

**AWS Application Load Balancer and Amazon ECS using dynamic port mapping**

This week I want to talk about ALB and ECS integration. As you know, recently AWS announced their new load balancing service, Application Load Balancer (ALB). ALB supports more advanced features than ELB with a lower cost. I won’t explain each of these features but, you can refer to [this documentation](https://aws.amazon.com/elasticloadbalancing/applicationloadbalancer/) to get more details about it. The new features are basically about:

Content-Based routing

Container-based Application Support ( Today I’ll give an example about this )

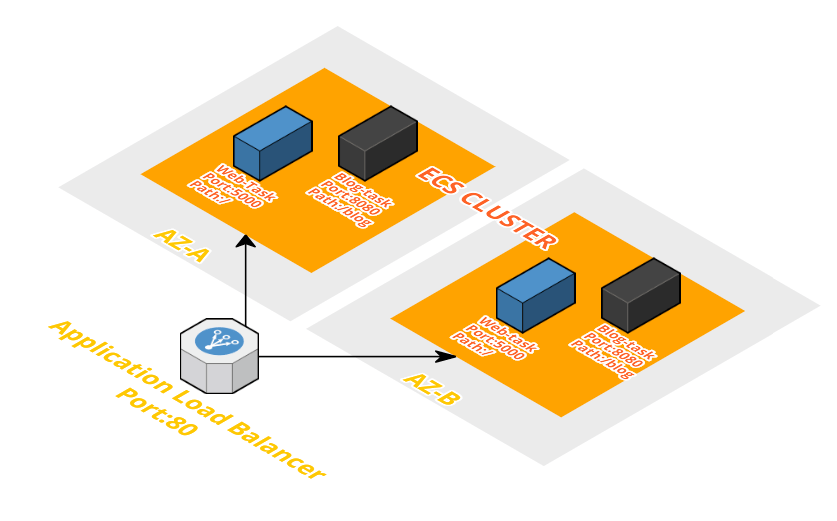
HTTP/2 Support

Websockets Support

Sticky Sessions

Better CloudWatch Metrics

As I mentioned, I want to show how we can use our dockerized applications with ALB. We will use following architecture that is shown below. As you see there are our webpage and blog page. These two applications will listen on different ports and will run on a ECS cluster. Our docker images will be stored in AWS ECS Registry. Also our ALB will use port 80 and will forward our request to backend using dynamic port mapping feature.



Our step for this demo:

Create Docker images

Create registry and push our images to it. (I’ll use AWS ECS Registry)

Create our task definitions

Create ALB

Run services

Let’s start by creating our images.

**1 – Create Docker images:**

As you see, our blog container will run a basic flask app which listens on port 8080 and the application will be reached by using “/blog” path. I’ll use this Dockerfile and create my image. Also we’ll have webpage container as well ( It has another Dockerfile. Only differences will be the port number (5000) and the path (“/”) )   
Dockerfile:

FROM centos MAINTAINER salk.onur@gmail.com RUN rpm -Uvh http://dl.fedoraproject.org/pub/epel/7/x86\_64/e/epel-release-7-8.noarch.rpm \ && yum update -y \ && yum install -y python-pip \ && pip install flask COPY . /src EXPOSE 8080 CMD cd /src && python blog.py

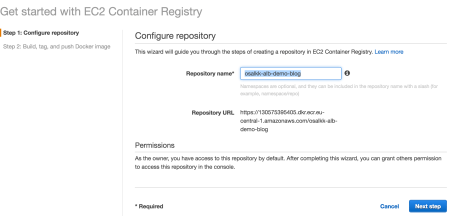
Application code, blog.py:

from flask import Flask from flask import render\_template app = Flask(\_\_name\_\_) @app.route('/blog') def blog(): return render\_template('blog.html',title='blog') if \_\_name\_\_ == '\_\_main\_\_': app.run(threaded=True,host='0.0.0.0',port=8080)

Now I can create my images using the commands below:

docker build -t osalkk-alb-demo-blog . docker build -t osalkk-alb-demo-web .

