

Deploying Docker containers on ECS

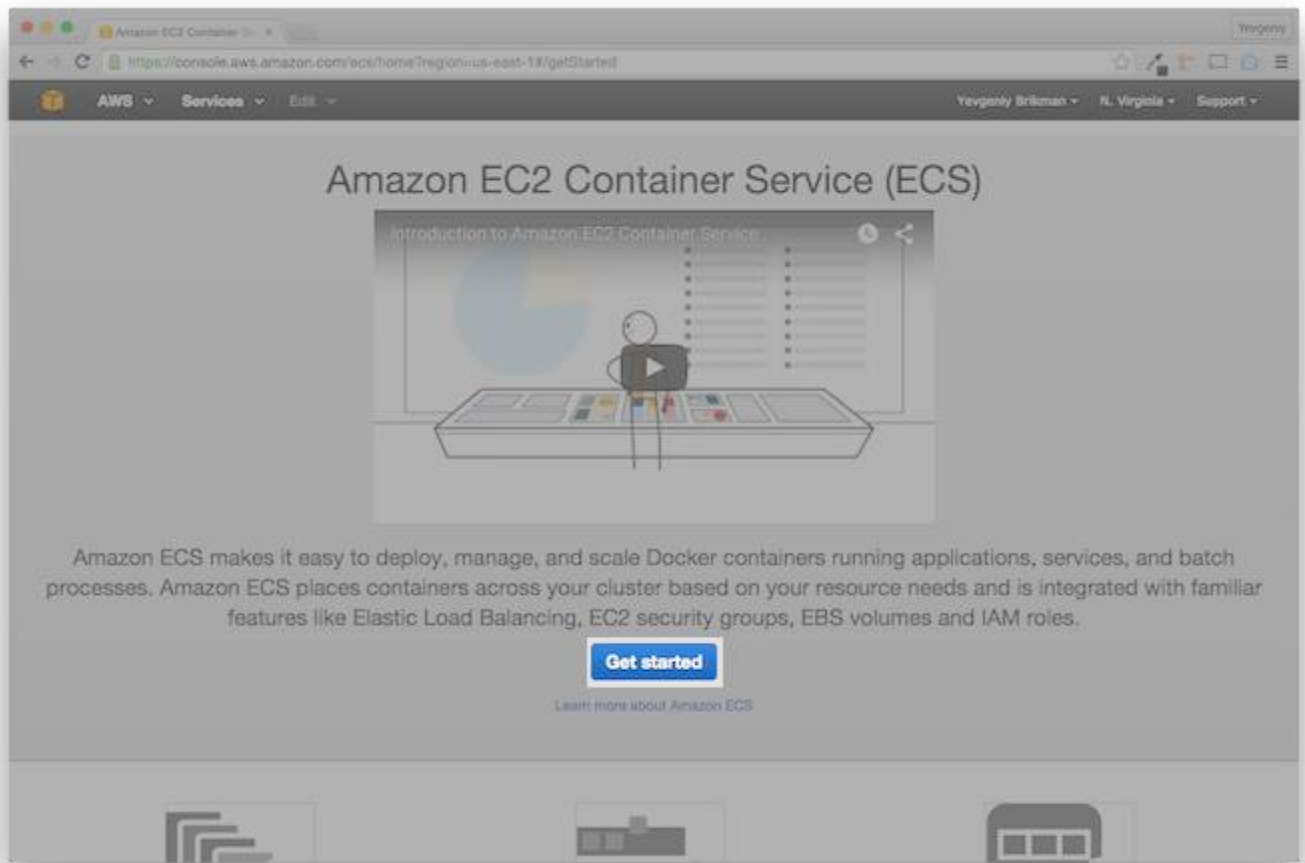
The idea behind ECS is that you create an *ECS Cluster*—which is a group of EC2 Instances managed by ECS—define what Docker containers you want to run, and ECS will take care of deploying those containers across the Cluster, rolling out new versions, and integrating with other AWS infrastructure. ECS can make it easier to manage multiple Docker containers running on multiple EC2 Instances—if you can figure out all the steps required to use it. These steps are:

1. Create an ECS Cluster
2. Create an ELB
3. Create IAM Roles
4. Create an Auto Scaling Group
5. Run Docker containers in your ECS Cluster
6. Update Docker containers in your ECS Cluster

To understand what all of these steps mean and how to do them, let's walk through an example.

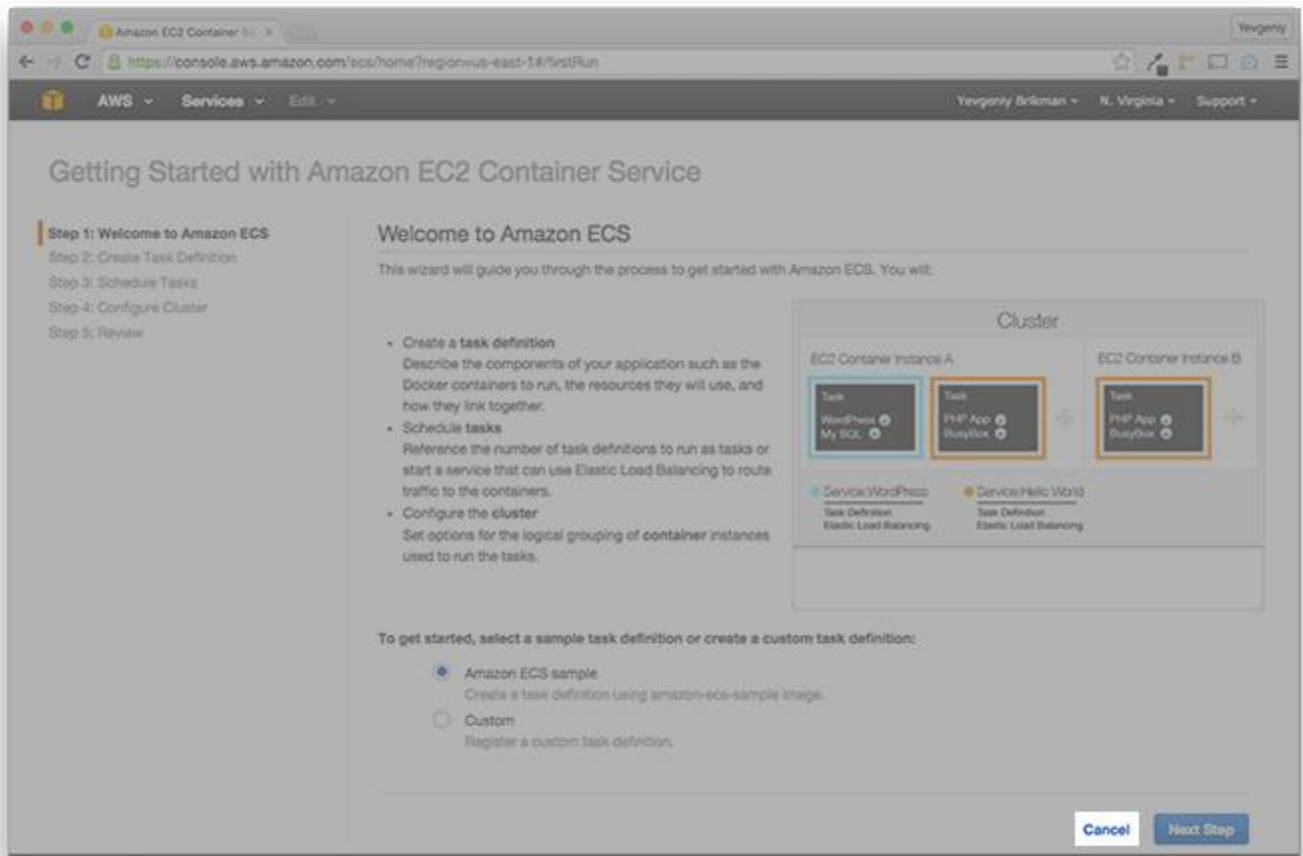
Creating a Cluster

Open up your [AWS Console](#) and click on the EC2 Container Service link to go to the [ECS Console](#). If this is your first time using ECS, you will be taken to a getting started page. Click the blue “Get started” button:



