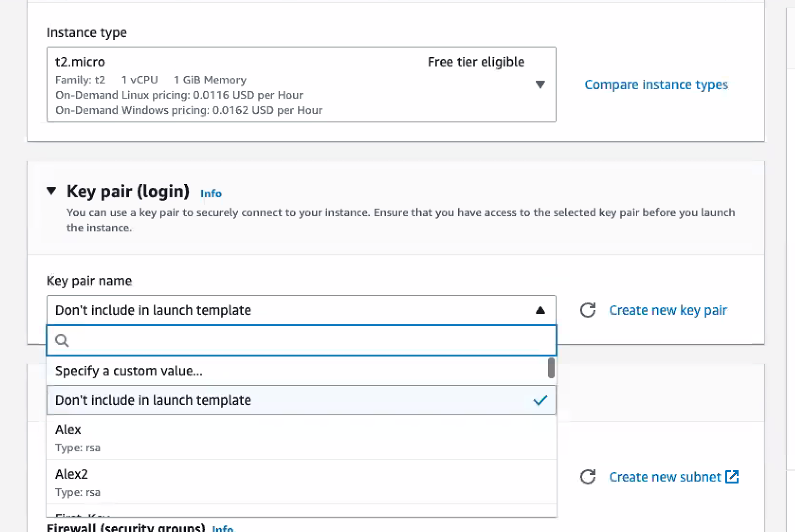
(Bir alçı kalıbı verecez ki LB benzer iki siteyi yönetsin.)



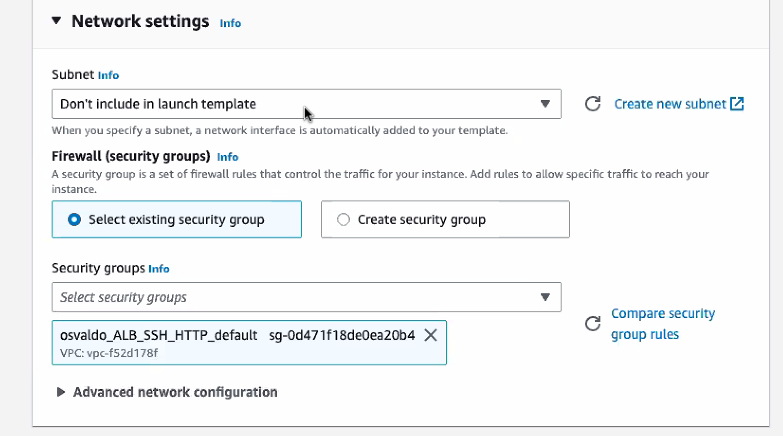
Tag ekliyoruz:



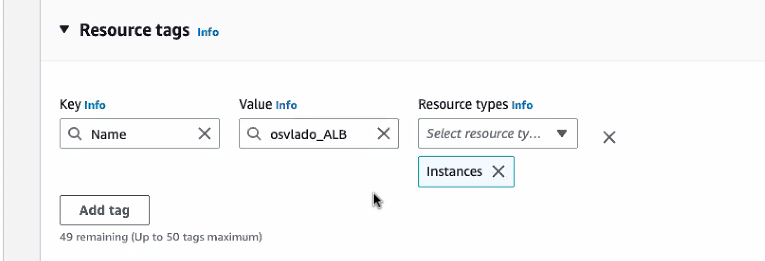




Oluşturduğumuz SG yi seçiyoruz:

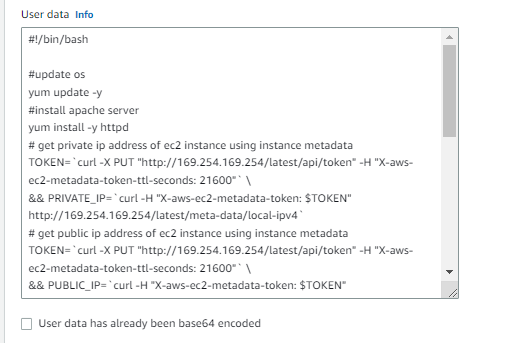


Reasource tag; bu alçı kalbının içinden kuşa vereceğimiz isim. Az önceki tag kalıba verdiğimiz isimdi.

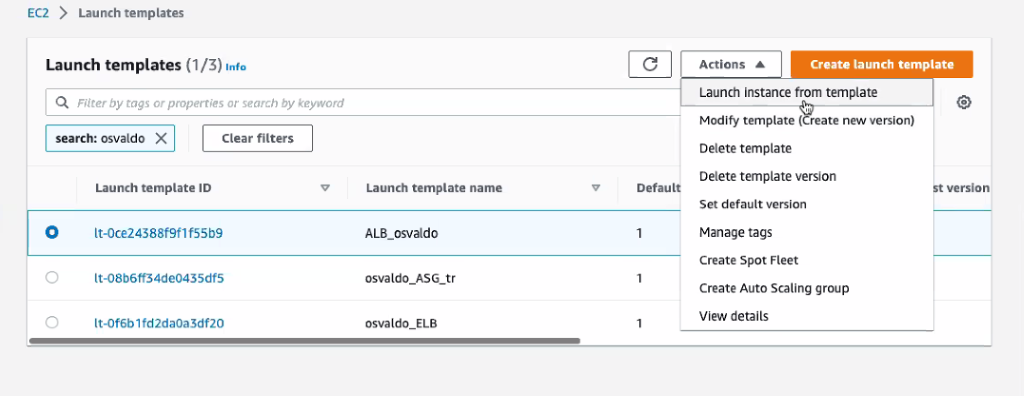


Instance lar yeterli şu anda.

