# Amazon ELBs [Elastic Loadbalancers]- Multi region architectures

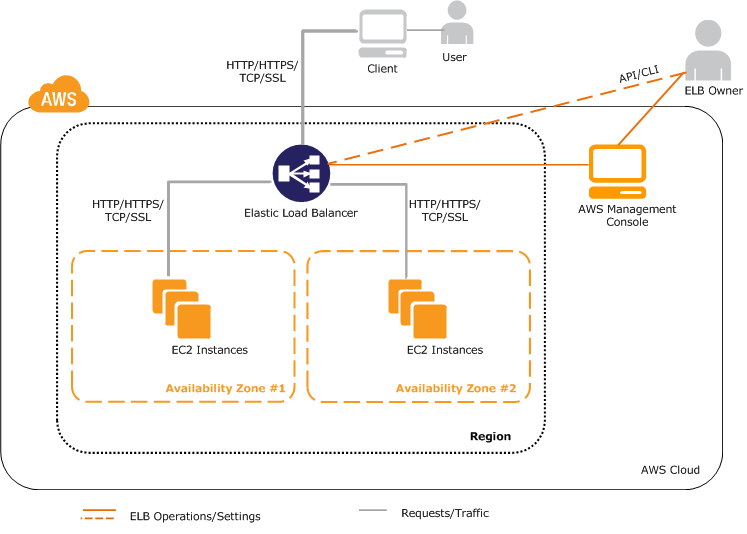
Elastic Load Balancing consists of two components: the load balancers and the controller service. The load balancers monitor the traffic and handle requests that come in through the Internet. The controller service monitors the load balancers, adding and removing load balancers as needed and verifying that the load balancers are functioning properly.

You have to create your load balancer before you can start using it. Each load balancer you create must have a unique Domain Name System (DNS) name. For example, if you create a load balancer named myLB in the us-east-1a, your load balancer might have a DNS name such as myLB-1234567890.us-east-1.elb.amazonaws.com.

You have to register the instances that you want to load balance with the load balancer. The instances are registered with the load balancer using the IP addresses associated with the instances. When an instance is stopped and then started, the IP address associated with your instance changes. This prevents the load balancer from routing traffic to your restarted instance. Elastic Load Balancing gives you the option to de-register your instance from the load balancer after you have stopped your instance, and then register the load balancer with your instance after you restart it.

**Architectural Overview of Elastic Load Balancing**

The following diagram shows how the various components of the Elastic Load Balancing work together.



This example assumes that you have created a load balancer, created a custom domain name and associated your load balancer with the domain name using a CNAME entry in DNS, and have registered your instances with it.

1. The client sends a URL request to DNS servers to access your application. The DNS server responds with a DNS name. For example, myLB-1234567890.us-east-1.elb.amazonaws.com.
2. The client looks for the resolution of the DNS name sent by the DNS server. The DNS entry is controlled by Amazon because your application instances are under the amazonaws.com domain. The Amazon DNS servers return one or more IP addresses.
3. The client then opens a connection to the machine at the provided IP address. The instance at this address is the load balancer you created.
4. The load balancer checks the health states of all the registered EC2 application instances within the selected Availability Zones and will begin routing traffic to instances that have met the healthy threshold defined in the health check configuration.
5. The load balancer routes the client request to the healthy EC2 application instance identified in the previous step. At this point, the client is communicating with one of your EC2 instances through your load balancer. The load balancer listeners can be configured to use either HTTP, HTTPS, TCP, or SSL protocols for both front-end connection (client to load balancer) and back-end connection (load balancer to back-end instance).

## Availability Zones and Regions

You can set up your Elastic Load Balancing to distribute incoming requests across EC2 instances in a single Availability Zone or multiple Availability Zones within a region. **Your load balancer does not distribute traffic across regions.**

For critical applications, you distribute incoming traffic across more than one Availability Zone. To distribute traffic across multiple Availability Zones, launch your Amazon EC2 instances in all the Availability Zones you plan to use, enable the Availability Zones for your load balancer, and then register your EC2 instances in all the enabled Availability Zones.