**2 – Create registry and push our images to it:**

Now I switch to AWS console and create my ECS registry as seen in the screenshot. ( This is only for blog image, for the web image the steps are the same )   


In the next step, I get help from “push commands” that AWS console shows us. First, I retrieve “docker login” output by using the command below. This will show a long command that will help us login to the registry.

aws ecr get-login --region eu-central-1

I use the “docker login” command:

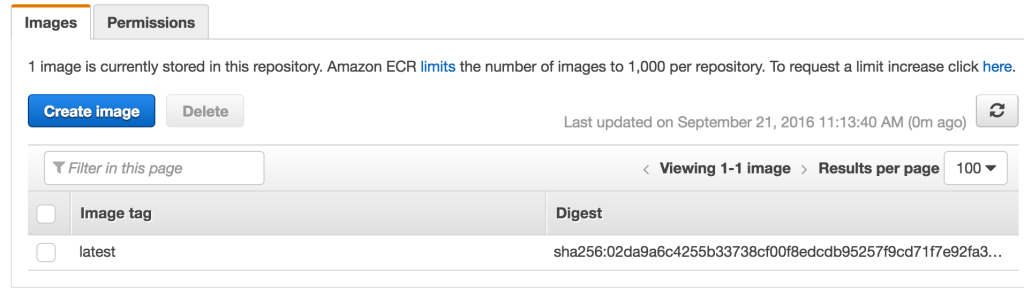
docker login -u AWS -p AQECAHh3zYOdtpBBSVw/.... -e none https://130575395405.dkr.ecr.eu-central-1.amazonaws.com

I tag my image:

docker tag osalkk-alb-demo-blog:latest 130575395405.dkr.ecr.eu-central-1.amazonaws.com/osalkk-alb-demo-blog:latest

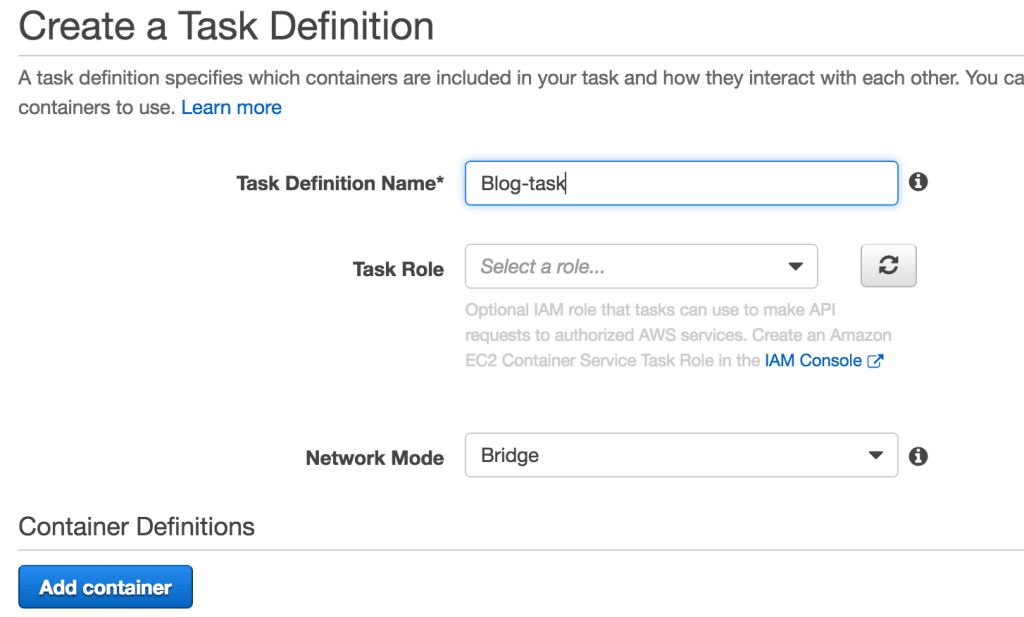
Finally, I push my image to AWS ECS Registry

docker push 130575395405.dkr.ecr.eu-central-1.amazonaws.com/osalkk-alb-demo-blog:latest