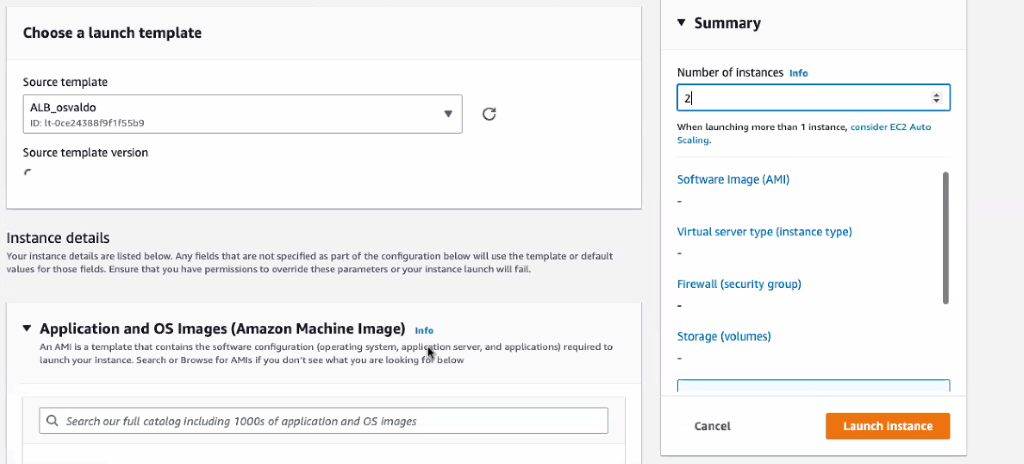
User Data ekliyoruz:



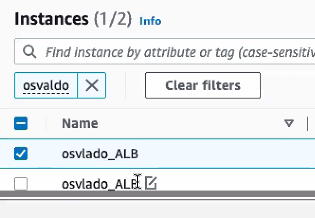
Create launch template dedik. Launch template imizi seçip Actions menüsünden Launch Instances From Template diyoruz:



İki tane instance kaldıracaz.



Launch instance dedik ve instance lara gidiyoruz. Resource tage koyduğumuz isimleri gördük. 1 ve 2 olarak kategorize edelim:



User Data nın içeriği:



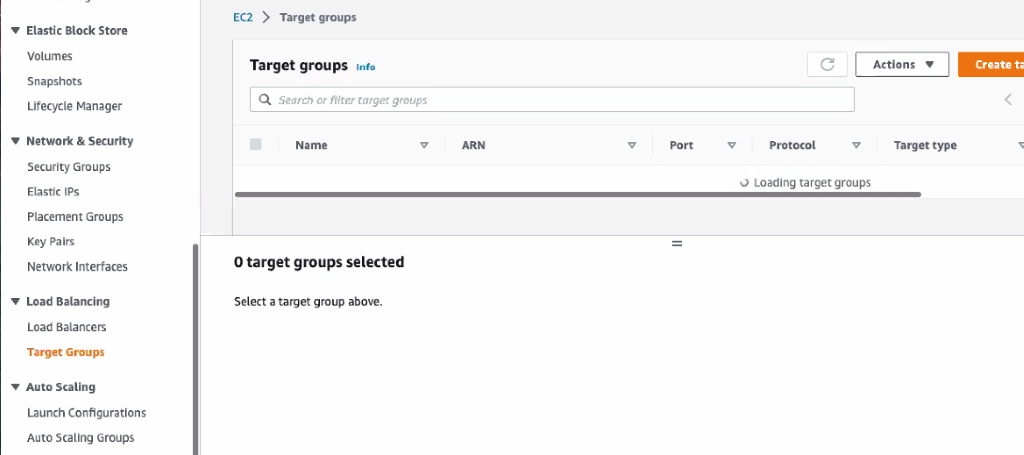
126 da metadata yazıyor. AWS bize bir çok metadata yı çekebilme imkanı sunuyor. Dokümantasyon da var.

Burada iki tane EC2 ayağa kalkacak ve bunların public ve private IP leri görülecek. Metadata bu bilgileri çekmiş olacak.

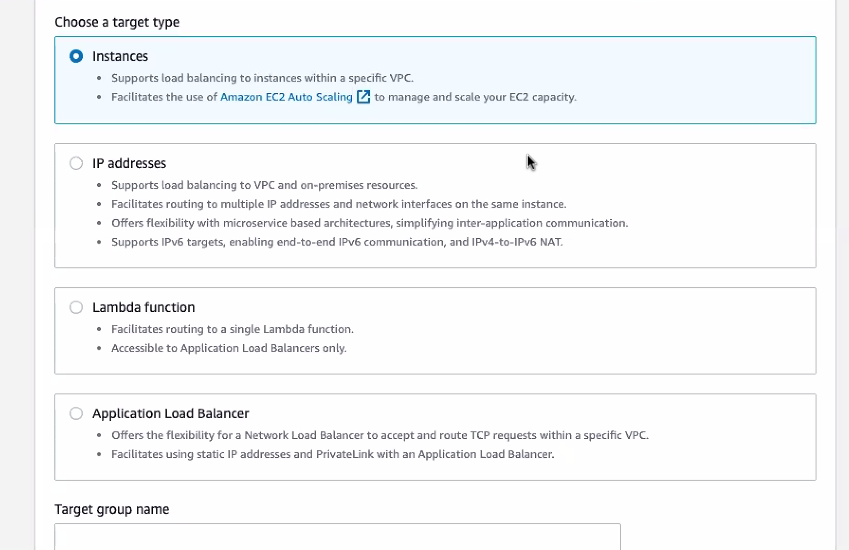
<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-metadata.html>

Biz bu IP bilgileri ile LB nin çalışıp çalışmadığını anlayacaz.