## Request Routing

Before a client sends a request to your load balancer, it first resolves the load balancer's domain name with the Domain Name System (DNS) servers. The DNS server uses DNS round robin to determine which load balancer node in a specific Availability Zone will receive the request.

The selected load balancer node then sends the request to healthy instances within the same Availability Zone using the leastconns routing algorithm. The leastconns routing algorithm favors back-end instances with the fewest connections or outstanding requests.

## Monitoring Elastic Load Balancing

Elastic Load Balancing provides the following metrics through Amazon CloudWatch (the documentation at <http://aws.amazon.com/documentation/cloudwatch/> provides detailed information on these metrics):

* Latency
* Request count
* Healthy hosts
* Unhealthy hosts
* Backend 2xx-5xx response count
* Elastic Load Balancing 4xx and 5xx response count

As you test your application, all of these metrics are important to watch. The particular items of interest are likely to be the Elastic Load Balancing 5xx response count, the backend 5xx response count, and latency.

#### Pre-Warming the Load Balancer

Amazon ELB is able to handle the vast majority of use cases for our customers without requiring "pre-warming" (configuring the load balancer to have the appropriate level of capacity based on expected traffic). In certain scenarios, such as when flash traffic is expected, or in the case where a load test cannot be configured to gradually increase traffic, we recommend that you [contact aws.amazon.com](https://aws.amazon.com/contact-us/)  to have your load balancer "pre-warmed". We will then configure the load balancer to have the appropriate level of capacity based on the traffic that you expect. We will need to know the start and end dates of your tests or expected flash traffic, the expected request rate per second and the total size of the typical request/response that you will be testing.

# Troubleshoot Elastic Load Balancing

Refer: http://docs.aws.amazon.com/ElasticLoadBalancing/latest/DeveloperGuide/elb-troubleshooting.html

Geographically Distributed Architecture with AWS

## Part-1 The Following diagram illustrates a sample AWS Multi Region HA architecture.

## 

**Now lets see how to address the above mentioned challenges:**  
 **Workload Migration:** Amazon S3 or EBS backed AMI’s will operate only in regional scope inside AWS. We need to create the same AMI’s in another AWS  region again for inter region HA architectures. Every time when a code deployment is made, applications need to synchronize the executable /jars/configuration files across the AWS regions. Use of Automated deployments like Puppet, Chef will make things easier for such ongoing deployment cases.  Point to Note: In addition to AMI's ; Amazon EBS, ElasticIP’s etc also operate in AWS Regional scope.

**Data Synch:** Any complex system will have data distributed in variety of Data sources like Database, NoSQL, Caches and File Storage.  Some of the preferred techniques which we recommend for AWS Multi Region Synch are:

* Database: MySQL Master-Slave replication, SQL Server 2012 HADR replication, SQL Server 2008 replication, Programmatic RDS replication
* File Storage: Gluster File Storage Replication, S3 Programmatic replication
* Cache: Since Cache replication across regions are too costly for many use cases, it is recommended to follow Cache warming inside every AWS Regions.
* Aspera for High Speed File Transfer

Since most these techniques are relying on Asynchronous Replication model, companies need to be aware of the Data loss, RPO and RTO they can incur in Architecting Multi Region AWS High Availability.

**Network Flow:**It is the ability to enable flow of network traffic between multiple AWS regions. Now let us see the points to consider in this:

* Since Amazon Elastic Load Balancers currently cannot transfer requests across AWS regions, it cannot be used for this Inter AWS Region High Availability Architecture
* Load Balancers/RP’s like Nginx or HAProxy deployed on EC2 on an AWS Region can do this, but during outages where an entire amazon region itself is affected, there are chances that the RP EC2’s are also affected and they cannot direct the requests to another AWS region effectively. This will lead to website failure.
* It is usually a recommended practice to achieve this User Network traffic re-direction at Managed DNS level. Using solutions like UltraDNS, Akamai or Route53 LBR (Latency based routing) we can shift or balance traffic between infrastructures hosted on different AWS regions.
* Using Amazon Route 53’s Latency Based Routing (LBR) feature, we can now have instances in several AWS regions and have requests from our end-users automatically routed to the region with the lowest latency. We need to enter the EC2 instance public IP, Elastic IP, or Elastic Load Balancer target in the Route 53 console for LBR to happen. This LBR feature can be used for designing GEO Distributed Infrastructures and High Availability Architectures across AWS regions. Behind the scenes, AWS is constantly gathering anonymous internet latency measurements and storing them in a number of Relational Database Service instances for processing. These measurements help them build large tables of comparative network latency from each AWS region to almost every internet network out there.
* Amazon ElasticIP’s are also not transferrable across AWS regions. FTP and other IP Based TCP endpoints used or hardcoded for App-to-App communication needs to be re-mapped or resolved using DNS accordingly. This is an important point to be considered in AWS Multi Region Deployments.

## Part 2:

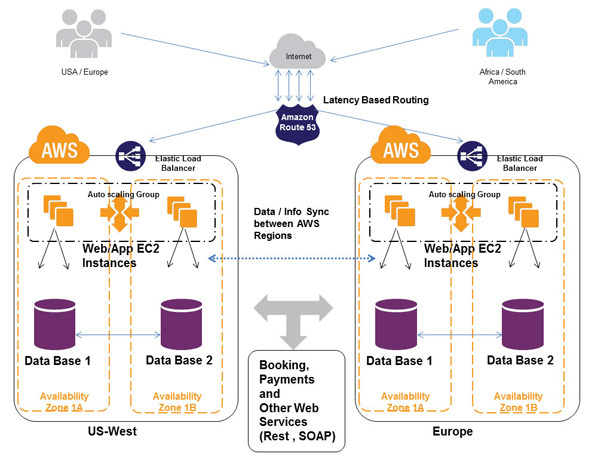
User requests originating all around the world are directed to the nearest AWS region or AWS region with lowest network latency (more precisely). For example, suppose you have Load balancers in the Sao Paulo, US-West,Europeregion and we have created a latency resource record set in Route 53 for each load balancer.

An end user in Paris enters the name of our domain in their browser, and the Domain Name System routes their request to a Route 53 name server. Route 53 refers to its data on latency between Paris and the EuropeEC2 region and between Paris and the Sao Paulo/USEC2 region.

If latency is lower between Paris and the Europe region( most of the times), Route 53 responds to the end user’s request with the IP address of your load balancer in the Amazon EC2 data center in Europe EC2 region.

If latency is lower between Paris and the Sao Paulo region, Route 53 responds with the IP address of the load balancer in the Amazon EC2 data center in Sao Paulo. This architecture rapidly cuts down the latency and gives the user a better experience. Also incase one of the regions is facingnetwork problems, the requests can be routed to alternate low latency region achieving High Availability at overall application level. Though this architecture has benefits it comes with various complexities depending upon the use case and technical stack used, we will uncover some of them below

The following diagram illustrates a simple multi-tiered technical stack on AWS used by the online cruise company.



* Route 53 is used as the DNS Layer
* Amazon CloudFrontis used for delivering Static assets
* Latency based routing is configured on Route53
* Elastic Load Balancers are created on US-West, Europe, Sao Paulo region
* Latency record sets are created in Route 53 and pointed to these Elastic Load Balancers in different regions
* Web/App EC2 instances are launched with Amazon Auto Scaling in Multiple-AZ’s. Infrastructure in every region is individually elastic and auto scaled. They can seamlessly expand and handle the traffic from a single geography and during regional outages they are capable to handle traffic directed from other regions as well.
* RDS MySQL can be used on Multi-AZ mode for storing data inside a single region. In case multi-region synchronization of data is required, it is suggested to use raw MySQL EC2 instances in M-MMM or M-SSS slave mode.
* All the HTTP, AJAX calls will pass through the Route 53 to the ELB. Amazon ELB will balance the request to the Web/App EC2 based on the algorithm configured. The Web/APP EC2 will access the database, process the result and send the response. The dynamic content is delivered from Web/App EC2 and static assets from CloudFront CDN. Some of the data that requires synchronization are replicated between AWS regions using SQS + custom programs, Web Services or MySQL Asynchronous replications.
* Centralized or Shared services (Booking, Payments etc) can be accessed over internet using HTTP/S, Web Services by the Web/App EC2. Shared services can be implemented in one of the AWS regions or on an External Data center as well.