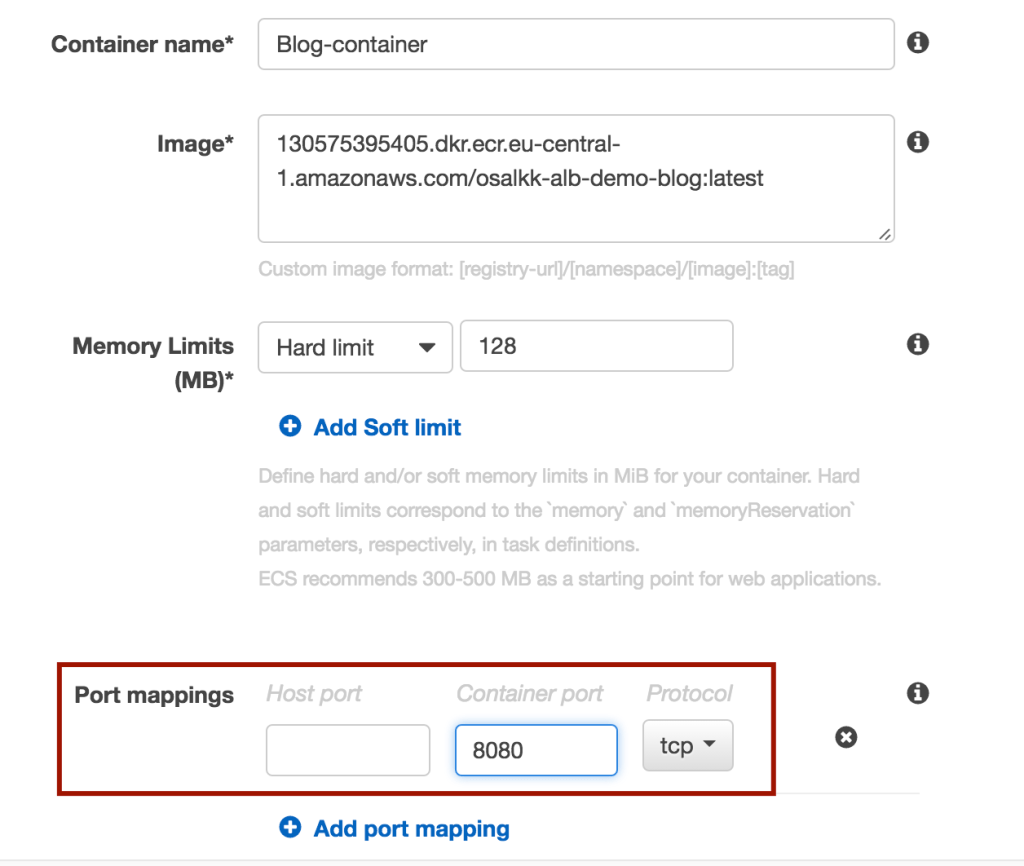
As we see, our images are stored in our repository. 

**3 – Create our task definitions:**

Now it’s time to create our tasks and services. In ECS console, I click “Create task definition” and proceed.



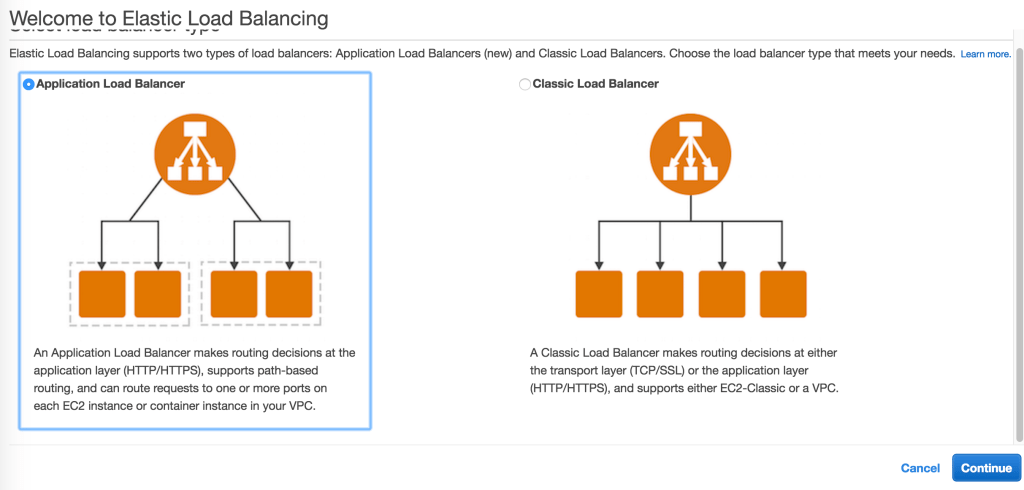
I give “Blog-task” name for my task and click “Add Container”. I complete the form as below. I leave the host port empty since we want it to be dynamically mapped. For the blog container I use 8080 and for the web  I use 5000 as port number. Then I click “Add” and “Create” to complete the task creation.



