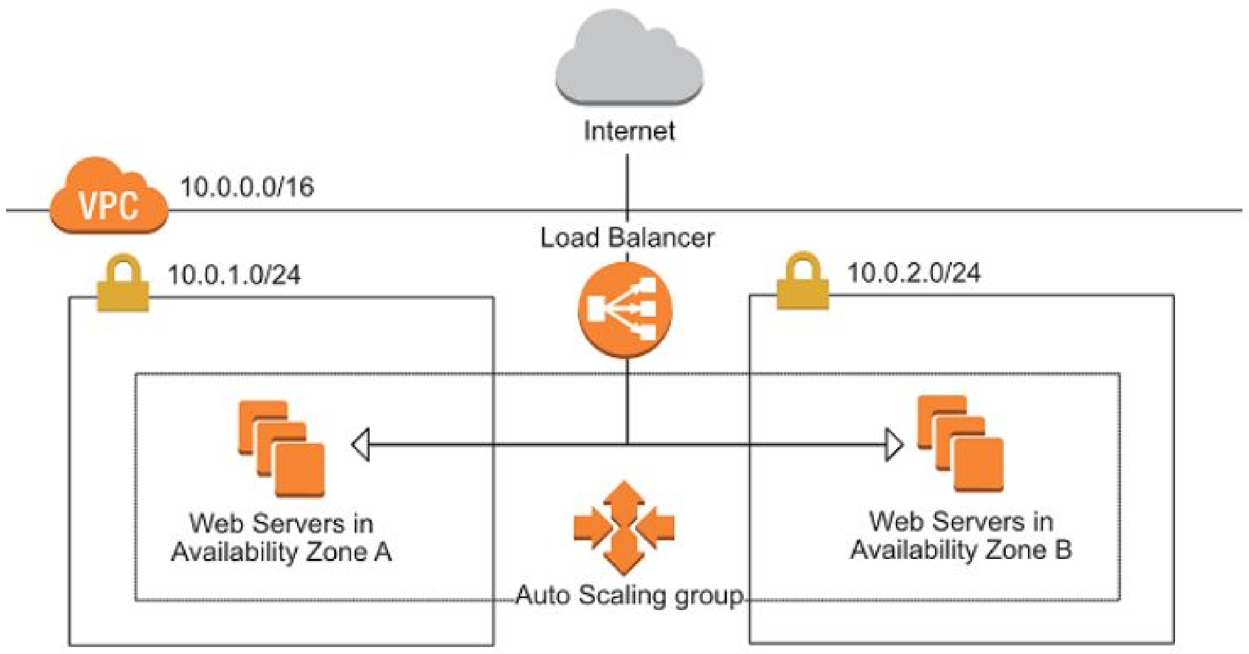
How to create a Scalable and Self Healing Architecture on AWS

Scalability and high availability have been the buzz word of the last past years in Software Industry. There is a very strong reason behind it. If you start off with a stack which runs fine in test environment and with little load shit hits the fan then you have made your life and your team's life miserable.

AWS loadbalancing and autoscaling let you create an elastic and self healing architecture with very little effort. Some people might argue about the cost of load balancer but I think getting a DNS endpoint instead of an IP makes the design simple and its worth the cost.



This post is about deploying a war file using s3 with AWS loadbalancing and autoscaling of EC2 instances. The purpose is to increase and decrease the number of EC2 instances when the demand changes. Webservers will be behind a load balancer so your application doesn't need to worry about dynamic changes in infrastructure.

Step — 1: S3 bucket as a repository for deployment files

An S3 bucket will serve as a repository to hold files which will be required for deployment. These can be the war files and configuration files for your tomcat for example.