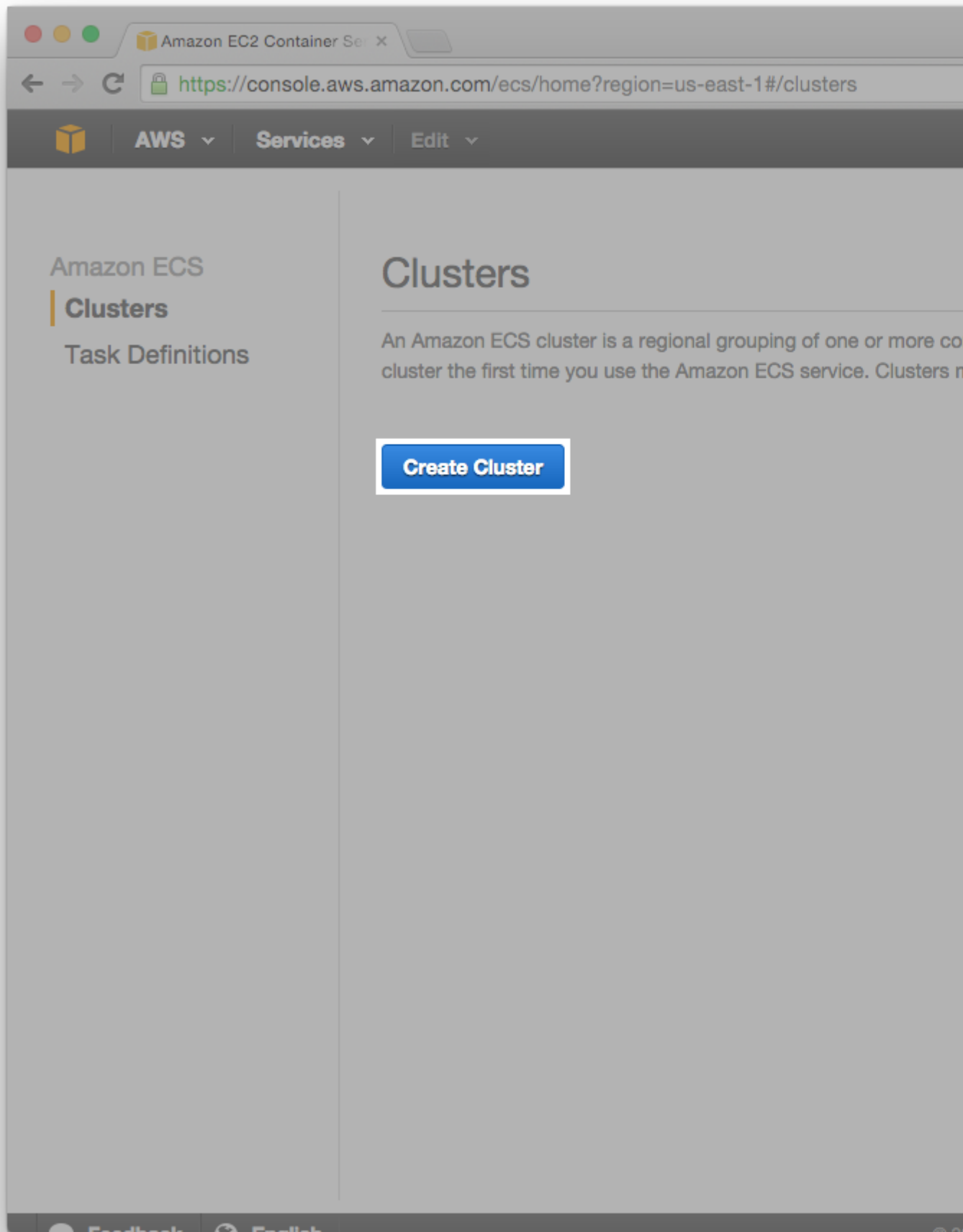
ECS Get Started

This takes you to a wizard that walks you through the process of using ECS, but I found the wizard confusing, and as you'll never be able to use the wizard again after this first time, it also doesn't teach you how to use the actual ECS UI. Therefore, I recommend clicking the cancel button in the bottom right corner:



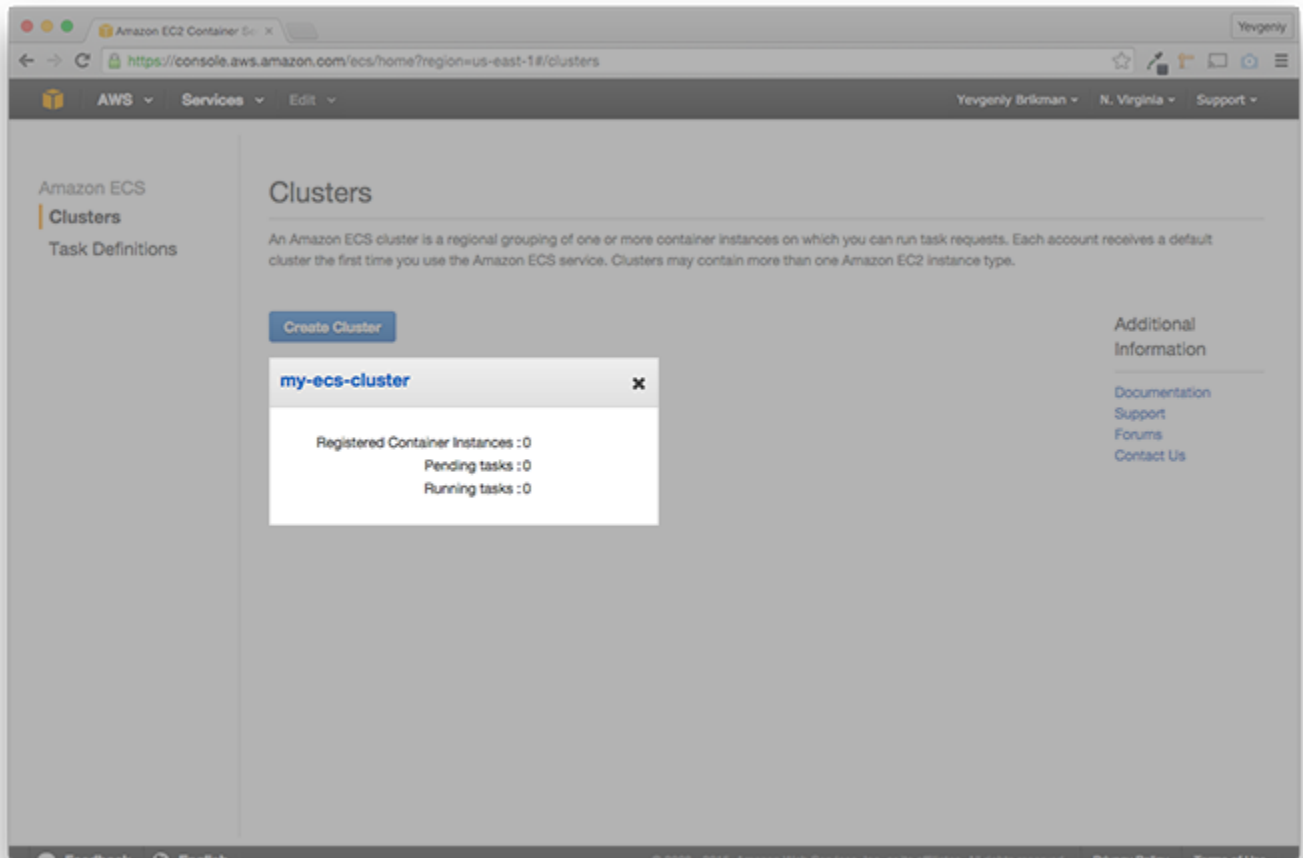
Click cancel to get out of the ECS wizard

This takes you to the Clusters page in the normal ECS UI. To create a Cluster, click the blue “Create Cluster” button:



Create ECS Cluster

Give the cluster a name, such as `my-ecs-cluster`, click the blue “Create” button, and your new Cluster will show up on the Clusters page:



Your new ECS Cluster

Notice how your Cluster shows zero “Registered Container Instances”. You need to create a bunch of new EC2 Instances and register them in the Cluster. Deploying, monitoring, and updating many EC2 Instances manually is tedious and error prone. A better solution is to use an *Auto Scaling Group* and an *Elastic Load Balancer*.

You can define an [Auto Scaling Group](#) to automatically launch multiple EC2 Instances based on rules you define. For example, you could define rules like “keep 5 EC2 Instances running at all times” or “always maintain a minimum of 3 EC2 Instances, but add one every time the CPU load is above 90% for more than 15 minutes, and remove one every time the CPU load drops