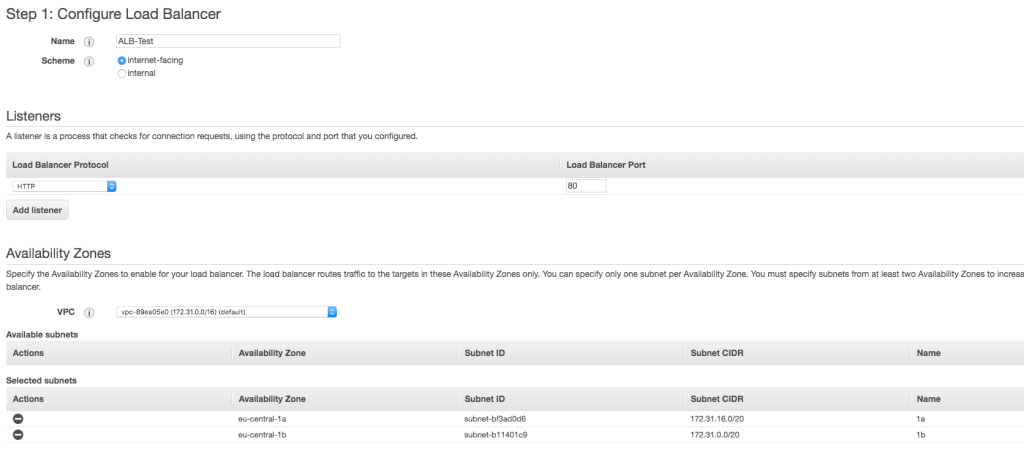
Before creating our services, we need to create our ALB.

**4- Create ALB:**

We are ready to create our ALB now. Let’s start by choosing Application Load Balancer



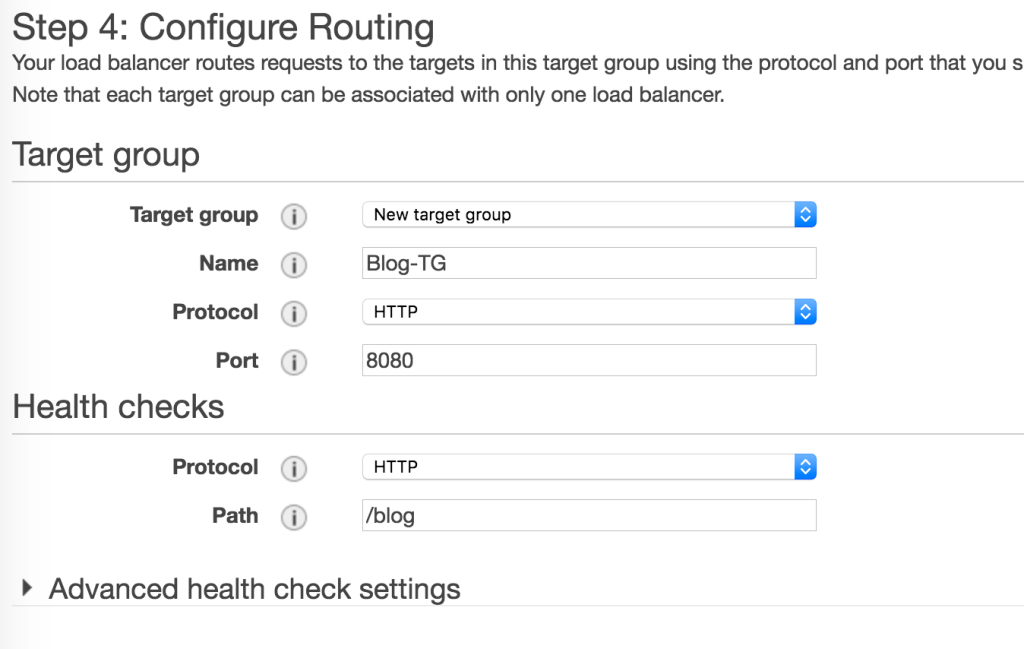
I name it as “ALB-Test” and choose my available subnets.



