zadara

Zadara Cloud Services - Load Balancing User Guide

Release 24.03

Zadara

Feb 24, 2025

CONTENTS

CHAPTER

ONE

LOAD BALANCING

1.1 Overview

In case of high volume application traffic which must be distributed between multiple VM instances, Zadara Cloud Services supports the definition of load balancers and target groups. Load balancers distribute the load between the VM instances defined in the target group. This capability provides the following benefits when compared to working with individual VM instances:

- Time required to complete a task is reduced.
- Fault tolerance and high availability can be provided in case of VM instance failure.

When you create a load balancer, you define a target group of instances that will share the work of processing requests from an application. The application then directs its requests to the load balancer, and the load balancer distributes the work among the instances in the load balancer's target group.

See the video demonstrating the basics of creating and configuring zCompute Load Balancers, Target Groups and Listeners:

✔ **Note:** The load balancing capability must be globally enabled and configured by a system administrator, before load balancers can be created and used.

1.2 Load Balancer

1.2.1 Creating a Load Balancer

To create a load balancer:

1. Navigate to **Home > Load Balancing > Load Balancers**. The list of currently defined load balancers is displayed.
2. To create a new load balancer, click **+ Create** from the top toolbar.
3. In the **Create Load Balancer** dialog which opens, enter the following:
    1. **Name**: The new load balancer's name.
    2. **Description**: An optional description for the load balancer.
    3. **Type**: The load balancer type determines the basis on which the load balancing is done. The following types are supported:
        - **ALB**: Application Load Balancer does distribution between targets based on HTTP/HTTPS sessions.

- **NLB**: Network Load Balancer does distribution between targets based on TCP sessions regardless of the application layer protocol.

4. **VPC**: From the dropdown, select the VPC to be associated with the load balancer.
5. **Subnet**: From the dropdown, select an existing subnet to be associated with the load balancer, or click **+** to define a new subnet.
6. **Elastic IP**: From the dropdown, select an existing Elastic IP to be associated with load balancer, or click **+** to define a new Elastic IP.
7. **Security Group**: Determine the Security Groups that are associated with the load balancer, on the basis of the following selection:
   - **Standard**: To select Security Groups that are created and managed by the user and can be updated at any time. It's the user responsibility to open the listeners' ports and to restrict to specific sources if required.

     From the dropdown, select existing Security Groups to be associated with load balancer, or click **+** to define and associate a new group.
   - **Managed**: To select Security Groups that are created and managed by the Load Balancer service. Listeners' ports will be automatically open to any incoming source.

   ✔ **Note:** This setting can't be changed after the load balancer creation.
8. **Instance Type**: From the dropdown, select the instance type (CPU, RAM, boot disk size) to be associated with load balancer and to cope with the expected load.
9. **IP Address**: Optional <place holder>
10. **High Availability**: Check or uncheck the high availability option. When selected, additional load balancers will be created for standby, to automatically replace a failed load balancer.
11. **Tags**: Enter tags that can be used for ease of identifying the load balancer and as a filter in searches.

4. Click **Finish**.

✔ **Note:** When you create a load balancer instance, the system creates a number of internal resources (including a VM). These are considered "protected resources." To preserve the integrity of the load balancer instance, the system prevents you from deleting or modifying these resources.

1.3 Target Groups

A target group is a group of instances to which a load balancer directs application traffic. The instances in this group collectively do the processing work that the application requires.

1.3.1 Creating a Target Group

To create a target group:

1. Navigate to **Home > Load Balancing > Target Groups**.

   The list of currently defined target groups is displayed.

2. To create a new target group, click **+ Create** from the top toolbar.

   The **Create Target Group** dialog will open.

   1. In the **Details** tab, enter the following:

      1. **Name**: The name of the new target group.
      2. **Description**: Optional description of the new target group.
      3. **Protocol**: Select the protocol used by the load balancer to access the target group.

         - **HTTP**: For an Application Load Balancer (ALB).
         - **TCP**: For a Network Load Balancer (NLB).

      4. **Default Port**: The target group port used by the load balancer to connect to this target group.
      5. **Sticky Session** - By default, a load balancer routes each request independently to the registered target group instance with the smallest load. However, you can use the sticky session feature to bind a user's session to a specific instance. This ensures that all requests from the user during the session are sent to the same instance.

         1. **Duration** - if sticky session is selected, enter session duration in seconds during which load balancer should consistently route the user's request to the same VM instance.