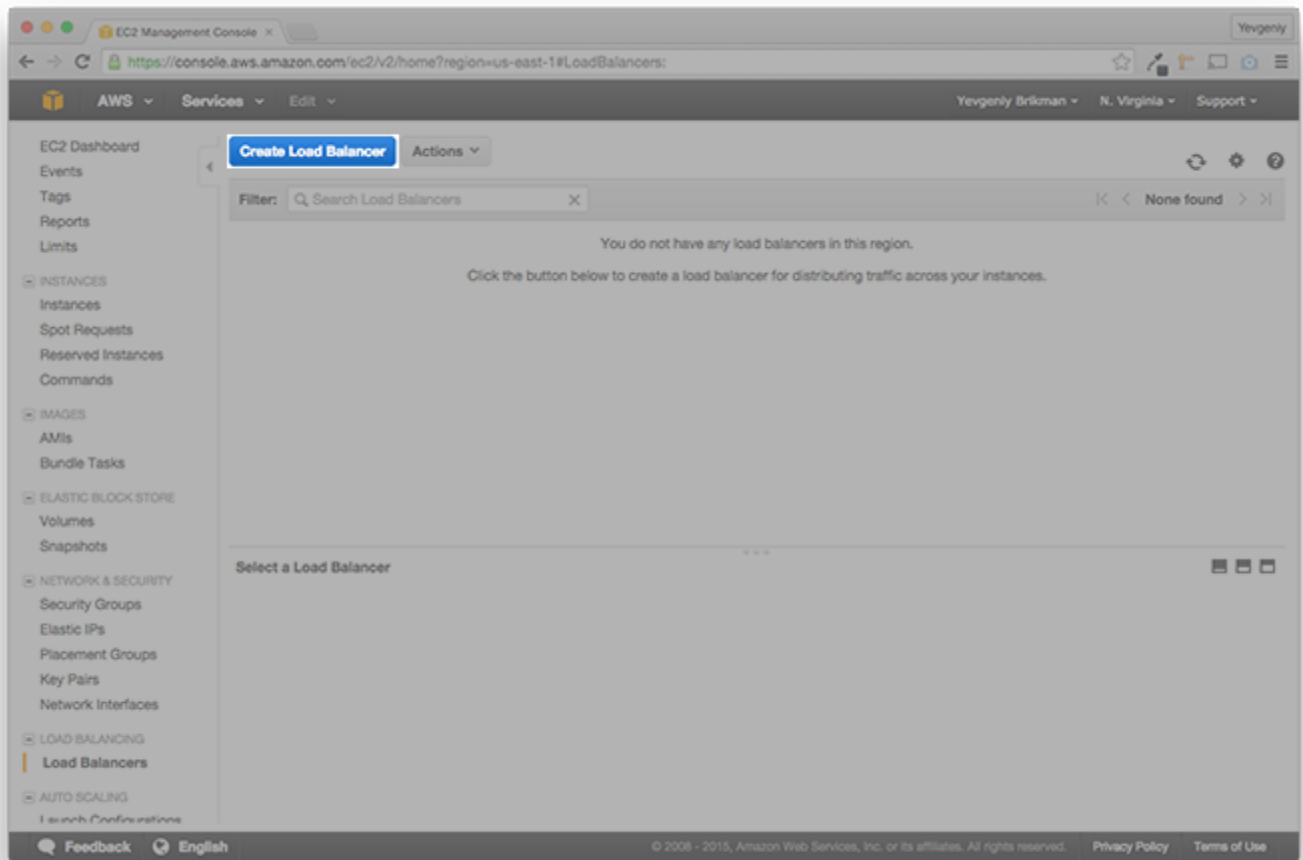
below 30% for more than 15 minutes” (you can feed the Auto Scaling Group information about the CPU load and other metrics from Amazon’s [CloudWatch](#) service).

You can use an [Elastic Load Balancer](#) (ELB) when you are running multiple EC2 Instances and you want to distribute load between them. The ELB monitors the health of your EC2 Instances, so if one goes down (due to a crash or an Auto Scaling Group reducing the number of instances) it can take it out of rotation or if a new one comes up (due to a new deployment or an Auto Scaling Group increasing the number of instances) it can add it to the rotation. Your users always send their requests to the ELB, so they are shielded from any changes within your data center.

Let’s create the ELB first and then move on to the Auto Scaling Group.

Creating an ELB

To create an ELB, open the [EC2 Console](#) (mouse over the “Services” menu at the top and click “EC2”), click the “Load Balancers” link in the bottom left, and click the blue “Create Load Balancer” button:




Create a Load Balancer

Give the ELB a name such as `ecs-load-balancer` and take a look at the “Listener Configuration” settings. The ELB can only route traffic from one port to another, such as routing all HTTP traffic that it gets on port 80 to port 80 of any EC2 Instances you attach to it. This limited feature set can be a problem, as we’ll discuss towards the end of the blog post. However, this configuration will work for our example, so leave those settings as-is and click the gray “Next: Assign Security Groups” button:

EC2 Management Console

← → ↺ https://console.aws.amazon.com/ec2/v2/home?region=us-east-1#CreateELBWizard

 AWS

Services

Edit

1. Define Load Balancer

2. Assign Security Groups

3. Configure Security Settings

4. Configure Health Checks

Step 1: Define Load Balancer

Basic Configuration

This wizard will walk you through setting up a new load balancer. Begin by giving your new load balancer a name. You also need to configure ports and protocols for your load balancer. Traffic from your clients can be routed to your load balancer with a standard web server on port 80.

Load Balancer name:

ecs-load-balancer

Create LB Inside:

My Default VPC (172.31.0.0/16)

Create an internal load balancer:

☐ [\(what's this?\)](#)

Enable advanced VPC configuration:

☐

Listener Configuration:

| Load Balancer Protocol | Load Balancer Port | Instance Protocol |
|------------------------|--------------------|-------------------|
| HTTP | 80 | HTTP |
| <div>Add</div> | | |

Feedback

English


© 2017 Amazon.com, Inc. or its affiliates. All rights reserved.

Give the Load Balancer a name

On the next page, click the “Select an existing security group” radio button, click the checkbox next to the Security Group you created earlier (`ssh-and-http-from-anywhere`), and click the gray “Next: Configure Security Settings” button:

EC2 Management Console

← → ↺ <https://console.aws.amazon.com/ec2/v2/home?region=us-east-1#CreateELBWizard>

 AWS

Services

Edit

1. Define Load Balancer

2. Assign Security Groups

3. Configure Security Settings

4. Configure Health Checks

Step 2: Assign Security Groups

You have selected the option of having your Elastic Load Balancer inside of a VPC, which allows you to assign a security group to the load balancer. This can be changed at any time.

Assign a security group: ☐ Create a new security group

☒ Select an **existing** security group

| | Security Group ID | Name | Description |
|-------------------------------------|-------------------|----------------------------|----------------|
| <input type="checkbox"/> | sg-7aaf7c1c | default | default VPC |
| <input checked="" type="checkbox"/> | sg-54a42932 | ssh-and-http-from-anywhere | Allow incoming |

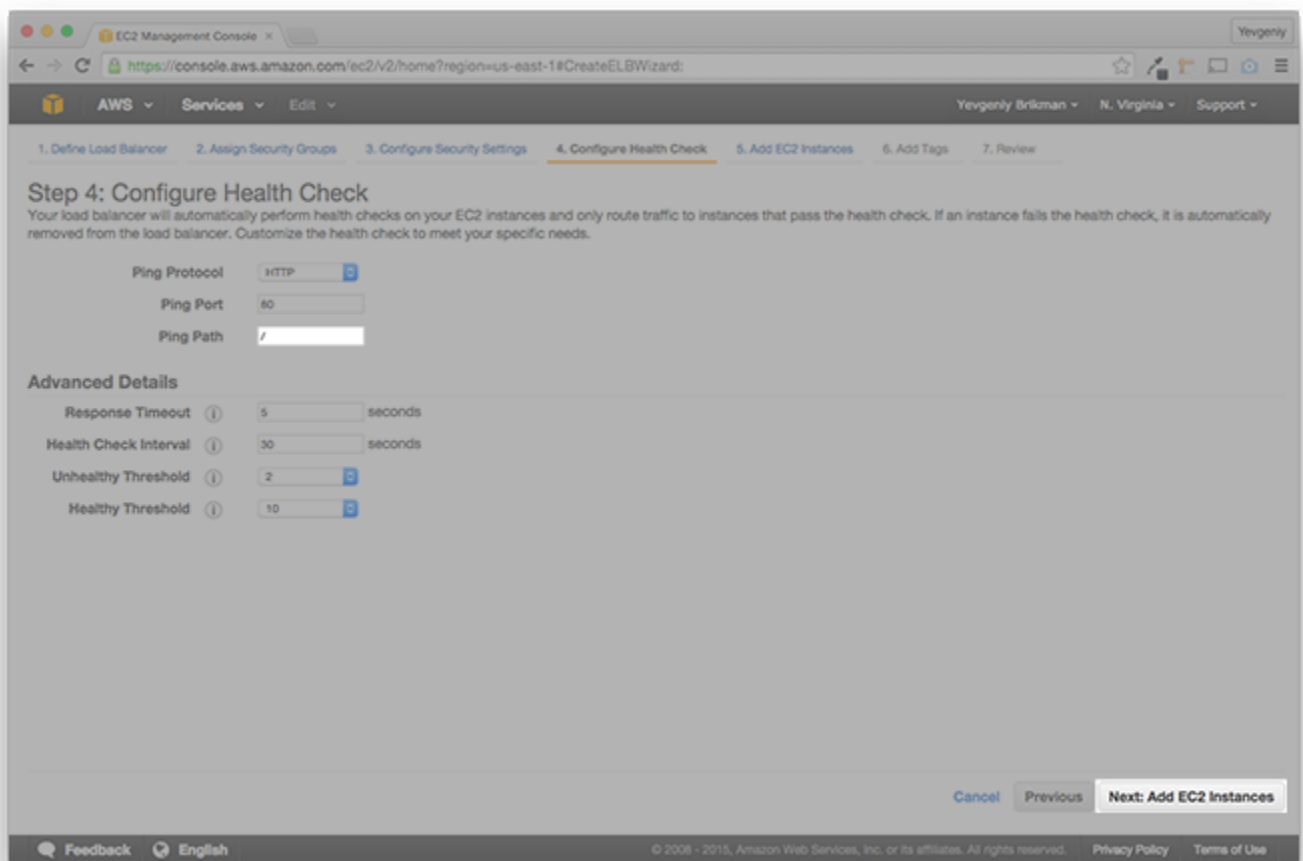
Feedback

English

© 2017 Amazon.com, Inc. or its affiliates. All rights reserved.