# How much does it cost to use Route 53 and LBR?

**Hosted Zones**

$0.50 per hosted zone / month for the first 25 hosted zones

$0.10 per hosted zone / month for additional hosted zones

**Standard Queries**

$0.500 per million queries – first 1 Billion queries / month

$0.250 per million queries – over 1 Billion queries / month

**Latency Based Routing Queries**

$0.750 per million queries – first 1 Billion queries / month  
$0.375 per million queries – over 1 Billion queries / month

**Alias queries for ELBs free of charge**

**What are the key benefits of using Route53Latency Based Routing?**

* Better performance than running in single AWS Region
* Improved reliability relative to running in a single region
* Easier implementation and much lower prices than traditional DNS solutions

**What are the negatives of using Route 53+Geo Distributed Architecture?**

* Route 53 LBR does not have much to crib about, but on the whole I would be happy if AWS Route 53 team can bring directional traffic routing and other algorithms available inproducts like UltraDNS to its portfolio in coming months. This will give architects better control in designing High Availability, Disaster Recovery and Geo Distributed solutions for more use cases.
* The sample Tech stack architecture that I had illustrated may not be applicablefor some use cases. In such scenarios , where there are more complex systems like Search Servers, Cache, NoSQL, Queue, ESB servers etcin your stack, things might get very complicated while designing Geo Distributed solutions.
* Since most of the AWS services operate on regional scope, it may pose problems while designing Geo Distributed architecture

# [Method to configure GEO DNS/LBR feature using AWS Route53](http://www.linuxfunda.com/2014/01/12/how-to-configure-geo-dns-feature-using-aws-route53/)

You can send visitors to different servers based on country of their IP address using Amazon Route 53 cloud based dns server. For example, if you have a server in America, and a server in Singapore, then you can easily route traffic for visitors in Asia go to the Singapore server and those in the rest of the world be served by the American server. This will results into the various kinds of benefits such as:

1. **Better performance** as you are sending web site visitors to their nearest web server.
2. **Reduced load** on origin.
3. **Geomarketing**/online advertising.
4. **Restricting content** to those geolocated in specific countries (I am not a big fan of DRM).
5. In some cases you can get potentially **lower costs** and more

## AWS Route 53 routing policy

From the Route 53 documents:

If your application is hosted on Amazon EC2 instances in multiple EC2 regions, you can reduce latency for your end users by serving their requests from the EC2 region for which network latency is lowest. Route 53 latency-based routing lets you use DNS to route end-user requests to the EC2 region that will give your users the fastest response.

It is possible to use Route 53's Latency Based Routing (LBR) feature with non-AWS endpoints or IP address. Route 53 don't restrict what IPs or CNAMEs you can tag with a region. AWS dns server will route traffic to those IPs/CNAMEs "as if" those IPs or CNAMEs were hosted in whichever AWS region you choose to tag. The following is the current list of regions supported by Route 53 LBR:

1. Nothern Virginia, US: us-east-1
2. Nothern California, US: us-west-1
3. Oregon, US: us-west-2
4. Ireland, EU: eu-west-1
5. Singapore, Asia: ap-southeast-1
6. Tokyo, Asia: ap-northeast-1
7. Sydney, Asia: ap-southeast-2
8. Sao Paulo, South America: sa-east-1

## Before you start with Route53 hosted zone configuration,

1. create two instances (one in us-east-1a and another one in ap-southeast-1 singapore)
2. Login to the instances and install apache webserver and post a sample webpage
3. Verify the correct loading of the web pages using the Public IP address of both servers