   2. Click **Next**.
   3. The **Health Check** tab configurations are used by the load balancer to determine whether the target is healthy:

      1. **Port**:

         - **Traffic Port**: The health check connection will be established on the port where the target accepts incoming traffic.
         - **Custom**: The health check connection will be established on a specified port for all targets in the group.

           Enter the Custom port number.

      2. **Interval**: The interval in seconds between health checks of an individual VM instance.

         - Minimum: 5 seconds.
         - Maximum: 300 seconds.

      3. **Timeout**: The time in seconds beyond which no response means a failed health check. The value must be less than the interval's maximum value.

         - Minimum: 2 seconds.
         - Maximum: 60 seconds.

      4. **Healthy Threshold**: The number of consecutive successful health checks needed to transition a VM instance to the healthy state.

         - Minimum: 2 seconds.
         - Maximum: 10 seconds.

5. **Unhealthy Threshold**: The number of consecutive failed health checks needed to transition a VM instance to the failed state.
   - Minimum: 2 seconds.
   - Maximum: 10 seconds.
6. **HTTP Health Check**: For health checks using the HTTP protocol, configure the following:
   1. **Path**: The ping path used by the load balancer to do a health check on VM instances in the target group.
   2. **HTTP Code(s)**: The HTTP health check response codes that determine a successful health check, and therefore a healthy target.

      The following formats are supported:
      - A single HTTP success code, for example, **200**.
      - Multiple HTTP success codes separated by commas, for example **200,301**.
      - A range of HTTP success codes, specified with "-" between the first and last code in the range, for example, 200-399.
   3. Click **Next**.

4. In the **Targets** tab, add targets to the group.
   - **Target Type**: Select the Target Type, click **Add**, and enter the target's parameters.

     ✔ **Note:** All targets in a Target Group must be of the same Target Type.

     The following Target Types are supported:
     - **Instance**:

       Enter the Instance type target's parameters:
       1. **Select Target VM**: From the dropdown, select a target VM instance.
       2. **Port**: Specify the port number on the target for communication with the load balancer.
     - **IP**:

       Enter the IP type target's parameters:
       1. **IP**: Enter the target's IP address, that is both in your region and is reachable from the load balancer.
       2. **Port**: Specify the port number on the target for communication with the load balancer.

   1. To configure additional targets in the Target Group click **Add**, and enter the target's parameters according to the Target Type.
   2. Click **Finish**.

1.4 Listeners

A listener is a process that monitors a specified port for incoming connection requests. The listener's rules determine how the load balancer routes these requests to the targets.

Every load balancer requires at least one listener, each of which must have one or more rules.

A listener's rule consists of a condition and a resulting action. When the condition is met, the load balancer executes the corresponding action for that rule.

When multiple rules are configured for a listener, they are evaluated sequentially based on their order. If the condition of the first rule is not met, the subsequent rules are assessed in turn. When a rule's condition is met and its corresponding action executed, no further subsequent rule is assessed for the incoming connection request. The final rule in the sequence includes a default condition indicating that none of the preceding rules matched the incoming request, and the load balancer will execute its corresponding action.

The order of all the rules except for the final one can be changed by dragging and dropping a rule into its preferred position in the sequence.

1.4.1 Creating a Listener

There are some differences in listener configuration options, depending on the type of load balancer:

- Creating a Listener for NLB type Load Balancer
- Creating a Listener for ALB type Load Balancer

1.4.2 Creating a Listener for NLB type Load Balancer

1. Navigate to **Home > Load Balancing > Target Groups**.

   The list of currently defined target groups is displayed.
2. Select the **NLB** type load balancer for which the listener is being created.
3. In the load balancer's lower pane's **Listeners** tab, click **+ Create**.
4. In the **Create Listener** dialog:
   1. **Protocol**: The Protocol is set to **TCP** and cannot be modified.
   2. **Port**: Enter the listener's port.
   3. **Rule**:

      ✔ **Note:**

      - It is not possible to create multiple rules for an NLB type load balancer's listener. Only the default rule is applicable and required. As a sole required rule, there is no condition for assessment.
      - The resulting **Action** is to **Forward** the incoming requests to a selected Target Group.

      1. **Action**: The rule's Action is to **Forward** the incoming requests to the Target Group specified in **Forward to** below. The Action cannot be modified for an NLB type load balancer's rule.
      2. **Forward to**: Select the destination Target Group from the dropdown, or click **+** to create a new one.

