NET304

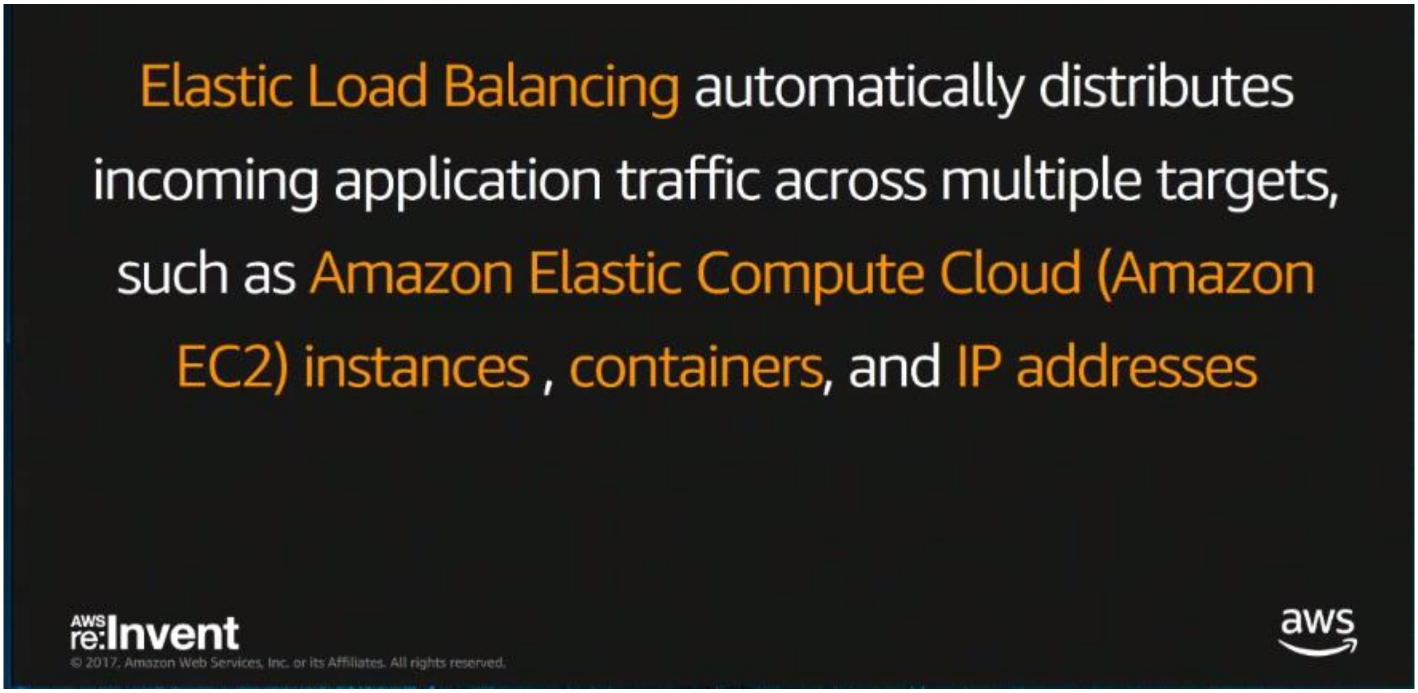
AWS re:INVENT

Deep Dive into the New Network Load Balancer

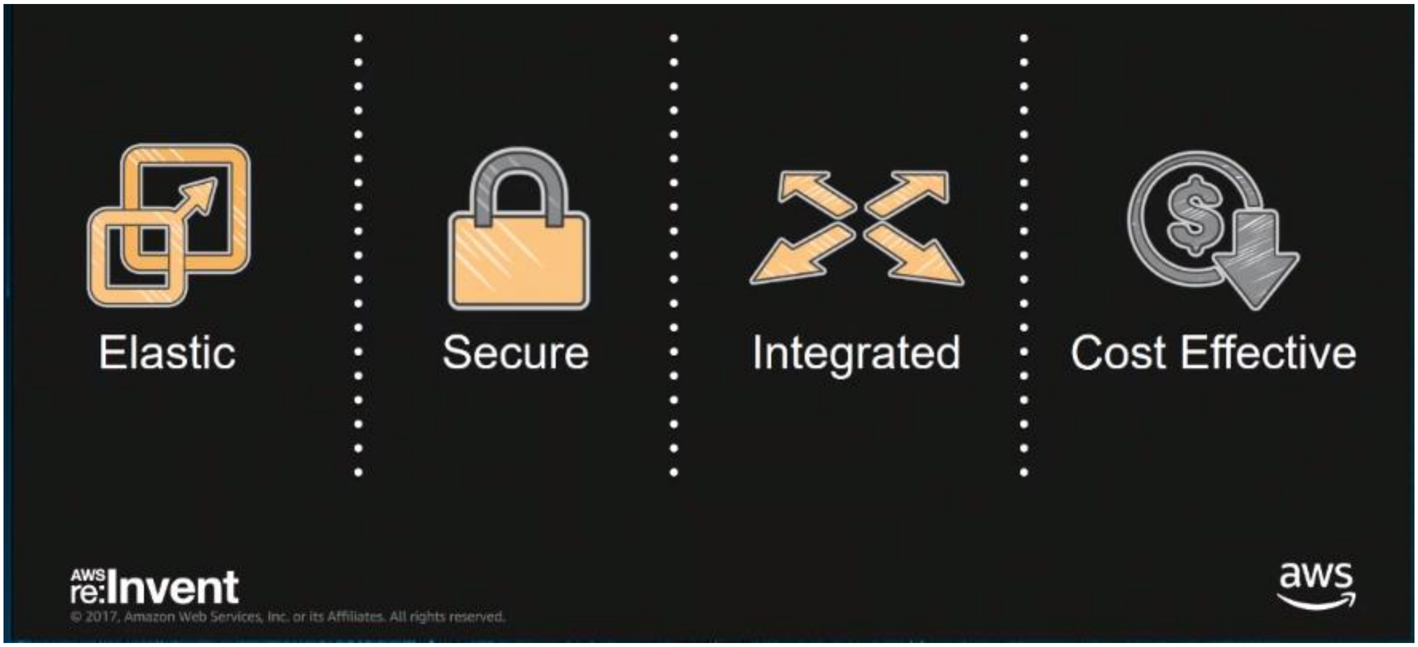
Pratibha Suryadevara, Narayan Subramaniam, Bryan McKenney(Loggly)

November 28, 2017

In this session, we explore the new Network Load Balancer that was launched as part of the Elastic Load Balancing service, which can load balance any kind of TCP traffic. This offers customers a high-performance, scalable, low-cost load balancer that can handle millions of requests per second with very low latencies, while maintaining high levels of performance. Come and learn more about this new Network Load Balancer.



Elastic Load Balancing automatically distributes incoming application traffic across multiple targets, such as Amazon Elastic Compute Cloud (Amazon EC2) instances , containers, and IP addresses



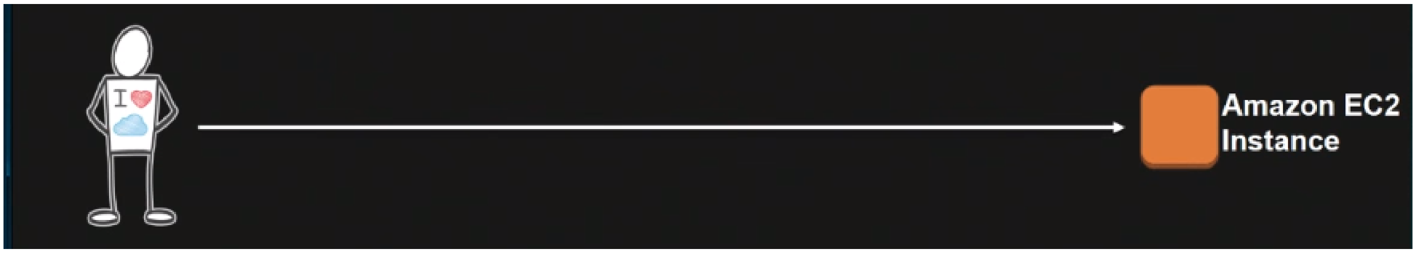
Elastic

Secure

Integrated

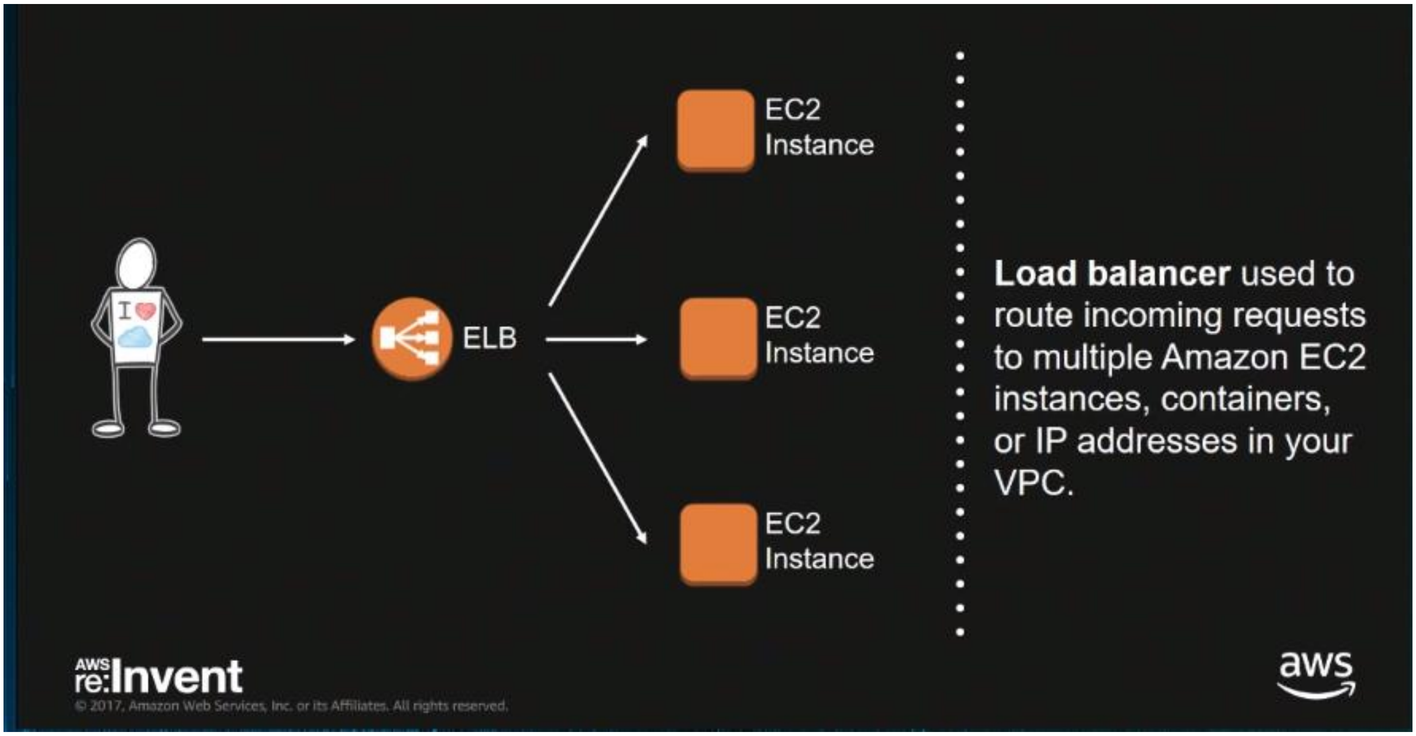
Cost Effective

These are the advantages of the ELB, you simply add the targets to the ELB and you get all the above benefits. ELB uses best in class ciphers and protocols to provide security to your LB and targets. ELB is also integrated with several AWS services like ECS, EKS, CF, Route53, etc



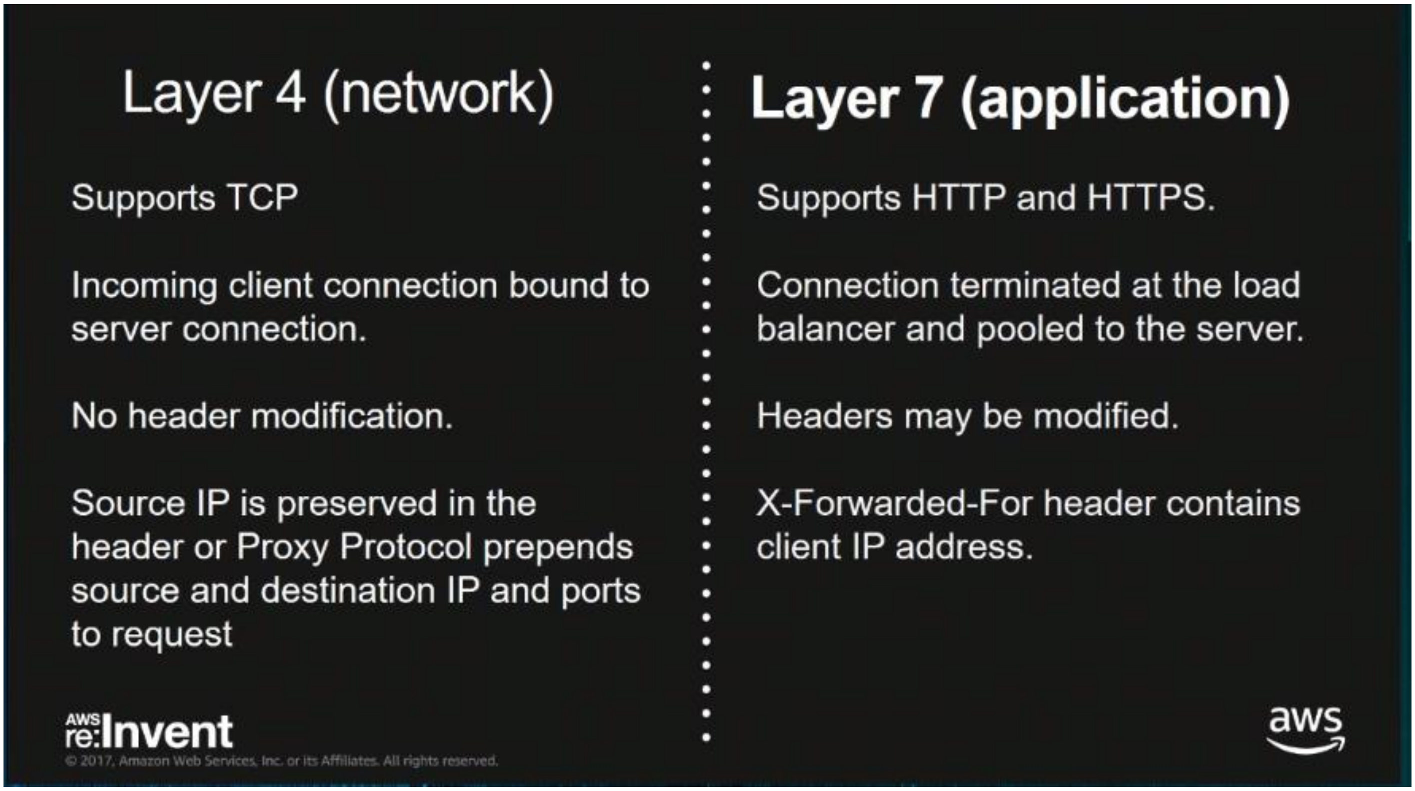
Amazon EC2 Instance

This instance doesn't scale for instant spike in traffic and is not secure



Load balancer used to route incoming requests to multiple Amazon EC2 instances, containers, or IP addresses in your VPC.

This is what is recommended when deploying an application in EC2



Layer 4 (network)

Supports TCP

Incoming client connection bound to server connection.

No header modification.

Source IP is preserved in the header or Proxy Protocol prepends source and destination IP and ports to request

Layer 7 (application)

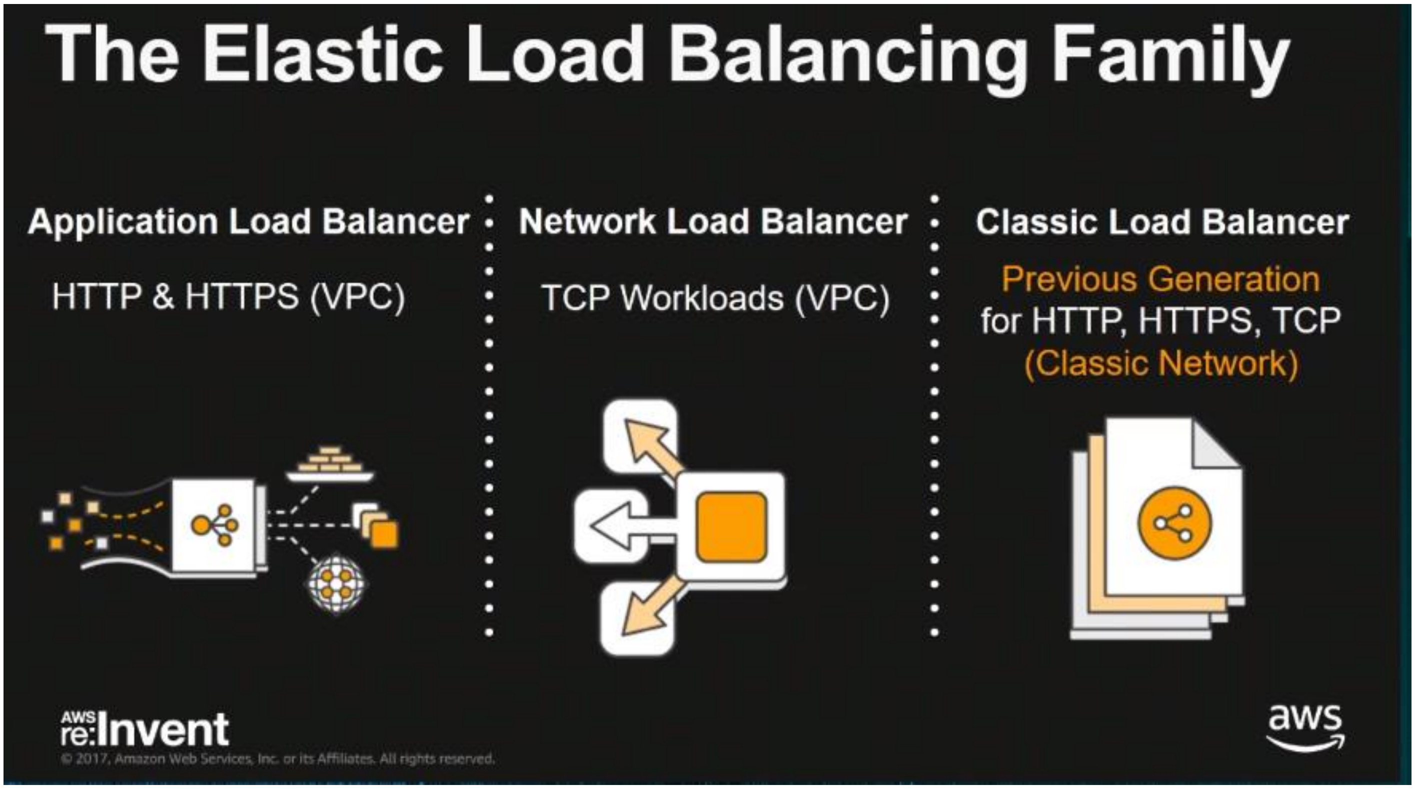
Supports HTTP and HTTPS.

Connection terminated at the load balancer and pooled to the server.

Headers may be modified.

X-Forwarded-For header contains client IP address.

These are the 2 types of LB available, it depends on what layer the LB runs in the OS stack



The Elastic Load Balancing Family

Application Load Balancer

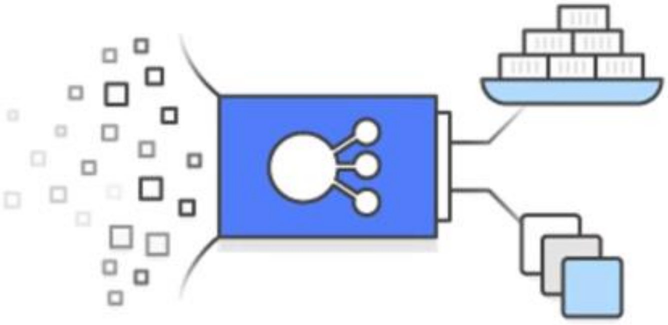
HTTP & HTTPS (VPC)

Network Load Balancer

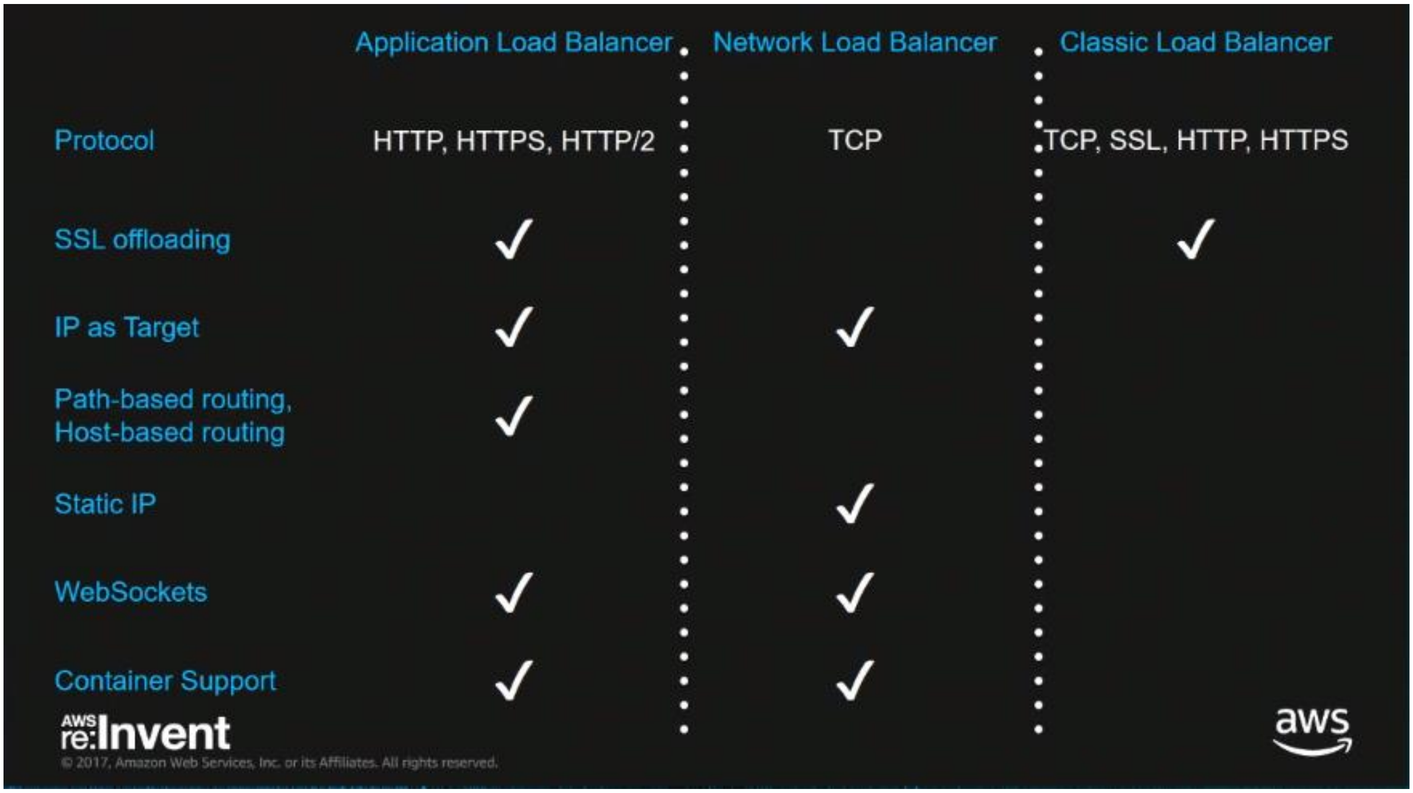
TCP Workloads (VPC)

Classic Load Balancer

Previous Generation
for HTTP, HTTPS, TCP
(Classic Network)

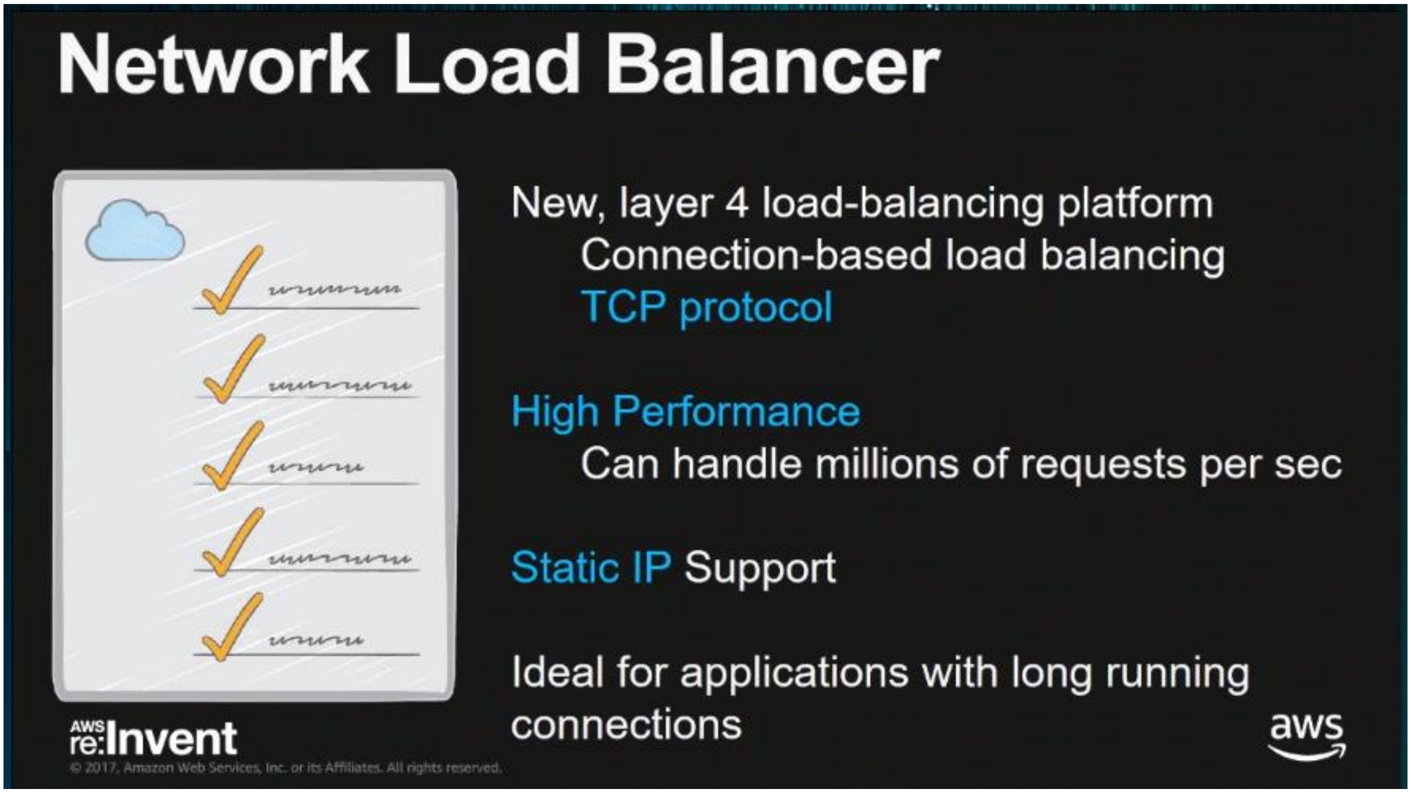


Network Load Balancer (NLB)



	Application Load Balancer	Network Load Balancer	Classic Load Balancer
Protocol	HTTP, HTTPS, HTTP/2	TCP	TCP, SSL, HTTP, HTTPS
SSL offloading	✓		✓
IP as Target	✓	✓	
Path-based routing, Host-based routing	✓		
Static IP		✓	
WebSockets	✓	✓	
Container Support	✓	✓	

Static IP means a single IP per AZ even in volatile traffic, it can be used for whitelisting valid IPs that we want to allow communication with.



Network Load Balancer

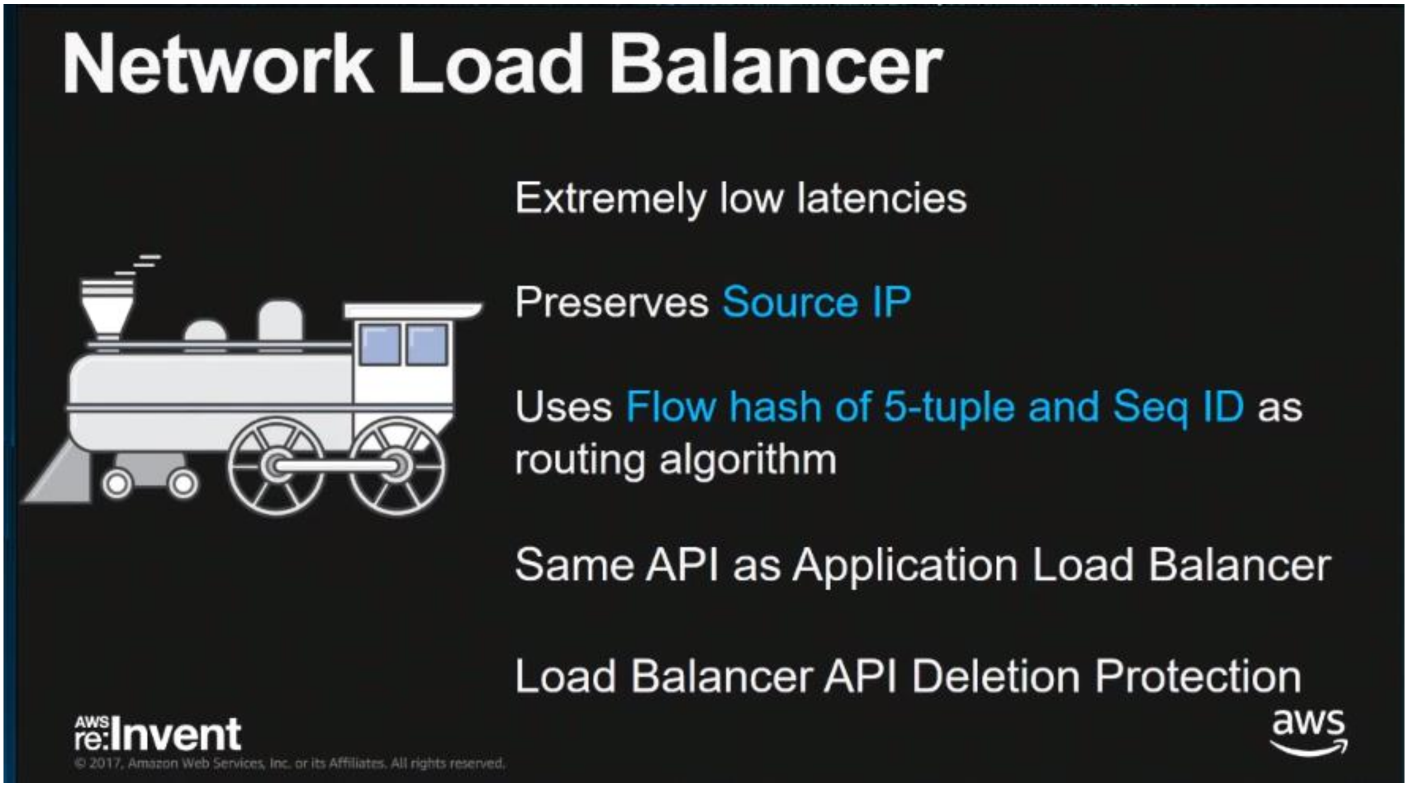
New, layer 4 load-balancing platform
- Connection-based load balancing
- TCP protocol

High Performance
- Can handle millions of requests per sec

Static IP Support

Ideal for applications with long running connections

NLB performs well with no cold starting and in very volatile traffic patterns, NLB also supports using static IPs per AZ so that you have a single IP being used as you scale and can whitelist in your firewalls and other devices sitting in front of your LBs.