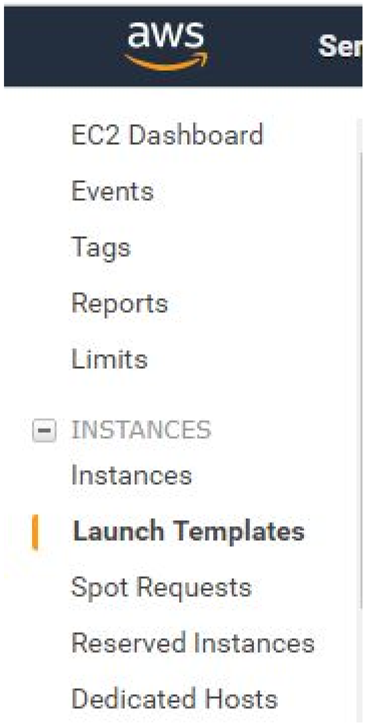
Step — 2: IAM Role for EC2 Instances for reading S3

Role policy

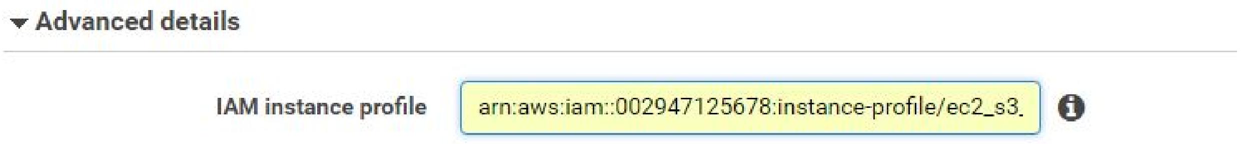
```
{ “Version”: “2012–10–17”,

“Statement”: [ { “ Effect”: “Allow”,

“Action”: [ “s3:Get*”, “s3:List*” ],

“Resource”: “*”

} ]
```

Step — 3: Create a Launch Template in Ec2 Dashboard

Launch template is a blueprint with configuration information of an instance. You will specify the AMI(Operating System), instance type, network settings and most importantly the bootstrap script which will create your webserver.



The role which you created earlier will need to be assigned to this launch template so the instances can read s3 bucket. You can do this in Advance tab



IAM instance profile: arn:aws:iam::002947125678:instance-profile/ec2_s3

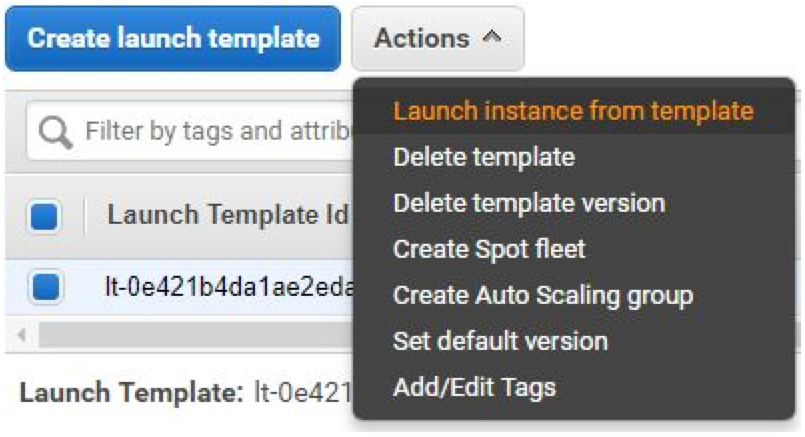
The bootstrap Script

In advanced details under User data put your bootstrap script. Example script

```
#!/bin/bash

sudo apt-get update

sudo apt-get -y upgrade

sudo apt-get -y install tomcat8

sudo apt install -y awscli

sudo aws — region eu-central-1 s3 cp s3://<your bucket>/<.war>/var/lib/tomcat8/webapps
```