         See Creating a Target Group.
   4. Click **Finish**.

1.4.3 Creating a Listener for ALB type Load Balancer

1. Navigate to **Home > Load Balancing > Target Groups**.

   The list of currently defined target groups is displayed.

2. Select the **ALB** type load balancer for which the listener is being created.
3. In the load balancer's lower pane's **Listeners** tab, click **+ Create**.
4. In the **Create Listener** dialog:
   1. **Protocol**: Select **HTTP** or **HTTPS**.

      For **HTTPS**, select or upload a **Default SSL Certificate**.

      Multiple SSL certificates per target group are supported for LBaaS. Each certificate configuration requires a certificate, a private key, and a certificate chain.

   2. **Port**: Enter the listener's port.
   3. **Rules**
      1. Define the listener's rules, starting with the default rule's action:
      2. To create additional rules, click **+Add Rule**.
         1. Click **+ Add Condition**, select one of the conditions from the dropdown and enter its parameter values:

Condition	Parameter values
Header	Header name and value patterns
Path	Path patterns
Host	Host names
Method	One or more of the HTTP methods: • **GET** • **HEAD** • **POST** • **PUT** • **PATCH** • **DELETE** • **OPTIONS**
IP	The CIDR blocks that contain the source IP addresses. Click **Add CIDR** to add another CIDR block.

         2. Click **+ Add Condition** to add another condition to the same rule.

            ✔ **Note:** Multiple conditions can be configured in a single rule for a listener of an ALB type load balancer.

      3. **Action**: Configure the rule's action, corresponding to the condition.

         Select the applicable action, and enter its parameter values:

Action	Parameter values
Forward	**Forward to**: Select the destination target group
Fixed Response	• **Status Code** The HTTP status code value • **Send Content** (optional). If selected, for **Content Type** select one of: – **text/plain** – **text/css** – **text/html** – **application/javascript** – **application/json** • **Response body**: Upload the response body file, up to a maximum size of 1 KB
Redirect	• **Protocol**: Leave it **Unchanged**, or select **HTTP** or **HTTPS** • **Port**: Enter the target port number, or select **Unchanged** to default to the target's original configured port • **Path**: The redirect URL that must be sent to the client's browser • **Query**: The query portion of the redirect URL • **Status Code**: – **301** (Moved Permanently) – **302** (Moved Temporarily)

3. To create an additional rule, click **+Add Rule**.

4. Click **Finish**.

CHAPTER

TWO

AWS-ELB

2.1 AWS-ELB-API

AWS API Reference	Ignored Param	Optional Parameters	Required Parameters	Unsupported Params
AddTags	[]	[]	ResourceArns Tags	[]
CreateListener	[]	Certificates	LoadBalancerArn Protocol Port DefaultActions	[]
CreateLoadBalancer	[]	Subnets SubnetMappings SecurityGroups Tags IpAddressType Scheme Type	Name	[]
CreateRule	[]	[]	Actions Conditions ListenerArn Priority	[]
CreateTargetGroup	[]	HealthCheckPath HealthCheckPort HealthCheckProtocol HealthCheckIntervalSeconds HealthCheckTimeoutSeconds HealthyThresholdCount UnhealthyThresholdCount TargetType	Name Protocol Port VpcId	[]
DeleteListener	[]	[]	ListenerArn	[]
DeleteLoadBalancer	[]	[]	LoadBalancerArn	[]
DeleteRule	[]	[]	RuleArn	[]
DeleteTargetGroup	[]	[]	TargetGroupArn	[]
DeregisterTargets	[]	[]	TargetGroupArn Targets	[]
DescribeListeners	[]	ListenerArns LoadBalancerArn	[]	[]
DescribeLoadBalancerAttributes	[]	[]	LoadBalancerArn	[]
DescribeLoadBalancers	[]	LoadBalancerArns Names	[]	[]
DescribeRules	[]	RuleArns ListenerArn	[]	[]
DescribeTags	[]	[]	ResourceArns	[]
DescribeTargetGroups	[]	[]	TargetGroupArn	[]

2.2 ELBv2 API Version Support

Zadara Cloud Services supports ELBv2, the 2015-12-01 API for Application Load Balancers and Network Load Balancers. It does not support the 2012-06-01 API for Classic Load Balancers.