Network Load Balancer

Extremely low latencies

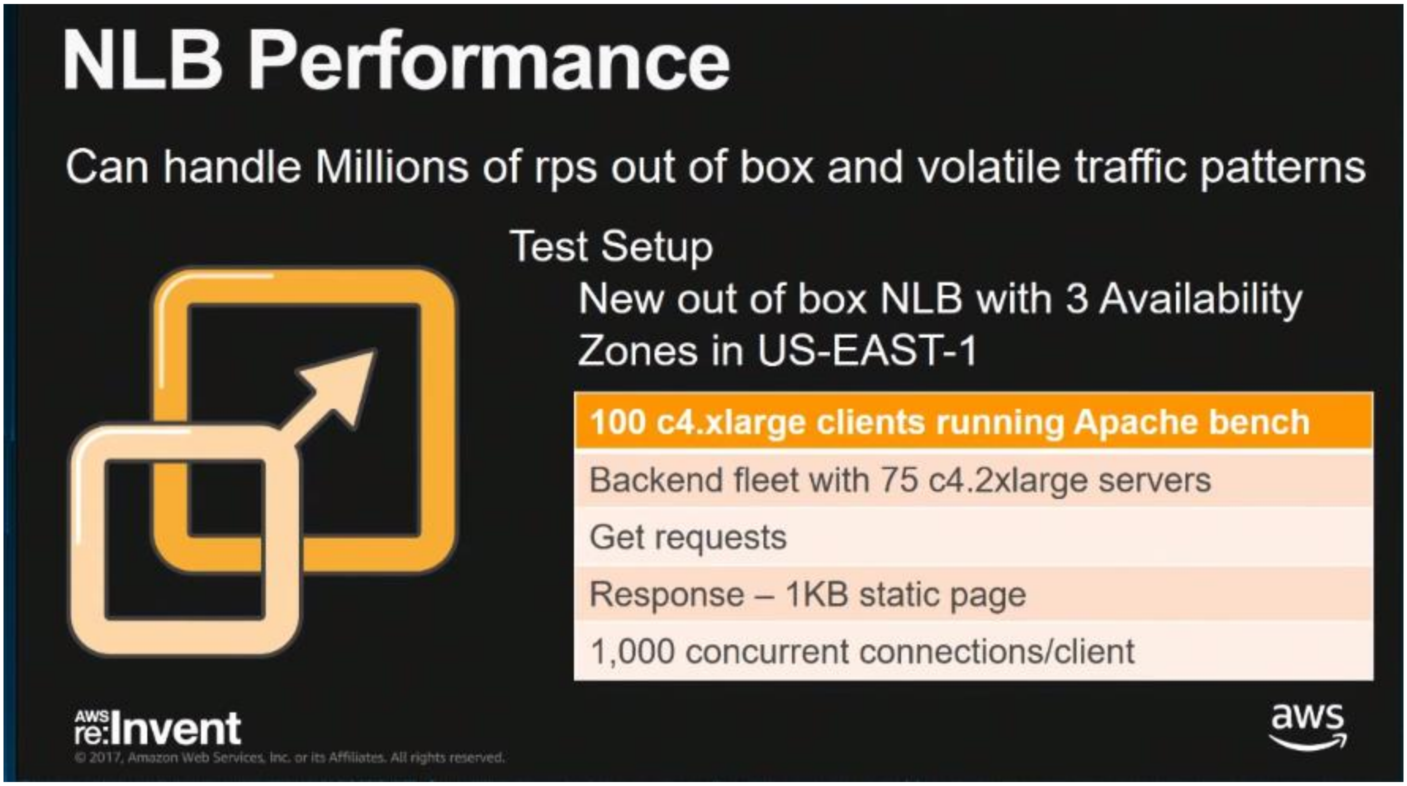
Preserves Source IP

Uses Flow hash of 5-tuple and Seq ID as routing algorithm

Same API as Application Load Balancer

Load Balancer API Deletion Protection

NLB uses the same set of new APIs as the ALB



NLB Performance

Can handle Millions of rps out of box and volatile traffic patterns

Test Setup

New out of box NLB with 3 Availability Zones in US-EAST-1

100 c4.xlarge clients running Apache bench

Backend fleet with 75 c4.2xlarge servers

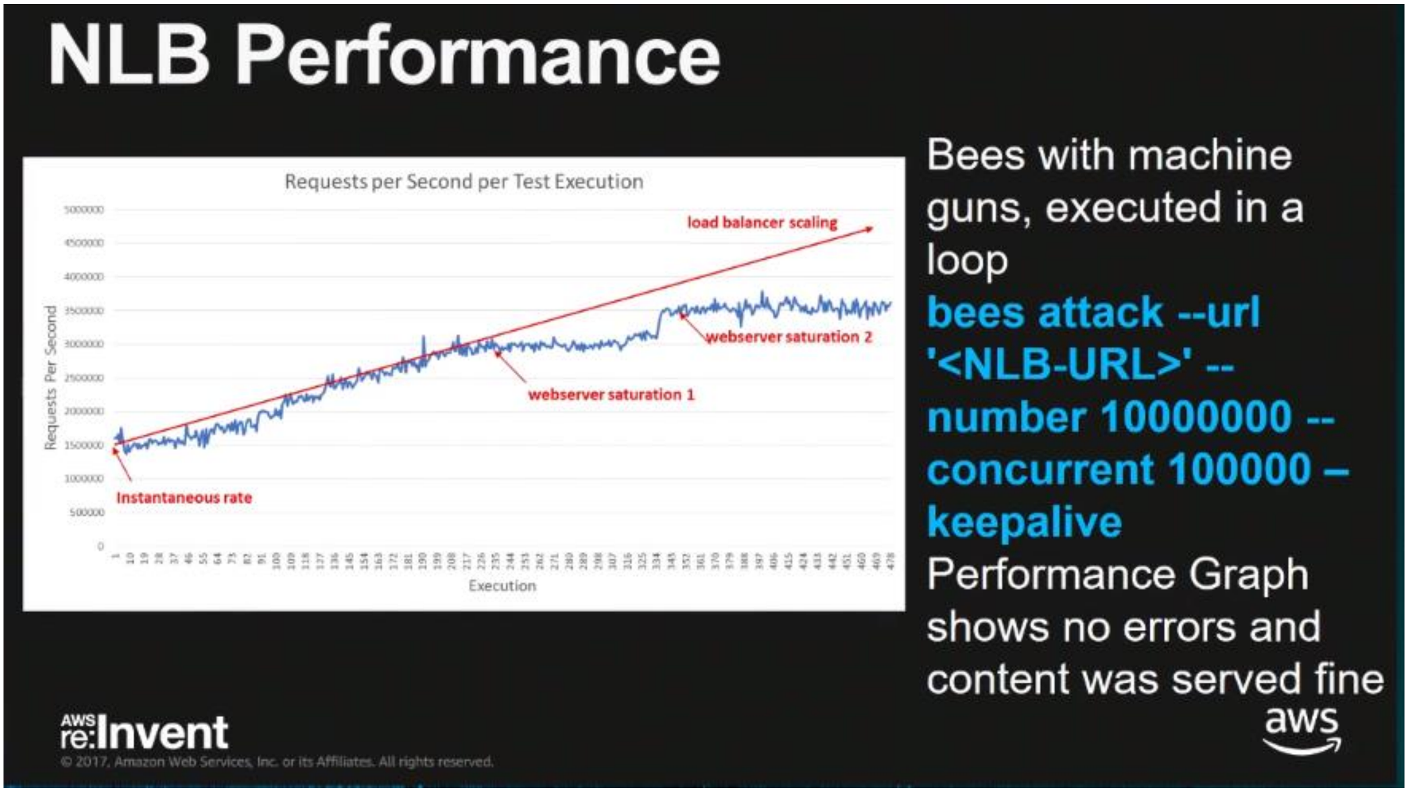
Get requests

Response – 1KB static page

1,000 concurrent connections/client

AWS re:Invent

aws



NLB Performance

Requests per Second per Test Execution

Bees with machine guns, executed in a loop

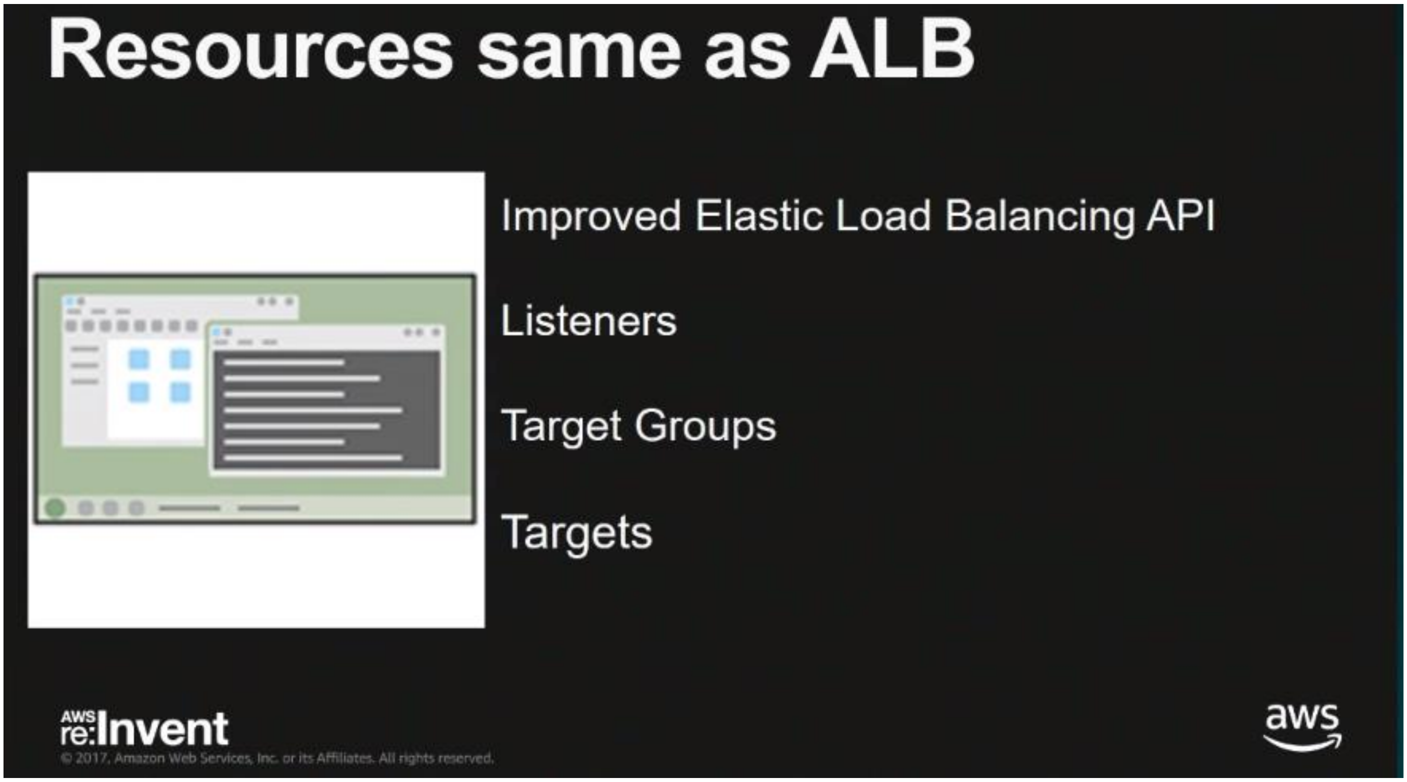
bees attack --url '<NLB-URL>' --number 10000000 --concurrent 100000 – keepalive

Performance Graph shows no errors and content was served fine

AWS re:Invent

aws

The NLB can linearly scale to tens of millions of requests per second



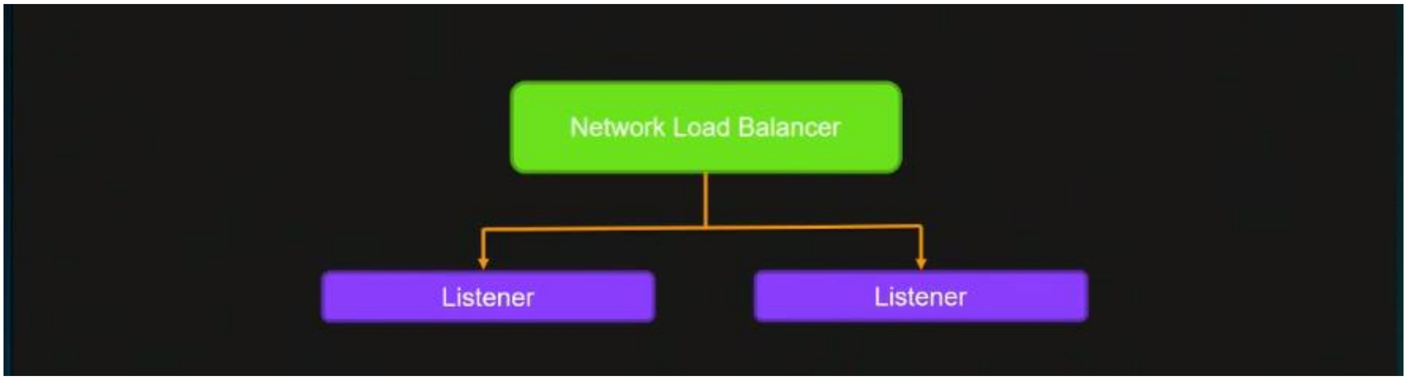
Resources same as ALB

Improved Elastic Load Balancing API

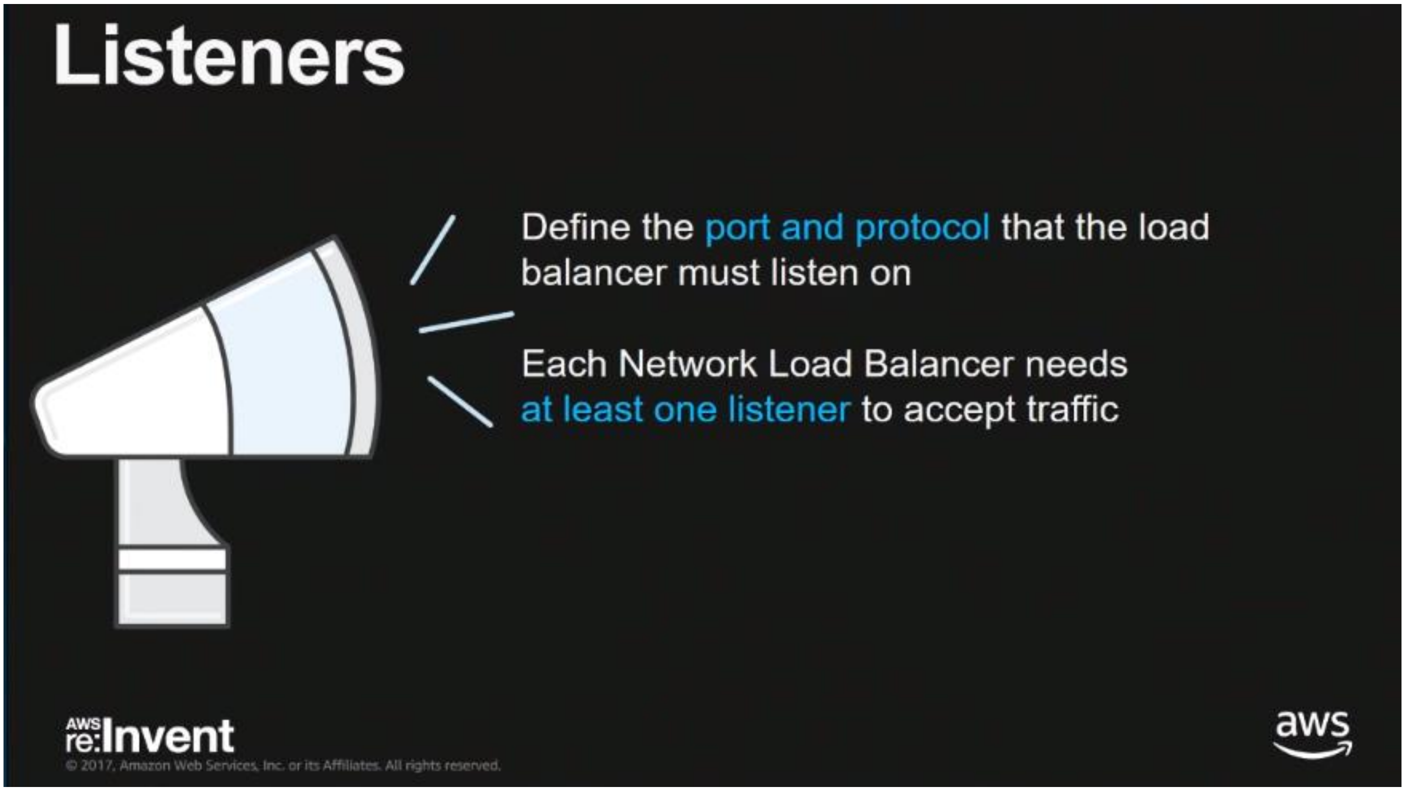
Listeners

Target Groups

Targets



We start by attaching listeners to the NLB as above

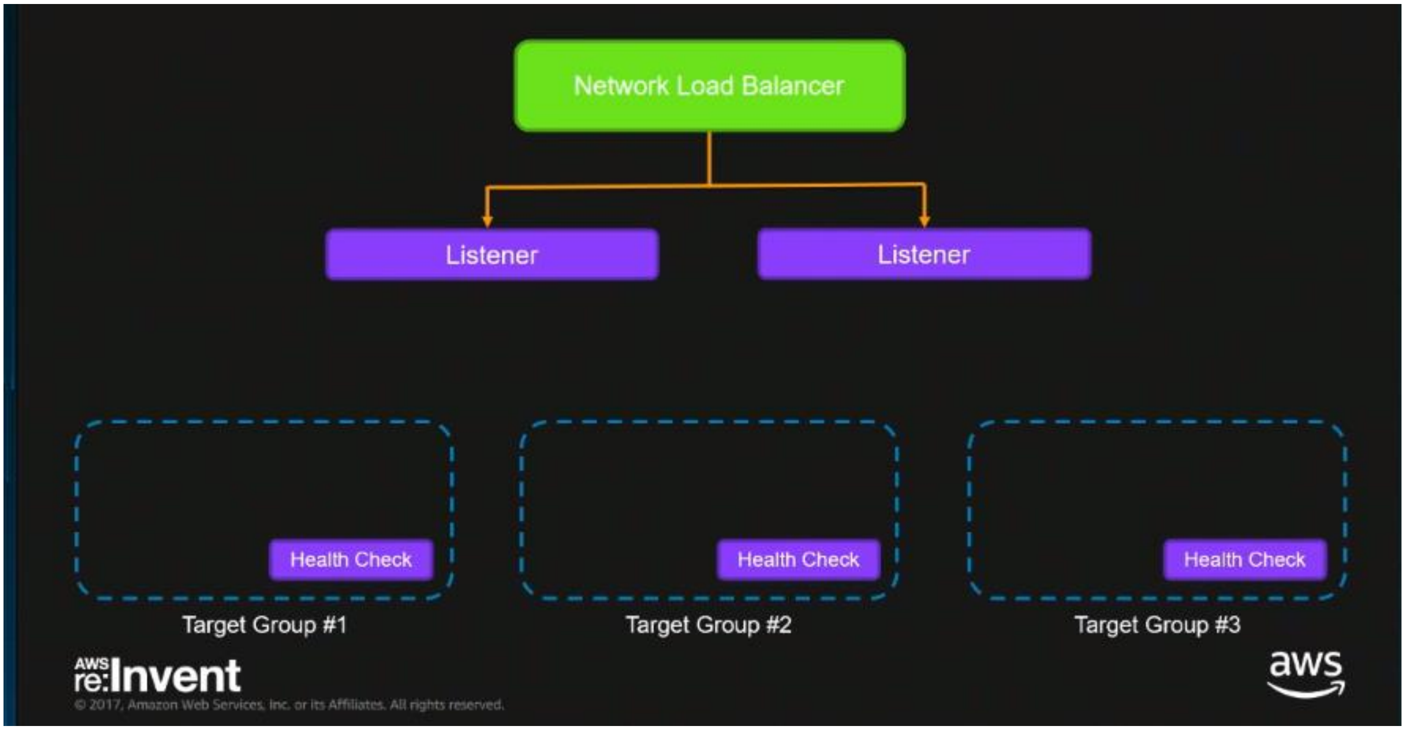


Listeners

Define the port and protocol that the load balancer must listen on

Each Network Load Balancer needs at least one listener to accept traffic

You need to have at least 1 listener before you start sending traffic through



Network Load Balancer

Listener

Listener

Health Check

Target Group #1

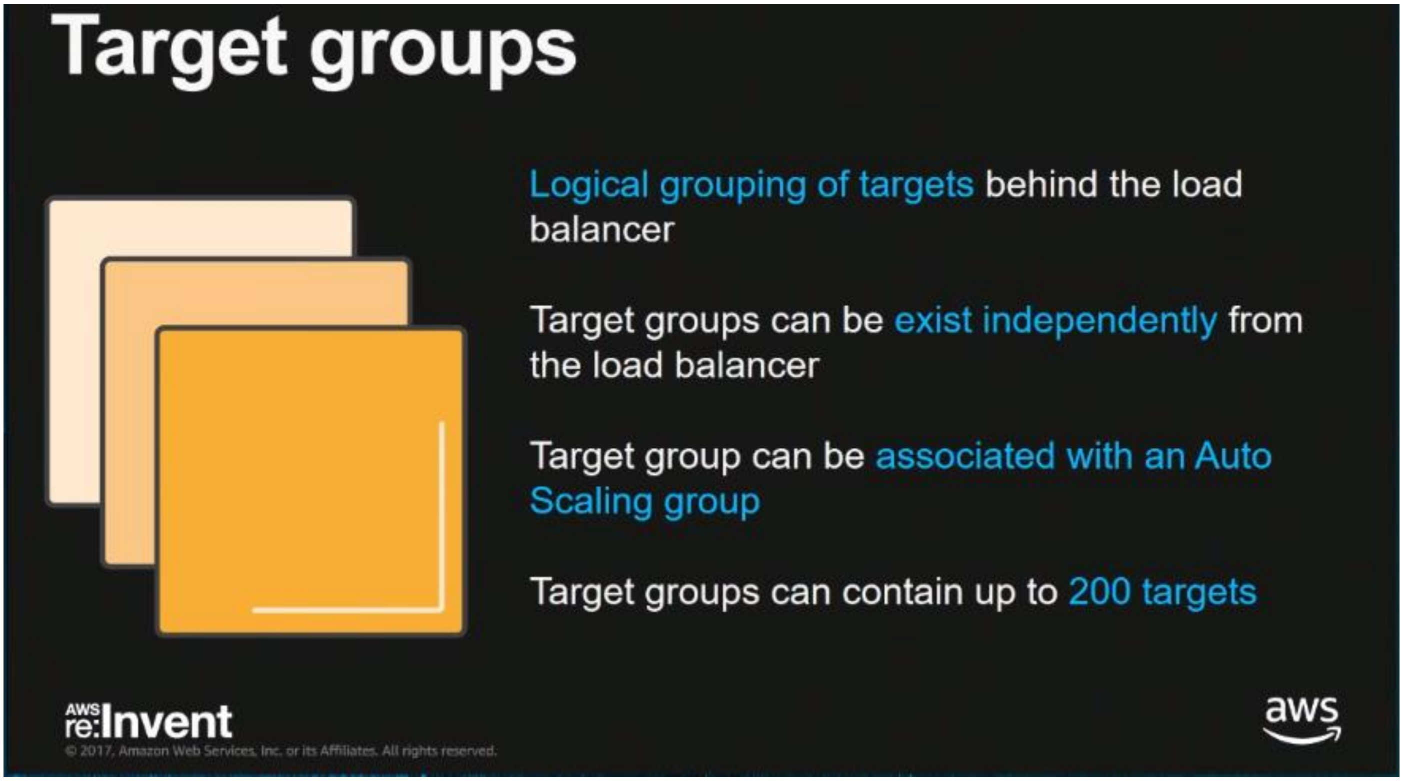
Health Check

Target Group #2

Health Check

Target Group #3

We then create or identify the target groups that we want to have the listener work with