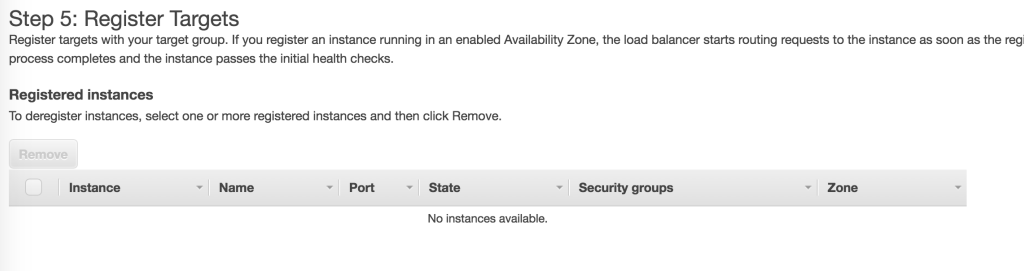
I select my security group



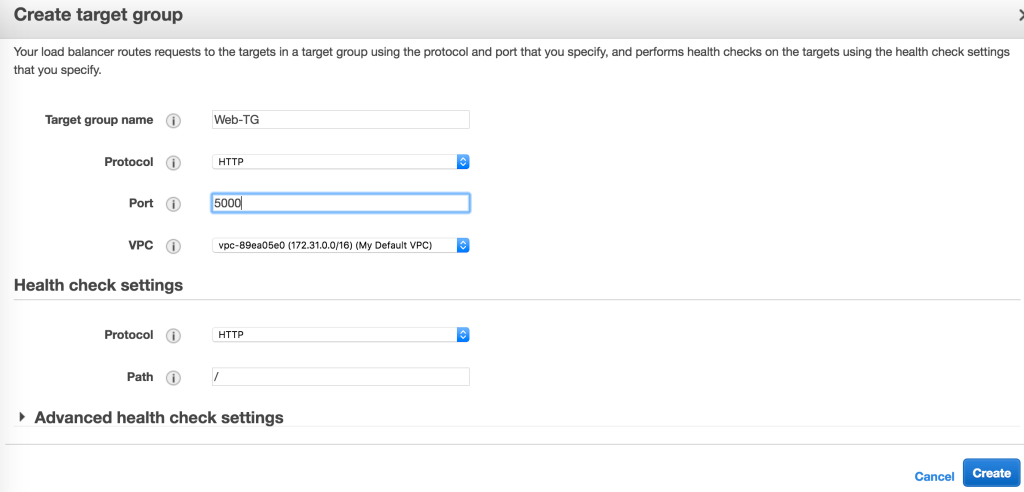
Now I have to configure the routings for my ALB. Here I add a new targeting group.  Remember that,  I use the port number as 8080 and the path as “/blog”.

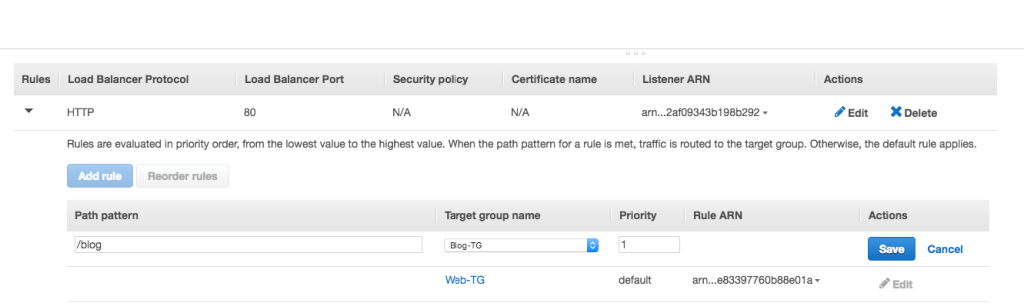


Since ECS will register targets, we don’t need to register instances manually. After that, we can review and finish creating our ALB.