2.3 AWS CLI elbv2 Examples

Zadara Cloud Services supports the AWS **elbv2** API/CLI. It does not support the **elb** API/CLI.

Here are some examples:

List the load balancers in your system. Note that the `LoadBalancerArn` is a Zadara Cloud Services UUID.

```
aws> elbv2 describe-load-balancers

----------------------------------------------------------------------------------------------------------------
↪--------------------------

|                                                    DescribeLoadBalancers                                     ▯
↪                          |

+--------------------------------------------------------------------------------------------------------------
↪--------------------------+

||                                                     LoadBalancers                                            ▯
↪                         ||

|+----------------------+---------------+----------------------------------------+-------------------+----
↪-------+-------+---------+|

||      CreatedTime     |    DNSName    |            LoadBalancerArn             | LoadBalancerName  | ▯
↪Scheme   | Type  |  VpcId  ||

|+----------------------+---------------+----------------------------------------+-------------------+----
↪-------+-------+---------+|

||  2018-10-01T11:05:32Z|  192.168.41.6 |  96ee2383-4a73-4494-bdf1-e879714c772b  |  dougtest         | ▯
↪internal |  None |  None   ||

|+----------------------+---------------+----------------------------------------+-------------------+----
↪-------+-------+---------+|

|||                                                  AvailabilityZones                                           ▯
↪                        |||

||+------------------------------------------------------------------------------------------------------------
↪------------------------+||

|||                                                     SubnetId                                                 ▯
↪                        |||

||+------------------------------------------------------------------------------------------------------------
↪------------------------+||

|||  subnet-6715c727b6f64ab9be2c5c3094e36473                                                                     ▯
↪                        |||
```

(continues on next page)

(continued from previous page)

```
||+-------------------------------------------------------------------------------------------------------------
↪------------------------+||

|||                                                    SecurityGroups                                          
↪                        |||

||+-------------------------------------------------------------------------------------------------------------
↪------------------------+||

|||  sg-76423026d14f40a48921c3368d85a0b5                                                                       
↪                        |||

|||  sg-fb329fe5ba2e40618bcba4aeb84adb0f                                                                       
↪                        |||

||+-------------------------------------------------------------------------------------------------------------
↪------------------------+||

|||                                                        State                                               
↪                        |||

||+-------------------------------------------------------+-----------------------------------------------------
↪------------------------+||

|||  Code                                                 |  active                                           
↪                        |||

||+-------------------------------------------------------+-----------------------------------------------------
↪------------------------+||



aws> elbv2 describe-load-balancers --load-balancer-arns 96ee2383-4a73-4494-bdf1-e879714c772b

-------------------------------------------------------------------------------------------------------------
↪---------------------------

|                                                  DescribeLoadBalancers                                       
↪                          |

+-------------------------------------------------------------------------------------------------------------
↪---------------------------+

||                                                    LoadBalancers                                             
↪                          ||

|+----------------------+---------------+--------------------------------------+-------------------+----
↪-------+-------+---------+|

||      CreatedTime     |    DNSName    |            LoadBalancerArn           | LoadBalancerName  | 
↪Scheme   | Type  |  VpcId  ||

|+----------------------+---------------+--------------------------------------+-------------------+----
↪-------+-------+---------+|

||  2018-10-01T11:05:32Z|  192.168.41.6 |  96ee2383-4a73-4494-bdf1-e879714c772b  |  dougtest         | 
```

(continues on next page)

(continued from previous page)

```
↪internal |  None |  None   ||

|+---------------------+---------------+--------------------------------------+------------------+----
↪-------+-------+---------+|

|||                                                  AvailabilityZones                                  
↪                        |||

||+-------------------------------------------------------------------------------------------------------
↪-----------------------+||

|||                                                     SubnetId                                        
↪                        |||

||+-------------------------------------------------------------------------------------------------------
↪-----------------------+||

|||  subnet-6715c727b6f64ab9be2c5c3094e36473                                                            
↪                        |||

||+-------------------------------------------------------------------------------------------------------
↪-----------------------+||

|||                                                   SecurityGroups                                    
↪                        |||

||+-------------------------------------------------------------------------------------------------------
↪-----------------------+||

|||  sg-76423026d14f40a48921c3368d85a0b5                                                                
↪                        |||

|||  sg-fb329fe5ba2e40618bcba4aeb84adb0f                                                                
↪                        |||

||+-------------------------------------------------------------------------------------------------------
↪-----------------------+||

|||                                                      State                                          
↪                        |||

||+-------------------------------------------------------+-----------------------------------------------
↪-----------------------+||

|||  Code                                                |  active                                     
↪                        |||

||+-------------------------------------------------------+-----------------------------------------------
↪-----------------------+||
```