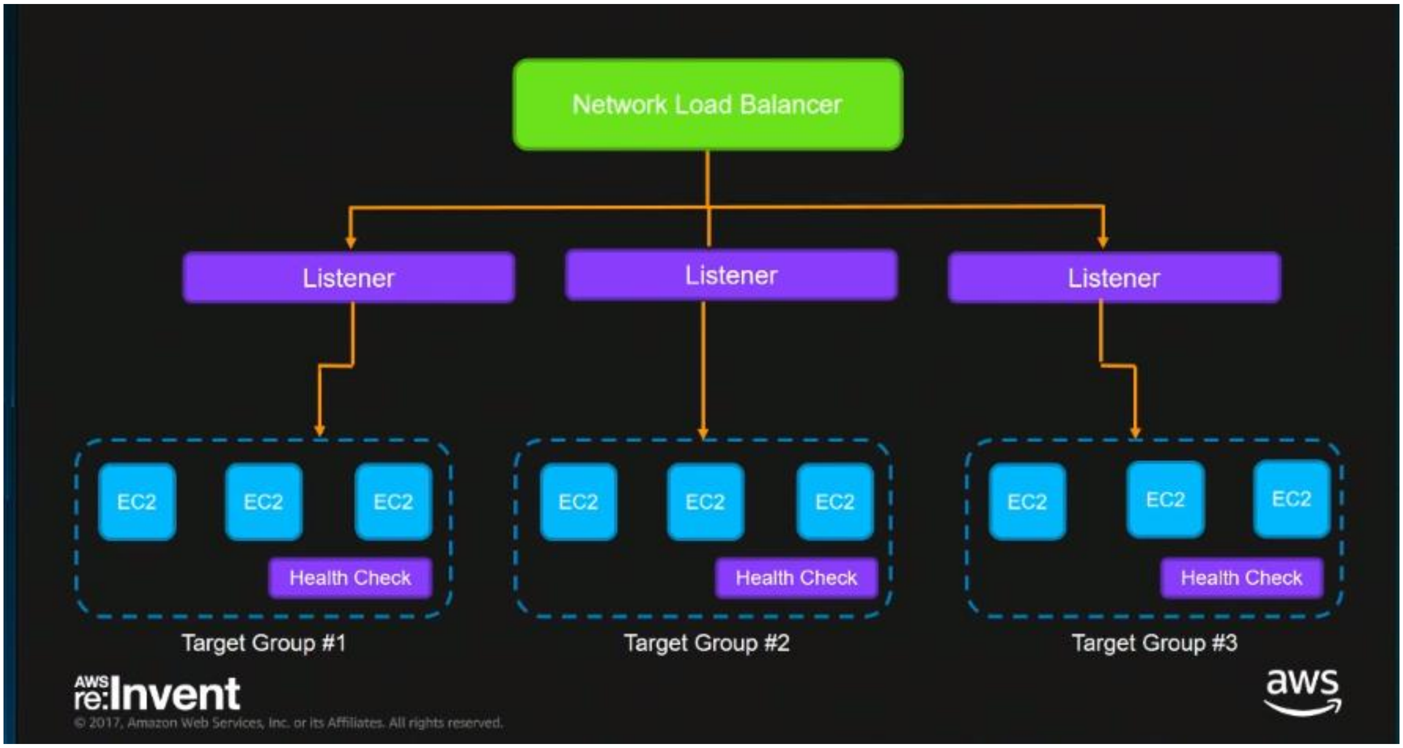
Target groups

Logical grouping of targets behind the load balancer

Target groups can be exist independently from the load balancer

Target group can be associated with an Auto Scaling group

Target groups can contain up to 200 targets



Network Load Balancer

Listener

Listener

Listener

EC2

EC2

EC2

Health Check

Target Group #1

EC2

EC2

EC2

Health Check

Target Group #2

EC2

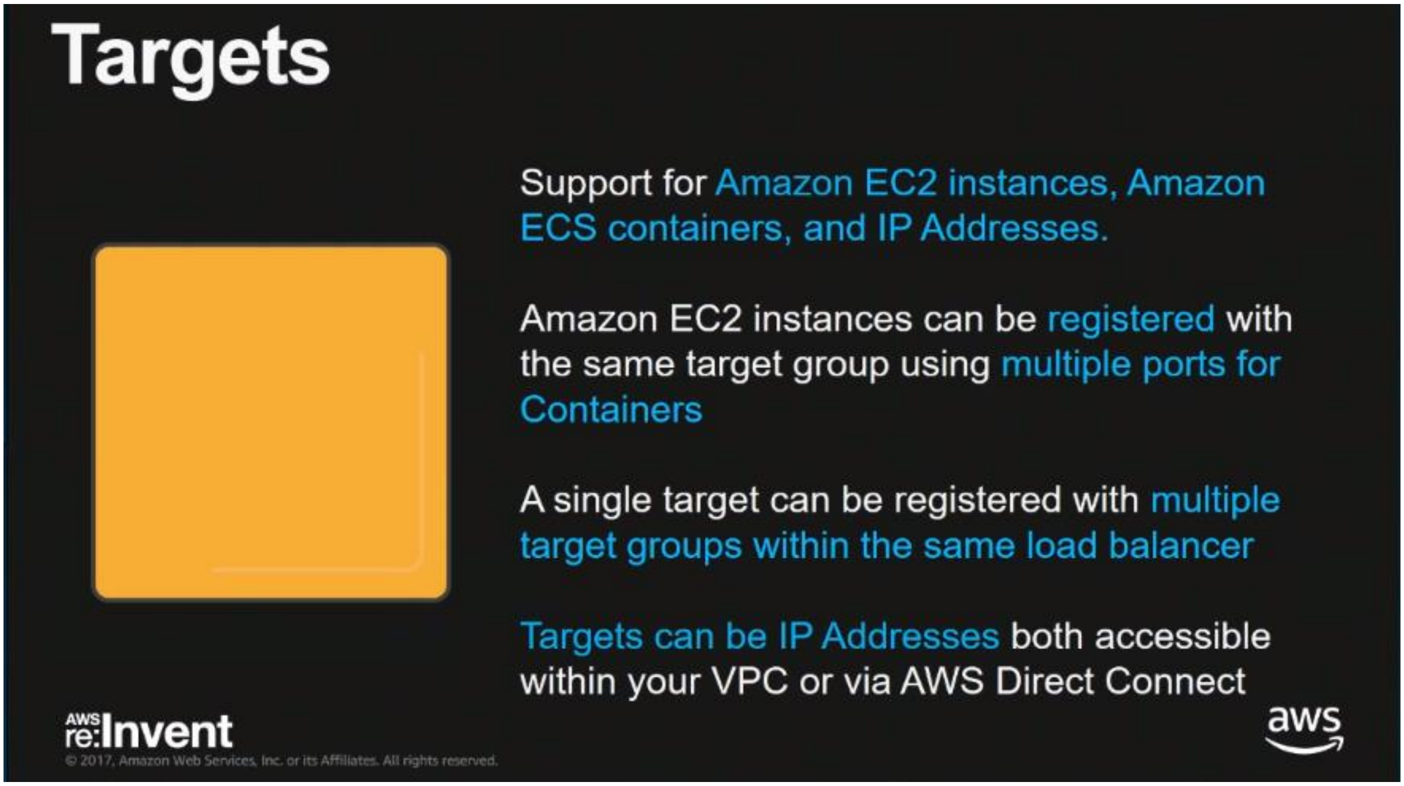
EC2

EC2

Health Check

Target Group #3

We then configure the health checks on the target groups to monitor the targets



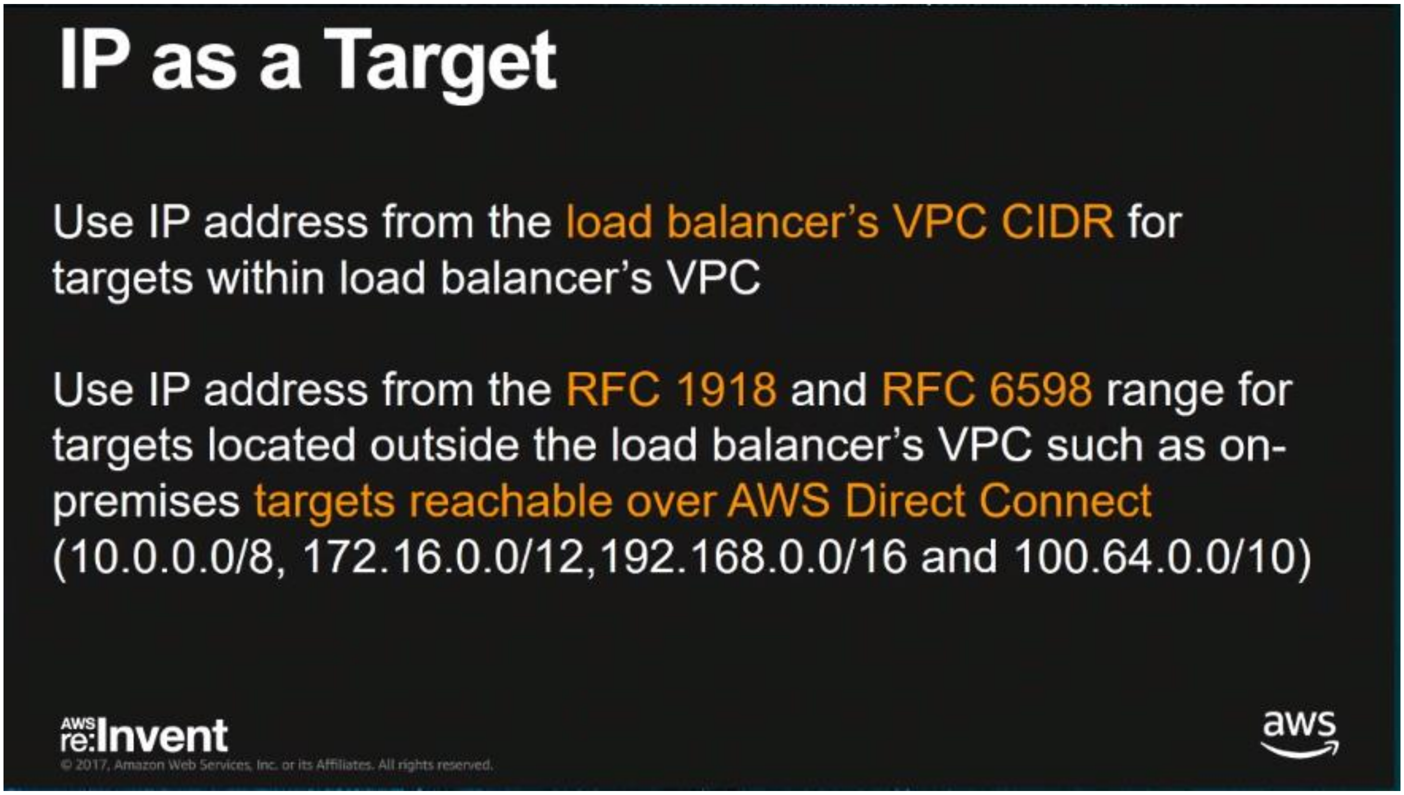
Targets

Support for Amazon EC2 instances, Amazon ECS containers, and IP Addresses.

Amazon EC2 instances can be registered with the same target group using multiple ports for Containers

A single target can be registered with multiple target groups within the same load balancer

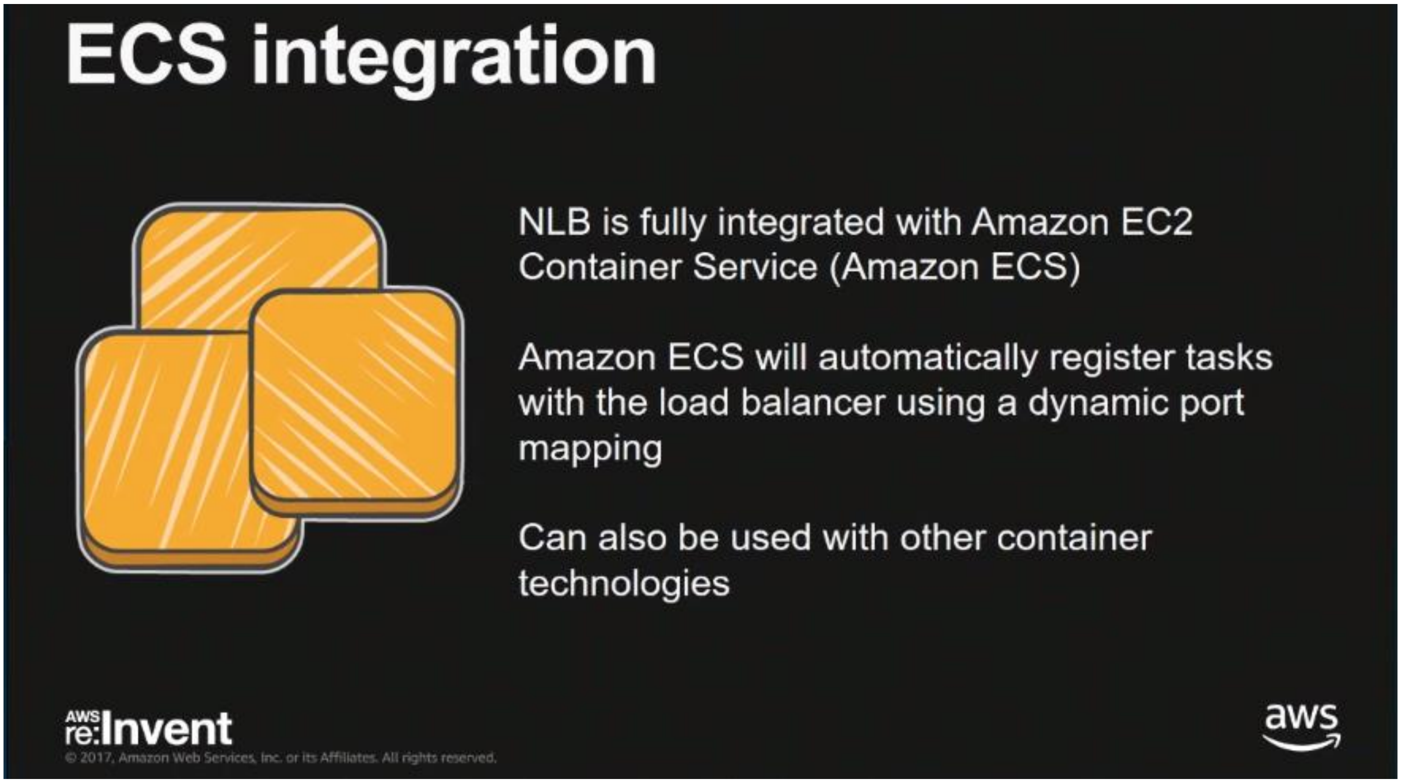
Targets can be IP Addresses both accessible within your VPC or via AWS Direct Connect



IP as a Target

Use IP address from the load balancer's VPC CIDR for targets within load balancer's VPC

Use IP address from the RFC 1918 and RFC 6598 range for targets located outside the load balancer's VPC such as on-premises targets reachable over AWS Direct Connect (10.0.0.0/8, 172.16.0.0/12,192.168.0.0/16 and 100.64.0.0/10)



ECS integration

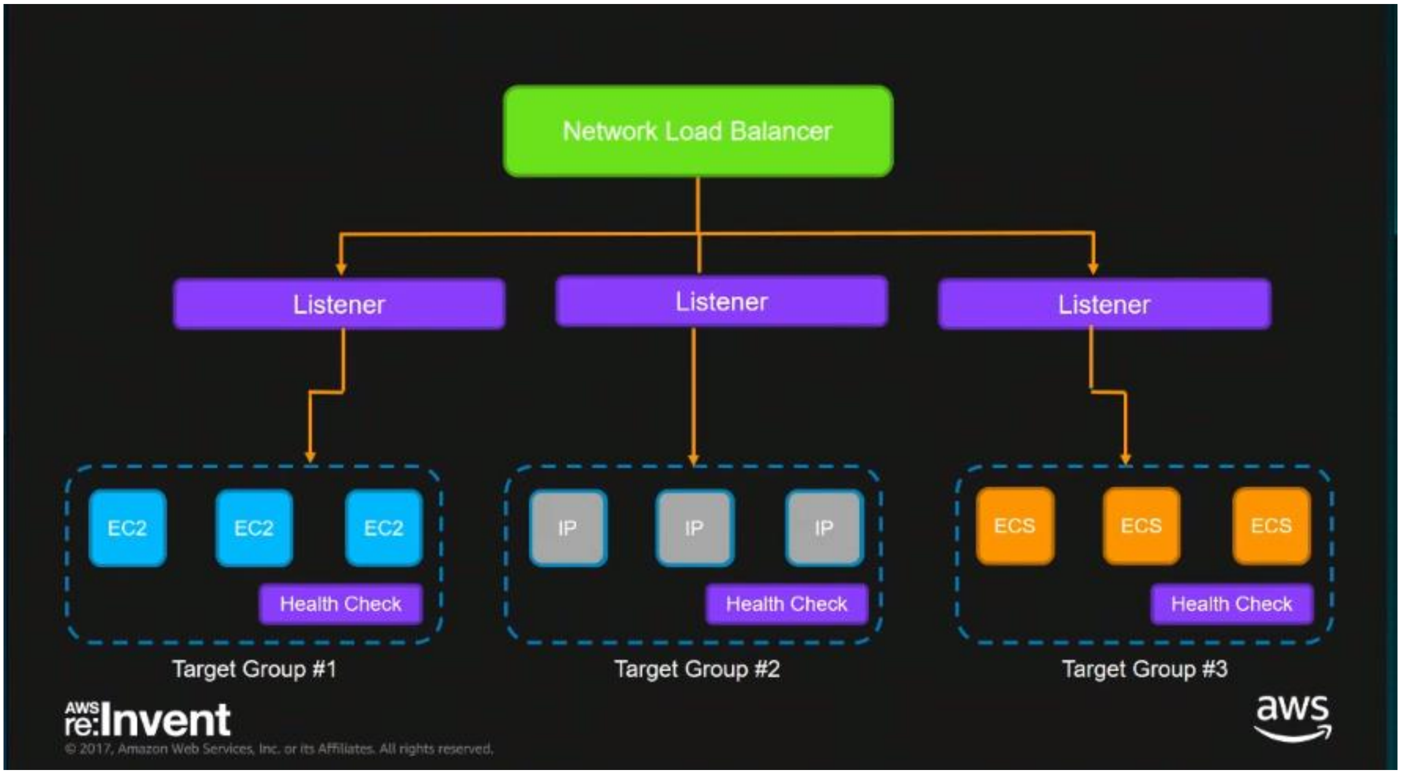
NLB is fully integrated with Amazon EC2 Container Service (Amazon ECS)

Amazon ECS will automatically register tasks with the load balancer using a dynamic port mapping

Can also be used with other container technologies

AWS re:Invent

aws

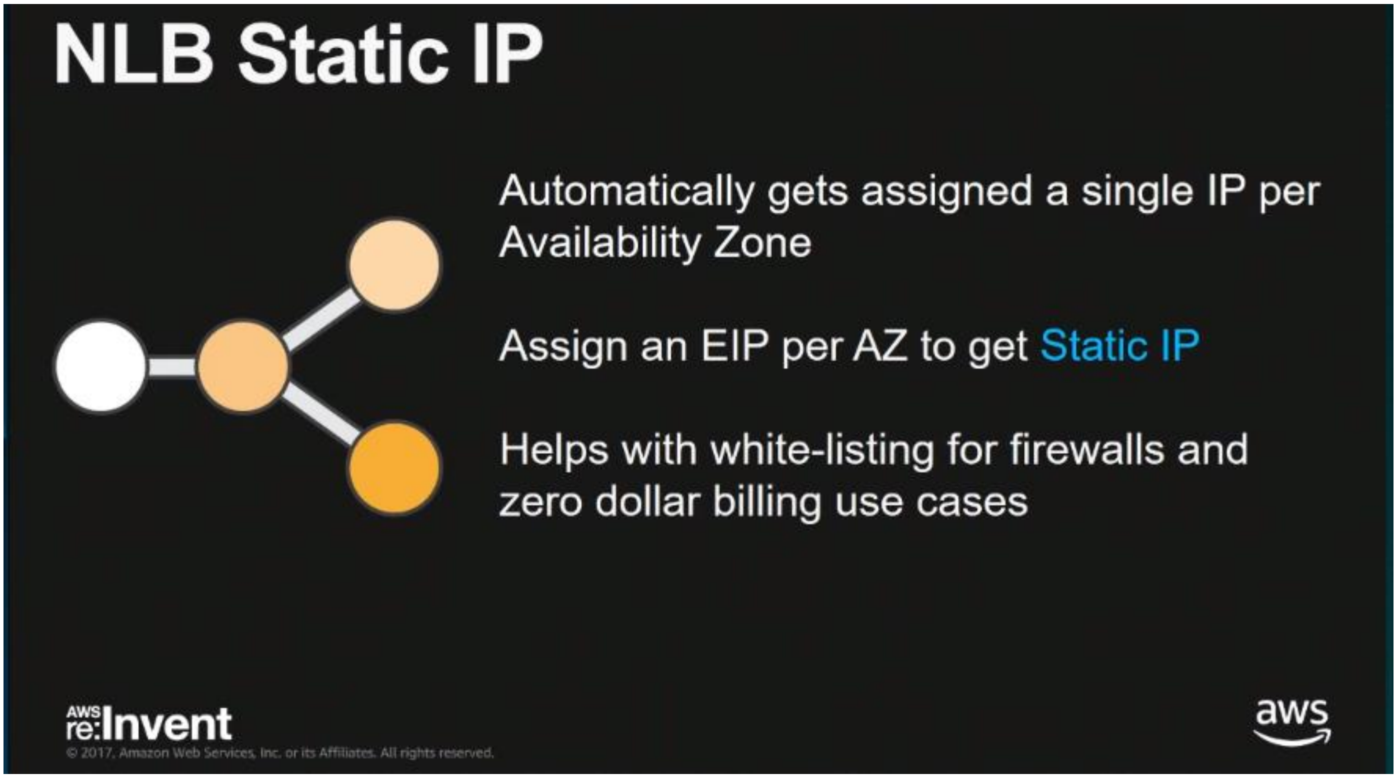


AWS re:Invent

aws

Let us now build up this architecture

NLB Other Key Features



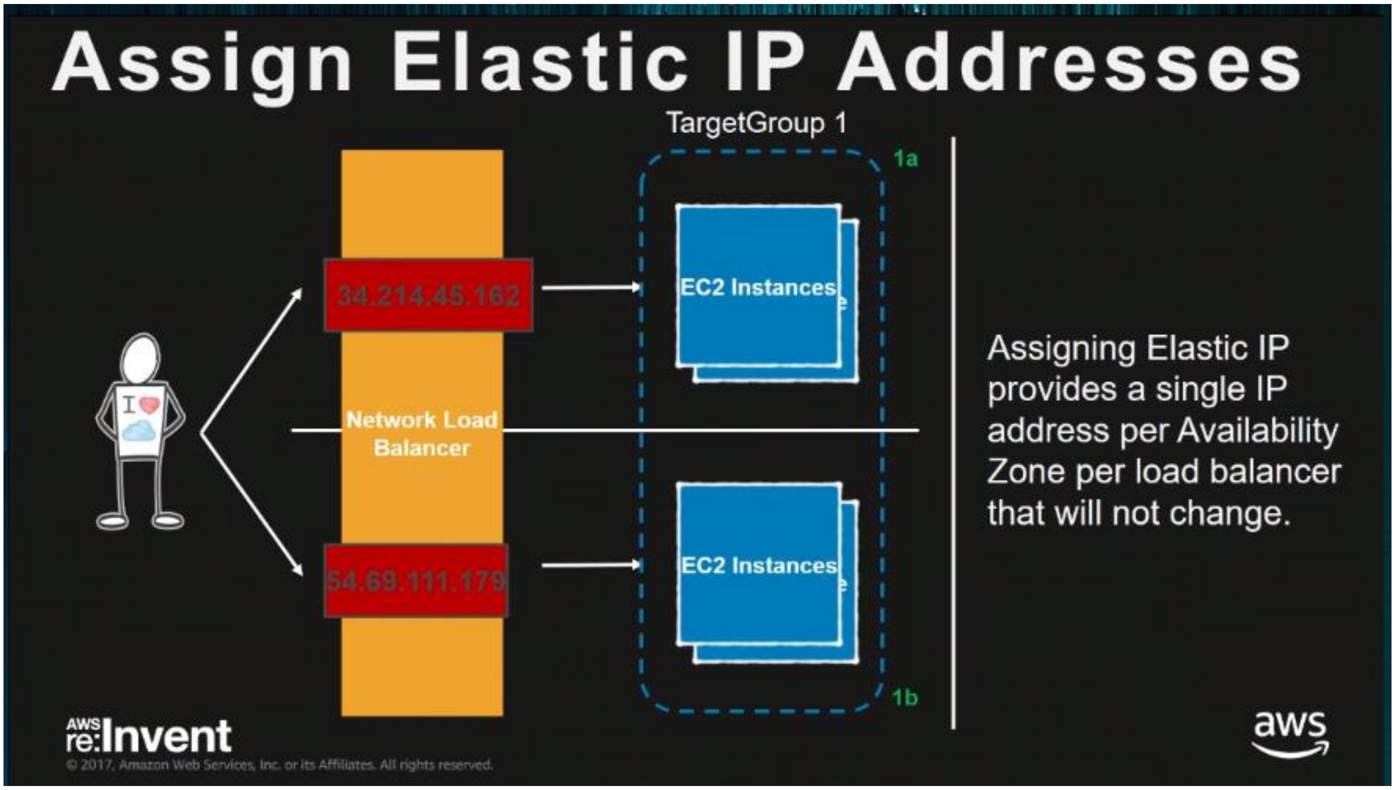
NLB Static IP

Automatically gets assigned a single IP per Availability Zone

Assign an EIP per AZ to get Static IP

Helps with white-listing for firewalls and zero dollar billing use cases

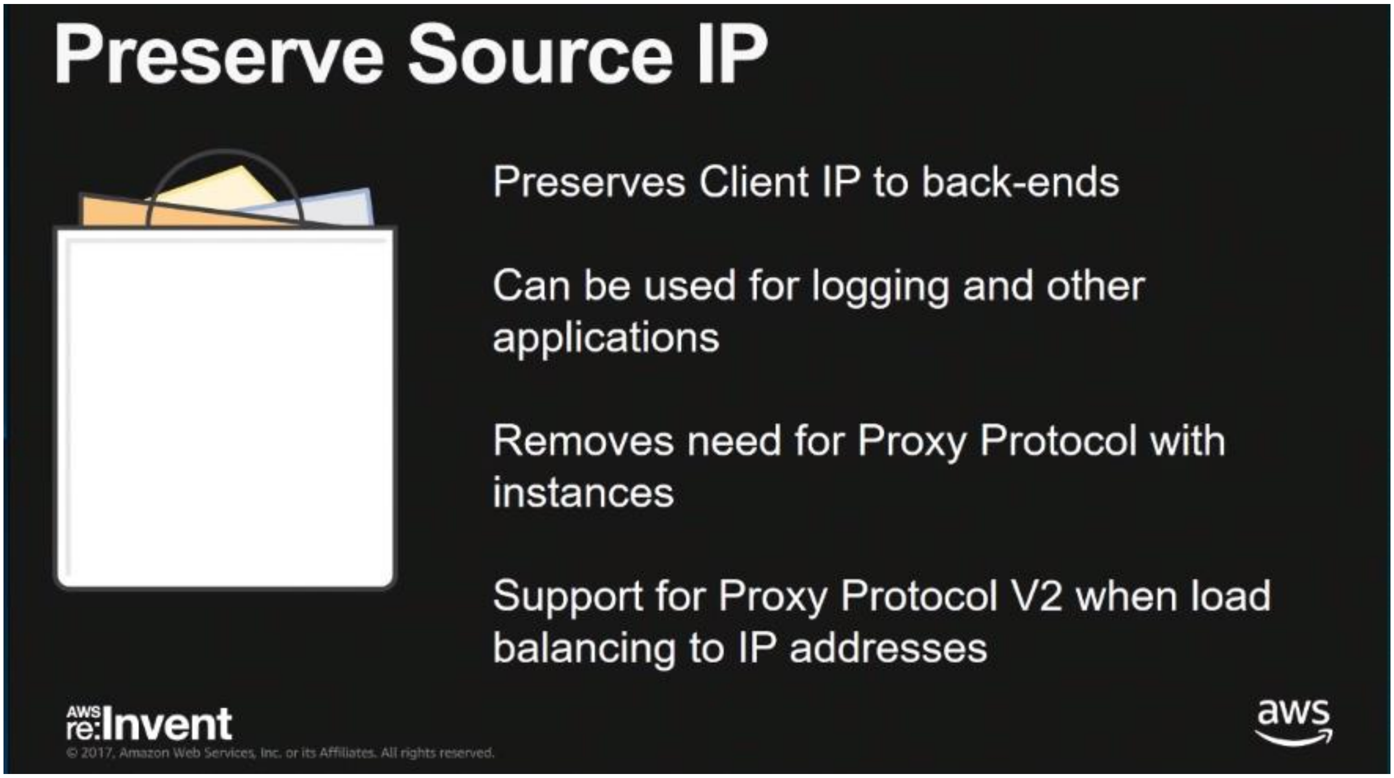
With NLB, you get a single IP per AZ when you create the NLB. If you do not want these IPs to change for the entire life of the LB, you can pick an EIP from your own EIP pool and assign it to the NLB. Static IP use has many use cases, the most common use case is in firewalls where you want to whitelist specific IPs to go across your firewalls.



Assign Elastic IP Addresses

Assigning Elastic IP provides a single IP address per Availability Zone per load balancer that will not change.

You can assign your own EIP per AZ to the NLB as above



Preserve Source IP

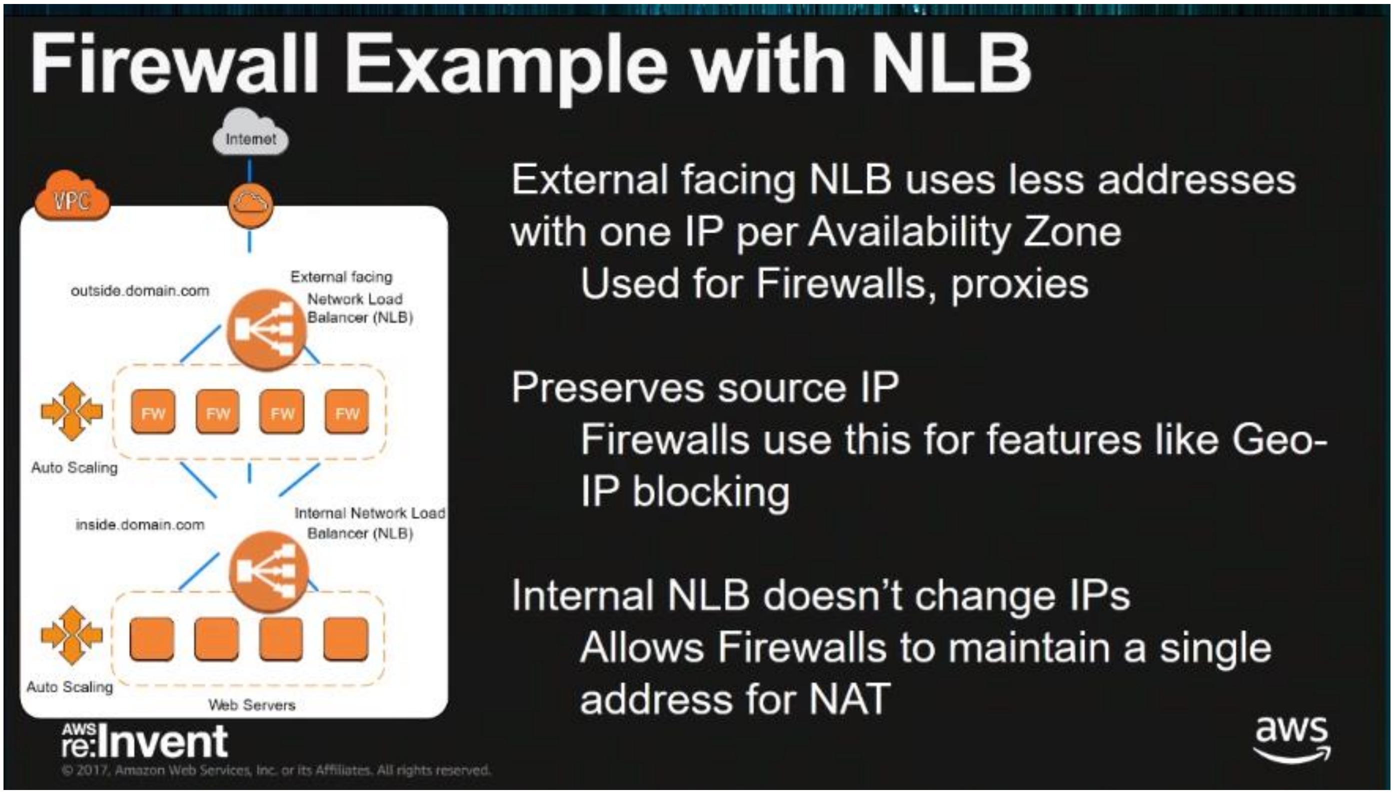
Preserves Client IP to back-ends

Can be used for logging and other applications

Removes need for Proxy Protocol with instances

Support for Proxy Protocol V2 when load balancing to IP addresses

NLB does not terminate the connection to the LB itself and acts as a pass through, the NLB is able to pass the source IP all the way to your backend instances unlike the CLB. You can now set up SGs on your backend targets to allow access to certain CIDR blocks.