Our next step is creating a new target group for web and adding a rule for it. Here I select 5000 as port number and  “/” as the path.

Then I select my ALB and edit the rules.   I add another rule and enter “/blog” as path pattern and “Blog-TG” as target group name( Previously, I had configured the “Blog-TG” so it was default. I edited it as “Web-TG”, don’t be confused with that ).

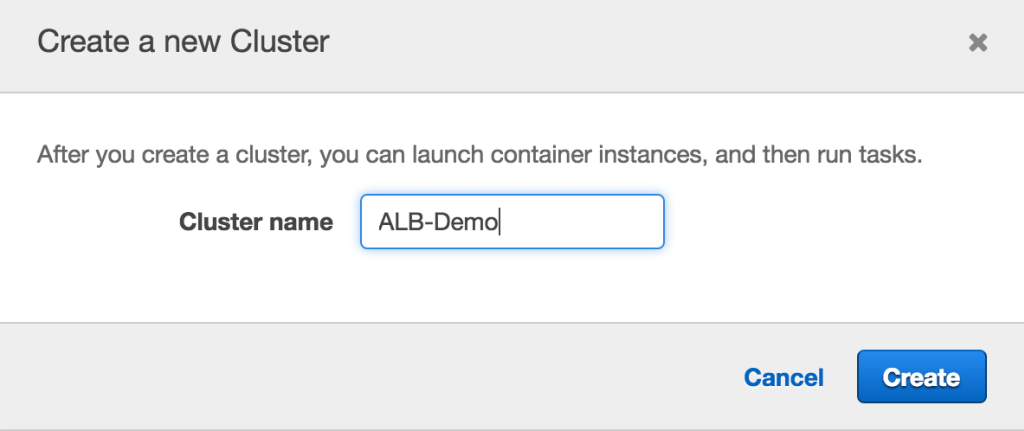


We can proceed with running our services and test our applications.

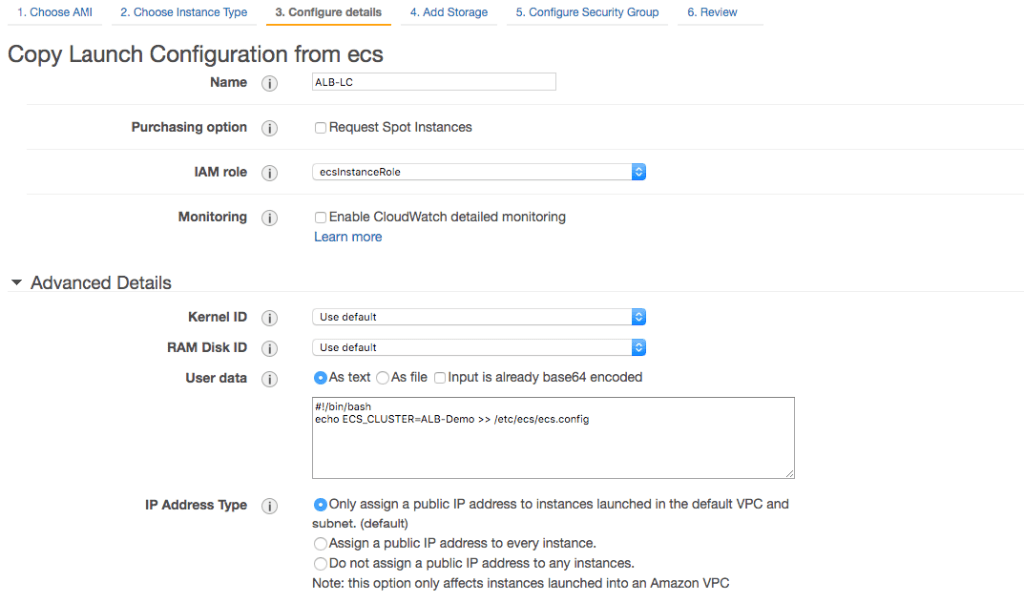
**5 – Run Services:**

We can now jump to the ECS console and configure our cluster and services.