Delete one of the load balancers you just listed, passing in the load balancer's LoadBalancerArn:

```
aws> elbv2 delete-load-balancer --load-balancer-arn 96ee2383-4a73-4494-bdf1-e879714c772b
```

✔ **Note:** **describe-load-balancers** takes `--load-balancer-arns` which is a comma separated list of UUIDs, and

delete-load-balancer takes `--load-balancer-arn` which is a single UUID.

2.4 Boto 3 for ELB

2.4.1 Example-Work with a Load Balancer and Target Group

This example shows you how you can use a load balancer to manage the instances in a target group.

```
import boto3
import botocore
import sys
import random

def main():

# Replace following parameters with your IP and credentials
CLUSTER_IP = '<API endpoint IP>'
AWS_ACCESS = '<AWS Access Key ID>'
AWS_SECRET = '<AWS Secret Access Key>'

# Example parameters
VPC_CIDR = '172.20.0.0/16'
SUBNET_CIDR = '172.20.10.0/24'
PORT_INTERNAL = 9090
PORT_EXTERNAL = 80
TARGETS_COUNT = 2
IMAGE_ID = '<Targets Image ID>'
INSTANCE_TYPE = '<Targets Instance Type>'

"""
This script shows and example of Boto3 ELB v2 integration with Symphony.
The scenario:
     1. Create VPC
     2. Create Internet-Gateway
     3. Attach Internet-Gateway
     4. Create Subnet
     5. Create Route-Table
     6. Create Route
     7. Associate Route-Table to Subnet
     8. Create Targets Security-Group
     9. Run target instances
     10. List load-balancers
     11. Create Load-Balancer Security-Groups
     12. Create load-balancer
     13. Create target-group
     14. Register instances to target-group
     15. Create Listener

This example was tested on versions:
- botocore 1.7.35
- boto3 1.4.7
"""

# The following will be used to differentiate entity names in this example
```

(continues on next page)

(continued from previous page)

```
run_index = '%03x' % random.randrange(2**12)

print ("Disabling warning for Insecure connection")
botocore.vendored.requests.packages.urllib3.disable_warnings(
    botocore.vendored.requests.packages.urllib3.exceptions.InsecureRequestWarning)

# creating a EC2 client connection to Symphony AWS Compatible region
ec2_client = boto3.client(service_name="ec2", region_name="symphony",
                          endpoint_url="https://%s/api/v2/ec2/" % CLUSTER_IP,
                          verify=False,
                          aws_access_key_id = AWS_ACCESS,
                          aws_secret_access_key=AWS_SECRET)

# creating a ELB client connection to Symphony AWS Compatible region
elb_client = boto3.client(service_name="elbv2", region_name="symphony",
                          endpoint_url="https://%s/api/v2/aws/elb" % CLUSTER_IP,
                          verify=False,
                          aws_access_key_id = AWS_ACCESS,
                          aws_secret_access_key=AWS_SECRET)

# Create VPC
create_vpc_response = ec2_client.create_vpc(CidrBlock=VPC_CIDR)

# check create vpc returned successfully
if create_vpc_response['ResponseMetadata']['HTTPStatusCode'] == 200:
    vpcId = create_vpc_response['Vpc']['VpcId']
    print("Created VPC with ID %s" % vpcId)
else:
    print("Create VPC failed")

#Create Internet Gateway
create_igw_response = ec2_client.create_internet_gateway()

# check create internet-gateway returned successfully
if create_igw_response['ResponseMetadata']['HTTPStatusCode'] == 200:
    igwId = create_igw_response['InternetGateway']['InternetGatewayId']
    print("Created InternetGateway with ID %s" % igwId)
else:
    print("Create InternetGateway failed")

#Attach Internet Gateway to VPC
attach_igw_response = ec2_client.attach_internet_gateway(InternetGatewayId=igwId,
                                                         VpcId=vpcId)

# check attach internet-gateway returned successfully
if attach_igw_response['ResponseMetadata']['HTTPStatusCode'] == 200:
    print("Attached InternetGateway with ID %s to VPC %s" % (igwId, vpcId))
else:
    print("Create InternetGateway failed")

#Create Subnet
create_subnet_response = ec2_client.create_subnet(CidrBlock=SUBNET_CIDR, VpcId=vpcId)

# check create subnet returned successfully
if create_subnet_response['ResponseMetadata']['HTTPStatusCode'] == 200:
    subnetId = create_subnet_response['Subnet']['SubnetId']
    print("Created Subnet with ID %s" % subnetId)
```