Firewall Example with NLB

External facing NLB uses less addresses with one IP per Availability Zone

- Used for Firewalls, proxies

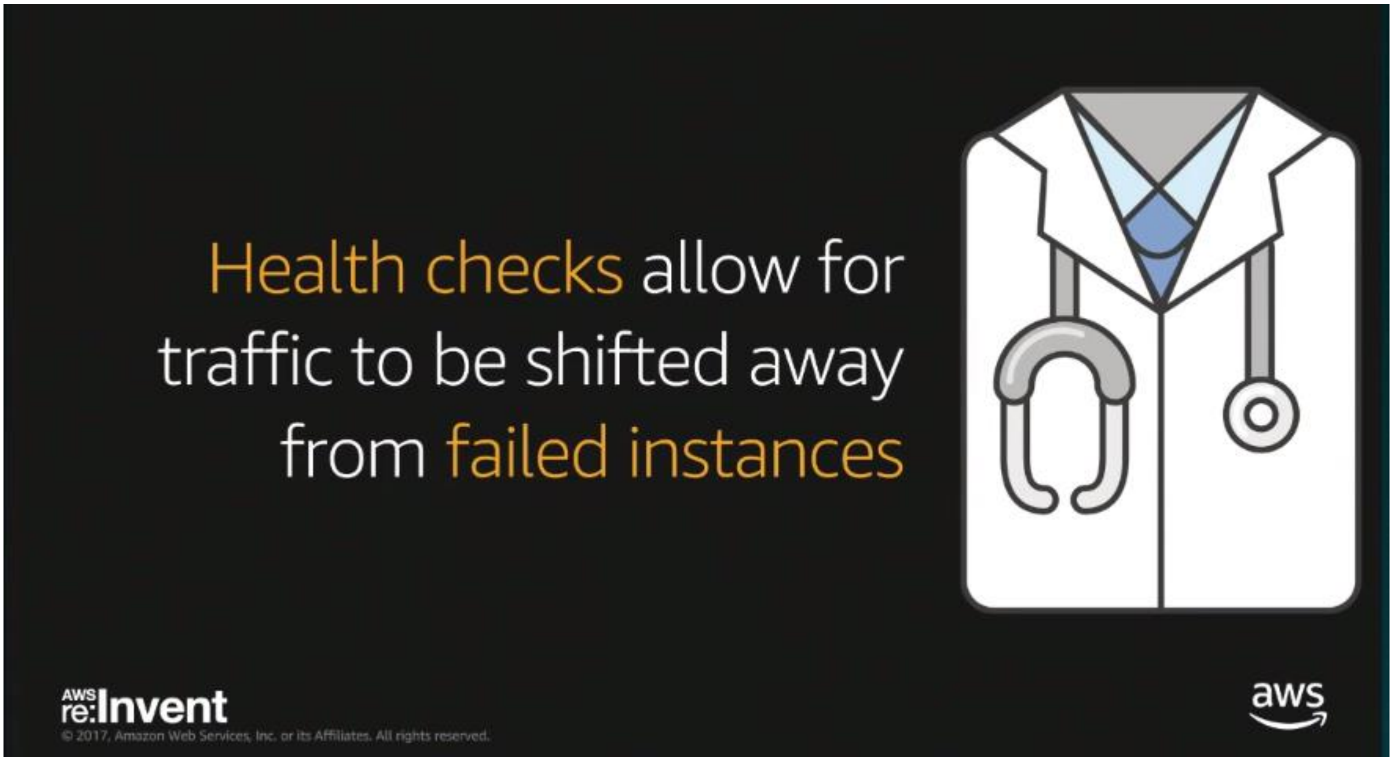
Preserves source IP

- Firewalls use this for features like Geo-IP blocking

Internal NLB doesn't change IPs

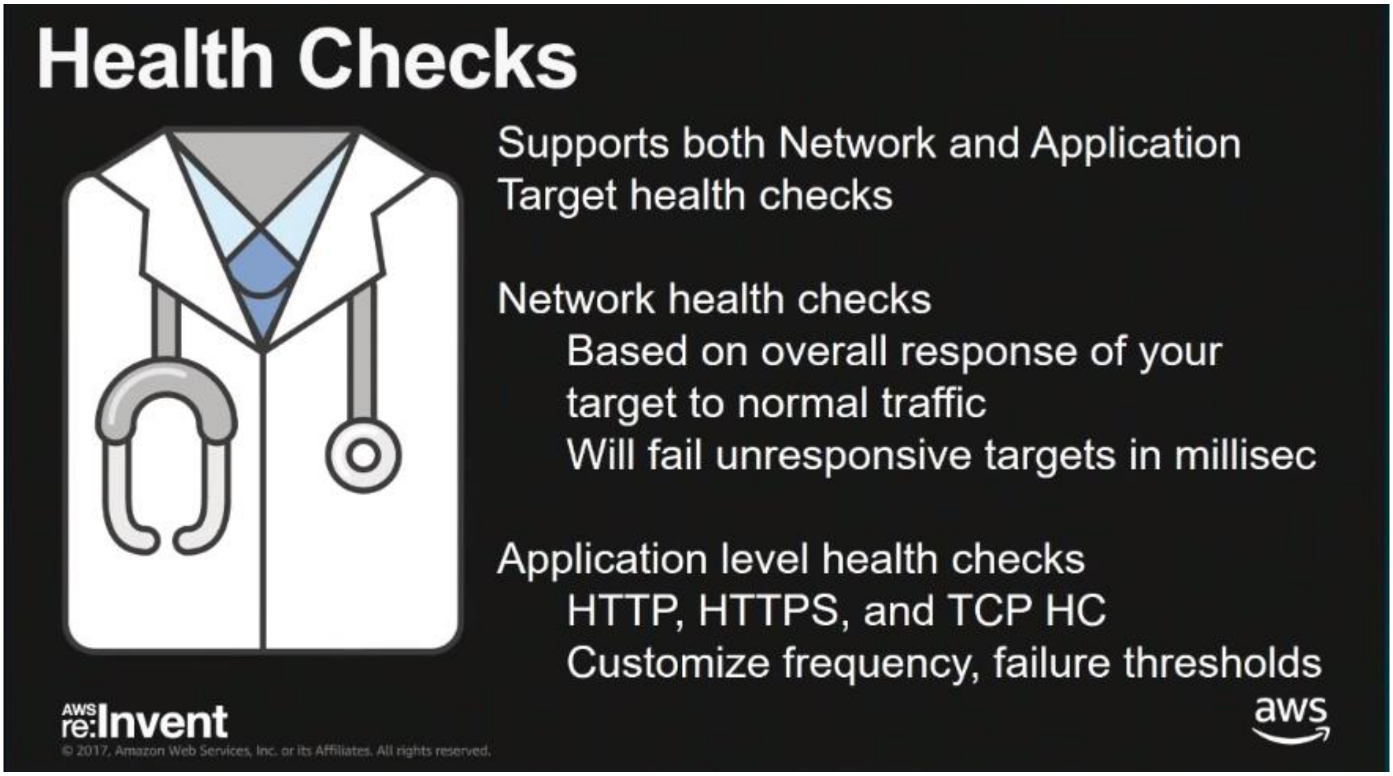
- Allows Firewalls to maintain a single address for NAT

Above is a real-life example where we have an external LB that is fronting a firewall, we then have an internal LB that is fronting a set of web servers. Because the external LB can pass through the source IPs of the requests to the firewalls, you can use the firewalls to do different things like geo-IP blocking. You can also use the sing static IP in NATs in your network.



Health checks allow for traffic to be shifted away from failed instances

re:Invent



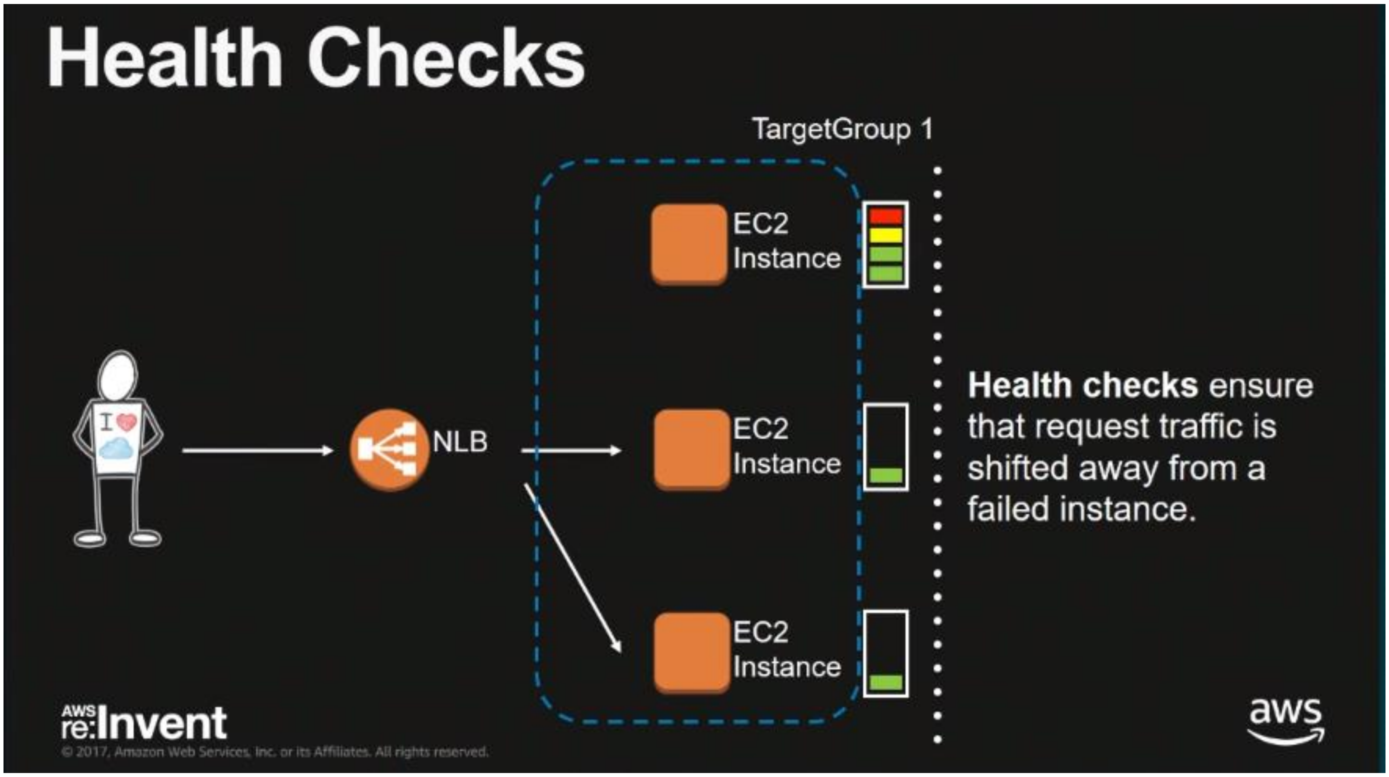
Health Checks

Supports both Network and Application Target health checks

Network health checks
- Based on overall response of your target to normal traffic
- Will fail unresponsive targets in millisec

Application level health checks
- HTTP, HTTPS, and TCP HC
- Customize frequency, failure thresholds

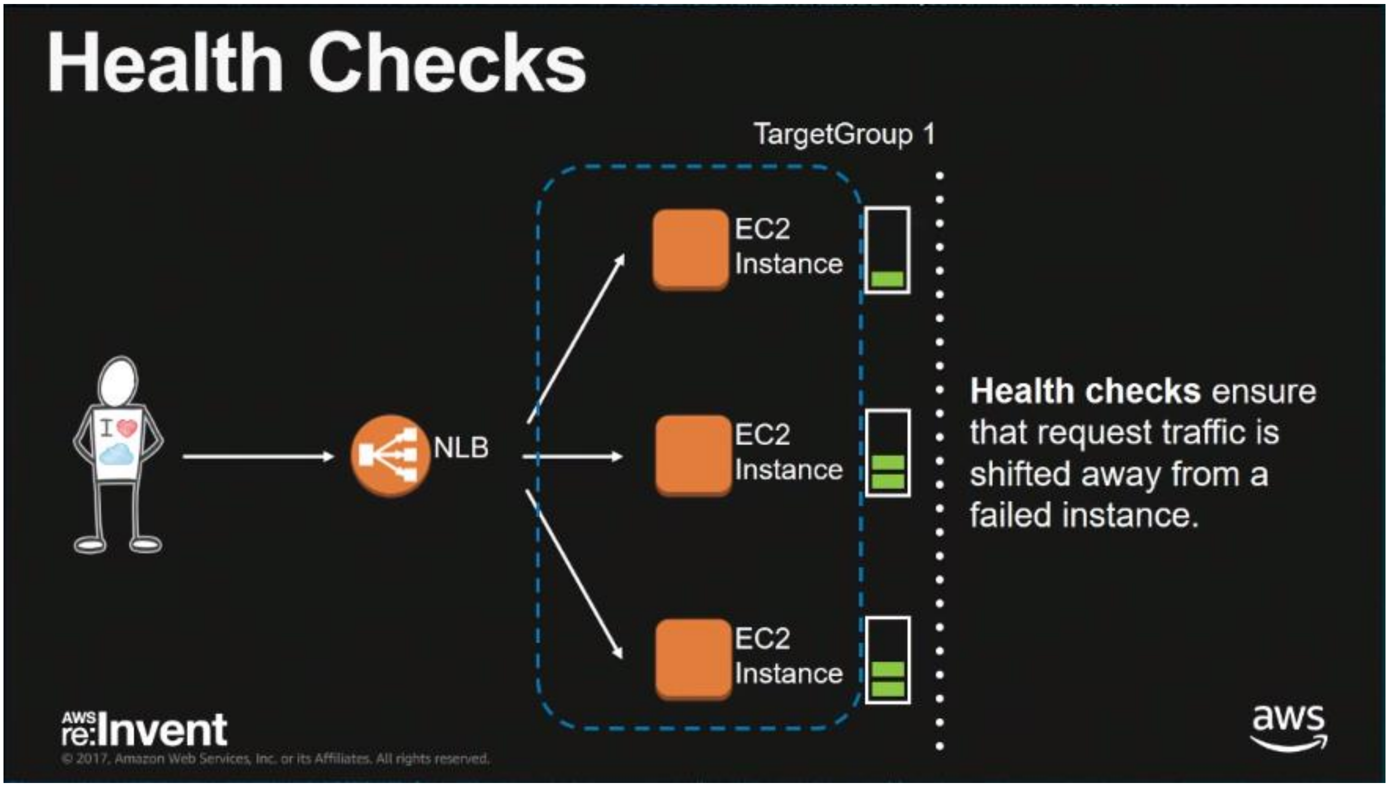
re:Invent



Health Checks

Health checks ensure that request traffic is shifted away from a failed instance.

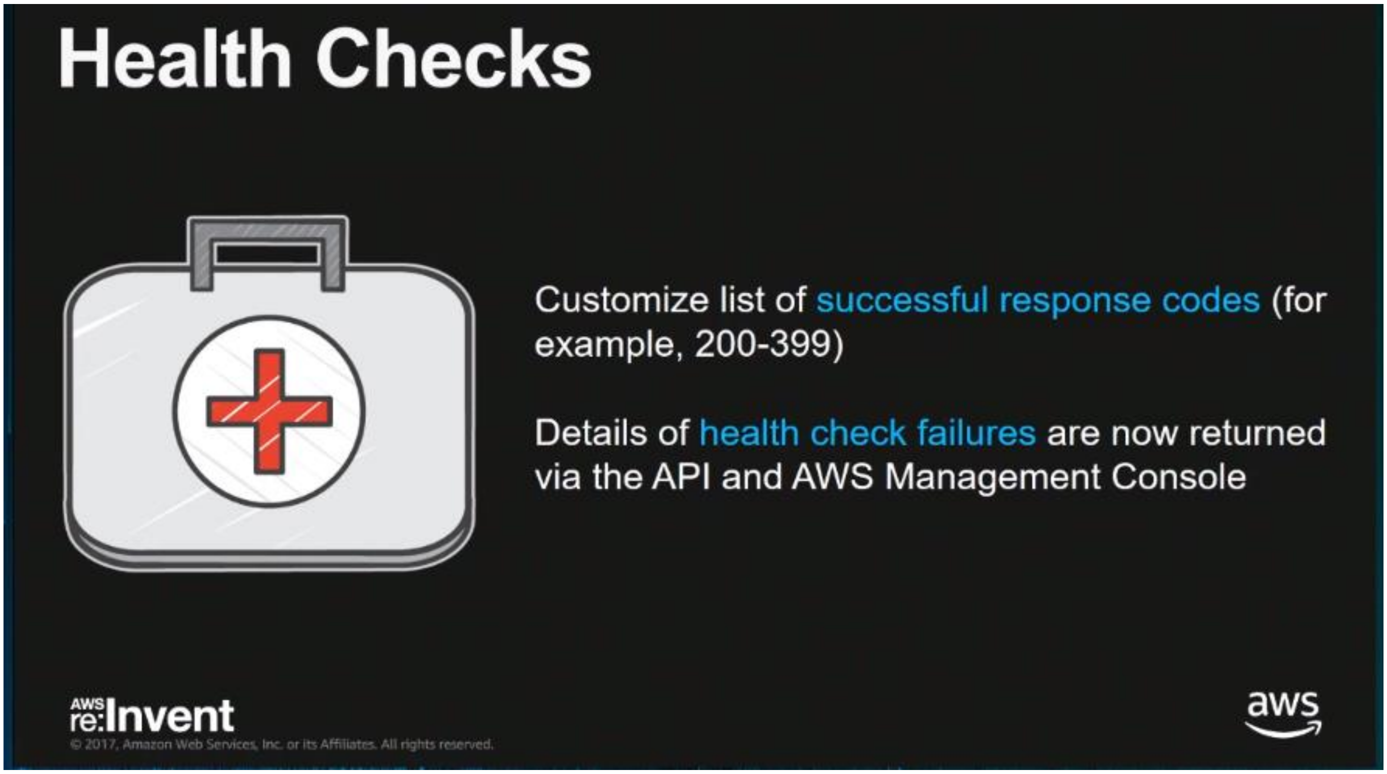
There is a command ***describe-target-health*** that can be used to see the health status of the target



Health Checks

Health checks ensure that request traffic is shifted away from a failed instance.

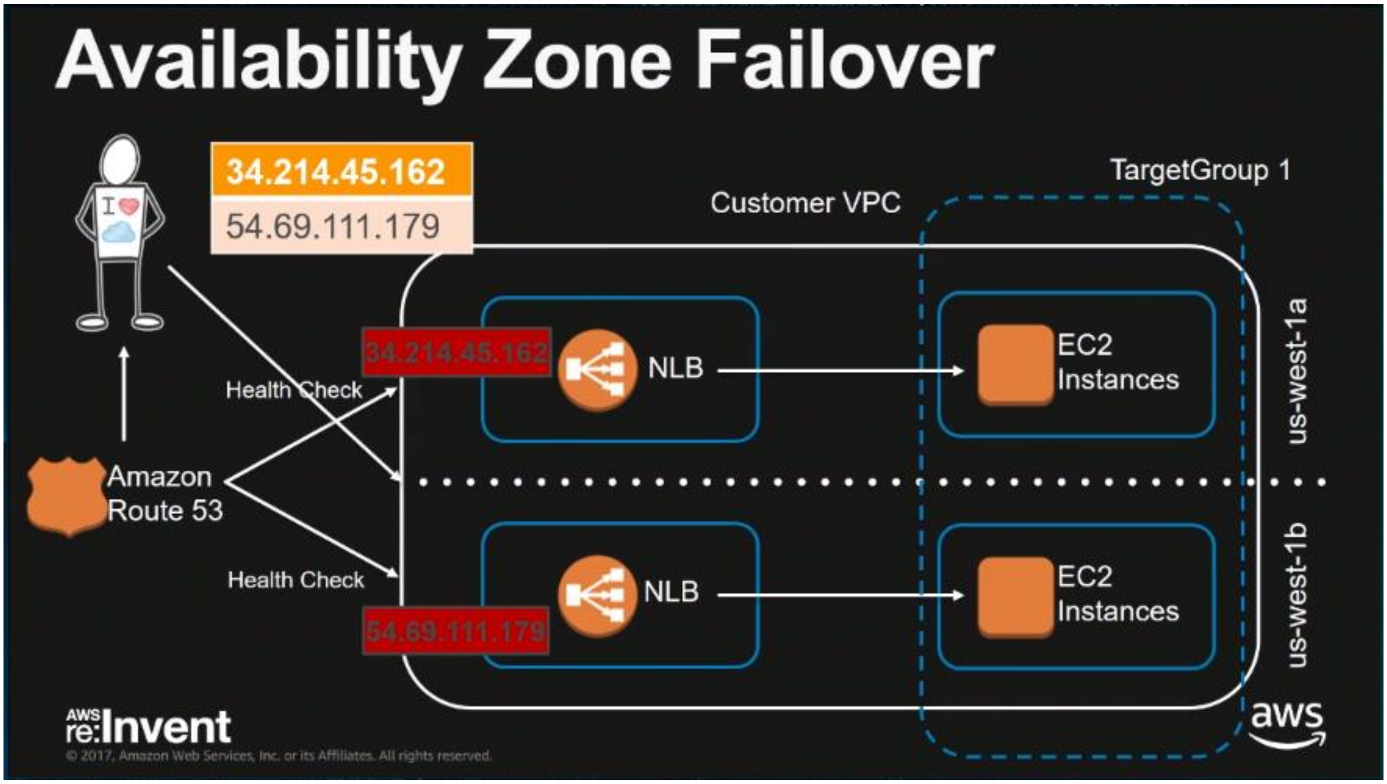
Now the load gets evenly distributed among the targets again



Health Checks

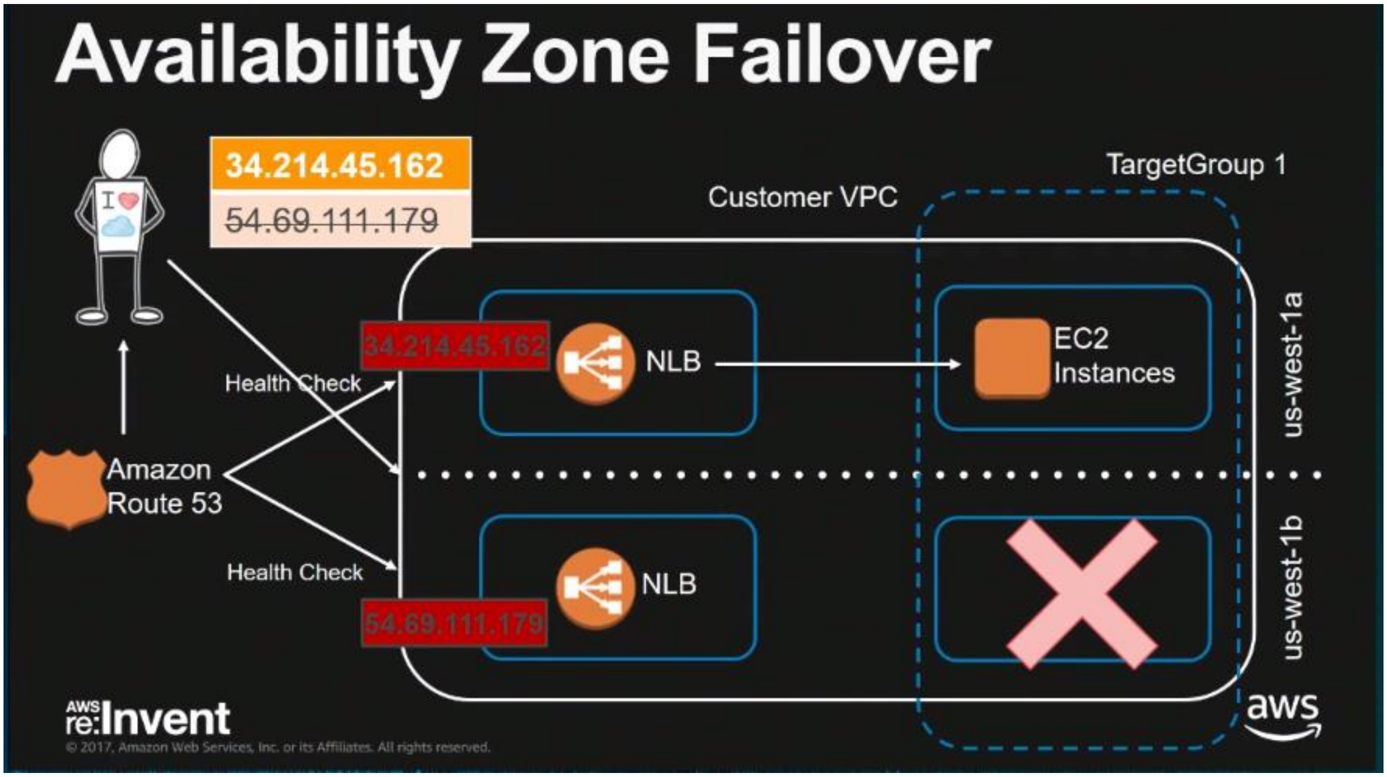
Customize list of successful response codes (for example, 200-399)

Details of health check failures are now returned via the API and AWS Management Console



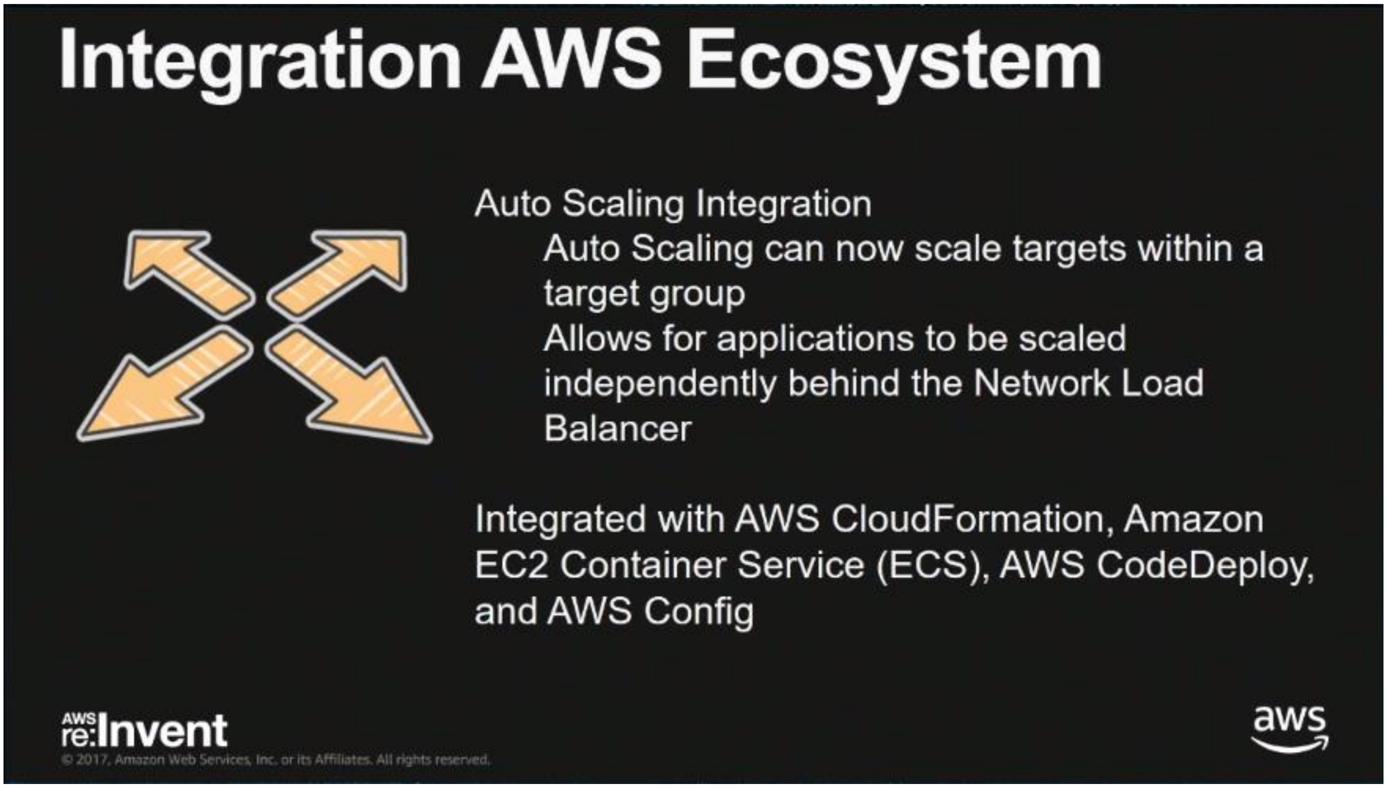
Availability Zone Failover

We have the NLB in 2 different AZs with different static address EIPs for the AZ NLB endpoints, the Route53 is resolving the DNS name of the NLB ***www.examplenlb.com*** to those 2 Static IP addresses.



Availability Zone Failover

If the backends in one AZ get unhealthy, Route53 will automatically start resolving to only the NLB in the healthy AZ. This is an easy DNS failover.



Integration AWS Ecosystem

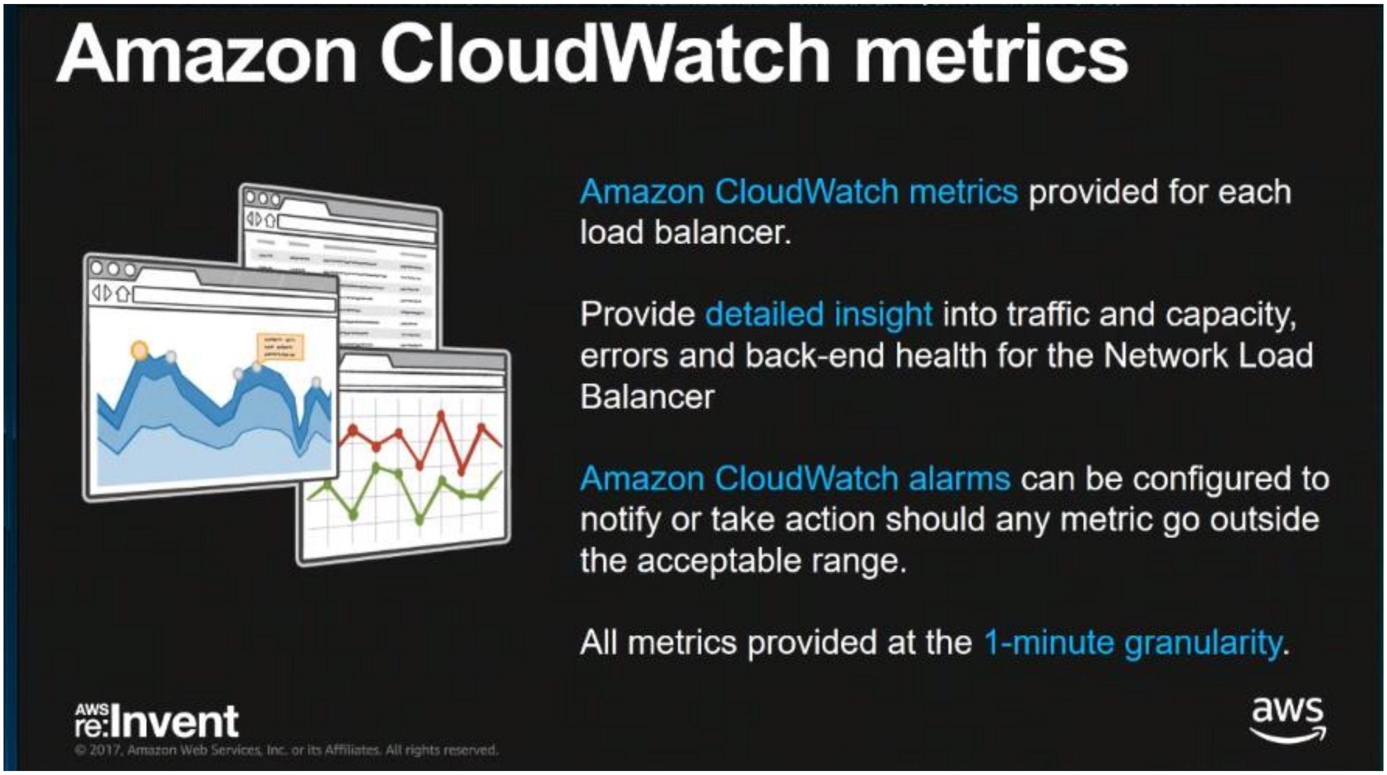
Auto Scaling Integration

- Auto Scaling can now scale targets within a target group
- Allows for applications to be scaled independently behind the Network Load Balancer

Integrated with AWS CloudFormation, Amazon EC2 Container Service (ECS), AWS CodeDeploy, and AWS Config

AWS re:Invent
aws

With ECS integration, you can use dynamic port mapping along with your containers and have the NLB front those applications.