Select the Security Group you created earlier

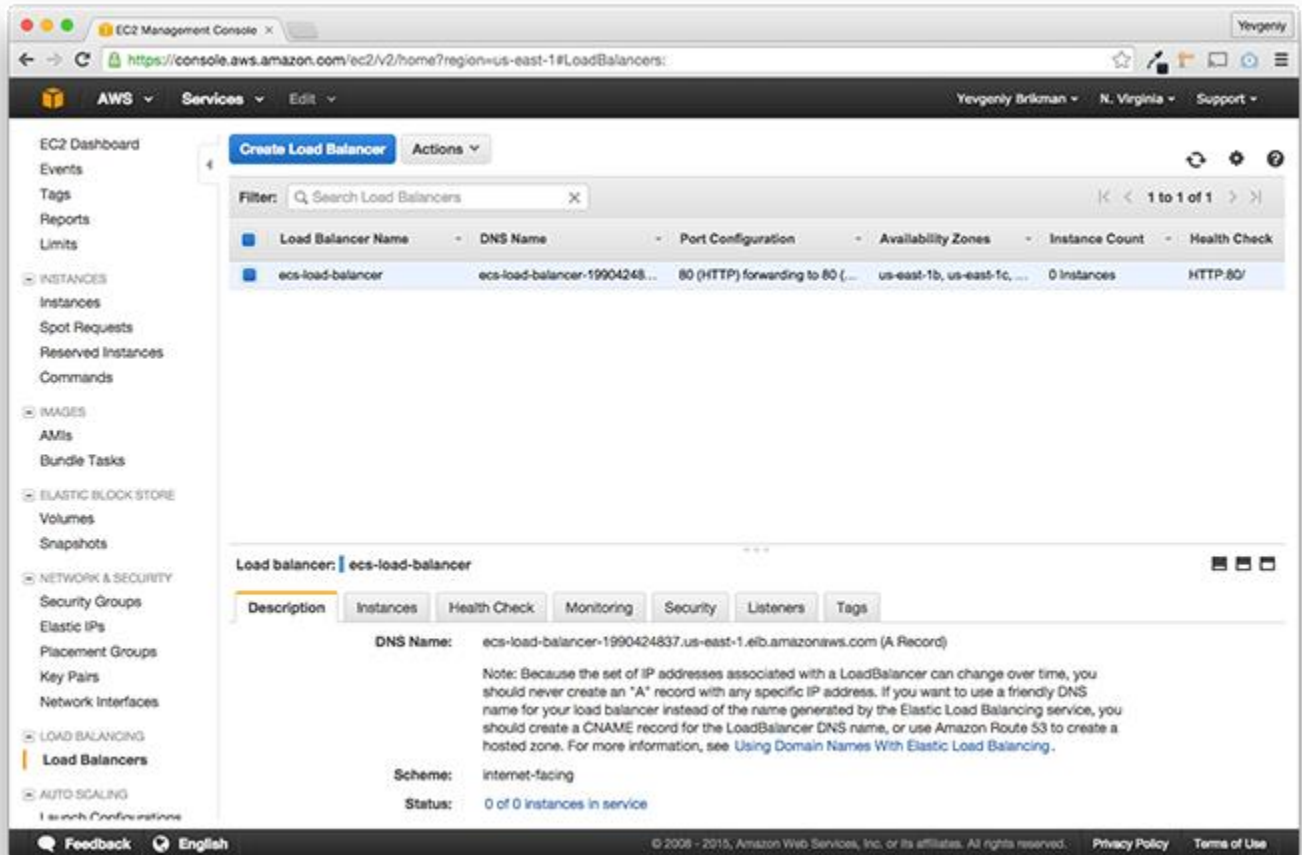
Ignore the warning (we aren't using SSL in this example) and click the gray "Next: Configure Health Check" button. The ELB uses a [Health Check](#) to periodically check if the EC2 Instances its routing to are actually up and running. It does this by sending a *Ping* (usually, an HTTP request) to a configurable URL on each EC2 Instance at a configurable interval. If the EC2 Instance responds with a 200 OK within a configurable period of time, it is considered healthy; if it doesn't, it is taken out of the ELB's rotation until future Pings mark it as healthy. For this tutorial, set the *Ping Path* to `/` (the only URL our Docker container knows how to handle) and click the gray "Add EC2 Instances" button:



Set the Health Check Ping Path to `/`

On the next page, you can manually add EC2 Instances to the ELB, but we're going to add Instances a different way (using an Auto Scaling Group), so skip this for now by clicking the gray "Next: Add Tags" button, then the blue "Review and Create" button, and finally, the blue

“Create” button. Once the ELB is created, click the blue “Close” button on the confirmation page and you should see your new ELB in the list:



Your newly created ELB

Before creating the Auto Scaling Group, we need to take a brief aside to deal with security in the form of IAM Roles.

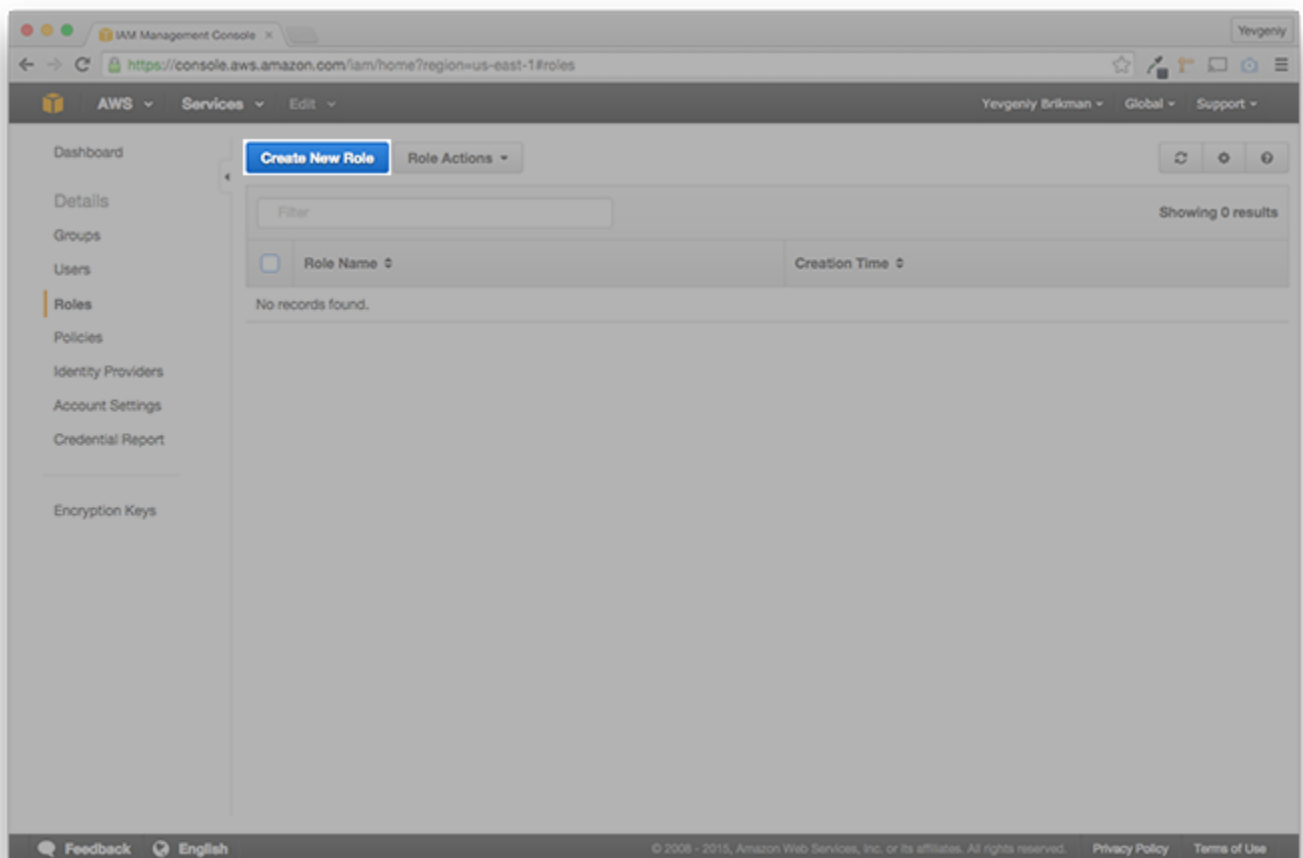
Creating IAM Roles

[IAM](#), which stands for Identity and Access Management, is the mechanism AWS uses to:

1. Define [Identities](#), such as a User, Group, or Role (authentication).
2. Define [Permissions and Policies](#) that specify what an Identity is or isn't allowed to do (authorization).