(continues on next page)

(continued from previous page)

```
else:
    print("Create Subnet failed")

#Create route table in the VPC
create_rtb_response = ec2_client.create_route_table(VpcId=vpcId)

# check create route-tables returned successfully
if create_rtb_response['ResponseMetadata']['HTTPStatusCode'] == 200:
    rtbId = create_rtb_response['RouteTable']['RouteTableId']
    print("Created Route Table ID %s in VPC %s" % (rtbId, vpcId))
else:
    print("Create route-tables failed")

#Add routing rule to route table
create_route_response = ec2_client.create_route(DestinationCidrBlock='0.0.0.0/0',
                                                GatewayId=igwId,
                                                RouteTableId=rtbId)

# check create route returned successfully
if create_route_response['ResponseMetadata']['HTTPStatusCode'] == 200:
    print("Created routing rule VPC with ID %s" % vpcId)
else:
    print("Create routing rule failed")

#Associate route table to subnet
associate_rtb_response = ec2_client.associate_route_table(RouteTableId=rtbId,
                                                          SubnetId=subnetId)

# check create route returned successfully
if associate_rtb_response['ResponseMetadata']['HTTPStatusCode'] == 200:
    print("Associated route table %s to subnet %s" % (rtbId, subnetId))
else:
    print("Associated route table failed")

#Create Security-Group
create_sg_response = ec2_client.create_security_group(GroupName='my_ELB_SG_%s' % run_index,
                                                      Description='Allow traffic for ELB',
                                                      VpcId=vpcId)

# check create security-group returned successfully
if create_sg_response['ResponseMetadata']['HTTPStatusCode'] == 200:
    sgId = create_sg_response['GroupId']
    print("Created security-group with ID %s" % sgId)
else:
    print("Create security-group failed")

#Allow Security-Group Rules - ICMP and TCP
allow_ingress_response = ec2_client.authorize_security_group_ingress(GroupId=sgId,
                                                                     IpPermissions=[
                                                                         {"IpProtocol": "icmp", "IpRanges
↪": [{"CidrIp": "0.0.0.0/0"}]},
                                                                         {"IpProtocol": "tcp", "FromPort":
↪ PORT_INTERNAL, "ToPort": PORT_INTERNAL, "IpRanges": [{"CidrIp": "0.0.0.0/0"}]}
                                                                     ])

# check allow ingress traffic returned successfully
if allow_ingress_response['ResponseMetadata']['HTTPStatusCode'] == 200:
```

(continues on next page)

(continued from previous page)

```
    print("Allow security group ingress for ICMP and TCP")
else:
    print("Allow security group ingress failed")

#Run instances
print ("Starting to run target instances")
run_instances_response = ec2_client.run_instances(ImageId=IMAGE_ID, InstanceType=INSTANCE_TYPE,
                                                  MaxCount=TARGETS_COUNT, MinCount=TARGETS_COUNT,
                                                  SecurityGroupIds=[sgId], SubnetId=subnetId)

# check run instances returned successfully
if run_instances_response['ResponseMetadata']['HTTPStatusCode'] == 200:
    targetIds = [instance['InstanceId'] for instance in run_instances_response['Instances']]
    print ("Created instances: " + ' '.join(p for p in targetIds))
else:
    print("Create instances failed")

def my_list_lbs():
    # list lbs
    lbs_list_response = elb_client.describe_load_balancers()

    # check lbs list returned successfully
    if lbs_list_response['ResponseMetadata']['HTTPStatusCode'] == 200:
        print ("LBs list: " + ' '.join(p for p in [lb['LoadBalancerName']
                                    for lb in lbs_list_response['LoadBalancers']]))
    else:
        print ("List lbs failed")

my_list_lbs()

# Create Security-group for Load Balancer
create_lb_sg_response = ec2_client.create_security_group(GroupName='internet-load-balancer_%s' % run_
↪index,
                                                         Description='Security Group for Internet-facing
↪LB',
                                                         VpcId=vpcId)

# check create security-group returned successfully
if create_lb_sg_response['ResponseMetadata']['HTTPStatusCode'] == 200:
    lbSgId = create_lb_sg_response['GroupId']
    print("Created LB security-group with ID %s" % lbSgId)
else:
    print("Create LB security-group failed")

# Allow Security-Group Ingress Rules
# TCP - external port from all sources
# ICMP - all
allow_ingress_response = ec2_client.authorize_security_group_ingress(GroupId=lbSgId,
                                                                     IpPermissions=[
                                                                         {"IpProtocol": "icmp", "IpRanges
↪": [{"CidrIp": "0.0.0.0/0"}]},
                                                                         {"IpProtocol": "tcp", "FromPort":
↪ PORT_EXTERNAL,
                                                                          "ToPort": PORT_EXTERNAL,
↪"IpRanges": [{"CidrIp": "0.0.0.0/0"}]}
                                                                     ])
```