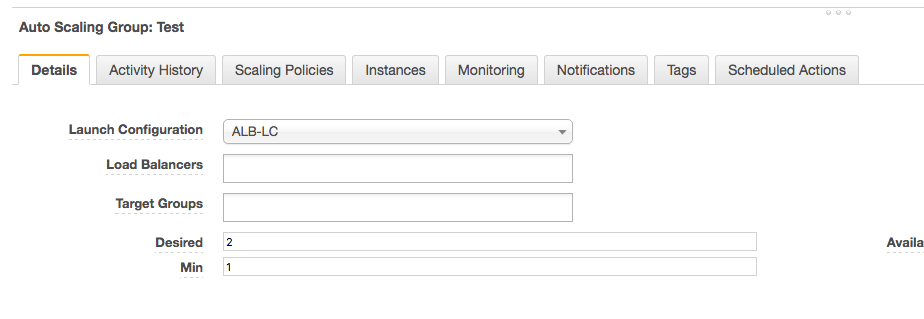
I create a new cluster as “ALB-Demo”.



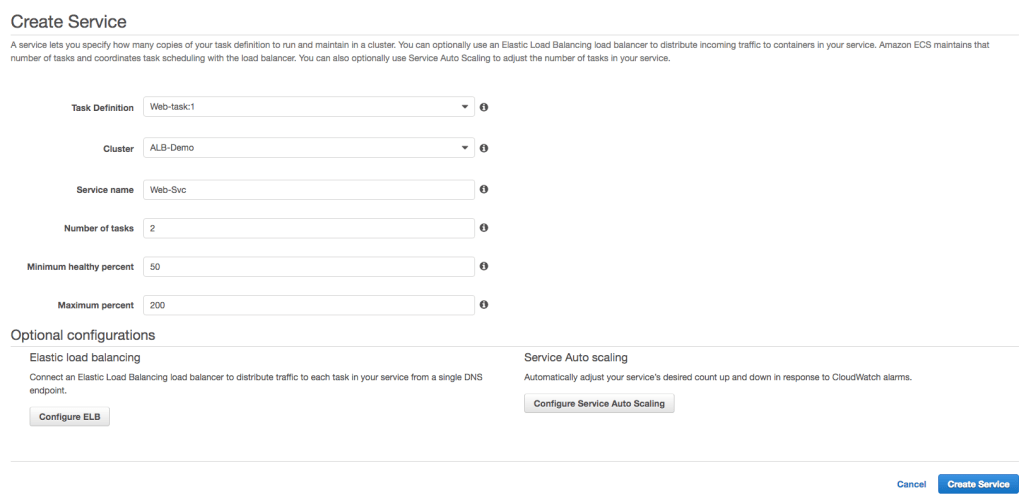
We need to launch instances for our ECS cluster. Here I won’t explain how to create autoscaling group and launching ECS instances ( If you need, you can refer [here](http://docs.aws.amazon.com/AmazonECS/latest/developerguide/launch_container_instance.html) ). I have already had an auto scaling group and a launch configuration. I copy it and change the user-data so it can register instances with my ECS Cluster.