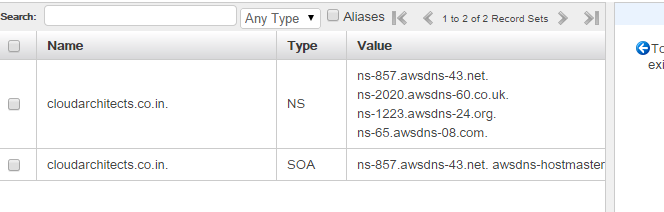
|  |  |
| --- | --- |
|  |  |
| **US-east-1a** | **Singapore(ap-southeast-1)** |

## Route 53 Configuration

First, open the Amazon Route 53 console at <https://console.aws.amazon.com/route53/> >

**Step 1: Create Hosted Zone**

1. Click on “Create Hosted Zone”
2. Write you domain name. For me it’s “cloudarchitects.co.in”
3. Put a comment for the domain
4. Click on “Create Hosted Zone”
5. Now you will find a row like the below image.



**Step 2: Add the Name Servers to your DNS (in my case it is godaddy.com)**

After adding a Hosted Zone in Route 53 you will get 4 Name Servers as “Delegation Set” (As shown in below image). Edit your DNS name server and put these values in it.

**Step 3:** **Add Record Set to Hosted Zone**

Now we will add a DNS record to our Hosted Zone.

### CNAME geodns settings for USA server

1. Set **Name** to www.l.
2. Choose **Type** to CNAME.
3. Set **TTL** to 30 seconds.
4. Set **Value** to w.usa.cloudarchitects.co.in.
5. Set **Routing Policy** to Latency.
6. Set **Region** to us-east-1.
7. Set **ID** to US-EAST, Data Center.
8. Click the **Create Record set** button to save the changes.

|  |
| --- |
|  |

**Step 4:** **Set an IP address for CNAME w.usa.cloudarchitects.co.in**

Visit the Amazon Route 53 console at <https://console.aws.amazon.com/route53/> > Choose your existing domain > Click on the **Go to Record Sets** button > Click on the **Create Record Set** button.

1. Set **Name** to w.usa.
2. Set **Type** to A - IPv4 address.
3. Set **TTL** to 30 seconds.
4. Set **Value** to 1.1.1.1. Please replace IP address 1.1.1.1 with your actual AWS endpoint or any other valid public IP address in US. (in this case the Instance running in us-east-1a)
5. Set **Routing Policy** to Simple.
6. Click the **Create Record set** button to save the changes.

|  |
| --- |
|  |

**Step 5: CNAME geodns settings for Singapore server**

1. Set **Name** to www.l.
2. Choose **Type** to CNAME.
3. Set **TTL** to 30 seconds.
4. Set **Value** to w.sg.cloudarchitects.co.in.
5. Set **Routing Policy** to Latency.
6. Set **Region** to ap-southeast-1.
7. Set **ID** to AP-SOUTHEAST, Data Center.
8. Click the **Create Record set** button to save the changes.