(continues on next page)

(continued from previous page)

```
# check allow ingress traffic returned successfully
if allow_ingress_response['ResponseMetadata']['HTTPStatusCode'] == 200:
    print("Allow security group ingress for ICMP and TCP")
else:
    print("Allow security group ingress failed")

# Allow Security-Group Egress Rules
# TCP - internal port to targets group
allow_egress_response = ec2_client.authorize_security_group_egress(GroupId=lbSgId,
                                                                   IpPermissions=[
                                                                       {"IpProtocol": "tcp",
                                                                        "FromPort": PORT_INTERNAL,
                                                                        "ToPort": PORT_INTERNAL,
                                                                        "UserIdGroupPairs": [{'GroupId':
↪sgId}]}
                                                                   ])

# check allow egress traffic returned successfully
if allow_egress_response['ResponseMetadata']['HTTPStatusCode'] == 200:
    print("Allow security group egress for ICMP and TCP")
else:
    print("Allow security group egress failed")

# create load-balancer
create_lb_response = elb_client.create_load_balancer(Name='my_lb_%s' % run_index,
                                                     Subnets=[subnetId],
                                                     SecurityGroups=[lbSgId],
                                                     Scheme='internet-facing')

# check create lb returned successfully
if create_lb_response['ResponseMetadata']['HTTPStatusCode'] == 200:
    lbId = create_lb_response['LoadBalancers'][0]['LoadBalancerArn']
    print "Successfully created load balancer %s" % lbId
else:
    print ("Create load balancer failed")

my_list_lbs()

# create target-group
create_tg_response = elb_client.create_target_group(Name='my_lb_tg_%s' % run_index,
                                                    Protocol='TCP',
                                                    Port=PORT_INTERNAL,
                                                    VpcId=vpcId)

# check create target-group returned successfully
if create_tg_response['ResponseMetadata']['HTTPStatusCode'] == 200:
    tgId = create_tg_response['TargetGroups'][0]['TargetGroupArn']
    print "Successfully created target group %s" % tgId
else:
    print ("Create target group failed")

# Register targets
targets_list = [dict(Id=target_id, Port=PORT_INTERNAL) for target_id in targetIds]
reg_targets_response = elb_client.register_targets(TargetGroupArn=tgId, Targets=targets_list)

# check register group returned successfully
if reg_targets_response['ResponseMetadata']['HTTPStatusCode'] == 200:
```

(continues on next page)

(continued from previous page)

```
    print "Successfully registered targets"
else:
    print ("Register targets failed")

# create Listener
create_listener_response = elb_client.create_listener(LoadBalancerArn=lbId,
                                                      Protocol='TCP', Port=PORT_EXTERNAL,
                                                      DefaultActions=[{'Type': 'forward',
                                                                       'TargetGroupArn': tgId}])

# check create listener returned successfully
if create_listener_response['ResponseMetadata']['HTTPStatusCode'] == 200:
    print "Successfully created listener %s" % tgId
else:
    print ("Create listener failed")

_name__ == '__main__':
sys.exit(main())
```

2.4.2 Boto 3 Quick Ref for ELB

```
import boto3
client = boto3.client('elbv2')
```

Commonly used methods
create_load_balancer
delete_load_balancer
describe_load_balancers
create_listener
delete_listener
describe_listeners
create_target_group
delete_target_group
describe_target_groups

Note: Other ELB methods are described here.