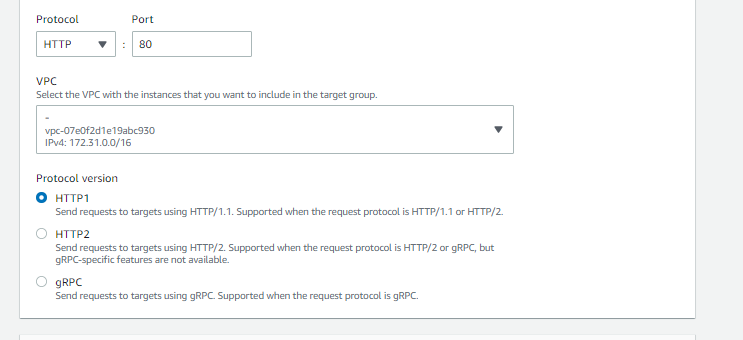
Şimdi target group oluşturacaz. LB nin yöneteceği instance lar lazım. Burada iki instance var ve iki instace lık group oluşturacaz.



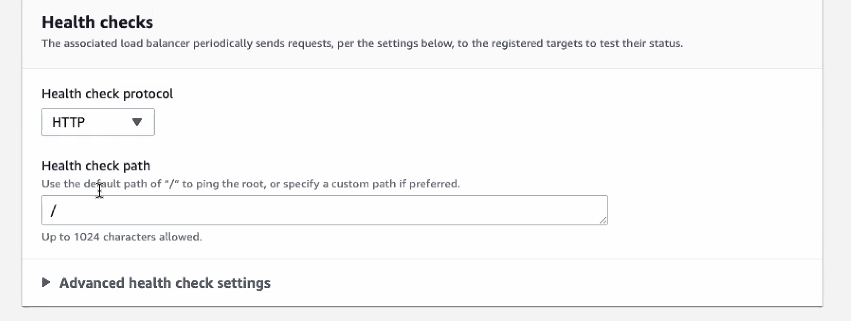
Sol menüde Load Balancing altından Target Group a geliyoruz. Ve Create diyoruz.



Instance lar, IP addressler ,Lamda function ya da App LB hedef kitle olabilir. İsim verdik.

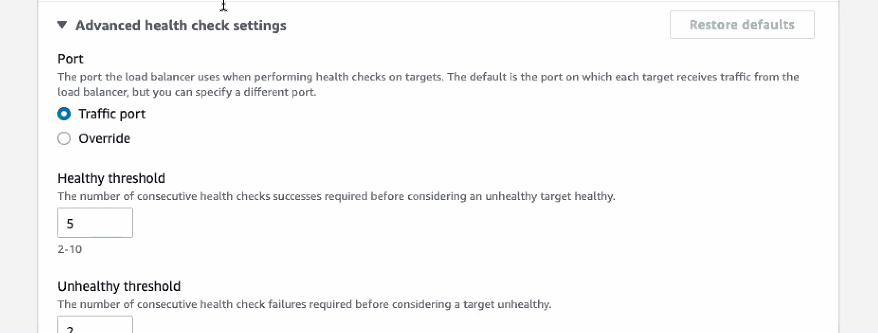


HTTP1 bir requeste bir cevap alıyor. HTTP2 bir requeste birden fazla cevap alıyor. GRPC google un ürettiği bir protokol. Bu bilgiler aynı kalacak.

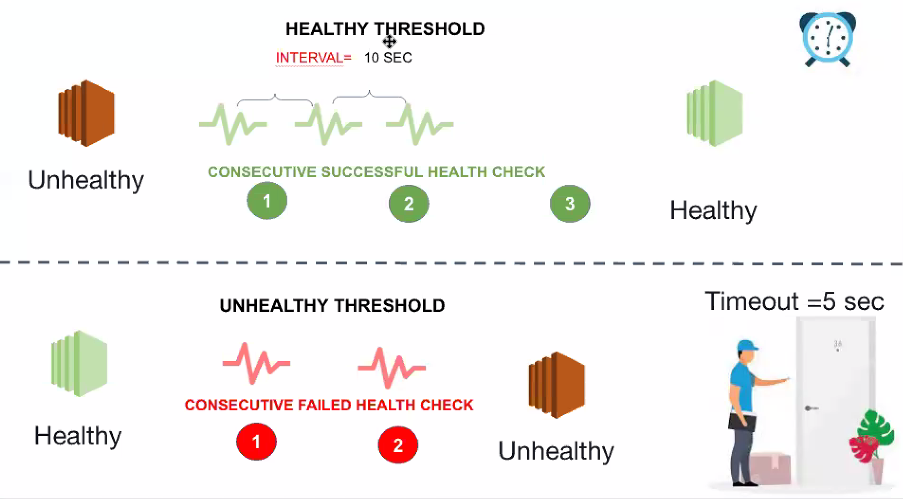


Health check önemli. Burada HTTP 80 portunda ve root sayfasında health check yap diyoruz (ana sayfada)

Advanced HC ye basıyoruz.