|  |
| --- |
|  |

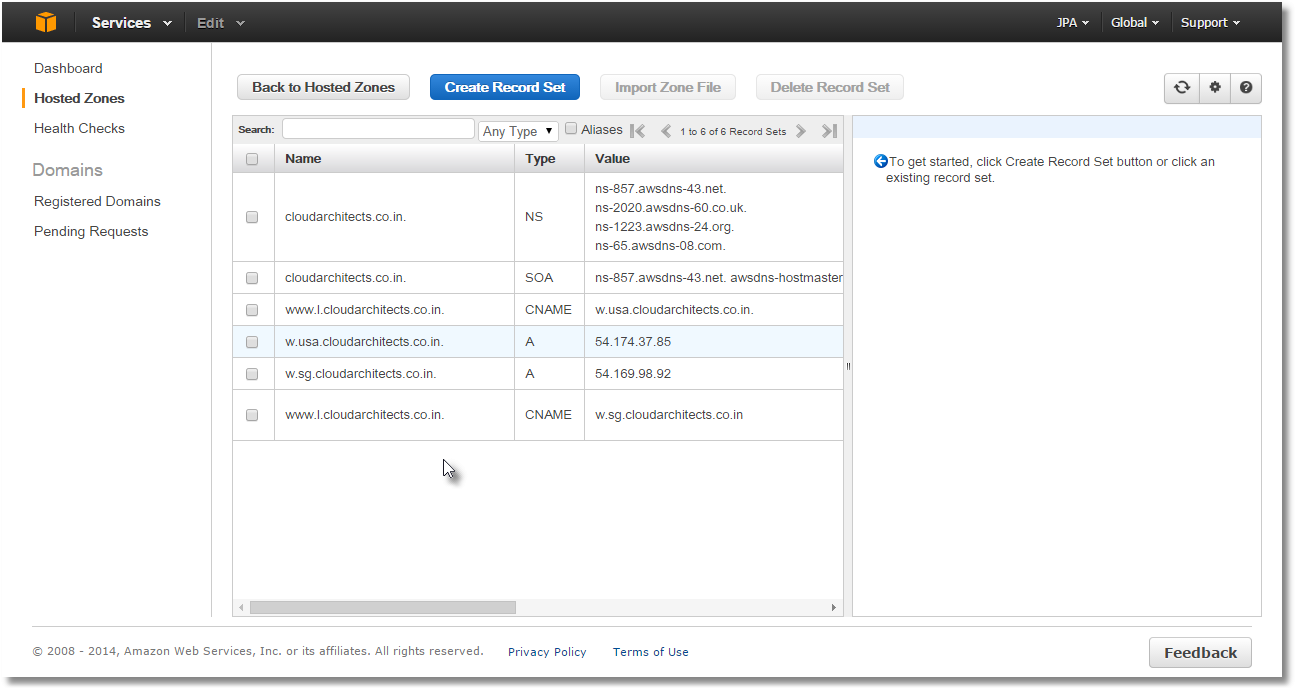
**Step 6: Set an IP address for CNAME w.sg.cloudarchitects.co.in**

Visit the Amazon Route 53 console at <https://console.aws.amazon.com/route53/> > Choose your existing domain > Click on the **Go to Record Sets** button > Click on the **Create Record Set** button.

1. Set **Name** to w.sg.
2. Set **Type** to A - IPv4 address.
3. Set **TTL** to 30 seconds.
4. Set **Value** to 2.2.2.2. Please replace IP address 2.2.2.2 with your actual AWS endpoint or any other valid public IP address in US. (in this case the Instance running in us-east-1a)
5. Set **Routing Policy** to Simple.
6. Click the **Create Record set** button to save the changes.

|  |
| --- |
|  |

**Once all Records are created, it will look like the below one:**

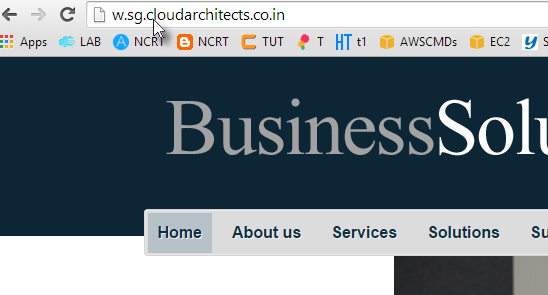


**Test both the DNSs:**

**1)**



2)



## Test it

Use the following dig or host dns lookup command line utilities to verify new settings:  
% dig +short www.l.cloudarchitects.co.in

1) dig using Singapore based DNS server [128.199.248.105 ns1.sg.dns.d0wn.biz Singapore OpenNIC Tier2 DNS powered by d0wn.biz ~ns1.sg.dns.d0wn.biz]

[ec2-user@ip-172-31-17-48 ~]$ dig +short www.l.cloudarchitects.co.in @128.199.248.105

w.sg.cloudarchitects.co.in.

54.169.98.92

2) US based DNS server [69.26.129.2 www.nalsinvestments.com Newbury Park]

[ec2-user@ip-172-31-17-48 ~]$ dig +short www.l.cloudarchitects.co.in @69.26.129.2

w.usa.cloudarchitects.co.in.

54.174.37.85