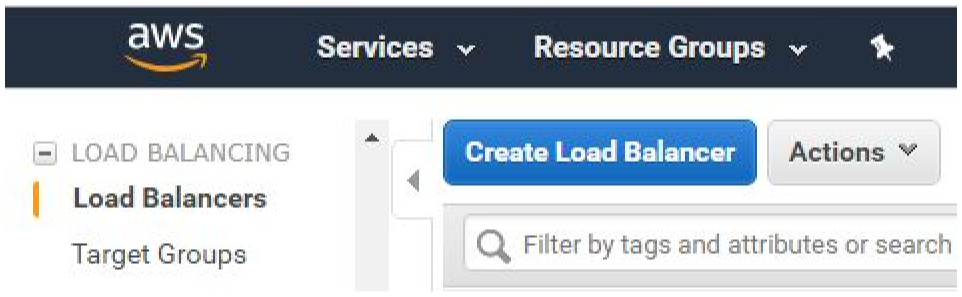
This script will install tomcat8 and awscli and then deploy your war file from your s3 bucket in tomcat.

Fire up your instance and verify if all is good.



Step — 4: Create a load balancer

Under Load Balancing click on Create Load Balancer



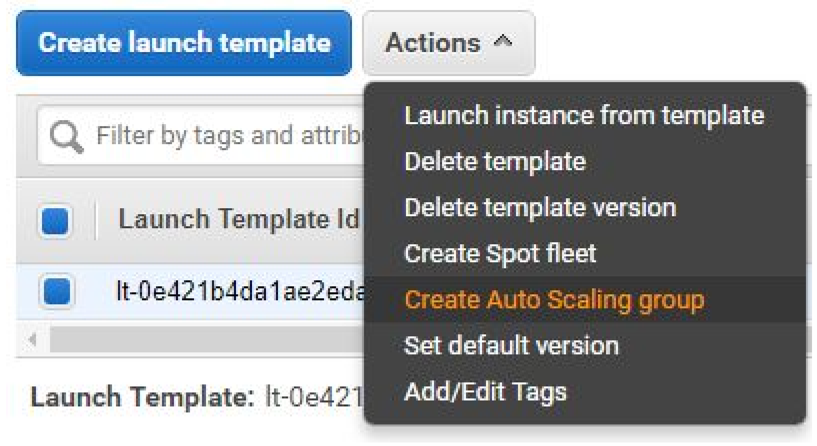
We will be using a Application Load Balancer for HTTP/HTTPS traffic. Follow the wizard and you will have a load balancer for your web traffic in no time.

Step — 5: Create a Target Group

Load balancer will forward the traffic to a target group. A listener rule in load balancer will be tied to a target group. Think of target group as a logical grouping of EC2 instances which will receive traffic from Load balancer.

Step — 6: Create Auto-scaling Group

Auto scaling groups are logical grouping of EC2 instances which share similar characteristics. For example you can have an auto scaling group of Web servers or Caching servers. To create an auto scaling group of your web servers. Choose your Launch template and click on Create Auto Scaling Group