Hangi portta bu kontrolü yapsın? Traffic port umuz 80, override diyerek başka port girebiliriz.



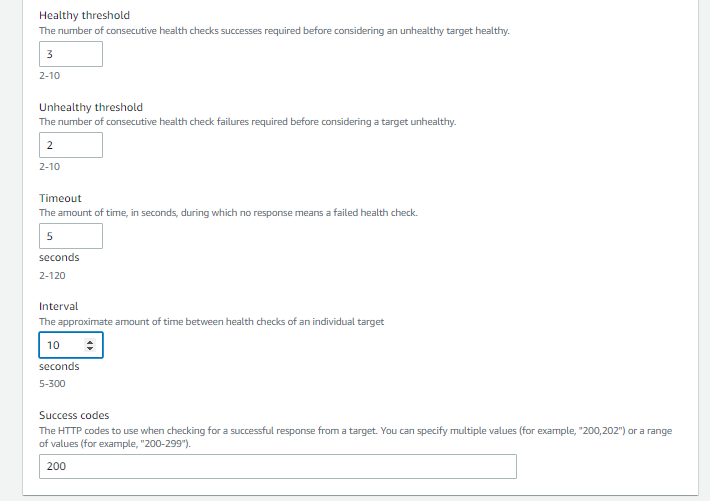
Healthy Threshold, unhealthy bir instance ı healthy ilan etmemiz için ardışık olarak (n) tane sağlıklı check geçirmiş olsun.

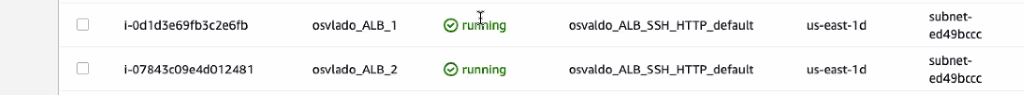
Interval, her check arasında (n) saniye bekle

Unhealthy threshold; ardışık olarak (n) tane sağlıksız check gelirse bunu unhealthy ilan et diyoruz. Yine arada aynı süre bekler.

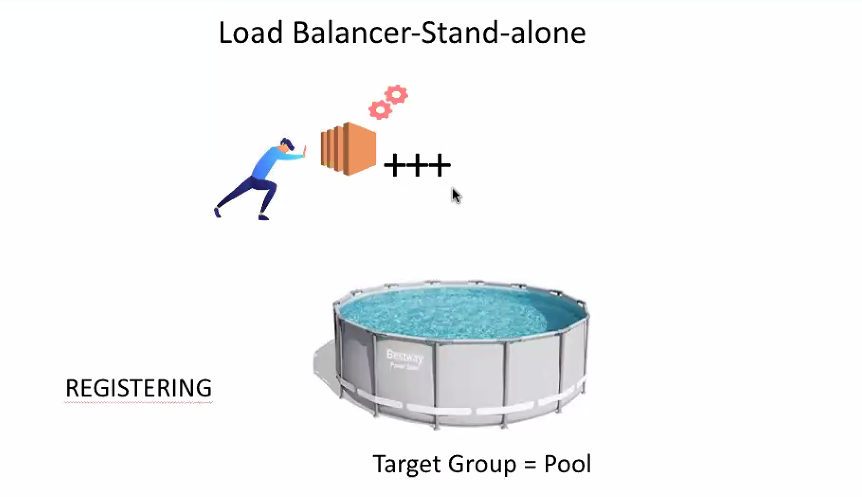
Timout ise (n) saniye süresince instance tan bir response (cevap) alamazsam bu health check failed olur.

Succes code u 200 dedik. Başarı kodu.

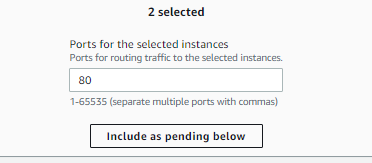


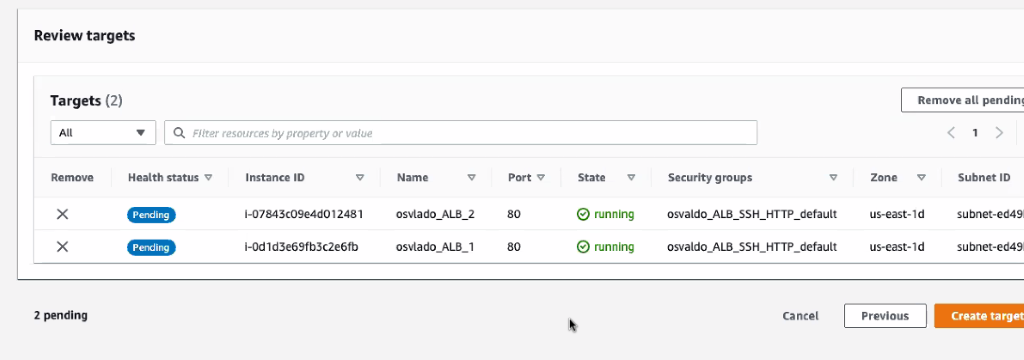


Target group oluştururken hangi instance ları havuzun içine koyacaz:

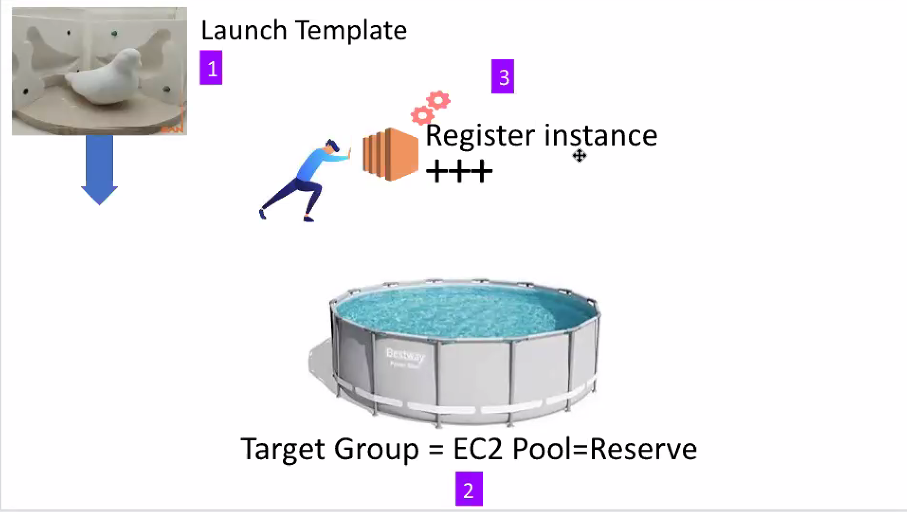


Seçip İnclude as pending below diyoruz:

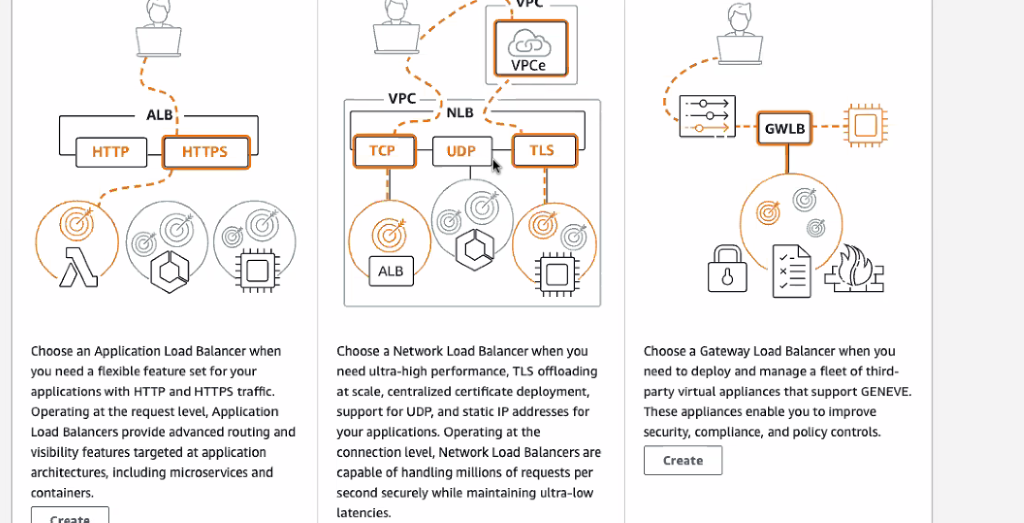




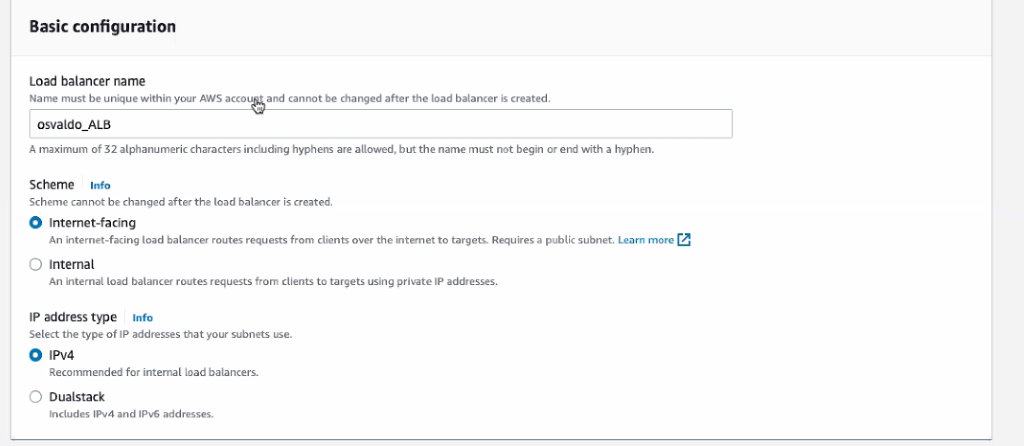
Create target group diyoruz.