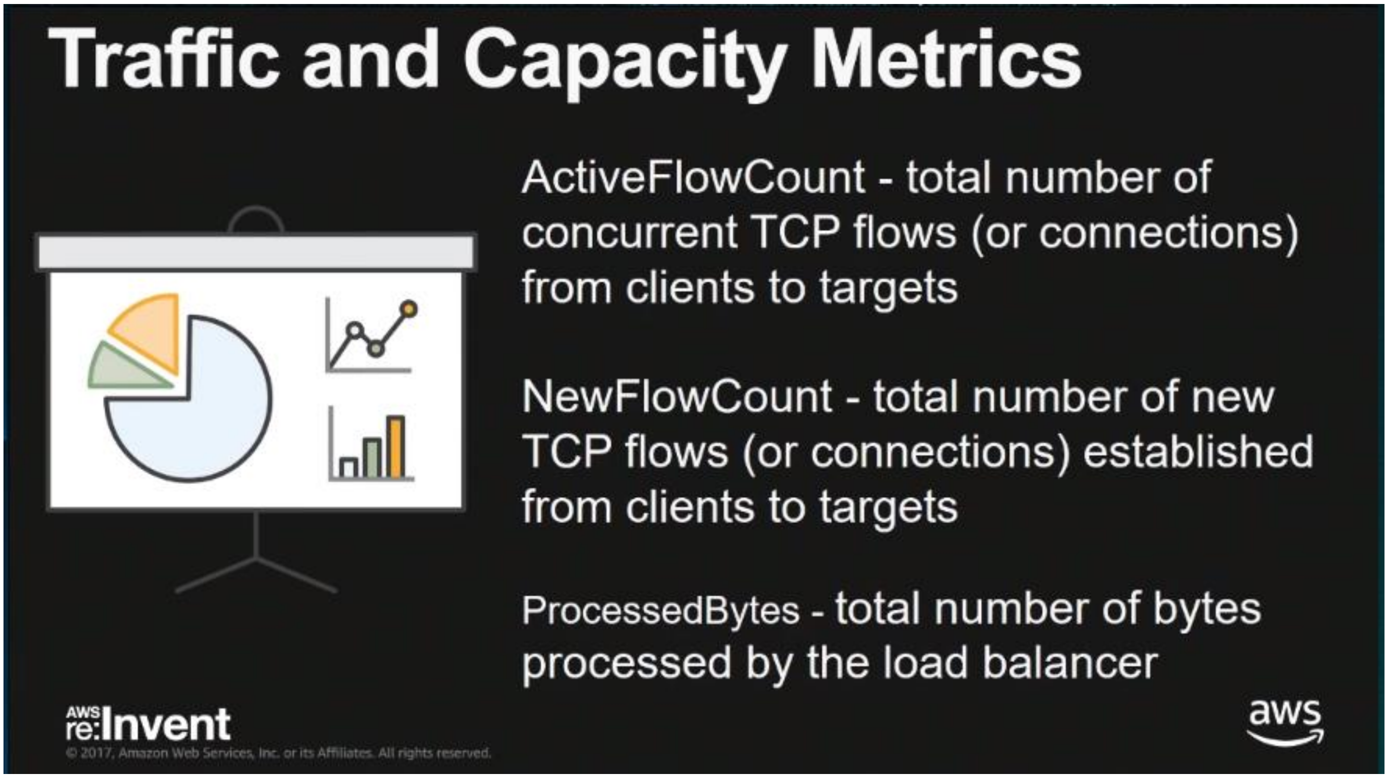
Amazon CloudWatch metrics

Amazon CloudWatch metrics provided for each load balancer.

Provide detailed insight into traffic and capacity, errors and back-end health for the Network Load Balancer

Amazon CloudWatch alarms can be configured to notify or take action should any metric go outside the acceptable range.

All metrics provided at the 1-minute granularity.

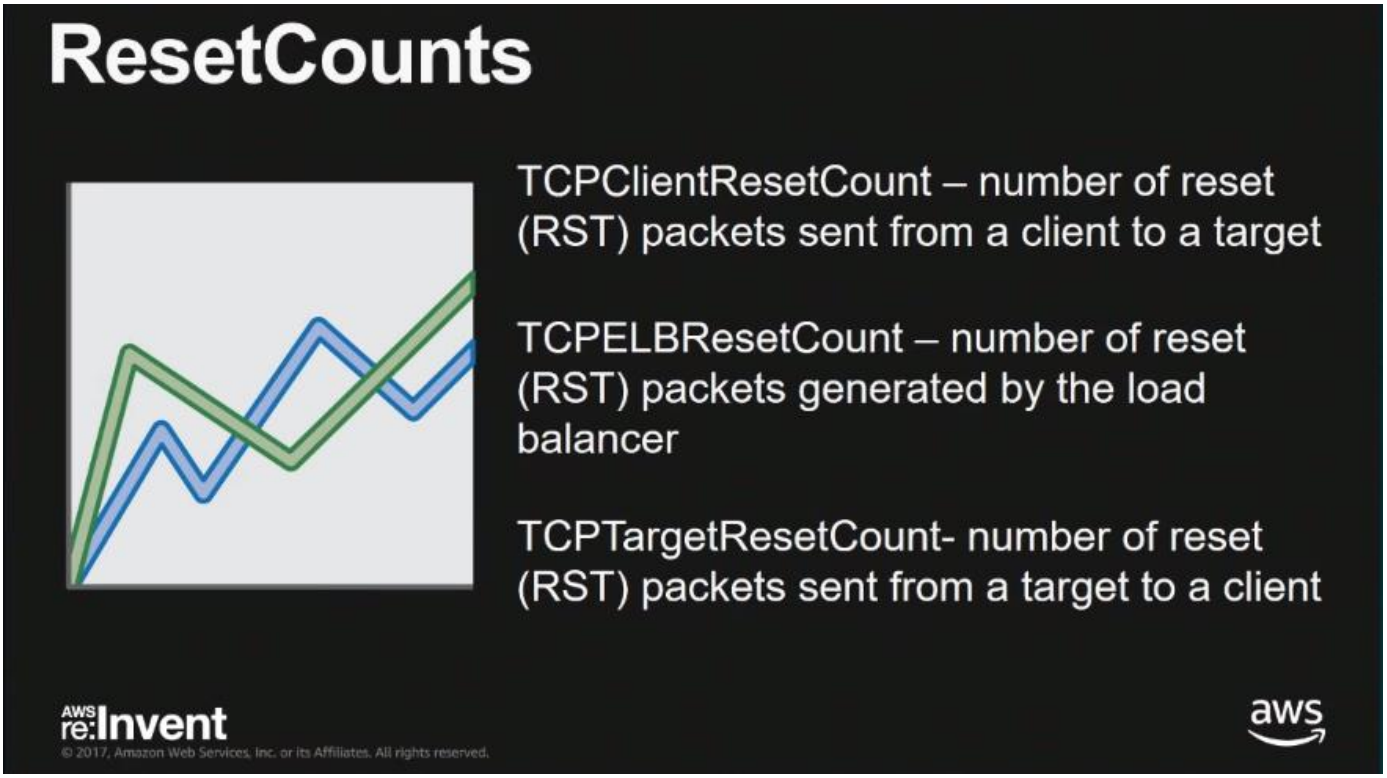


Traffic and Capacity Metrics

ActiveFlowCount - total number of concurrent TCP flows (or connections) from clients to targets

NewFlowCount - total number of new TCP flows (or connections) established from clients to targets

ProcessedBytes - total number of bytes processed by the load balancer

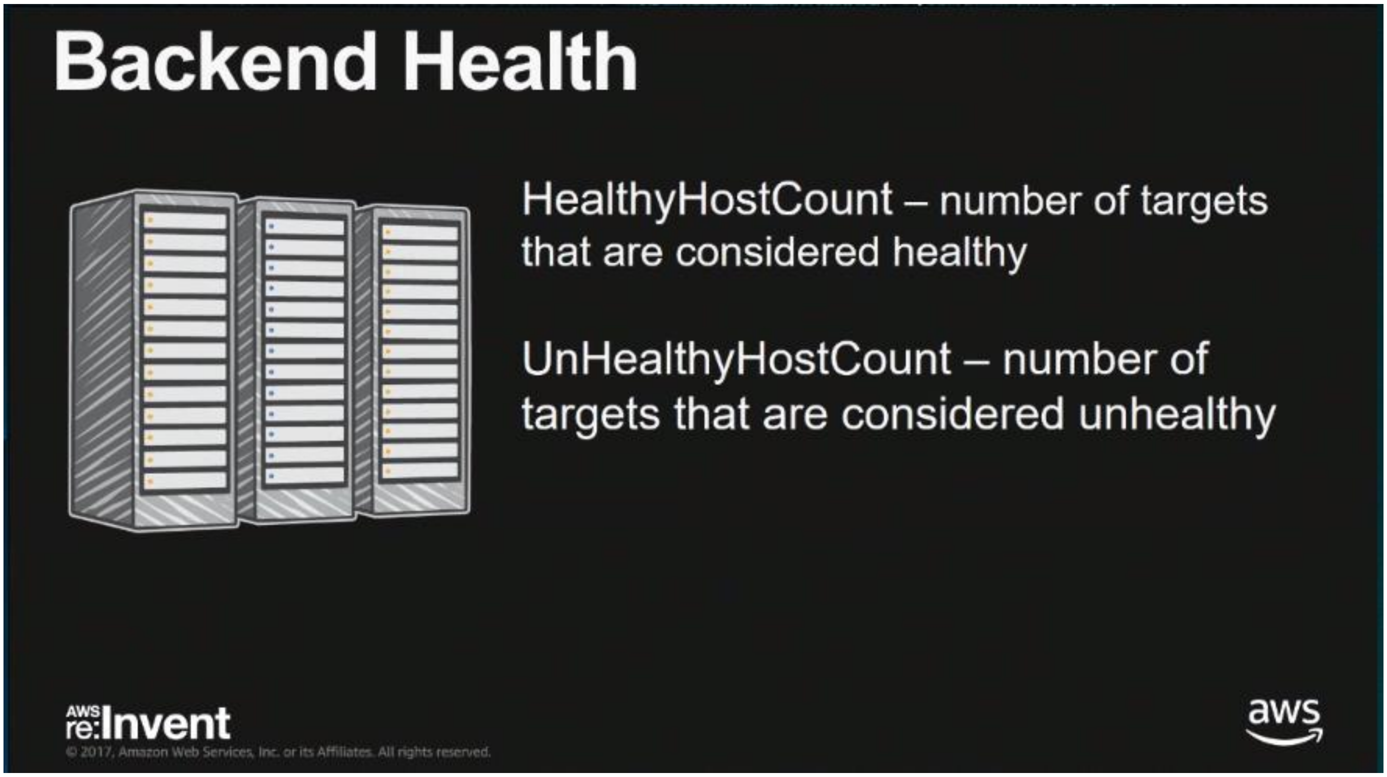


ResetCounts

TCPClientResetCount – number of reset (RST) packets sent from a client to a target

TCPELBResetCount – number of reset (RST) packets generated by the load balancer

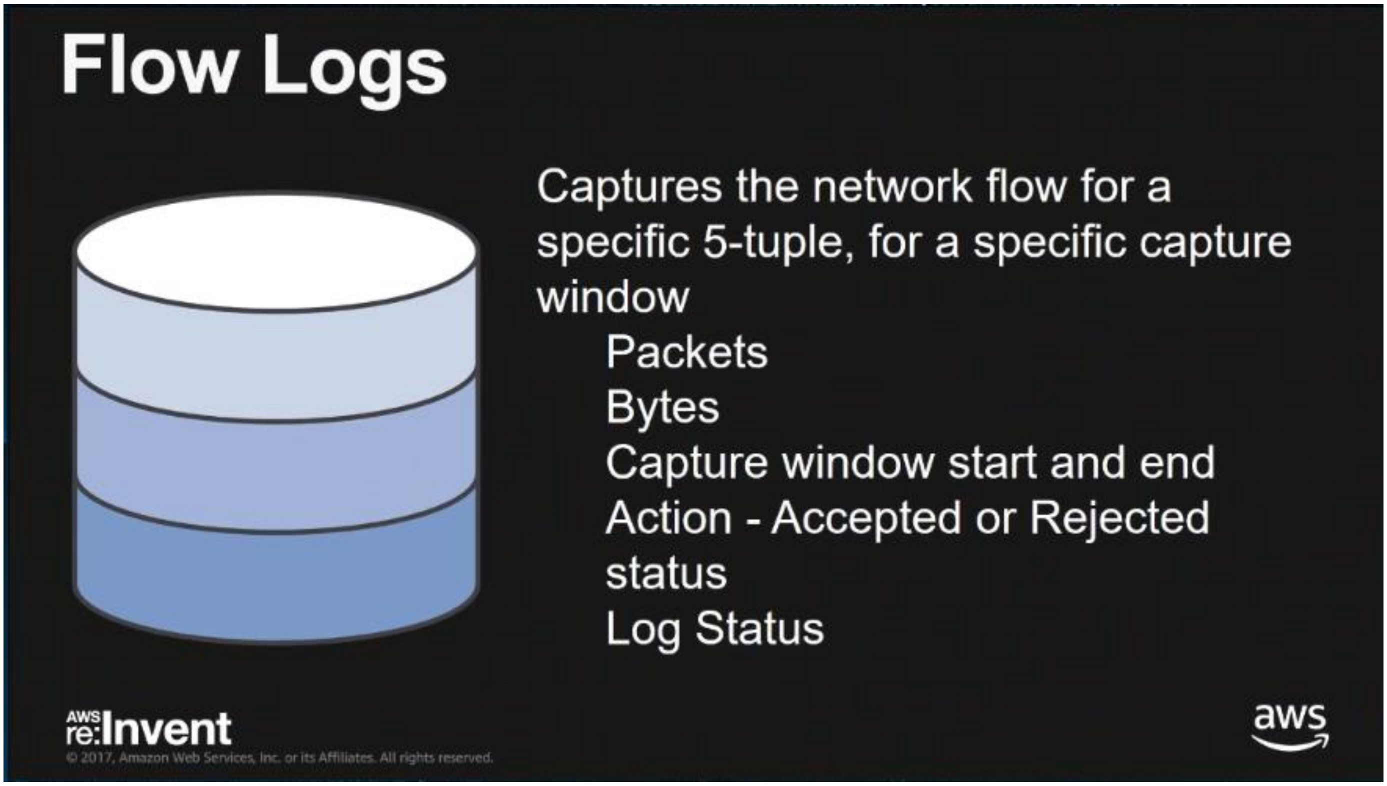
TCPTargetResetCount- number of reset (RST) packets sent from a target to a client



Backend Health

HealthyHostCount – number of targets that are considered healthy

UnHealthyHostCount – number of targets that are considered unhealthy



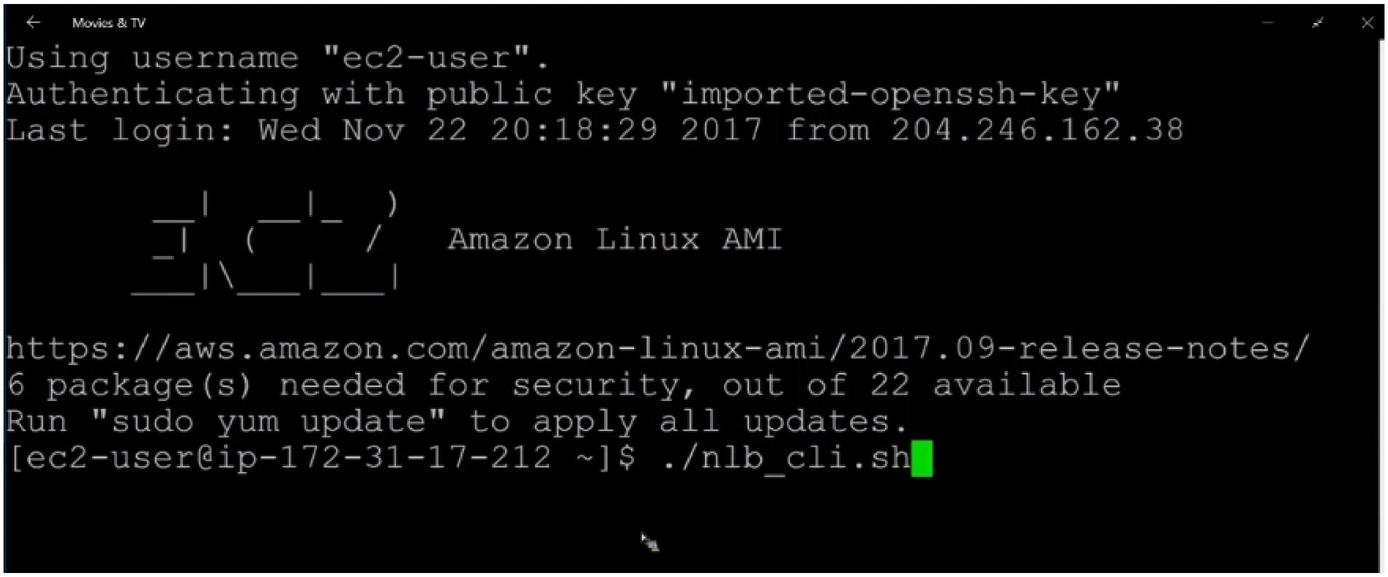
Flow Logs

Captures the network flow for a specific 5-tuple, for a specific capture window

- Packets
- Bytes
- Capture window start and end
- Action - Accepted or Rejected status
- Log Status

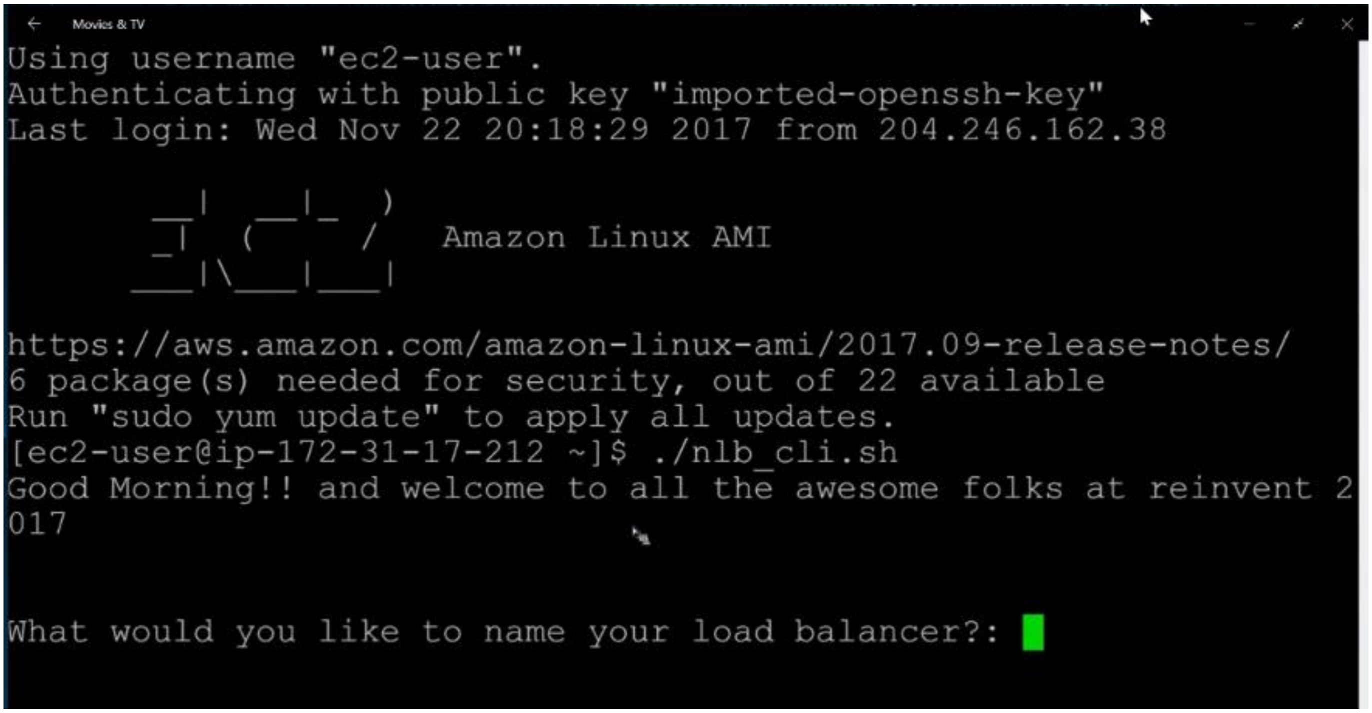
Demo for NLB API and CONSOLE

Let us see how the NLB API works



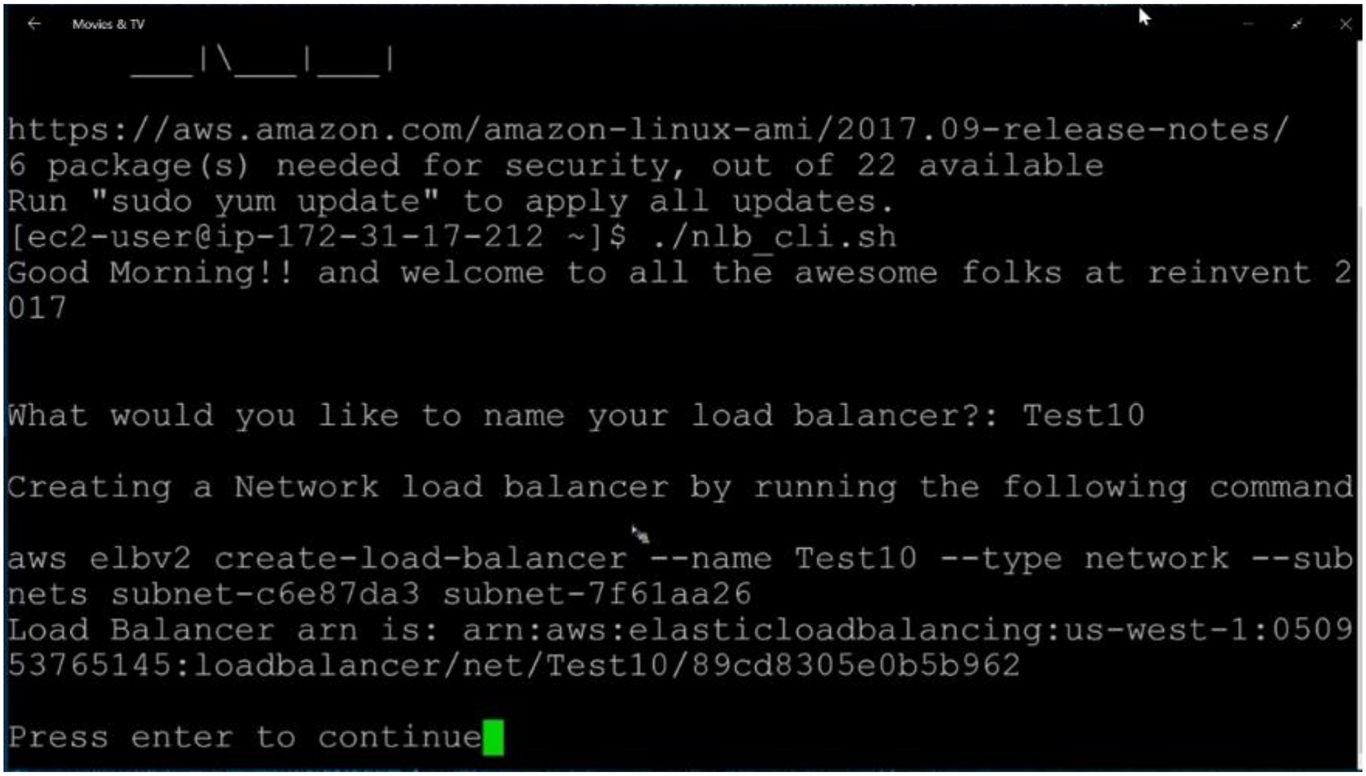
```
Using username "ec2-user".
Authenticating with public key "imported-openssh-key"
Last login: Wed Nov 22 20:18:29 2017 from 204.246.162.38

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2017.09-release-notes/
6 package(s) needed for security, out of 22 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-17-212 ~]$ ./nlb_cli.sh
```

We have an EC2 instance running that is running a bash script that will create an NLB and the various resources like targets, target groups, and a listener. This will help us see the API structure that can be used for creating an NLB.



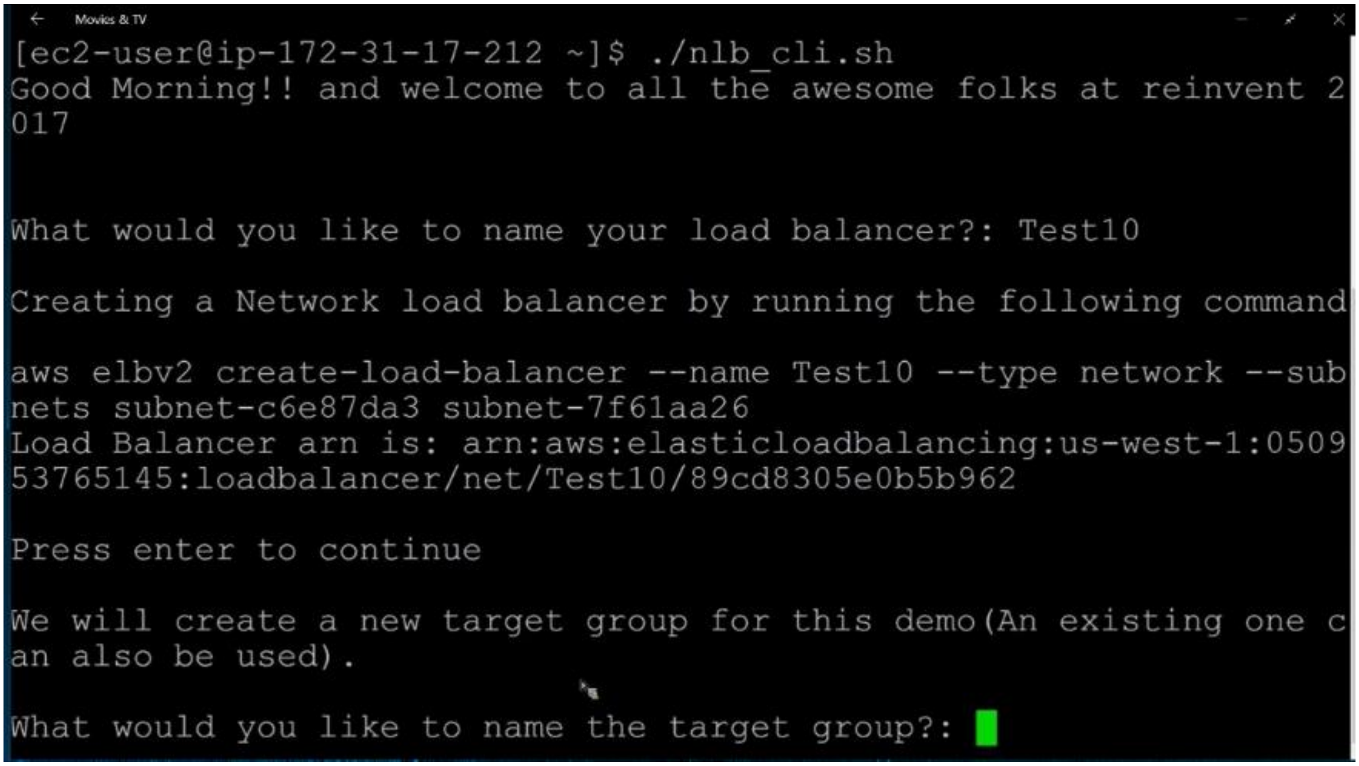
```
Using username "ec2-user".
Authenticating with public key "imported-openssh-key"
Last login: Wed Nov 22 20:18:29 2017 from 204.246.162.38

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2017.09-release-notes/
6 package(s) needed for security, out of 22 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-17-212 ~]$ ./nlb_cli.sh
Good Morning!! and welcome to all the awesome folks at reinvent 2
017


What would you like to name your load balancer?:
```

The script asks what name you want to give your NLB



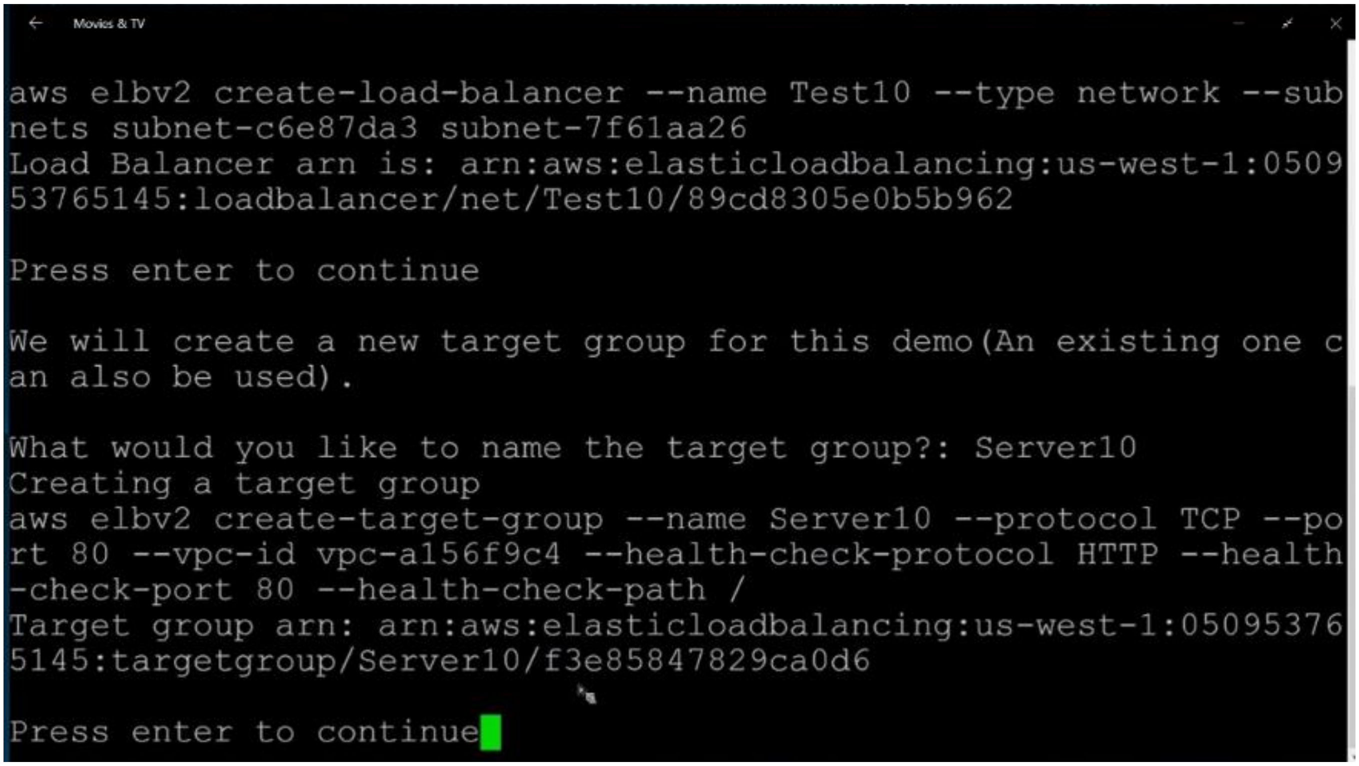
```
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2017.09-release-notes/
6 package(s) needed for security, out of 22 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-17-212 ~]$ ./nlb_cli.sh
Good Morning!! and welcome to all the awesome folks at reinvent 2
017


What would you like to name your load balancer?: Test10

Creating a Network load balancer by running the following command

aws elbv2 create-load-balancer --name Test10 --type network --sub
nets subnet-c6e87da3 subnet-7f61aa26
Load Balancer arn is: arn:aws:elasticloadbalancing:us-west-1:0509
53765145:loadbalancer/net/Test10/89cd8305e0b5b962

Press enter to continue
```

We choose the name ***Test10***, then the script issues the ***$ aws elbv2 create-load-balancer - -name Test10 - -type network - -subnets subnet-c6e87da3 subnet-7f61aa26*** command to create the NLB. We then get the NLB's ***arn*** value back after it has been created successfully. We have used the 2 subnets that we created in 2 different AZs for HA.