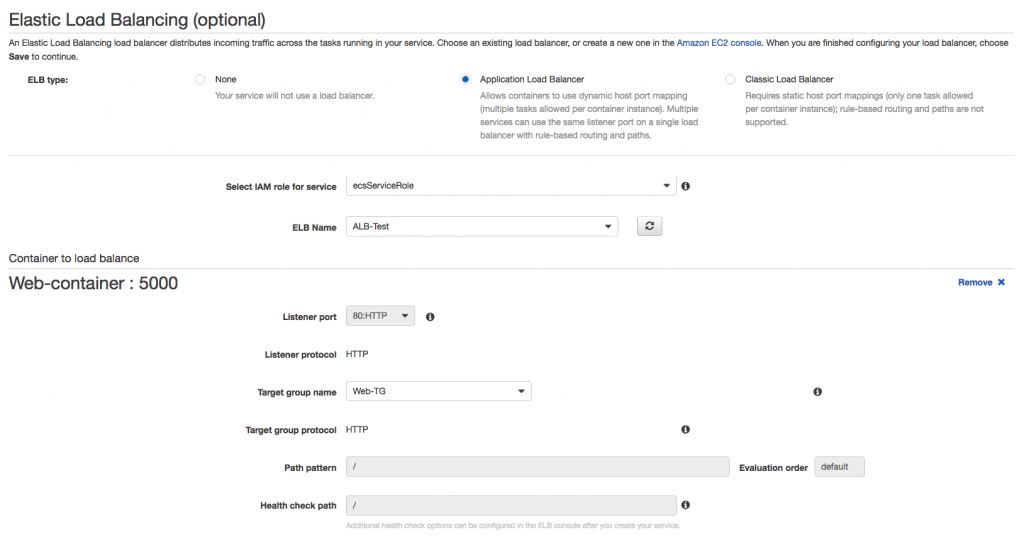
Then I edit my autoscaling group and select my new launch configuration, set the desired to 2 and save.



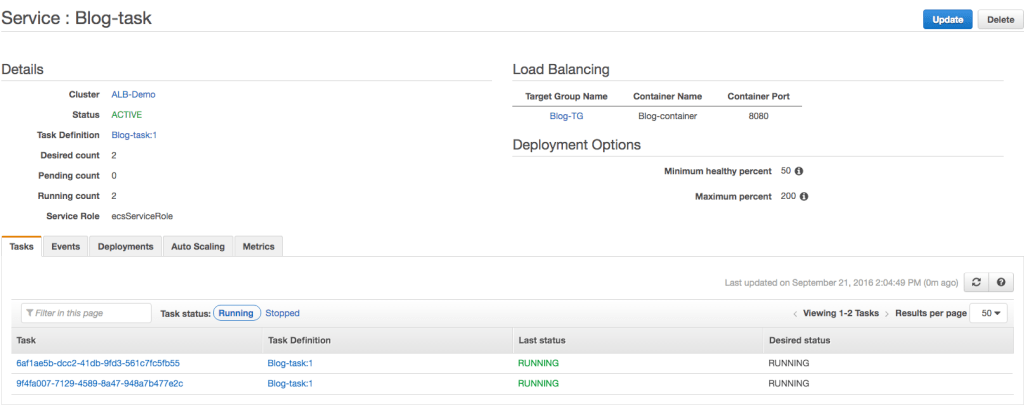
After a while, my instances are registered in my cluster. Now let’s configure the services. Here I select my task definition, set the number of tasks to 2 and click “Configure ELB”.



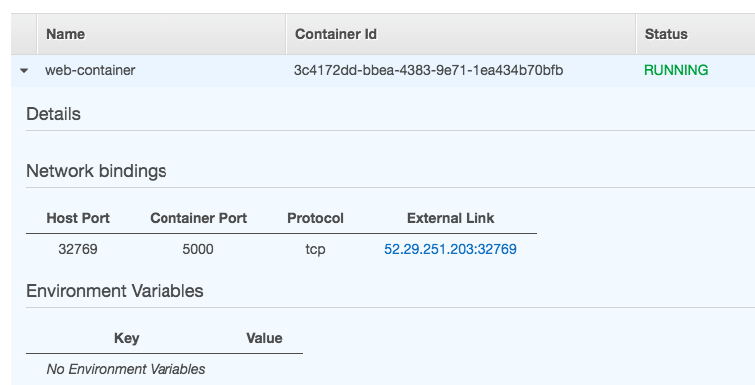
At ELB screen, I select “ALB-Test” as ELB and select “Web-TG” as target group name ( all other fields are populated as I select ). Then I finish creating the service. Also I make the same steps for blog service.



As soon as the tasks are running, I can test my ALB and the applications.



You can also view the dynamic port mapping if you check a task detail. As seen below, port number “32769” is assigned as host port for a web task.



Finally let’s see if our applications are running. Here are the results both for “/” and “/blog” paths.



As you can see, configuring ALB with container-based applications is easy. I hope you find this post useful. If you have any questions or comments, please feel free to write and don’t forget to share this post please.