Şimdi sol menüden Load Balancer a geçiyoruz ve create Load Balance diyoruz:

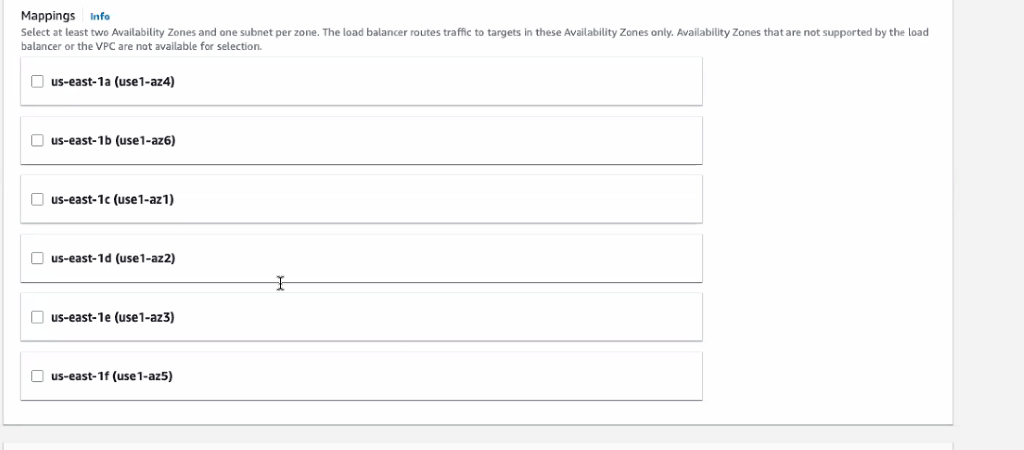


Biz App LB yi seçiyoruz. İsim yazıyoruz:



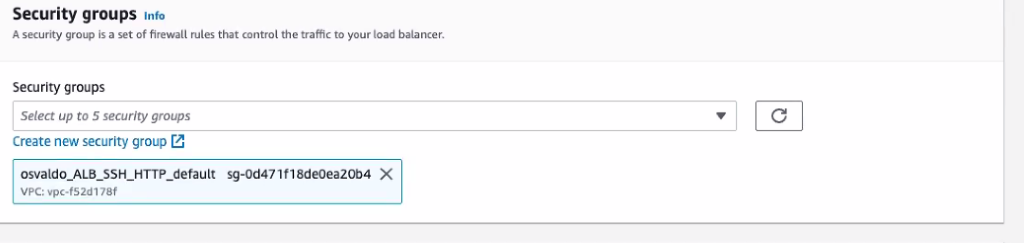
Internet-facing public internetten gelen requestlerdir. Internal da arka planda içerideki aplikasyonun da loadu ayarlanabilir. (bankacıların kendi kullandığı programlar diyebiliriz.)

Dualstack dersek IPv6 yı da işin içine katabiliriz.

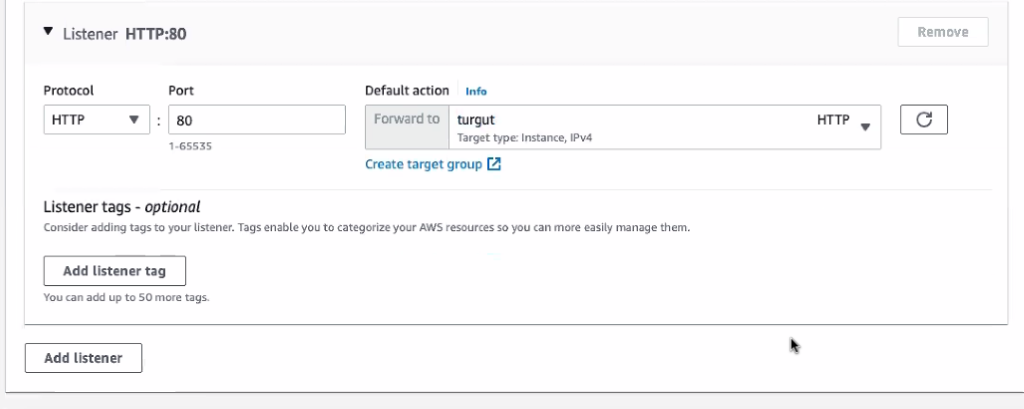


Mappings de LB hangi AZ lerde hizmet versin. Best practice olarak hepsi seçilir. Çünkü instance ın hangi AZ de olduğunu bilmeyebiliriz. Tıkladıkça subnet açılır onlara karışmıyoruz.

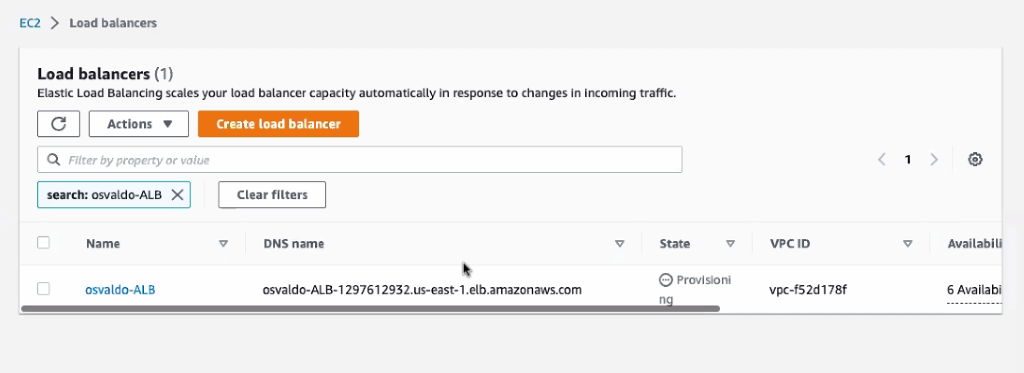
Default SG yi iptal edip oluşturduğumuz SG yi seçiyoruz:



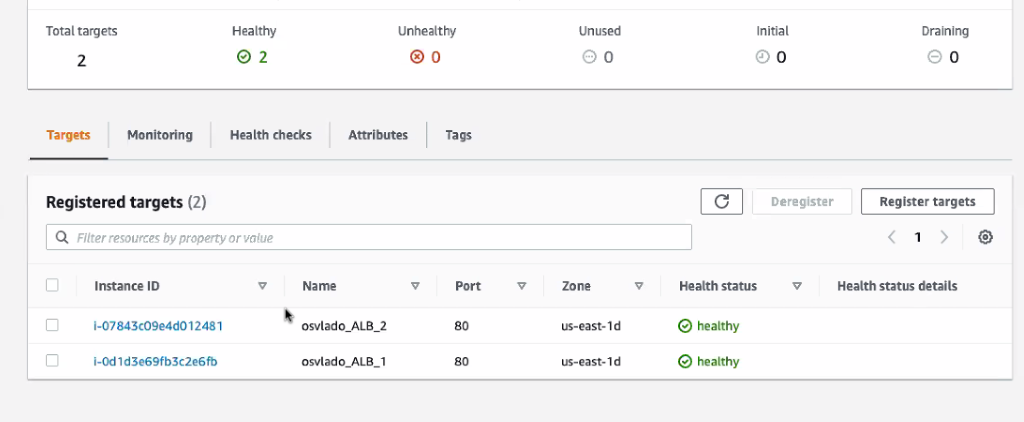
Target group u seçiyoruz:

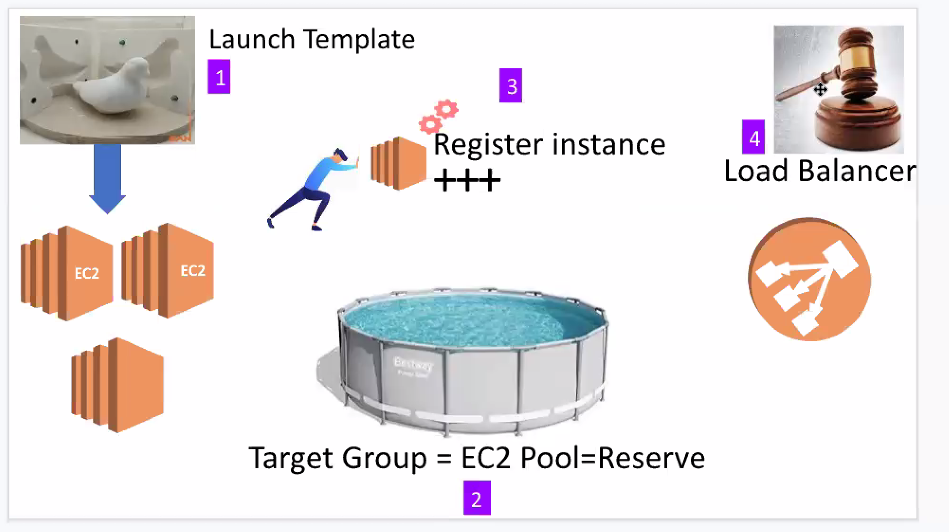


Add listener diyerek başka target grouplar da ekeleyebiliriz. Create diyoruz. LB miz üretiliyor:

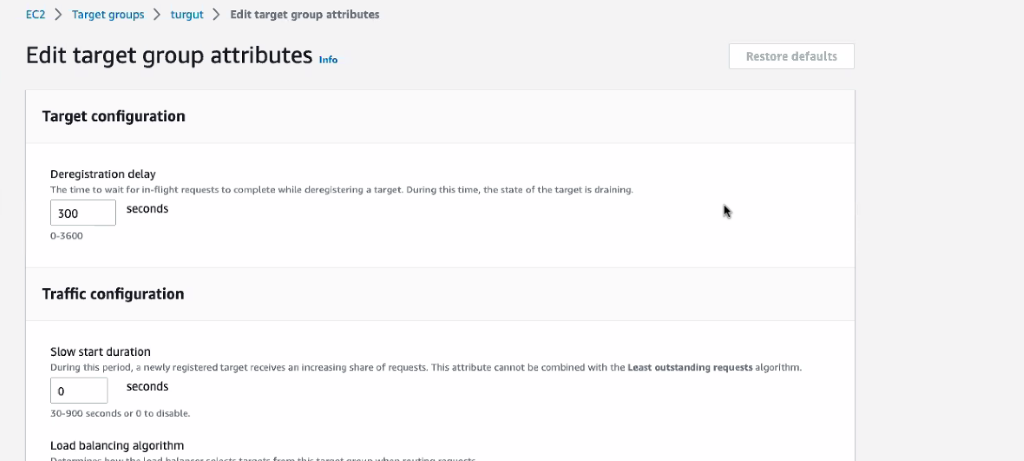


LB mizi kurduk:





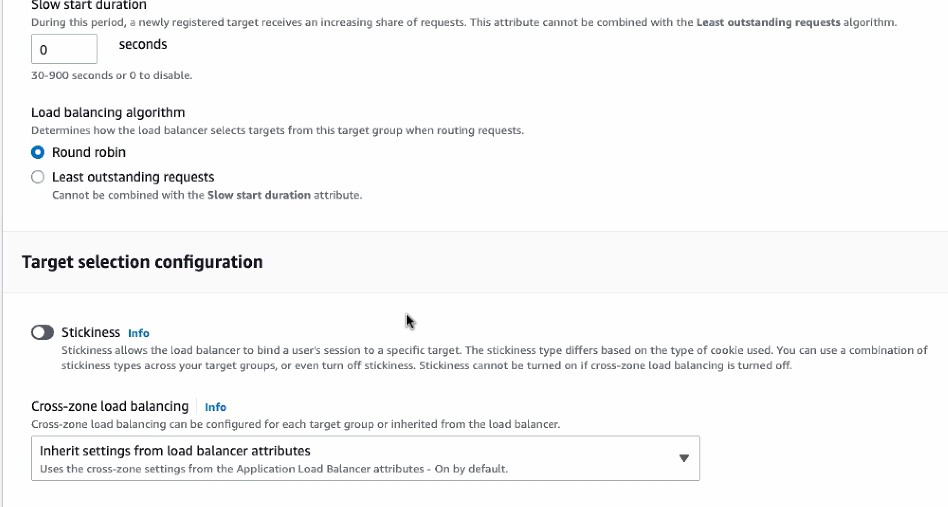
Target group a gelip Attributes a basıyoruz:



Slow start duration yükü yavaş yavaş vermek için kullanılır.

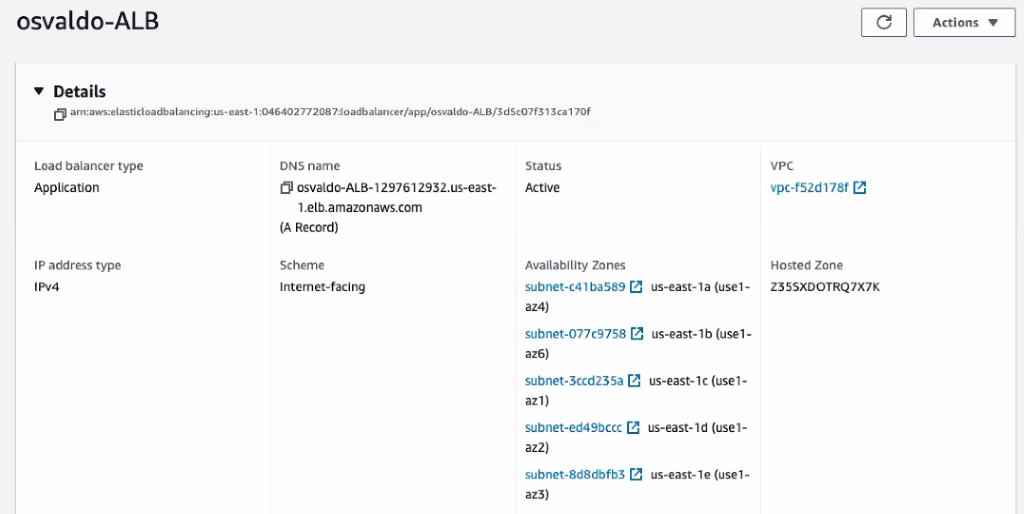
Round robin sırasıyla instance lara yükü dğıtır.

Least outstanding requests: yükün ağırlığına göre dağıtır.

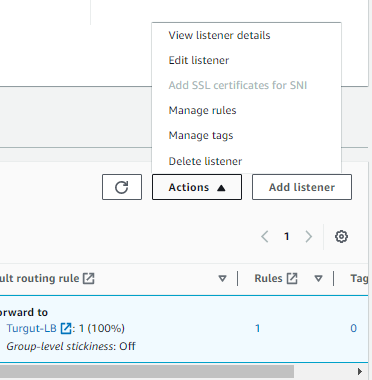


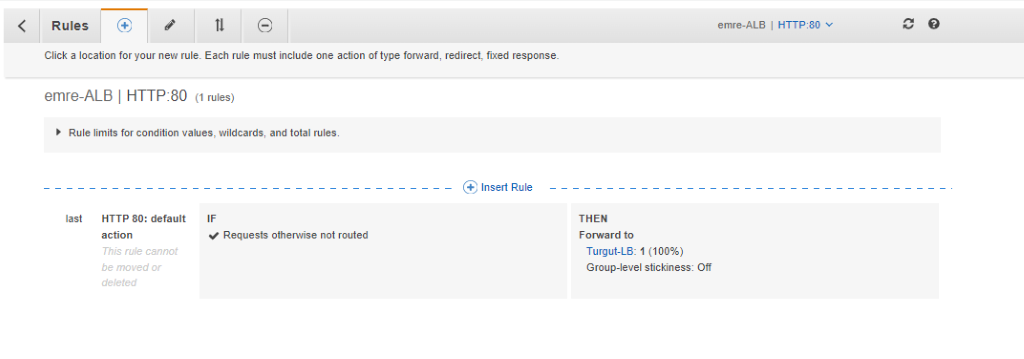
Stickiness; bankacılık veya e develtte randevu alırken işlemin sürekliliği gerektiğinde bir faaliyeti tek bir server yapsın load balance değiştirmesin. Bunu stickiness sağlar.

LB miz aktif:



LB yi seçip actions dan manage rules diyerek rule ekeleyebiliyoruz:





Önce load balancer ı silecez.

Sonra instance ları temrinate edecez.

Target group kullanılmadıkça sorun olmaz ama karışıklık olmaması için silinebilir.

LAunch template de aynı şekilde.