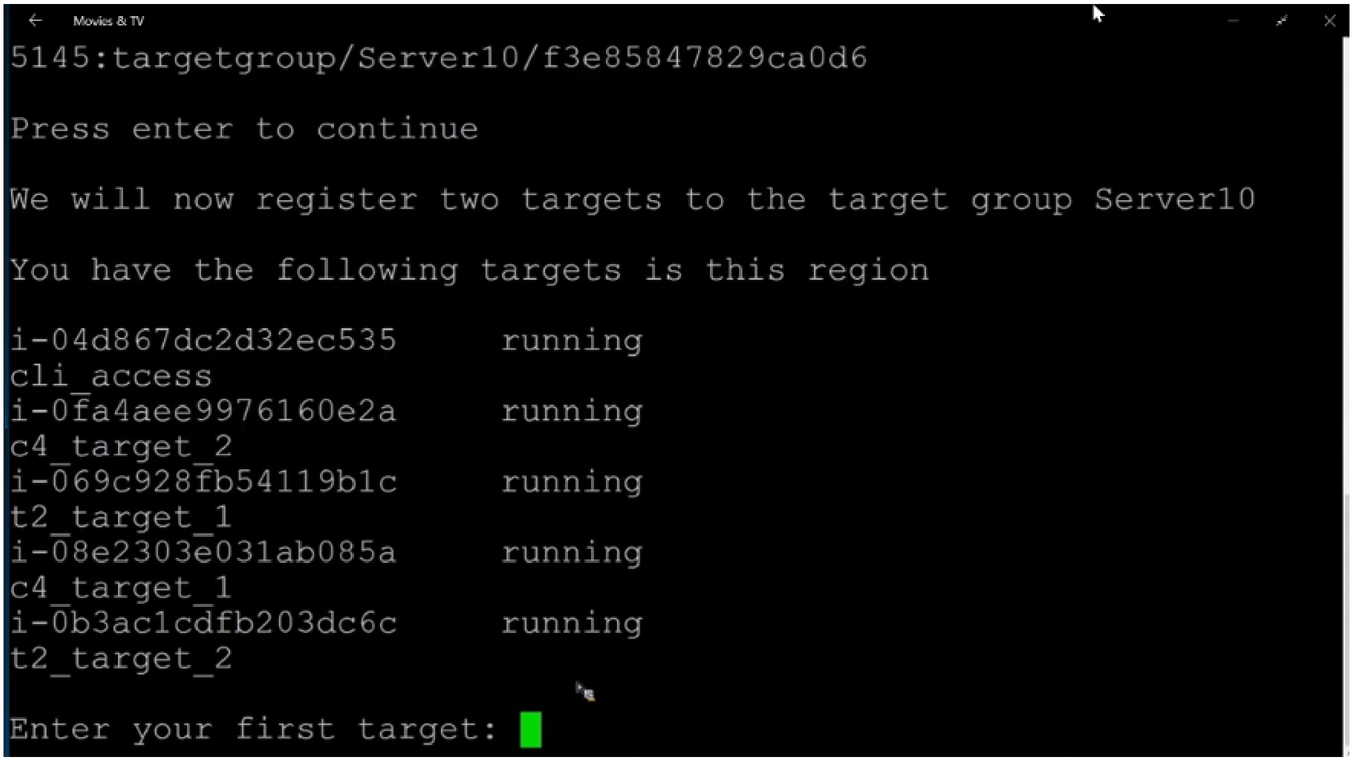
```
[ec2-user@ip-172-31-17-212 ~]$ ./nlb_cli.sh
Good Morning!! and welcome to all the awesome folks at reinvent 2
017


What would you like to name your load balancer?: Test10

Creating a Network load balancer by running the following command

aws elbv2 create-load-balancer --name Test10 --type network --sub
nets subnet-c6e87da3 subnet-7f61aa26
Load Balancer arn is: arn:aws:elasticloadbalancing:us-west-1:0509
53765145:loadbalancer/net/Test10/89cd8305e0b5b962

Press enter to continue

We will create a new target group for this demo(An existing one c
an also be used).

What would you like to name the target group?: 
```

The script then asks if we want to create a new target group to use with a lister for this NLB.



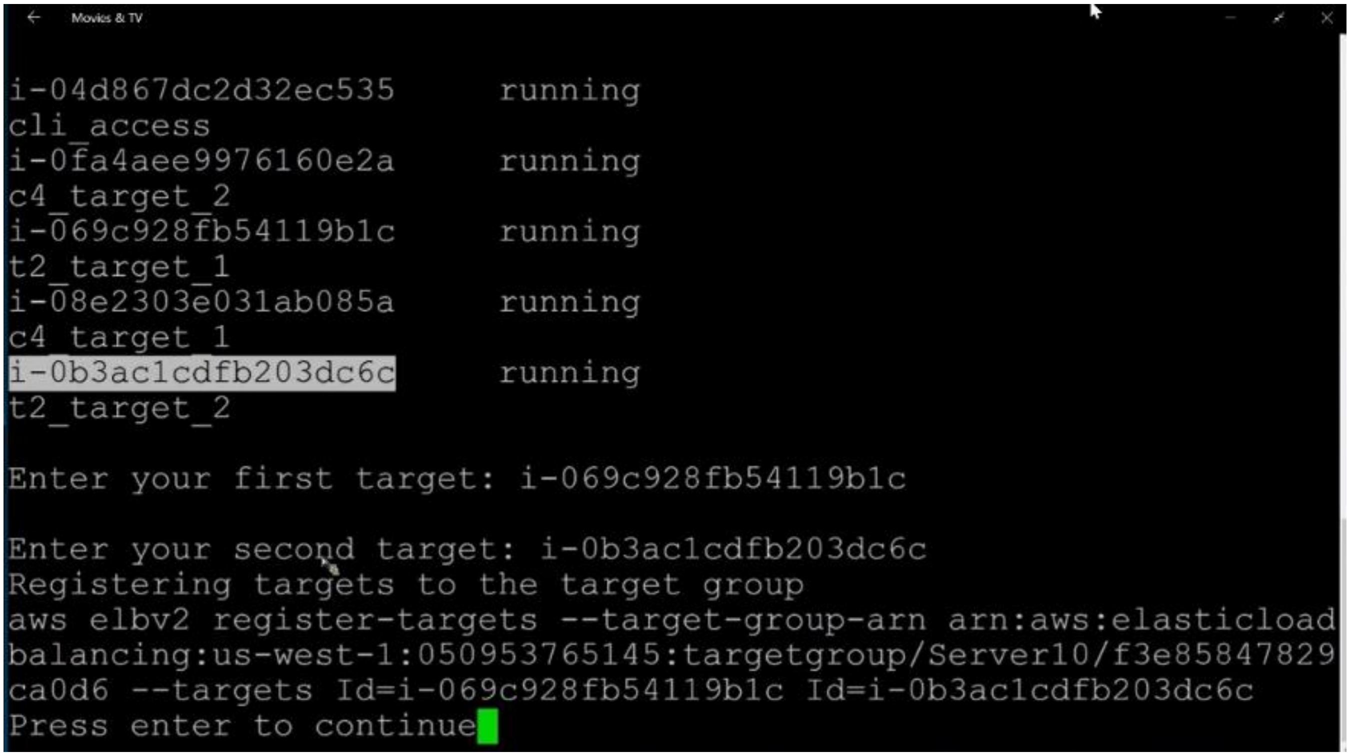
```
aws elbv2 create-load-balancer --name Test10 --type network --sub
nets subnet-c6e87da3 subnet-7f61aa26
Load Balancer arn is: arn:aws:elasticloadbalancing:us-west-1:0509
53765145:loadbalancer/net/Test10/89cd8305e0b5b962

Press enter to continue

We will create a new target group for this demo(An existing one c
an also be used).

What would you like to name the target group?: Server10
Creating a target group
aws elbv2 create-target-group --name Server10 --protocol TCP --po
rt 80 --vpc-id vpc-a156f9c4 --health-check-protocol HTTP --health
-check-port 80 --health-check-path /
Target group arn: arn:aws:elasticloadbalancing:us-west-1:05095376
5145:targetgroup/Server10/f3e85847829ca0d6

Press enter to continue
```

We choose the target group name Server10 and the script then creates a new target group for this NLB with the ***$ aws elbv2 create-target-group - -name Server10 - -protocol TCP - -port 80 - -vpc-id vpc-a156f9c4 - -health-check-protocol HTTP - -health-check-port 80 - -health-check-path /*** command. It then returns back to us the newly created target group’s ***arn*** value.



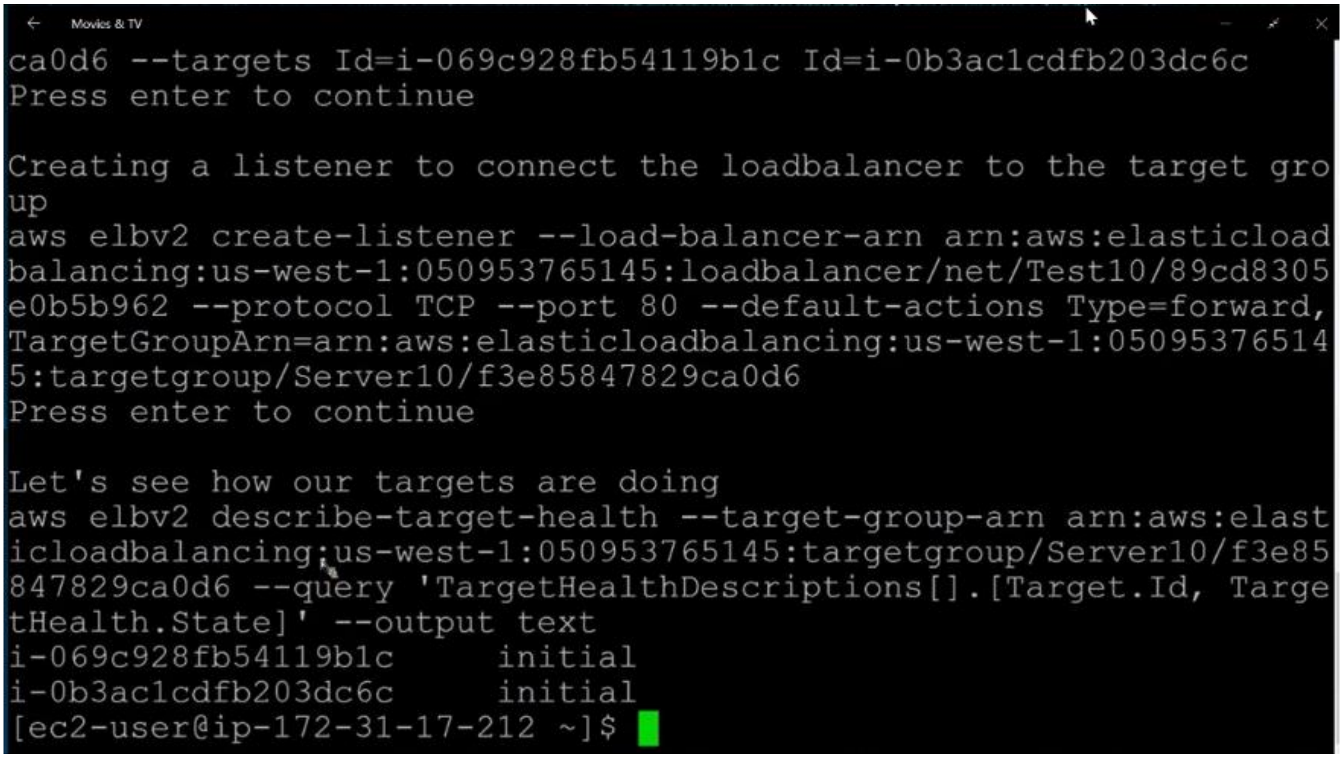
```
5145:targetgroup/Server10/f3e85847829ca0d6

Press enter to continue

We will now register two targets to the target group Server10

You have the following targets is this region

i-04d867dc2d32ec535     running
cli_access
i-0fa4aee9976160e2a     running
c4_target_2
i-069c928fb54119b1c     running
t2_target_1
i-08e2303e031ab085a     running
c4_target_1
i-0b3ac1cdfb203dc6c     running
t2_target_2

Enter your first target:
```

Now it is going to ask that we ***register targets that we want to use with this NLB*** by showing us the list of all available targets that we can select to add to this NLB.



```
i-04d867dc2d32ec535     running
cli_access
i-0fa4aee9976160e2a     running
c4_target_2
i-069c928fb54119b1c     running
t2_target_1
i-08e2303e031ab085a     running
c4_target_1
i-0b3ac1cdfb203dc6c     running
t2_target_2

Enter your first target: i-069c928fb54119b1c

Enter your second target: i-0b3ac1cdfb203dc6c
Registering targets to the target group
aws elbv2 register-targets --target-group-arn arn:aws:elasticload
balancing:us-west-1:050953765145:targetgroup/Server10/f3e85847829
ca0d6 --targets Id=i-069c928fb54119b1c Id=i-0b3ac1cdfb203dc6c
Press enter to continue
```

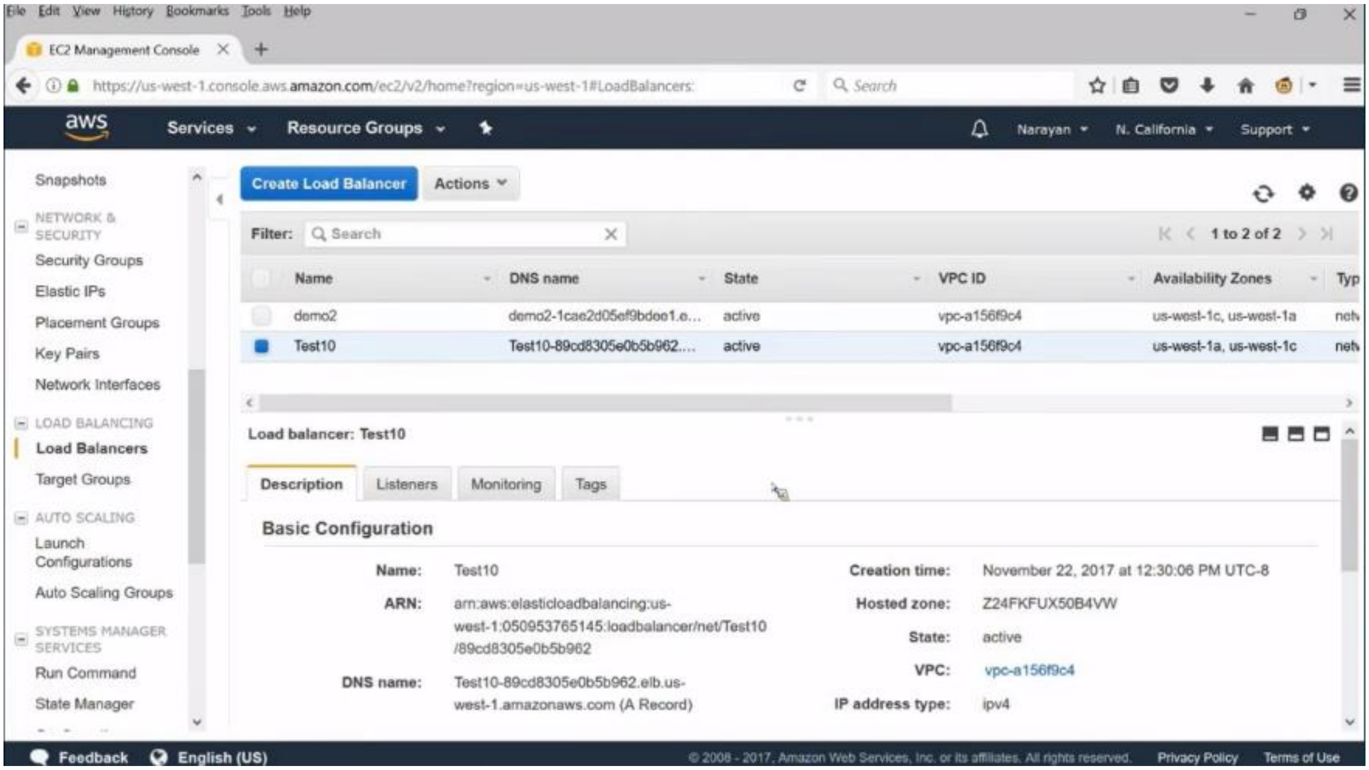
We select 2 target group to have them registered with this NLB as target groups, the script the issues the ***$ aws elbv2 register-targets - -target-group-arn arn:aws:elasticloadbalancing:us-west-1:0509673673676:targetgroup/Server10/f3e85847829ca0d6 - -targets Id=<target-group 1> Id=<target-group 2>*** command to have the 2 target groups registered as target groups for this NLB.



```
ca0d6 --targets Id=i-069c928fb54119b1c Id=i-0b3ac1cdfb203dc6c
Press enter to continue

Creating a listener to connect the loadbalancer to the target gro
up
aws elbv2 create-listener --load-balancer-arn arn:aws:elasticload
balancing:us-west-1:050953765145:loadbalancer/net/Test10/89cd8305
e0b5b962 --protocol TCP --port 80 --default-actions Type=forward,
TargetGroupArn=arn:aws:elasticloadbalancing:us-west-1:05095376514
5:targetgroup/Server10/f3e85847829ca0d6
Press enter to continue

Let's see how our targets are doing
aws elbv2 describe-target-health --target-group-arn arn:aws:elast
icloadbalancing:us-west-1:050953765145:targetgroup/Server10/f3e85
847829ca0d6 --query 'TargetHealthDescriptions[].[Target.Id, Targe
tHealth.State]' --output text
i-069c928fb54119b1c     initial
i-0b3ac1cdfb203dc6c     initial
[ec2-user@ip-172-31-17-212 ~]$
```

The script then goes and automatically creates a listener for us using the ***$ aws elbv2 create-listener - -load-balancer-arn arn:aws:elasticloadbalancing:us-west-1:0509673673676:loadbalancer/net/Test10/89cd8305e0b5b962 - -protocol TCP - -port 80 - -default-actions Type=forward,TargetGroupArn=arn:aws:elasticloadbalancing:us-west-1: 0509673673676:targetgroup/Server10/f3e85847829ca0d6*** command. We are using the default action type of 'forward' so that any connection that comes into our NLB at port 80 with the TCP protocol, will be forwarded to the target group called Server10. This then uses the 5-tuple hash algorithm and the sequenceId to pick the specific backend to send the request to within the Server10 target group.

We can then use the script to check the status of our targets using the ***$ aws elbv2 describe-target-health - -target-group-arn arn:aws:elasticloadbalancing:us-west-1: 0509673673676:targetgroup/Server10/ f3e85847829ca0d6 –query 'TargetHealthDescriptions[].[target.Id, TargetHealth.State]' - -output text*** command as above. We get the result back that the 2 target groups are been initialized and the healthchecks are still in progress for the targets in that particular target group.



Name	DNS name	State	VPC ID	Availability Zones	Typ
demo2	demo2-1cae2d05ef9bdee1.e...	active	vpc-a156f9c4	us-west-1c, us-west-1a	netv
Test10	Test10-89cd8305e0b5b962....	active	vpc-a156f9c4	us-west-1a, us-west-1c	netv

Load balancer: Test10

Basic Configuration

Name: Test10

ARN: arn:aws:elasticloadbalancing:us-west-1:050953765145:loadbalancer/net/Test10/89cd8305e0b5b962

DNS name: Test10-89cd8305e0b5b962.elb.us-west-1.amazonaws.com (A Record)

Creation time: November 22, 2017 at 12:30:06 PM UTC-8

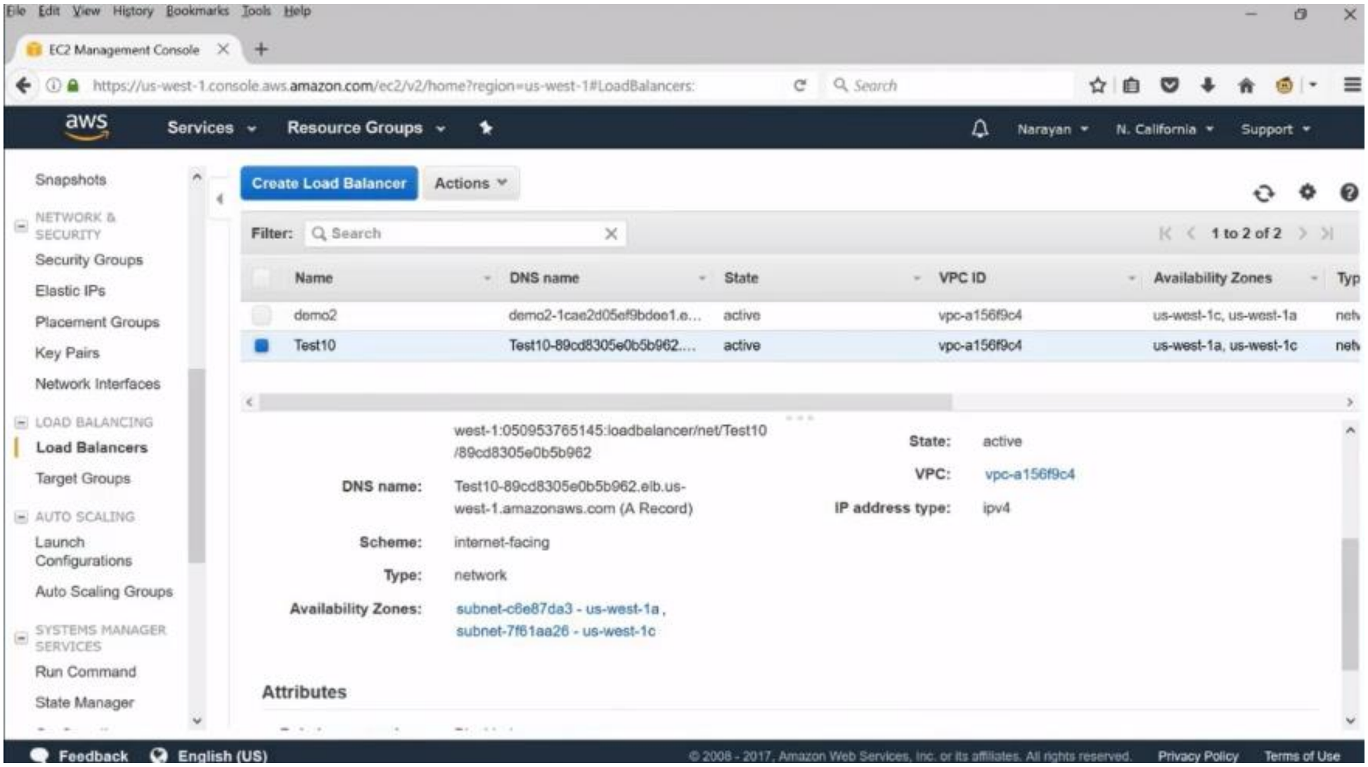
Hosted zone: Z24FKFUX50B4VW

State: active

VPC: vpc-a156f9c4

IP address type: ipv4

We can now go to the Load Balancers section of the EC2 service console and refresh the console, we can see that the Test10 NLB has been created



Name	DNS name	State	VPC ID	Availability Zones	Typ
demo2	demo2-1cae2d05ef9bdee1.e...	active	vpc-a156f9c4	us-west-1c, us-west-1a	netv
Test10	Test10-89cd8305e0b5b962....	active	vpc-a156f9c4	us-west-1a, us-west-1c	netv

west-1:050953765145:loadbalancer/net/Test10/89cd8305e0b5b962

DNS name: Test10-89cd8305e0b5b962.elb.us-west-1.amazonaws.com (A Record)

Scheme: internet-facing

Type: network

Availability Zones: subnet-c6e87da3 - us-west-1a , subnet-7f61aa26 - us-west-1c

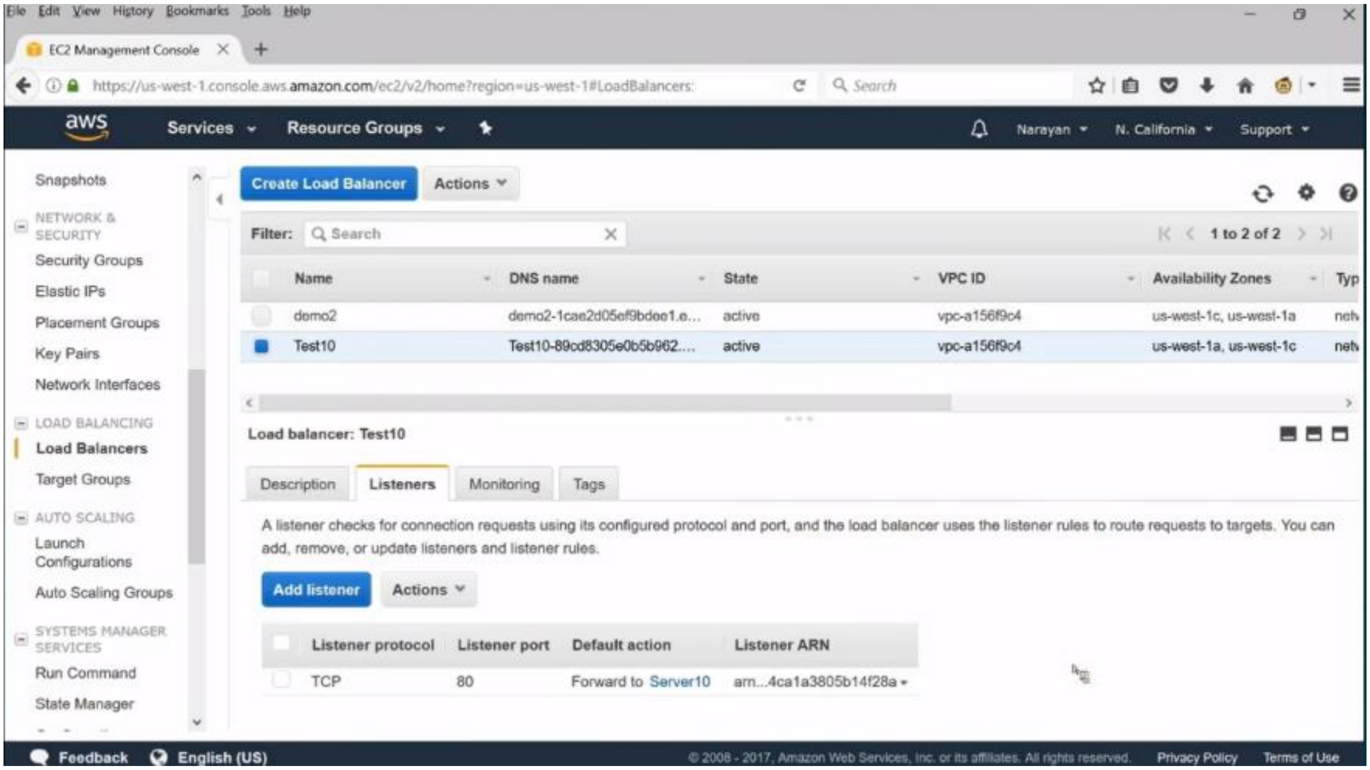
State: active

VPC: vpc-a156f9c4

IP address type: ipv4

Attributes

We see that it is a network type LB, internet-facing with 2 subnets in 2 different AZs,



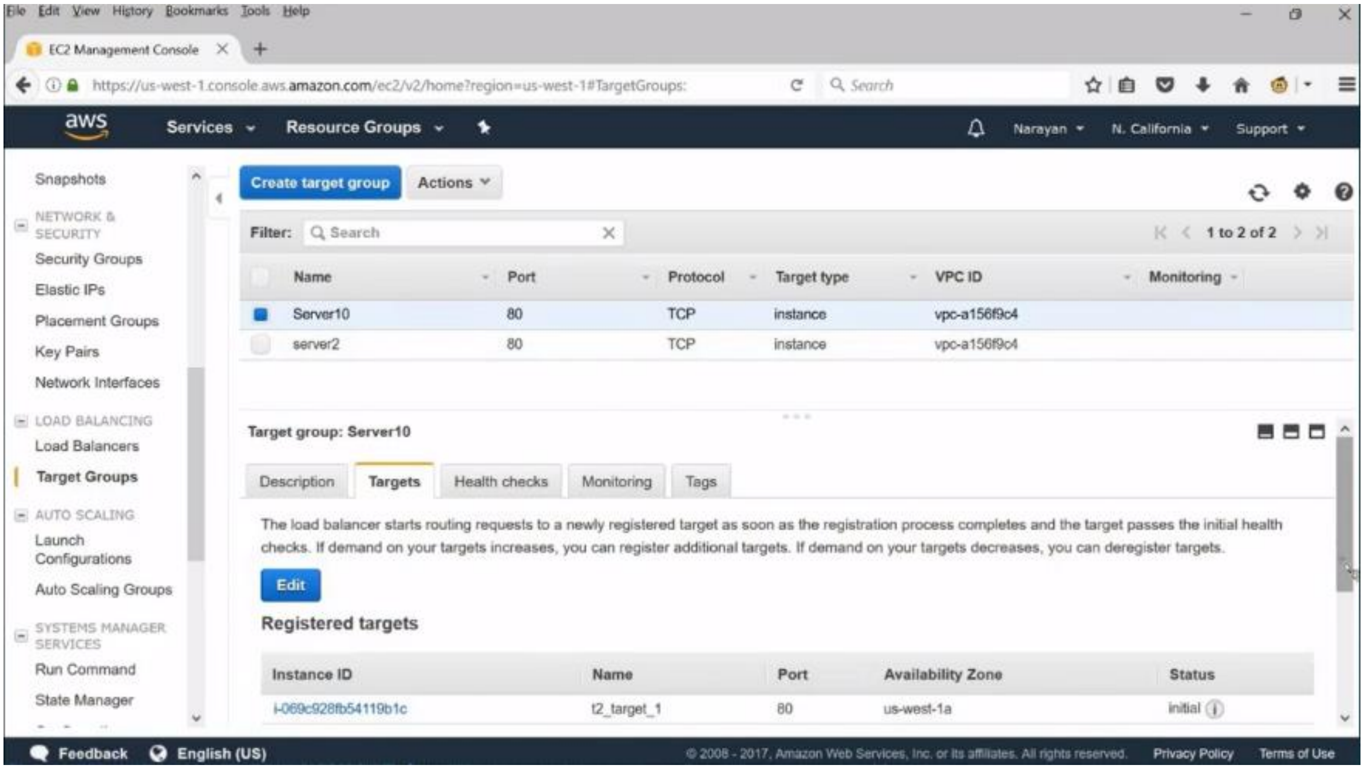
Name	DNS name	State	VPC ID	Availability Zones	Typ
demo2	demo2-1cae2d05ef9bdee1.e...	active	vpc-a156f9c4	us-west-1c, us-west-1a	netv
Test10	Test10-89cd8305e0b5b962....	active	vpc-a156f9c4	us-west-1a, us-west-1c	netv

Load balancer: Test10

A listener checks for connection requests using its configured protocol and port, and the load balancer uses the listener rules to route requests to targets. You can add, remove, or update listeners and listener rules.