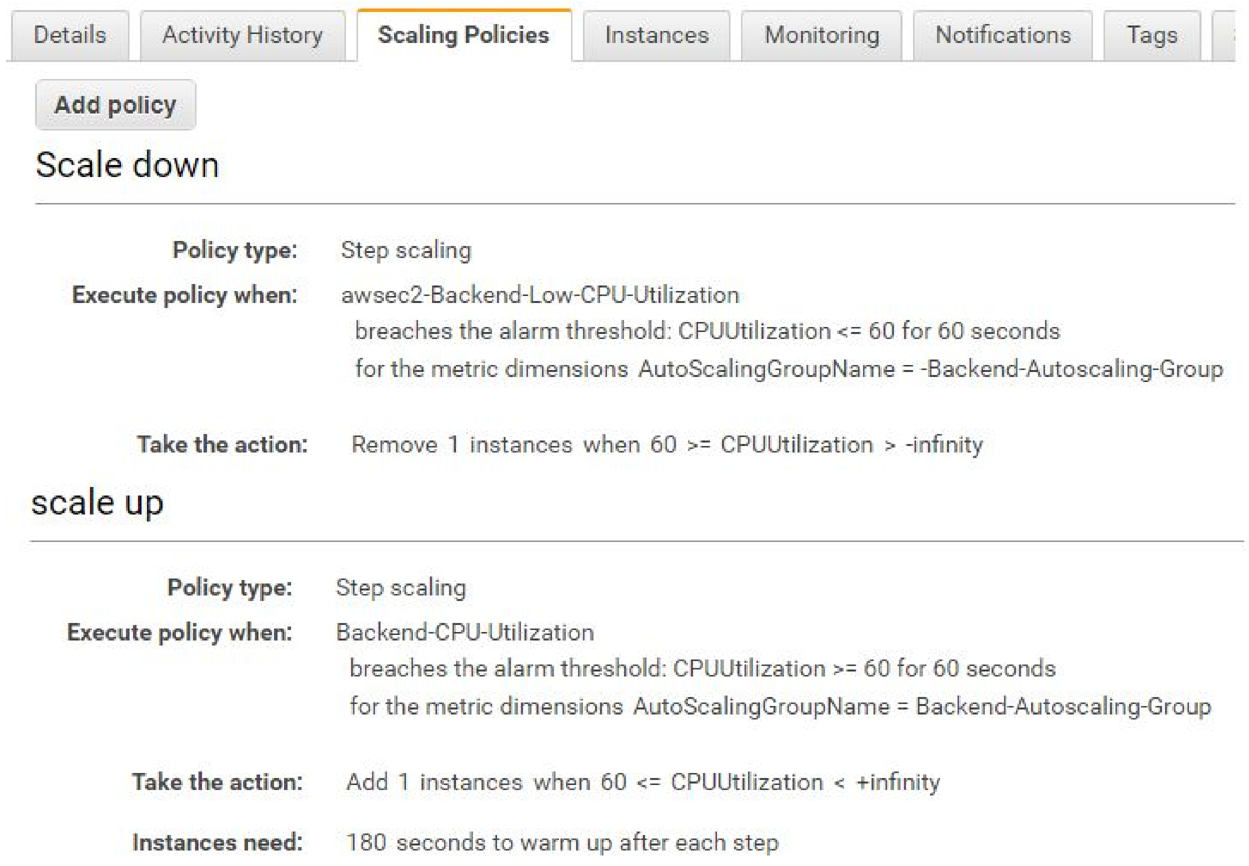
During configuration make sure to keep the health check type to ELB instead of instance health check. This will enable scaling based on your health check done by load balancer, which can be application layer checking. Instead of just relying on instance being alive.

> **Important:** Associate Auto scaling group with target group so all the instances launched by auto scaling will belong to the target group associated with load balancer.

Scaling Policy

This example policy will add an instance when average CPU utilization crosses 60% and remove an instance when CPU utilization goes down below 60%

[illegible]



Scale down

Policy type: Step scaling

Execute policy when: awsec2-Backend-Low-CPU-Utilization
breaches the alarm threshold: CPUUtilization <= 60 for 60 seconds
for the metric dimensions AutoScalingGroupName = -Backend-Autoscaling-Group

Take the action: Remove 1 instances when 60 >= CPUUtilization > -infinity

scale up

Policy type: Step scaling

Execute policy when: Backend-CPU-Utilization
breaches the alarm threshold: CPUUtilization >= 60 for 60 seconds
for the metric dimensions AutoScalingGroupName = Backend-Autoscaling-Group

Take the action: Add 1 instances when 60 <= CPUUtilization < +infinity

Instances need: 180 seconds to warm up after each step

You are all done. Time to test scaling

Ssh into one of the instance and install stress utility

on a ubuntu machine

sudo apt-get install stress

sudo stress — cpu 8 — timeout 420

Wait for 5 minutes(Default time for cloud watch to trigger an alarm). If everything is working fine, auto scaling will trigger and instance will start. Your load balancer will start to route the traffic towards your new instance.