Listener protocol	Listener port	Default action	Listener ARN
TCP	80	Forward to Server10	arn...4ca1a3805b14f28a

For this Test10 NLB, we also have a port 80 listener with TCP protocol that has the default action to forward to the target group called Server10.



Name	Port	Protocol	Target type	VPC ID	Monitoring
Server10	80	TCP	instance	vpc-a156f9c4	
server2	80	TCP	instance	vpc-a156f9c4	

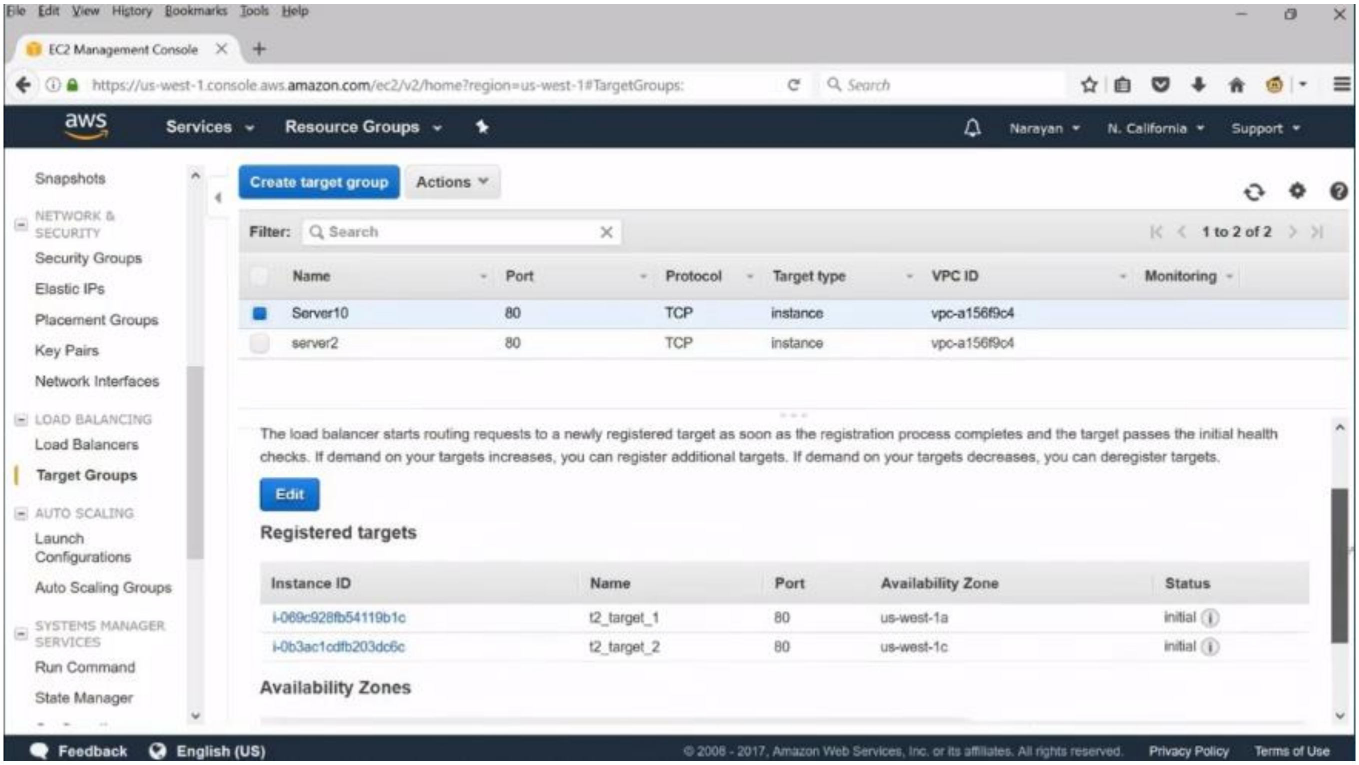
Target group: Server10

The load balancer starts routing requests to a newly registered target as soon as the registration process completes and the target passes the initial health checks. If demand on your targets increases, you can register additional targets. If demand on your targets decreases, you can deregister targets.

Registered targets

Instance ID	Name	Port	Availability Zone	Status
i-069c928fb54119b1c	t2_target_1	80	us-west-1a	initial

We can choose the Target Groups section to see the Server10 target group as above.



	Name	Port	Protocol	Target type	VPC ID	Monitoring
■	Server10	80	TCP	instance	vpc-a156f9c4	
□	server2	80	TCP	instance	vpc-a156f9c4	

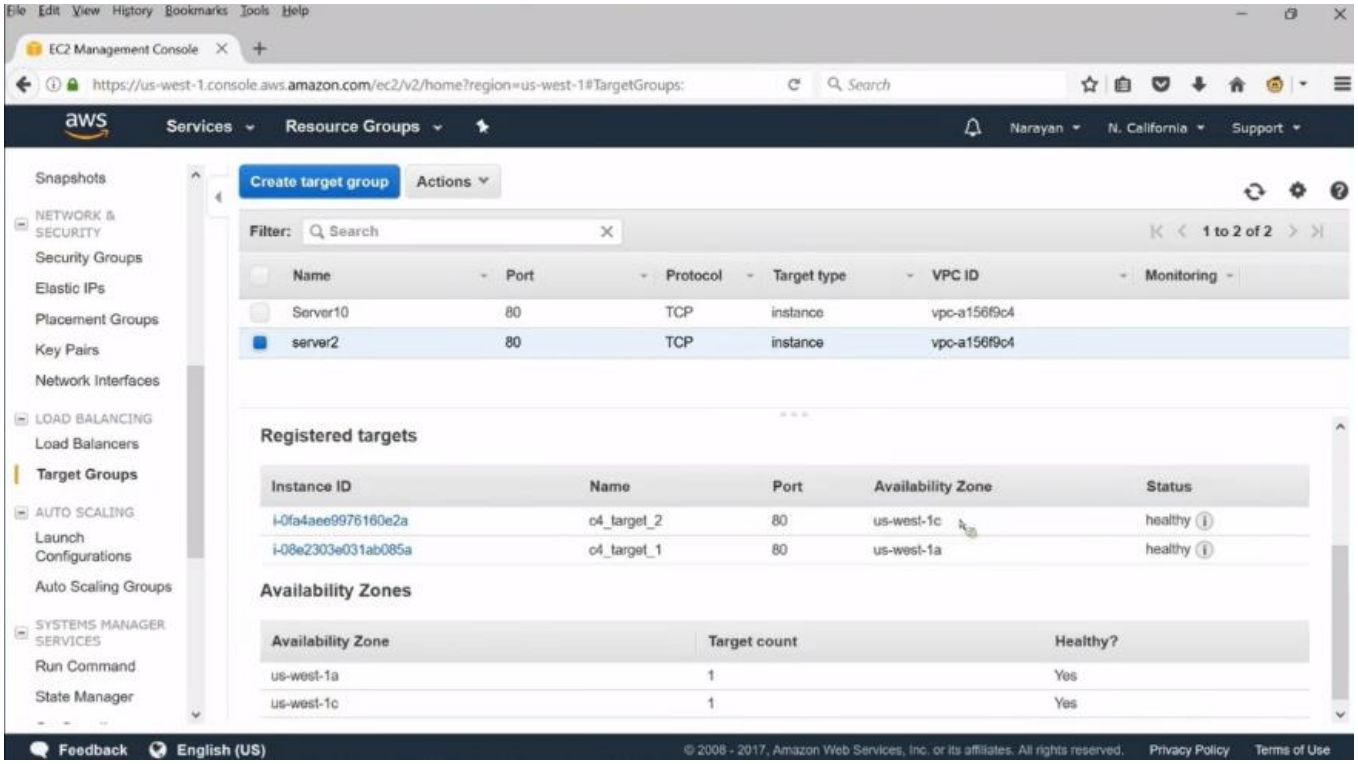
The load balancer starts routing requests to a newly registered target as soon as the registration process completes and the target passes the initial health checks. If demand on your targets increases, you can register additional targets. If demand on your targets decreases, you can deregister targets.

Registered targets

Instance ID	Name	Port	Availability Zone	Status
i-069c928fb54119b1c	t2_target_1	80	us-west-1a	initial
i-0b3ac1cdfb203dc6c	t2_target_2	80	us-west-1c	initial

Availability Zones

We can see the 2 registered targets that we created, one in each AZ. They both have their status as saying the health checks are still being initialized also.



	Name	Port	Protocol	Target type	VPC ID	Monitoring
□	Server10	80	TCP	instance	vpc-a156f9c4	
■	server2	80	TCP	instance	vpc-a156f9c4	

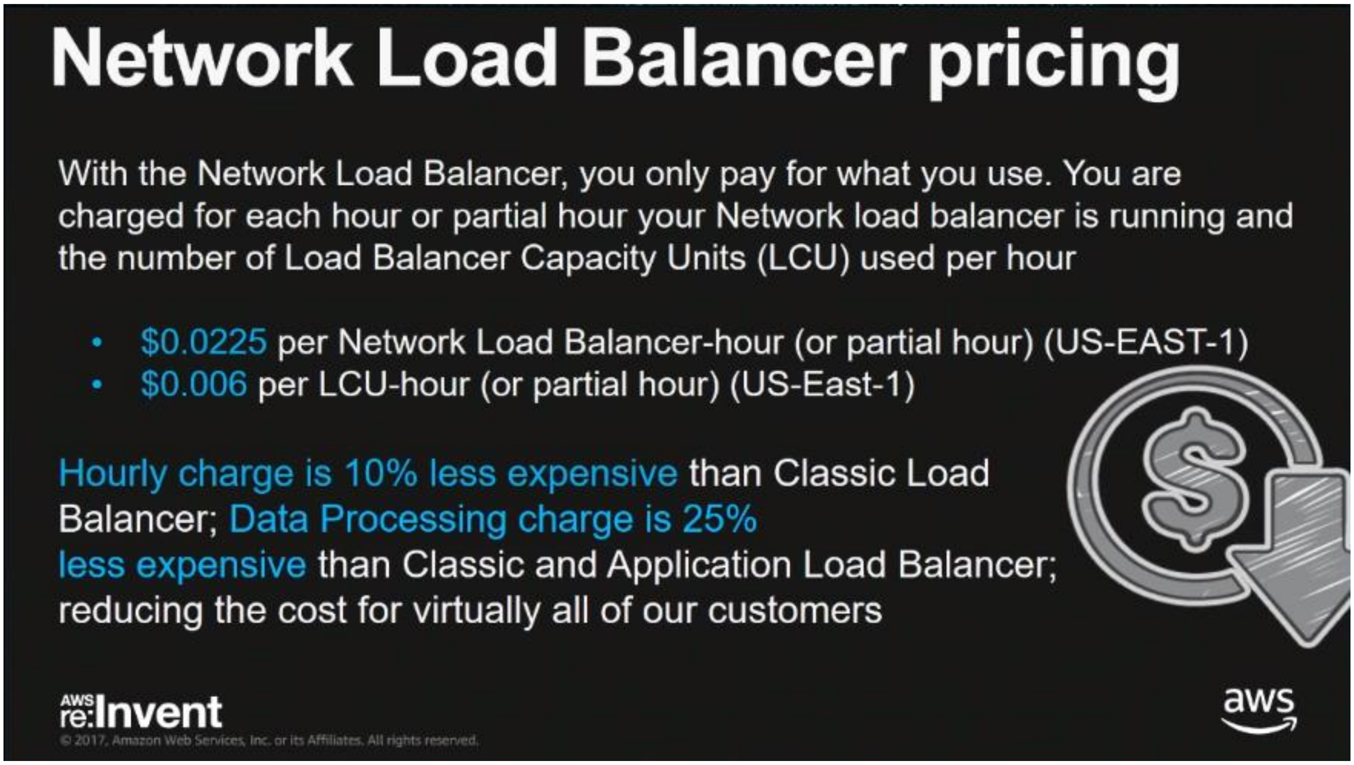
Registered targets

Instance ID	Name	Port	Availability Zone	Status
i-0fa4aee9976160e2a	c4_target_2	80	us-west-1c	healthy
i-08e2303e031ab085a	c4_target_1	80	us-west-1a	healthy

Availability Zones

Availability Zone	Target count	Healthy?
us-west-1a	1	Yes
us-west-1c	1	Yes

This is what will happen once the targets are completed and in a healthy state, their status will go from 'initial' to 'healthy' as seen above

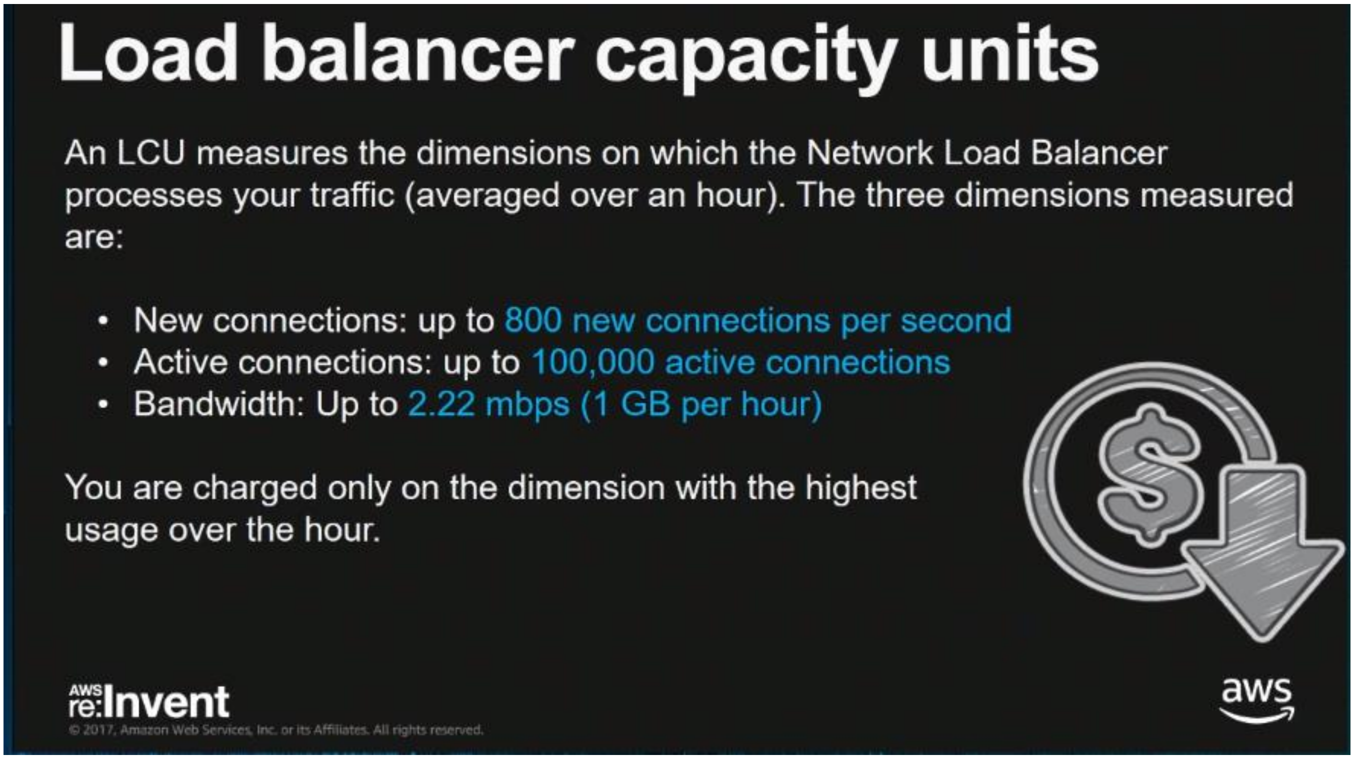


Network Load Balancer pricing

With the Network Load Balancer, you only pay for what you use. You are charged for each hour or partial hour your Network load balancer is running and the number of Load Balancer Capacity Units (LCU) used per hour

- $0.0225 per Network Load Balancer-hour (or partial hour) (US-EAST-1)
- $0.006 per LCU-hour (or partial hour) (US-East-1)

Hourly charge is 10% less expensive than Classic Load Balancer; Data Processing charge is 25% less expensive than Classic and Application Load Balancer; reducing the cost for virtually all of our customers

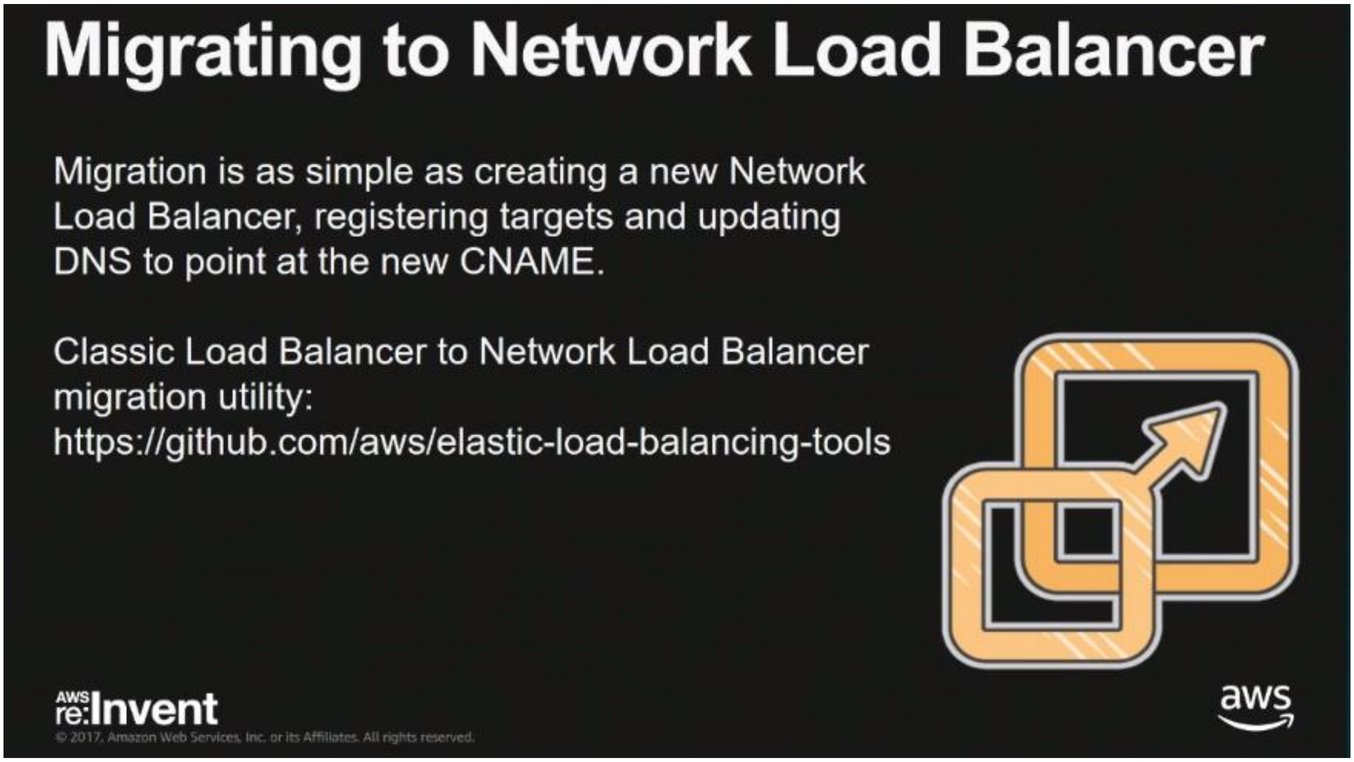


Load balancer capacity units

An LCU measures the dimensions on which the Network Load Balancer processes your traffic (averaged over an hour). The three dimensions measured are:

- New connections: up to 800 new connections per second
- Active connections: up to 100,000 active connections
- Bandwidth: Up to 2.22 mbps (1 GB per hour)

You are charged only on the dimension with the highest usage over the hour.

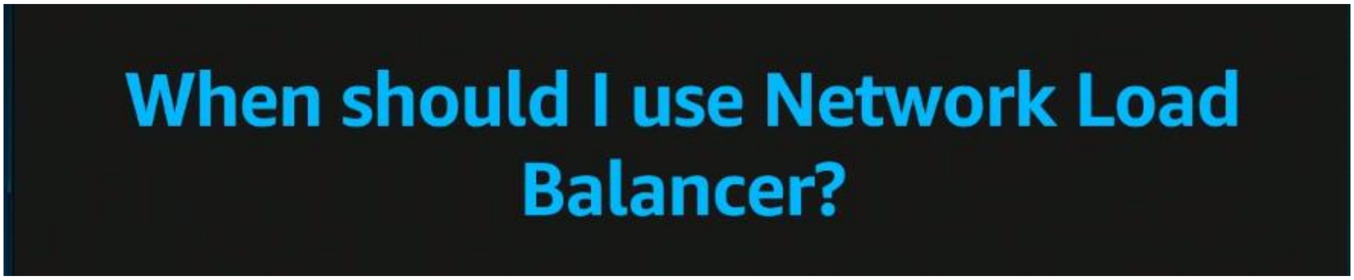


Migrating to Network Load Balancer

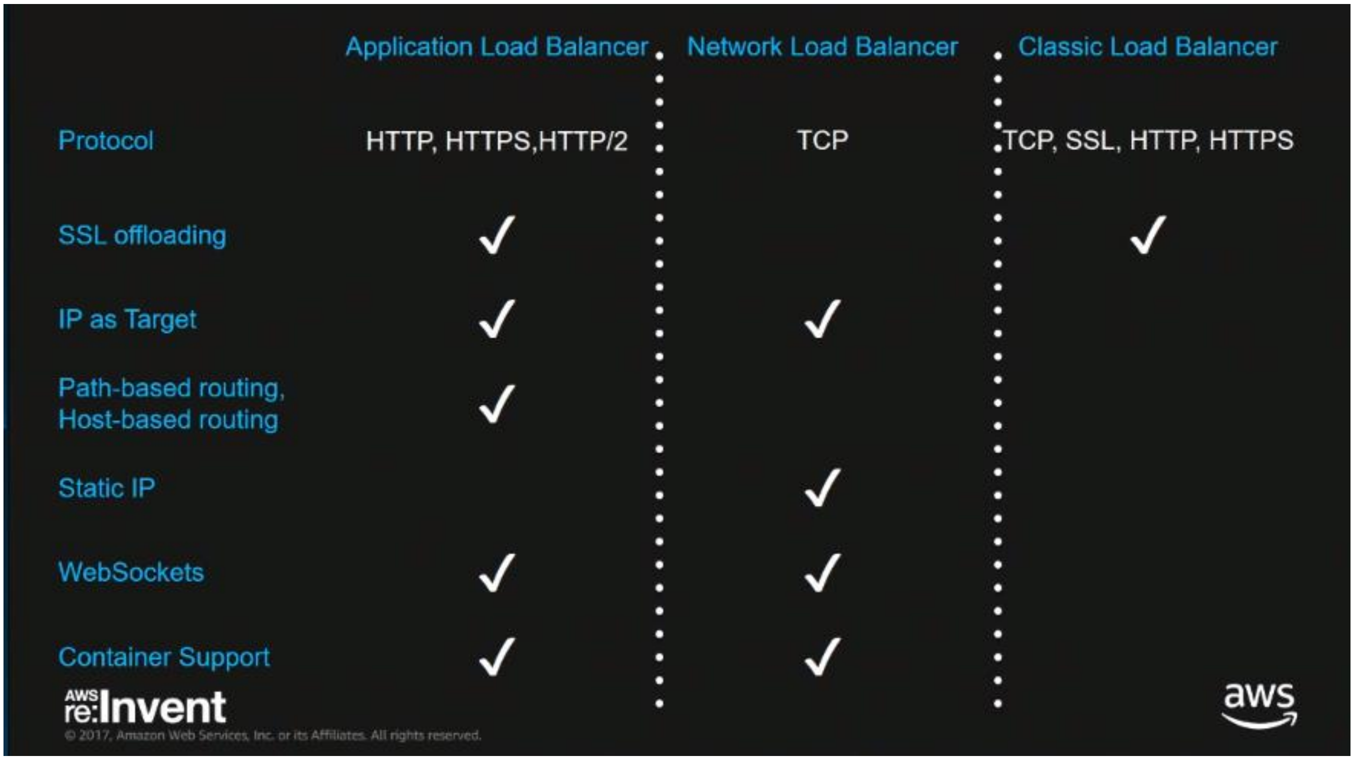
Migration is as simple as creating a new Network Load Balancer, registering targets and updating DNS to point at the new CNAME.

Classic Load Balancer to Network Load Balancer migration utility:
https://github.com/aws/elastic-load-balancing-tools

AWS re:Invent

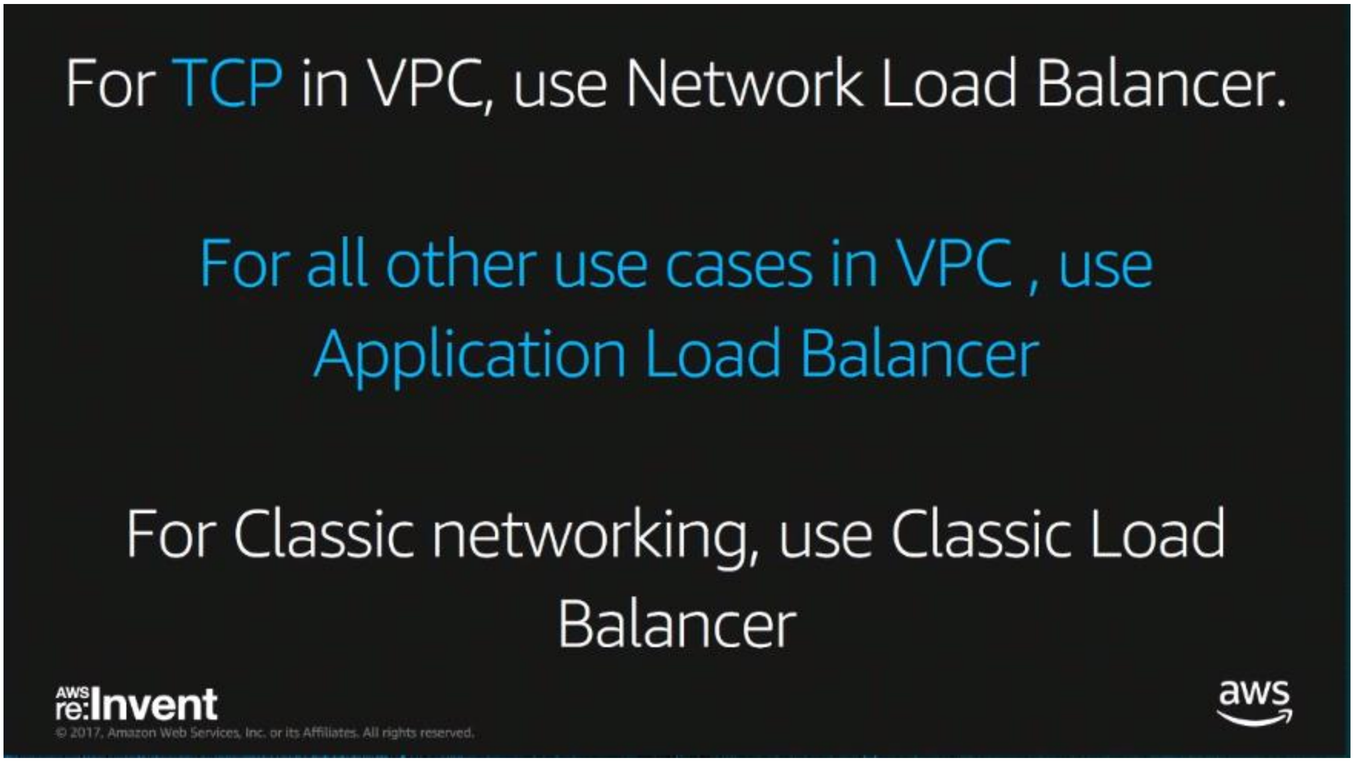


When should I use Network Load Balancer?



	Application Load Balancer	Network Load Balancer	Classic Load Balancer
Protocol	HTTP, HTTPS,HTTP/2	TCP	TCP, SSL, HTTP, HTTPS
SSL offloading	✓		✓
IP as Target	✓	✓	
Path-based routing, Host-based routing	✓		
Static IP		✓	
WebSockets	✓	✓	
Container Support	✓	✓	

AWS re:Invent



For TCP in VPC, use Network Load Balancer.

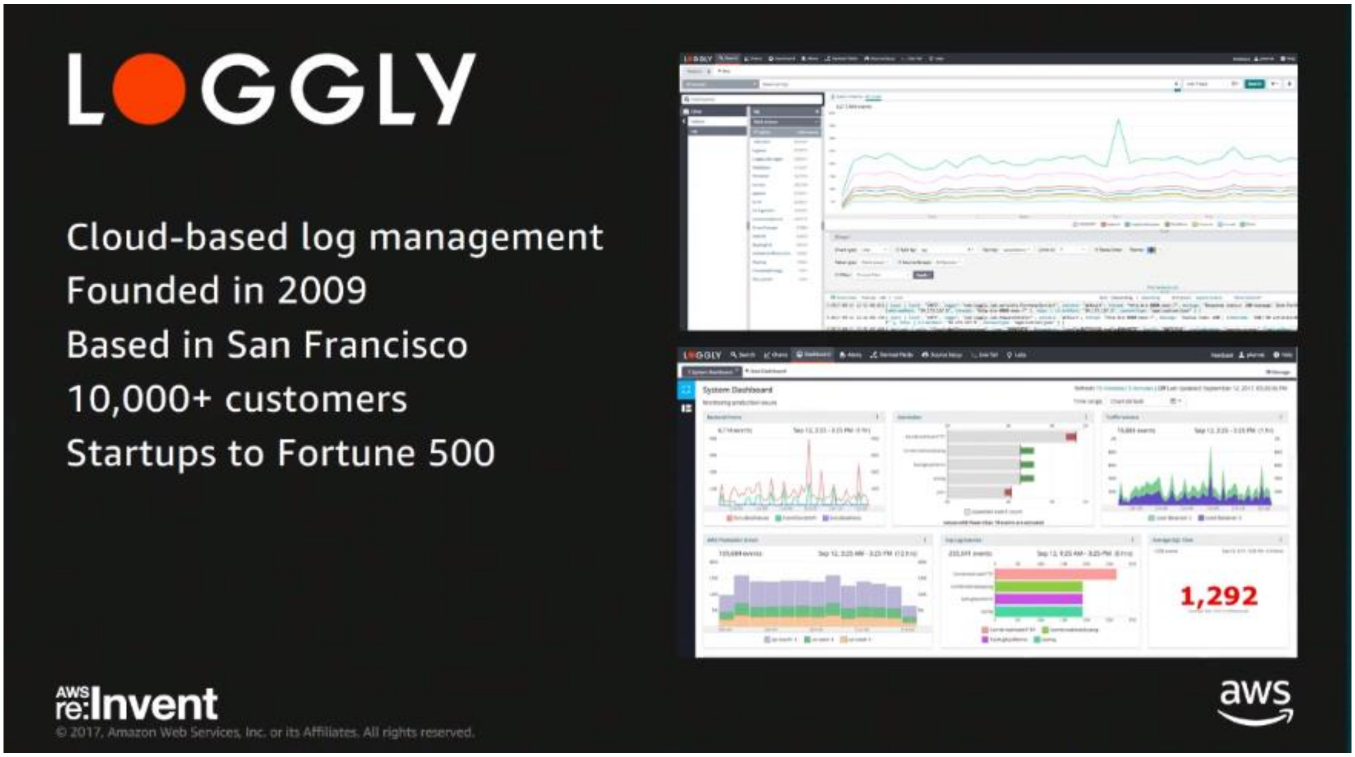
For all other use cases in VPC , use Application Load Balancer

For Classic networking, use Classic Load Balancer

AWS re:INVENT

AMAZON NLB AT LOGGLY

BRYAN McKENNEY, HEAD OF OPERATIONS

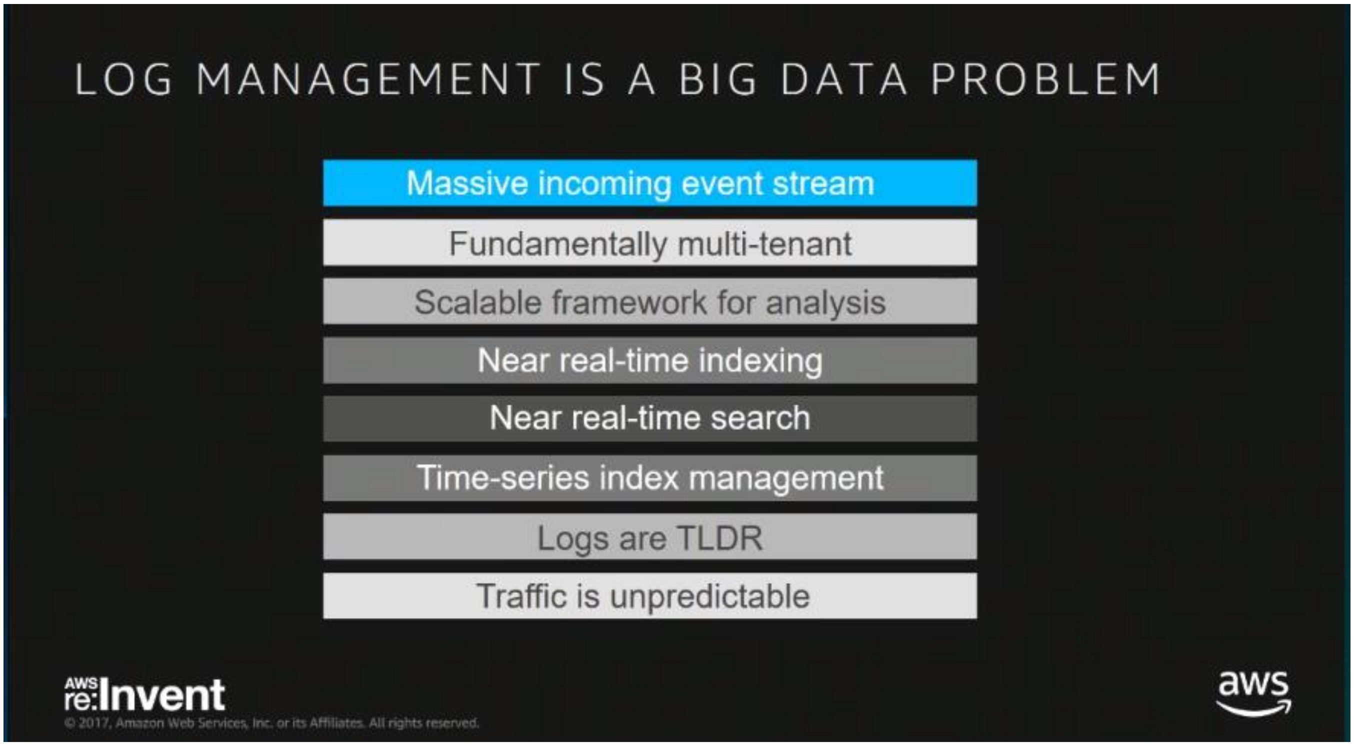


LOGGLY

Cloud-based log management
Founded in 2009
Based in San Francisco
10,000+ customers
Startups to Fortune 500

AWS re:Invent

aws



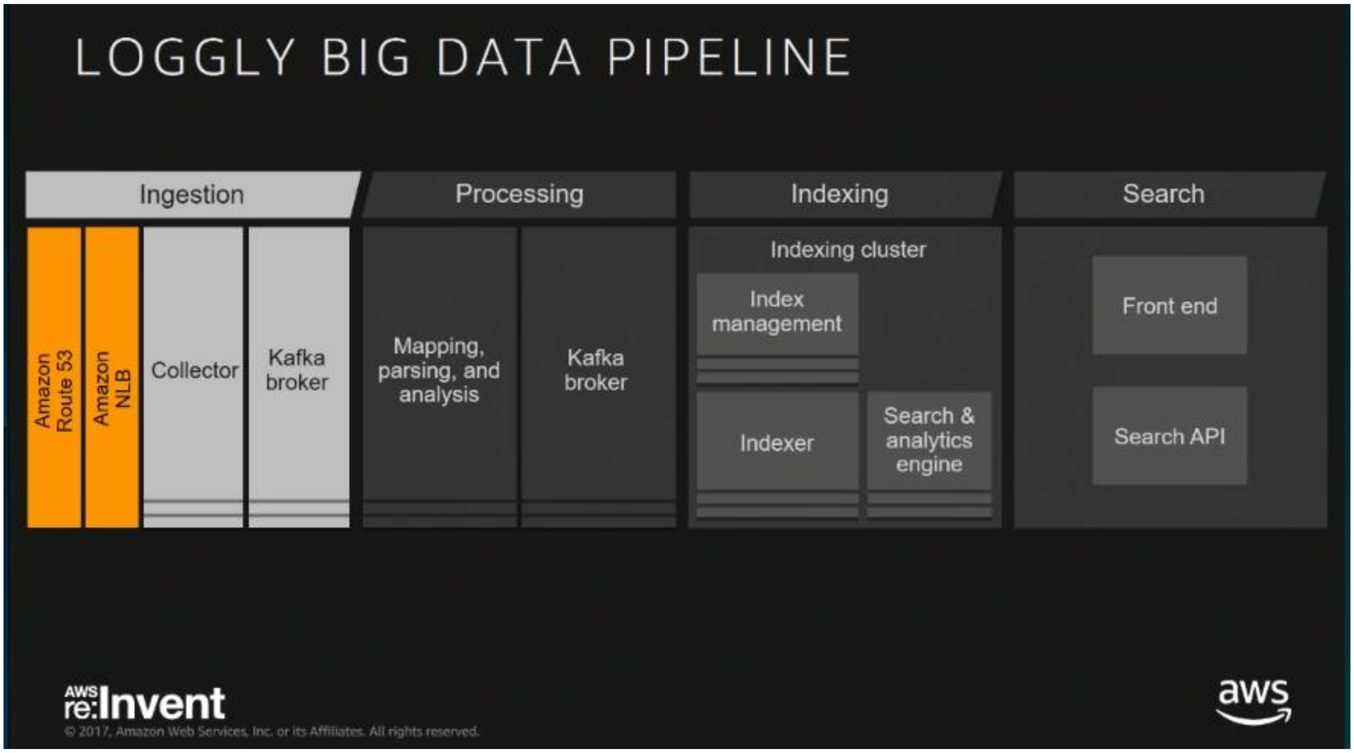
LOG MANAGEMENT IS A BIG DATA PROBLEM

- Massive incoming event stream
- Fundamentally multi-tenant
- Scalable framework for analysis
- Near real-time indexing
- Near real-time search
- Time-series index management
- Logs are TLDR
- Traffic is unpredictable

AWS re:Invent

aws

Big data services require a very resilient architecture.



LOGGLY BIG DATA PIPELINE

Ingestion

Amazon Route 53

Amazon NLB

Collector

Kafka broker

Processing

Mapping, parsing, and analysis

Kafka broker

Indexing

Indexing cluster

Index management

Indexer

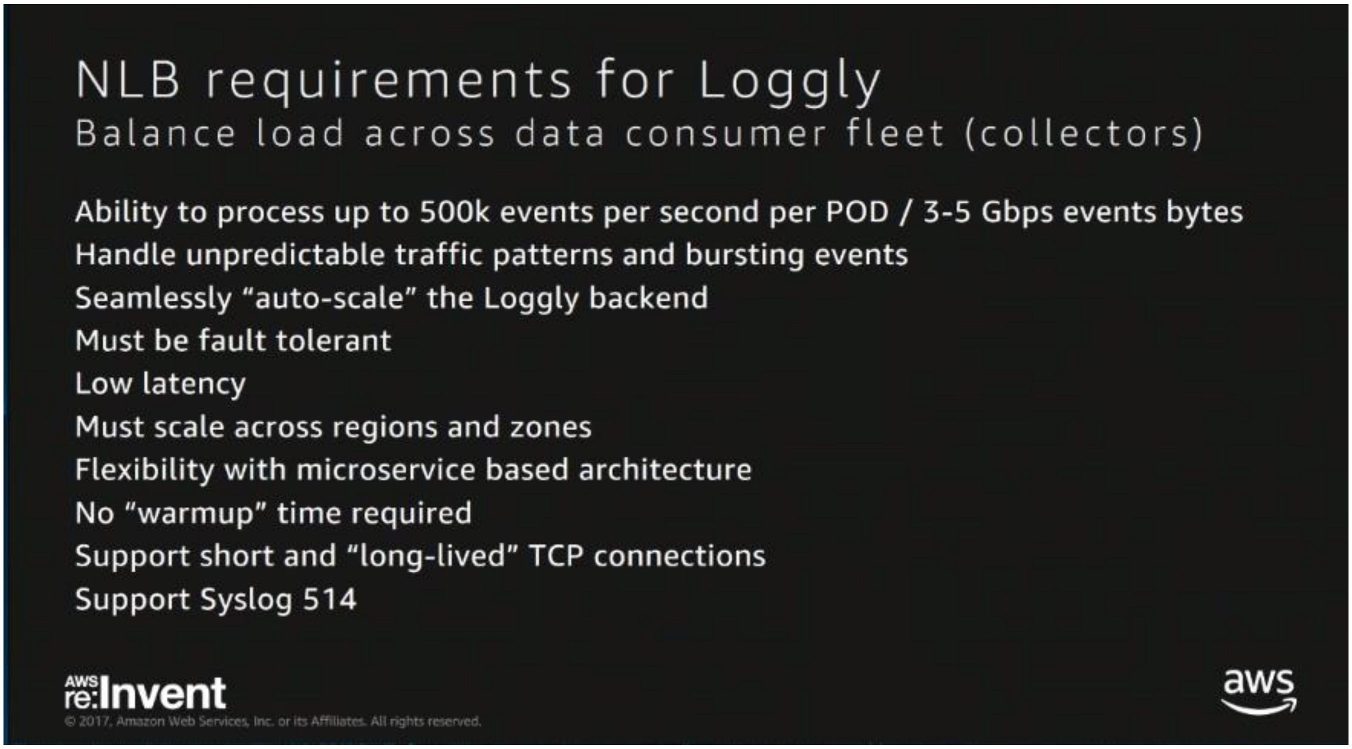
Search & analytics engine

Search

Front end

Search API

Loggly uses AWS route53 and NLB products to load balance data ingestion traffic, this is the entry point for customer logs into the pipeline. NLB evenly distributes the incoming data to a fleet of collectors where data validation is done, then pass on the data for processing, indexing, and make search available for customers. Routing and distributing loads to healthy collectors in the pipeline is one of the most important services.



NLB requirements for Loggly

Balance load across data consumer fleet (collectors)

Ability to process up to 500k events per second per POD / 3-5 Gbps events bytes

Handle unpredictable traffic patterns and bursting events

Seamlessly “auto-scale” the Loggly backend

Must be fault tolerant

Low latency

Must scale across regions and zones

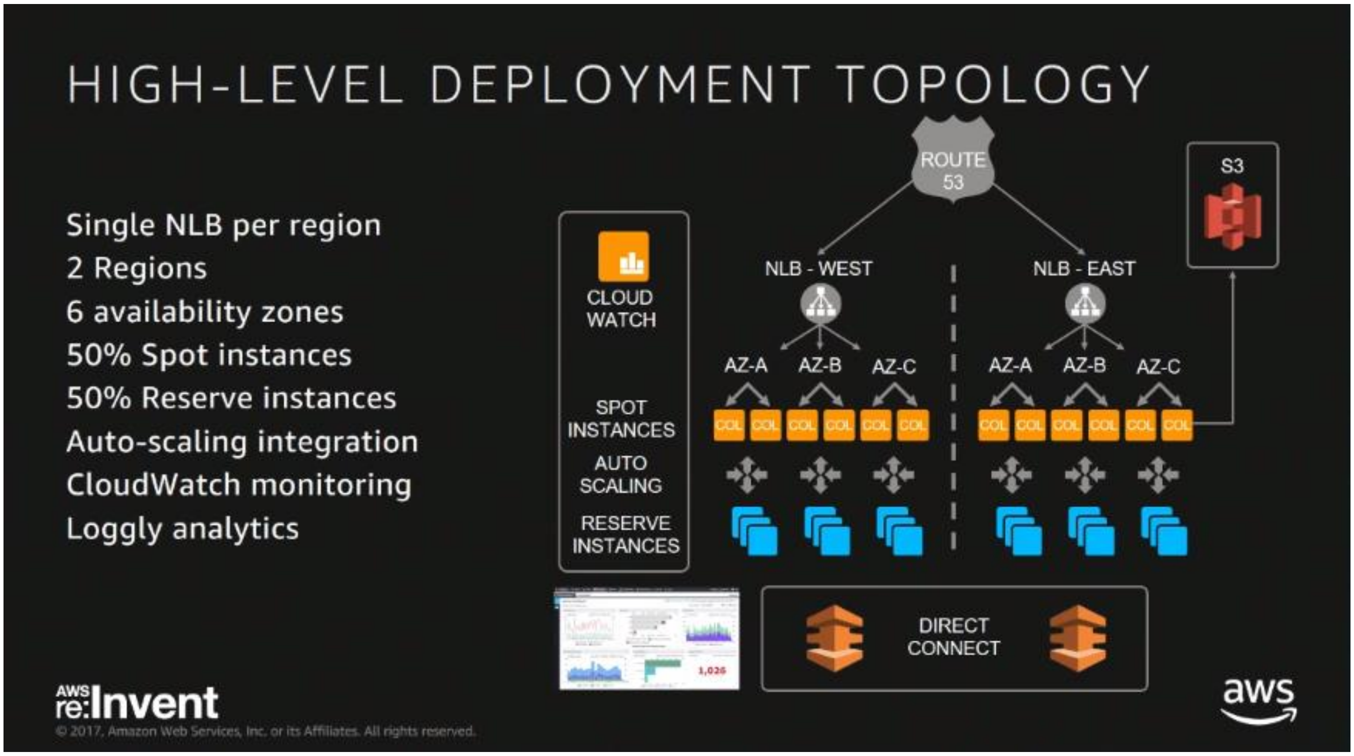
Flexibility with microservice based architecture

No “warmup” time required

Support short and “long-lived” TCP connections

Support Syslog 514

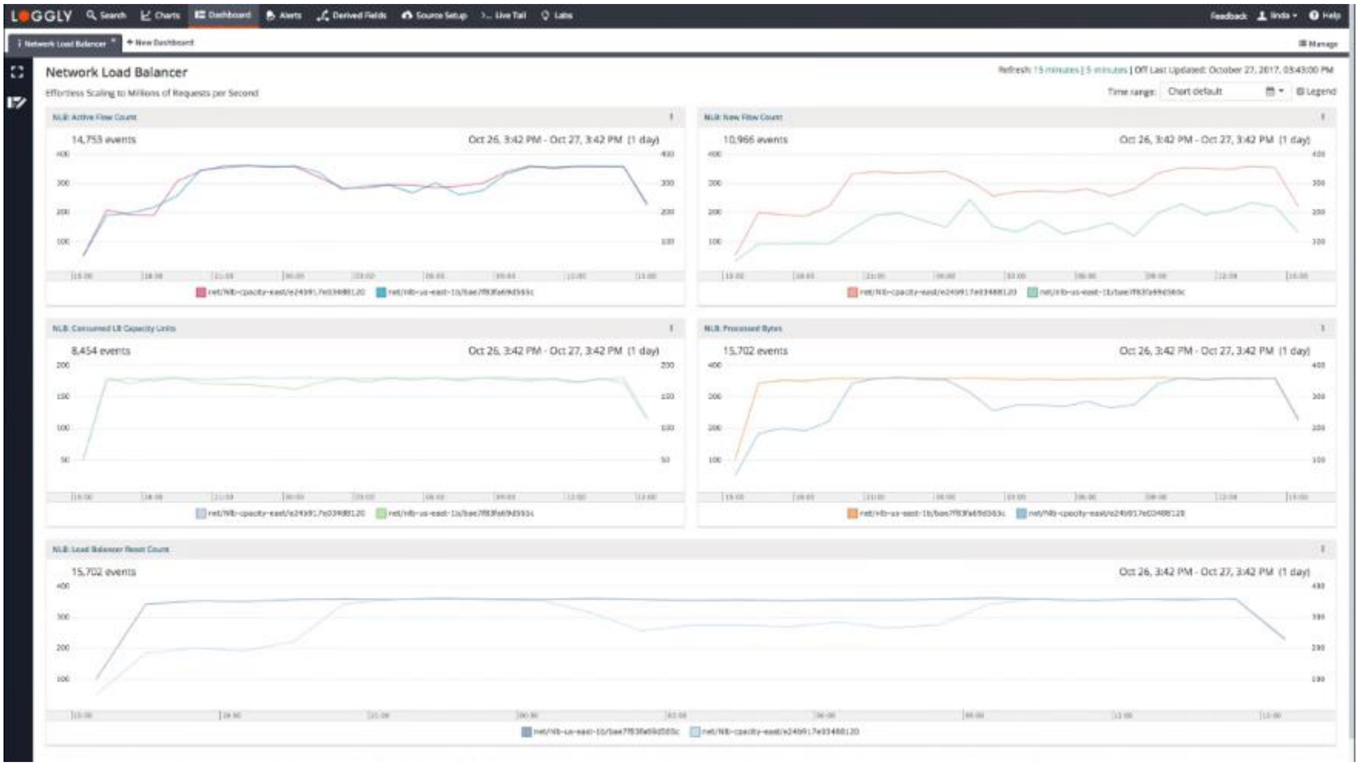
This system can adapt to different customer streaming profiles



HIGH-LEVEL DEPLOYMENT TOPOLOGY

Single NLB per region
2 Regions
6 availability zones
50% Spot instances
50% Reserve instances
Auto-scaling integration
CloudWatch monitoring
Loggly analytics

Loggly operates 2 NLBs, one in the west and the other in the east. We have 50% spot fleet instances upfront supported by another 50% spot fleet in the back to balance risk and cost for this topology. We use CloudWatch and ***Loggly Management Analytics*** to provide critical insights into our ***Ingestion Pipeline*** and Consumer Endpoint Health.



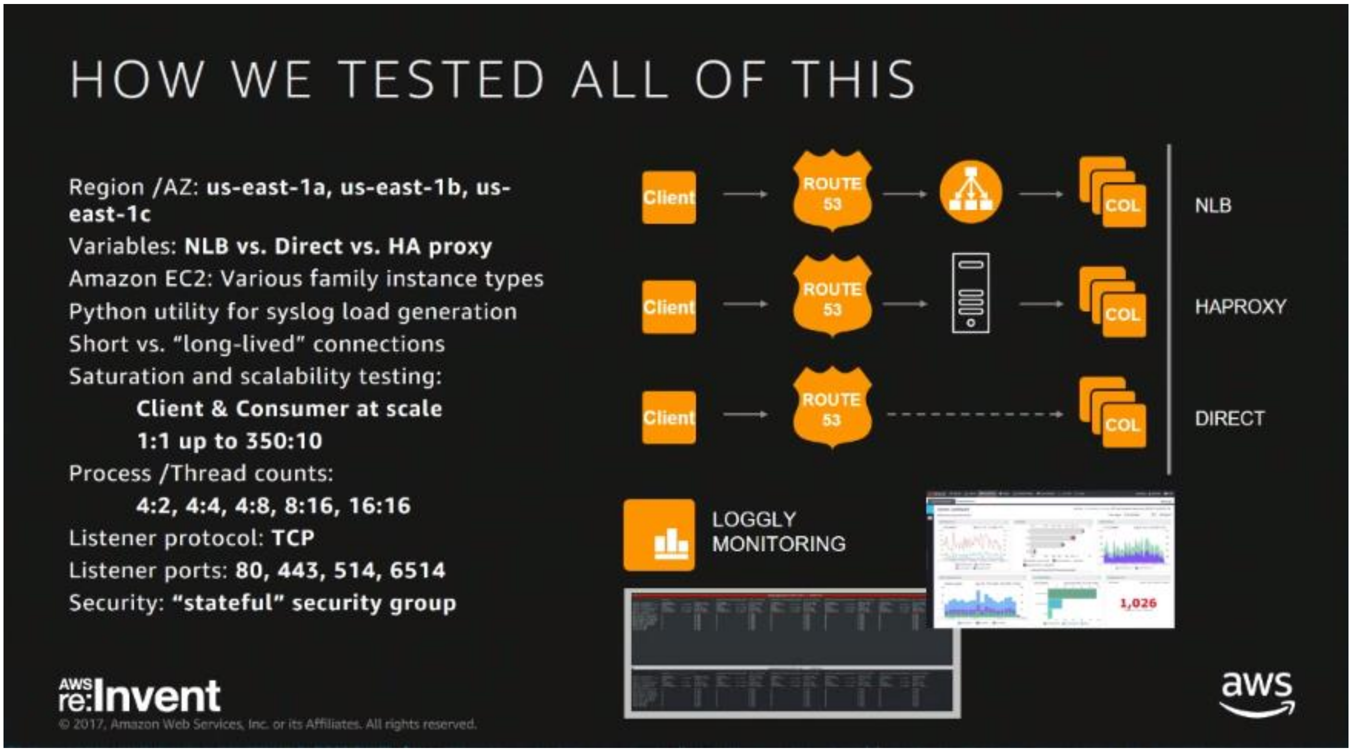
Network Load Balancer

Effortless Scaling to Millions of Requests per Second

Refresh: 15 minutes | 5 minutes | Off Last Updated: October 27, 2017, 03:43:00 PM

Time range: Chart default

This is one of the view used by operations and network engineers to check the health of incoming data flow, and the critical data patterns



HOW WE TESTED ALL OF THIS

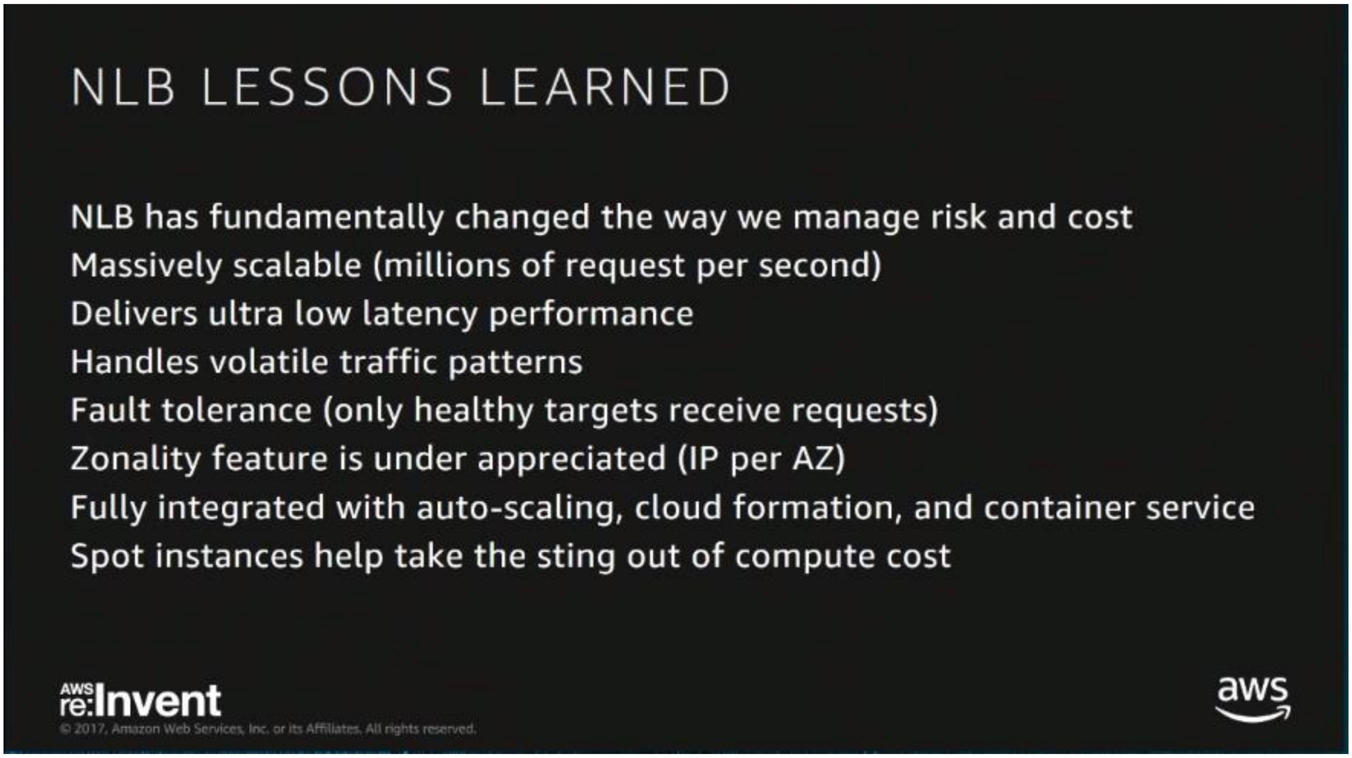
Region /AZ: **us-east-1a, us-east-1b, us-east-1c**
Variables: **NLB vs. Direct vs. HA proxy**
Amazon EC2: Various family instance types
Python utility for syslog load generation
Short vs. "long-lived" connections
Saturation and scalability testing:
  Client & Consumer at scale
  1:1 up to 350:10
Process /Thread counts:
  4:2, 4:4, 4:8, 8:16, 16:16
Listener protocol: **TCP**
Listener ports: **80, 443, 514, 6514**
Security: **"stateful" security group**

AWS re:Invent

aws

To test all these during the PoC phase. We started with historical review of all the incoming traffic volume data, throughput and performance benchmarks, then we used these historical data to establish specific NLB acceptance criteria that needs to be met. Our use case scope was to test both syslog long-lived connections and short-lived HTTP connections. We also need a balanced comparative testing perspective, so we setup 3 test harness variables Route53, NLB, and a generic HAProxy setup for sanity testing.

We started with 1 client sending simulated load to 1 connector through the NLB to establish a baseline, this establish the direct baseline for the HAProxy and the Direct testing. We then ramped up to 100s of clients sending loads to 10s of collectors, adjusting processing and threads to simulate customer profiles. The test results verified that the NLB can consistently manage load as the direct testing could with negligible differences.



NLB LESSONS LEARNED

NLB has fundamentally changed the way we manage risk and cost
Massively scalable (millions of request per second)
Delivers ultra low latency performance
Handles volatile traffic patterns
Fault tolerance (only healthy targets receive requests)
Zonality feature is under appreciated (IP per AZ)
Fully integrated with auto-scaling, cloud formation, and container service
Spot instances help take the sting out of compute cost

AWS re:Invent

aws