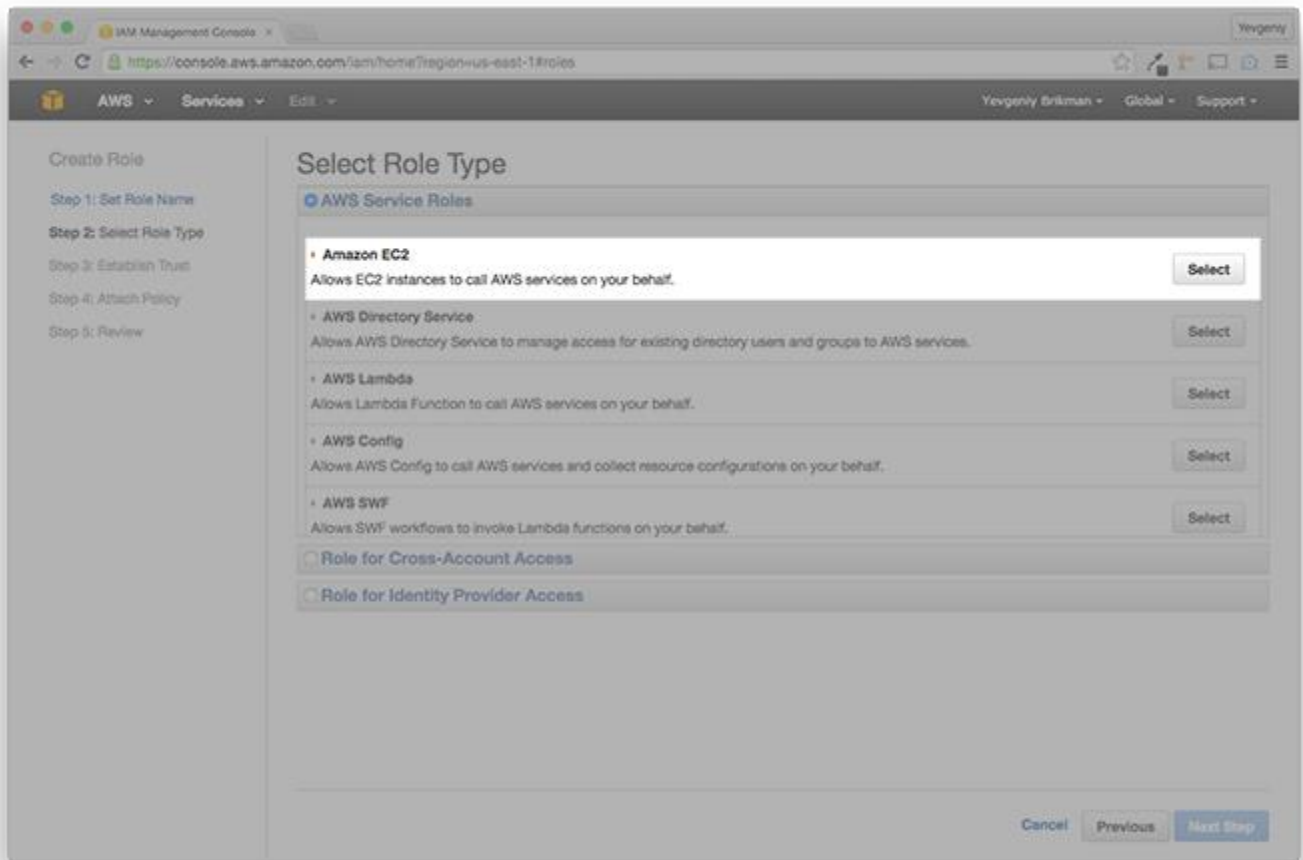
For example, later in this post, you are going to create a bunch of EC2 Instances that, when they boot up, need to register themselves in your ECS Cluster. By default, your EC2 Instances do not have the Permissions to talk to an ECS Cluster, so you need to create an [IAM Role](#)—an Identity with set of Permissions—and attach it to your EC2 Instances (for more info, see [Amazon ECS Container Instance IAM Role](#)).

Head over to the [IAM Console](#) (you can mouse over the “Services” menu and click the “IAM” link), click the “Roles” link on the left side, and click the blue “Create New Role” button:



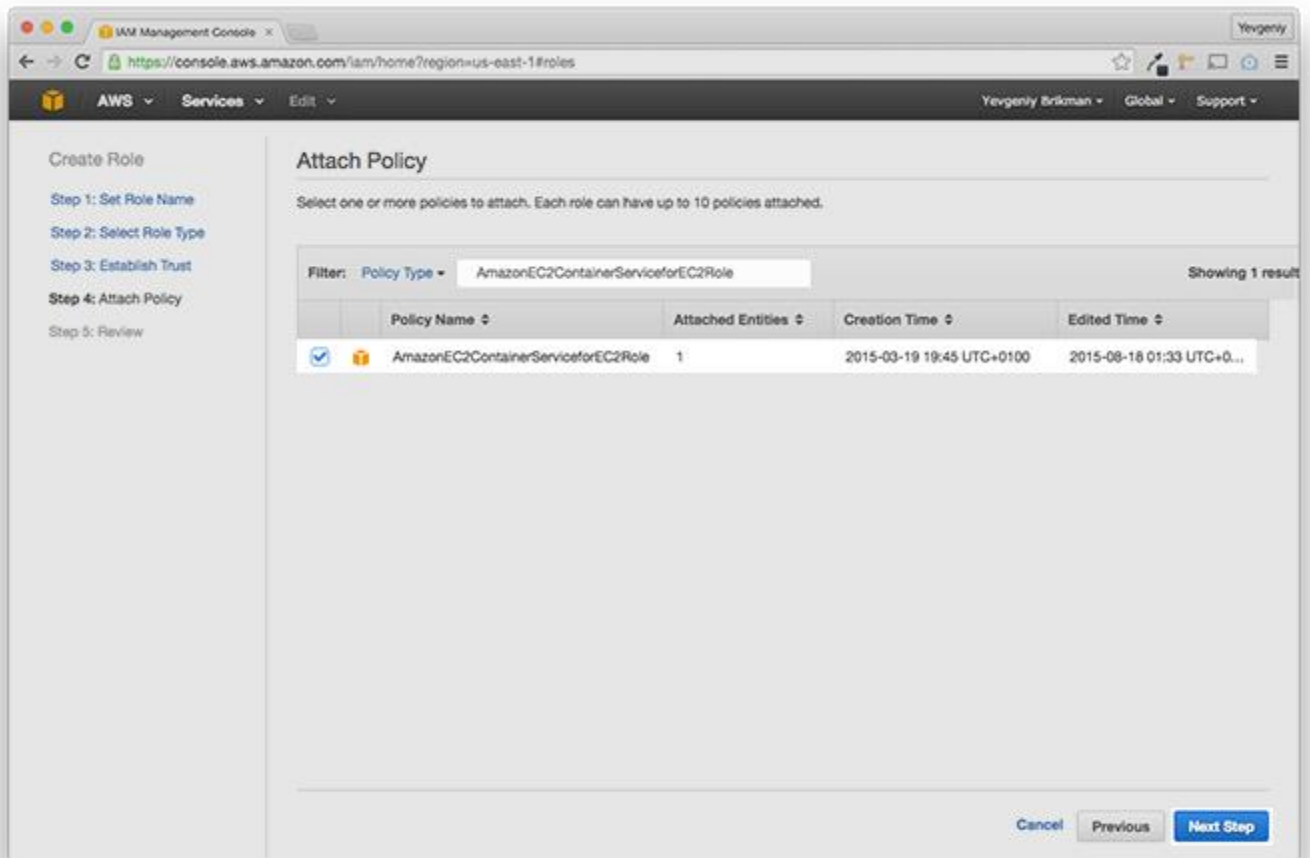
Create an IAM Role

Give the role a name, such as `ecs-instance-role`, and click the blue “Next Step” button in the bottom right. From the “AWS Service Roles” list, click the gray “Select” button next to `Amazon EC2`:



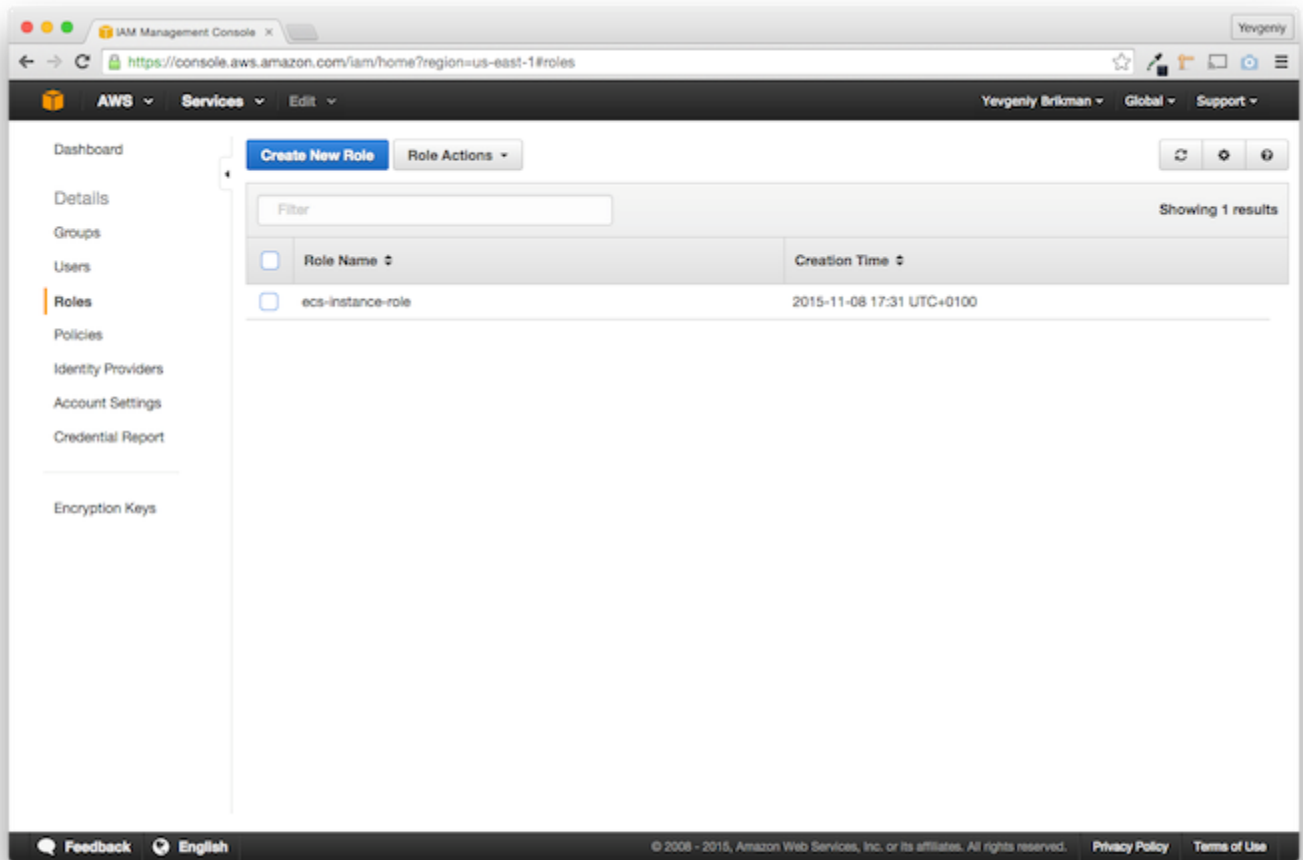
Select the Amazon EC2 Service Role

On the next page, search for `AmazonEC2ContainerServiceforEC2Role` (this is a pre-defined Policy that has all the Permissions you need), click the checkbox next to `AmazonEC2ContainerServiceforEC2Role`, and click the blue “Next Step button”:



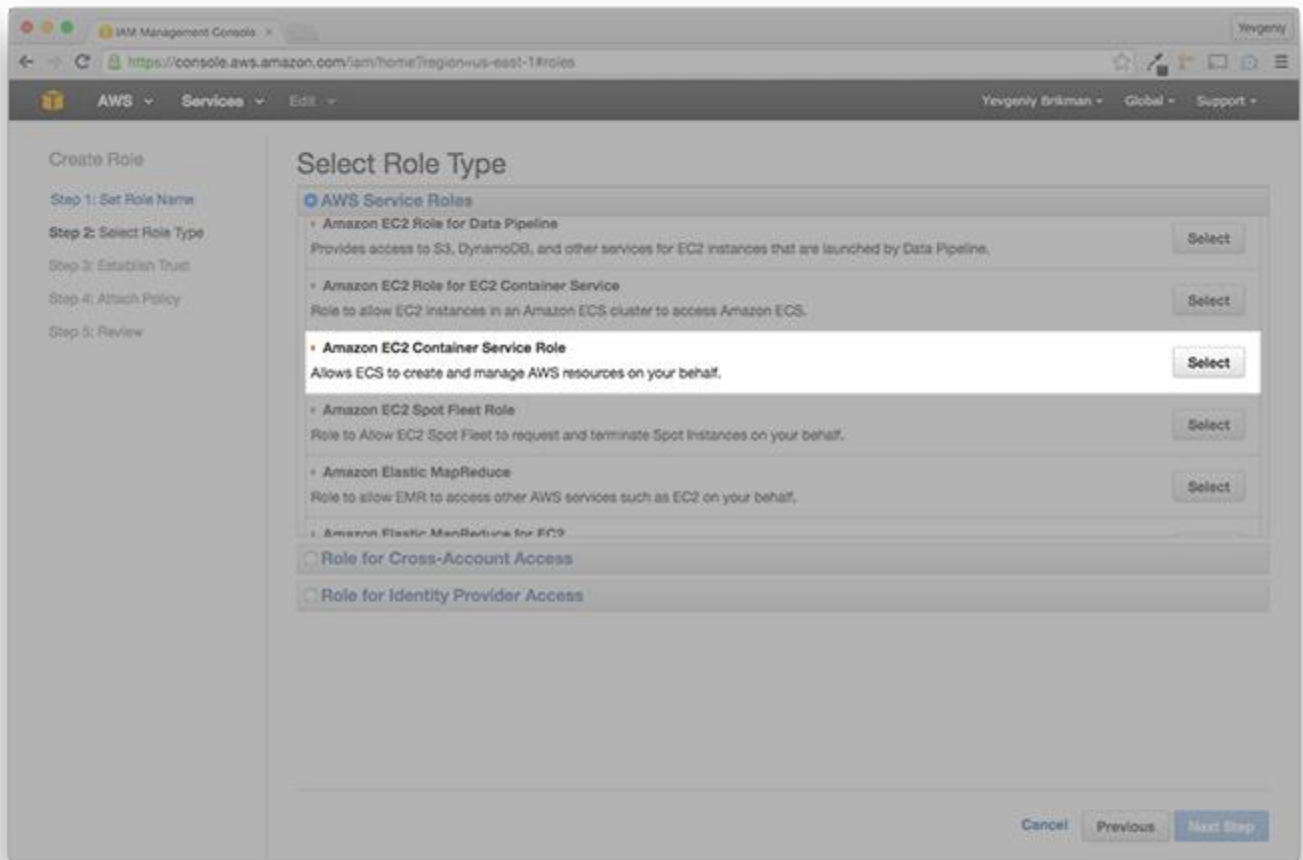
Check AmazonEC2ContainerServiceforEC2Role

Click the blue “Create Role” button and you should see your new IAM Role in the list:



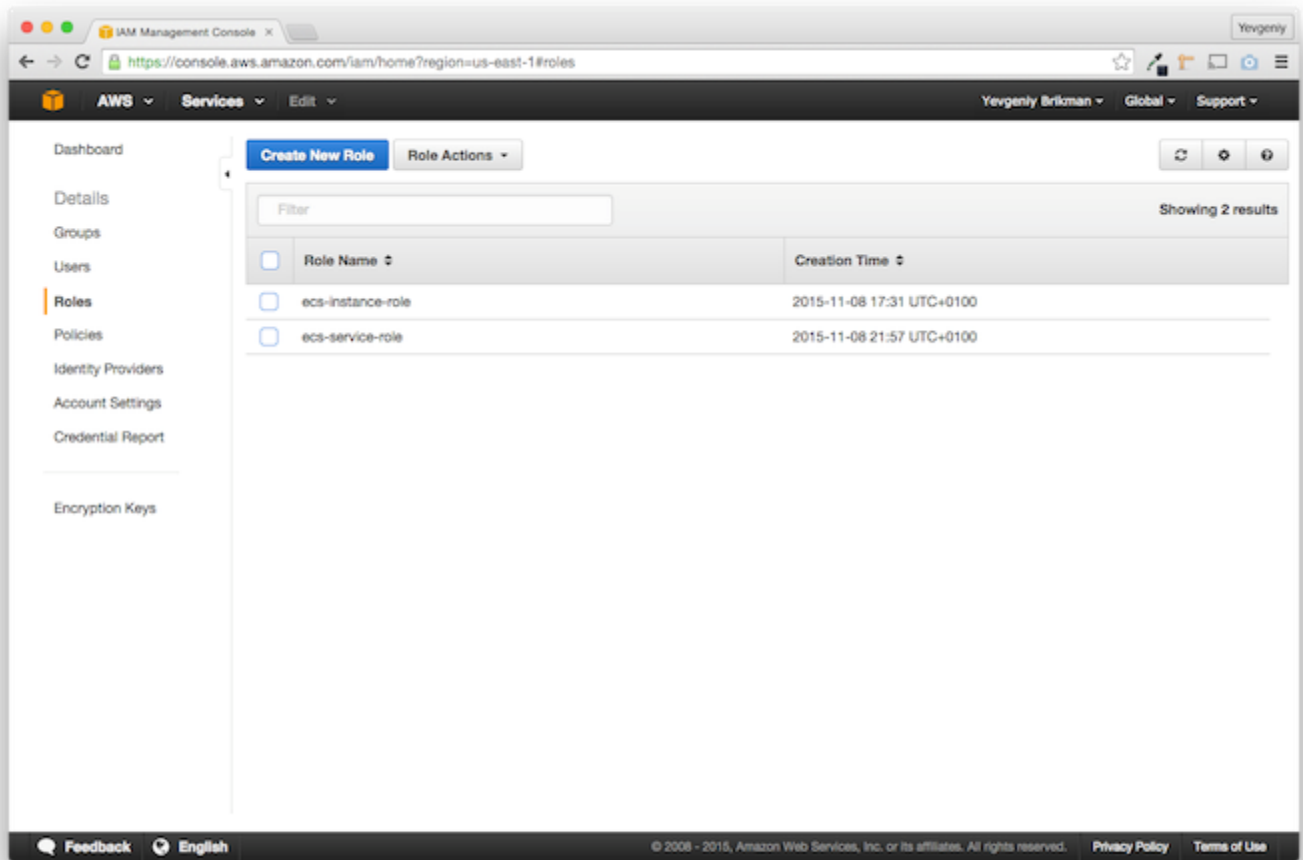
The new IAM Role

You need to create a similar IAM Role to allow the ECS Cluster to talk to your ELB so it can notify the ELB when it is deploying or undeploying Docker containers (see [Amazon ECS Service Scheduler IAM Role](#) for more info). Click the blue “Create New Role” button again, give the role a name such as `ecs-service-role`, and click the blue “Next Step” button in the bottom right. From the “AWS Service Roles” list, click the gray “Select” button next to `Amazon EC2 Container Service Role`:



Select the Amazon EC2 Container Service Role

Click the checkbox next to the `AmazonEC2ContainerServiceRole` (another pre-defined Policy, and the only one in the list), click the blue “Next Step” button in the bottom right, and then click the blue “Create Role” button. You should now have two IAM roles:

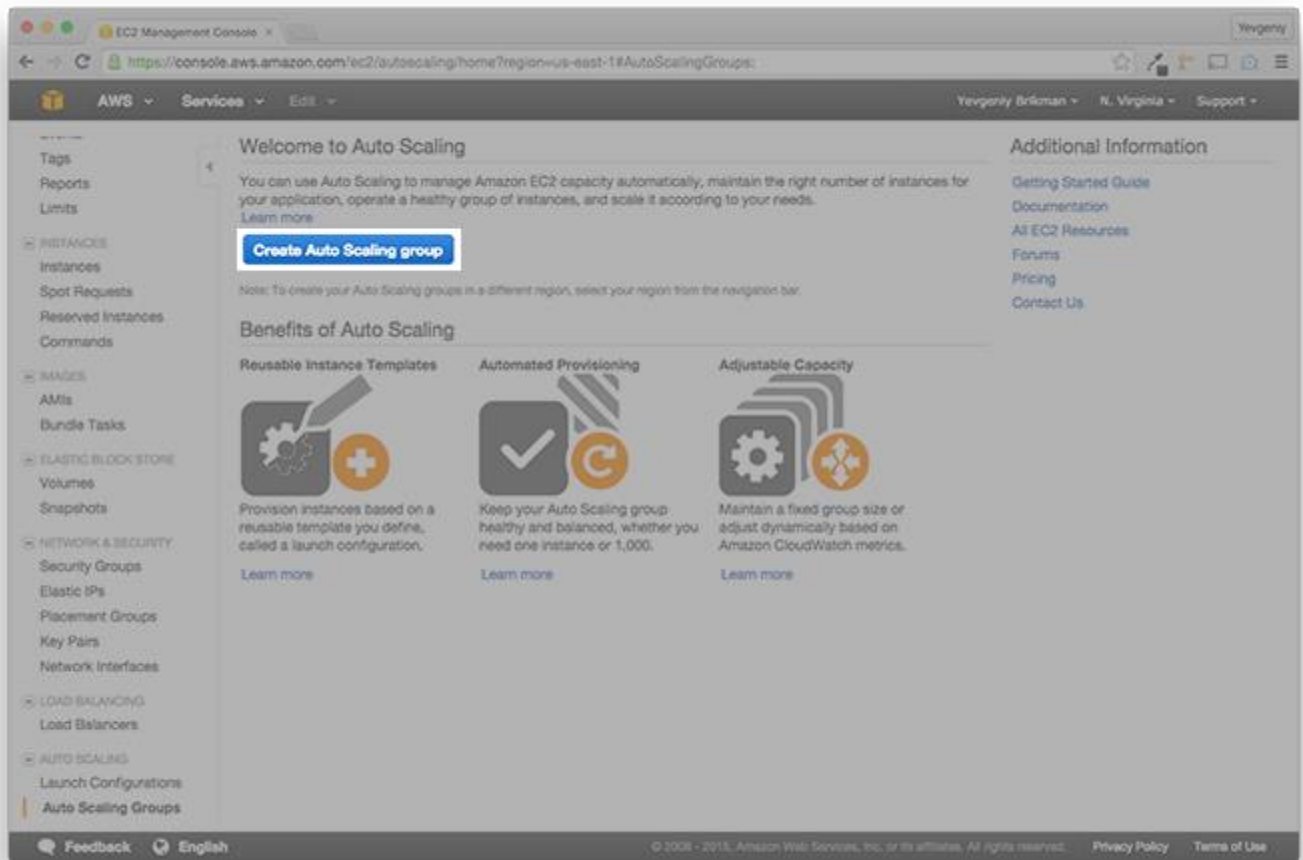


The EC2 Instance Role and the ECS Service Role

With all that out of the way, you can finally create your Auto Scaling Group.

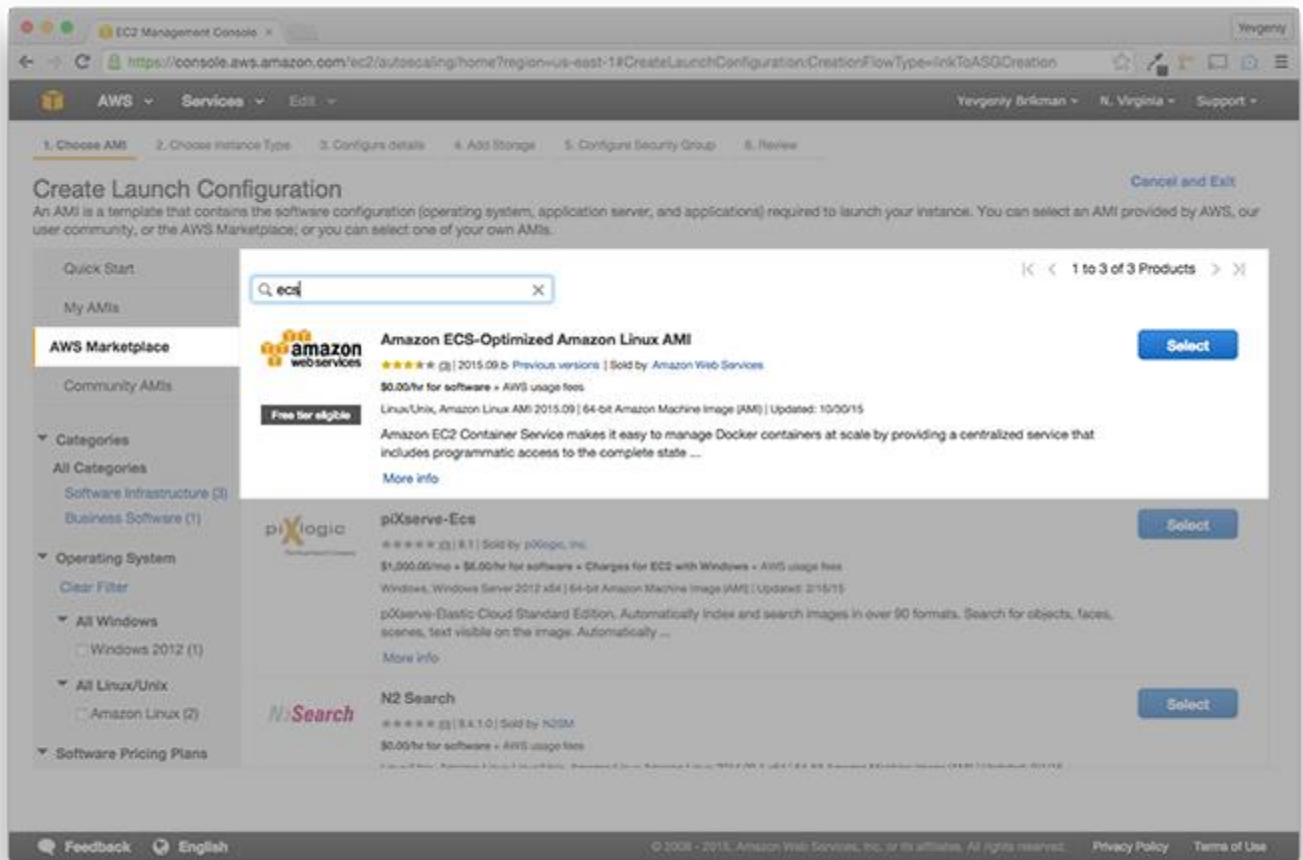
Creating an Auto Scaling Group

To create an Auto Scaling Group, open the [EC2 Console](#) (mouse over the “Services” menu at the top and click “EC2”), click the “Auto Scaling Groups” link in the bottom left, and click the blue “Create Auto Scaling Group” button:



Create an Auto Scaling Group

The first step in creating an Auto Scaling Group is to define a [Launch Configuration](#). This is a reusable template that defines what kind of EC2 Instances the Auto Scaling Group should launch, including the AMI, instance type, security group, and all the other details you saw when launching an EC2 Instance manually in the first part of this tutorial. Click the blue “Create launch configuration” button in the bottom-right corner to go to the AMI selection page. Instead of using the Amazon Linux AMI as before, click the “AWS Marketplace” tab on the left, search for `ECS`, and select the `Amazon ECS-Optimized Amazon Linux AMI` (don’t worry, it’s part of the AWS free tier):



Use the Amazon ECS-Optimized Amazon Linux AMI

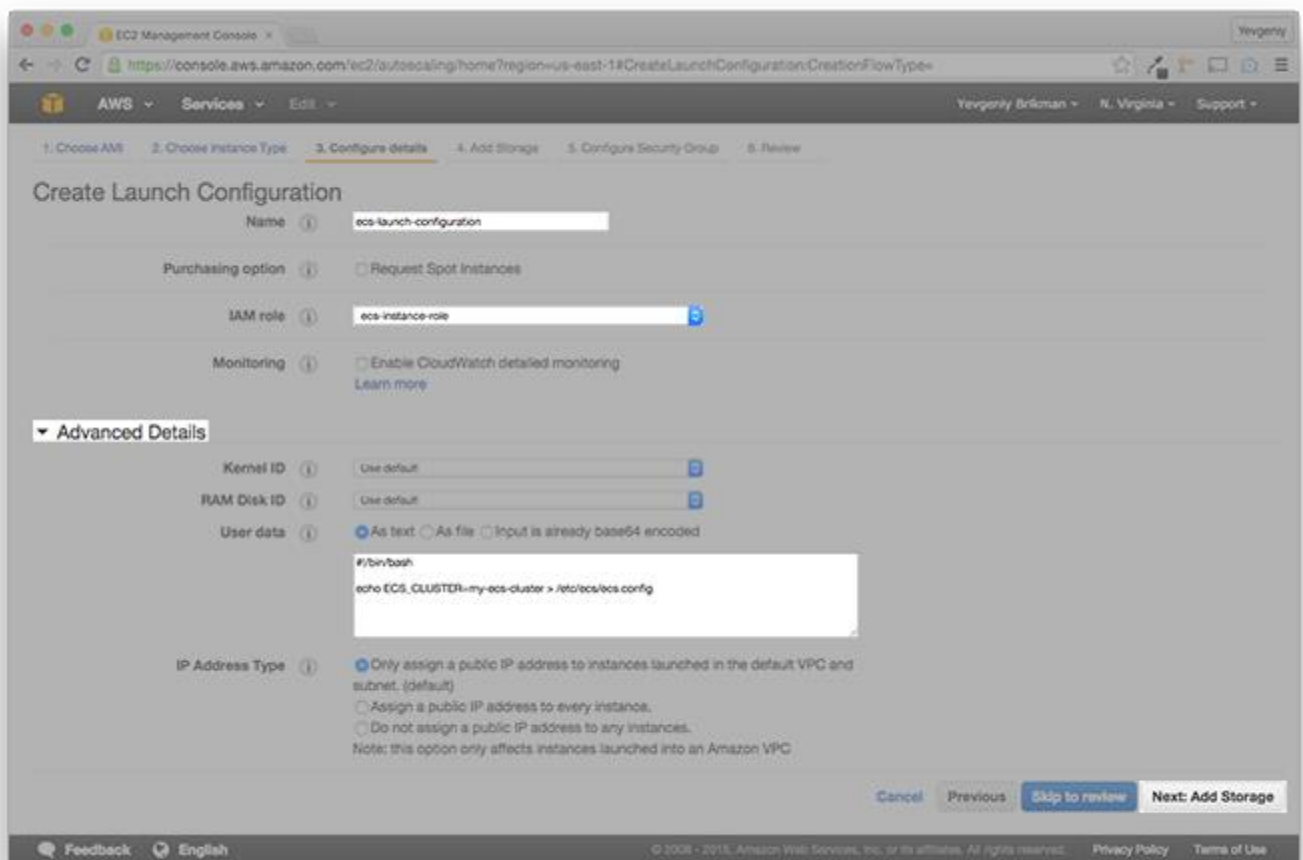
(Note, if this is your first time using the AWS Marketplace, you may have to accept the terms of service, as mentioned in [this comment by Petri Sirkkala](#).)

This Amazon ECS-Optimized Amazon Linux AMI includes the [ECS Container Agent software](#) that knows how to register this EC2 Instance in your ECS Cluster. How convenient! The only thing it needs is the name of your ECS Cluster, which we'll get to in just a moment. On the next page, select `t2.micro` as the instance type, and click the gray "Next: Configure details" button. Give the Launch Configuration a name, such as `ecs-launch-configuration`, select the instance IAM Role you created earlier from the drop-down list (`ecs-instance-role`), and then click the "Advanced Details" link to open up the bottom section. Find the text box labeled "User data" and enter the following shell script into it:

```
#!/bin/bash
```

```
echo ECS_CLUSTER=my-ecs-cluster > /etc/ecs/ecs.config
```

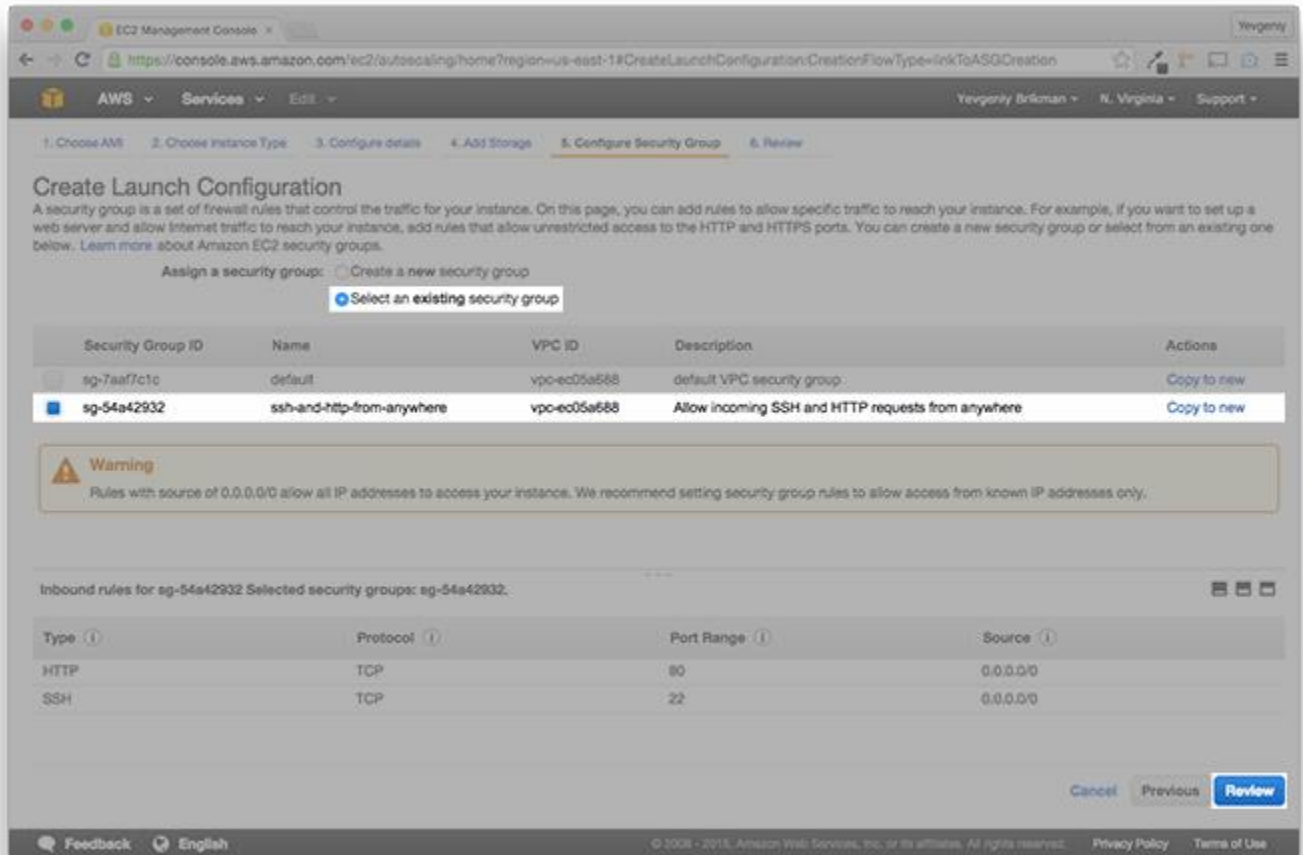
[User data](#) is a place you can put custom shell scripts that the EC2 Instance will run right after booting. The shell script above puts the name of your ECS Cluster (`my-ecs-cluster`, in this example) into the `/etc/ecs/ecs.config` file. The ECS Container Agent knows to look into this file, so this is how you provide it the name of your ECS Cluster. If you don't specify a name, the Agent will use `Default` (see [Amazon ECS Container Agent Configuration](#) for more info). Your Launch Configuration should look something like this:



Name the launch configuration and use your newly created IAM Role

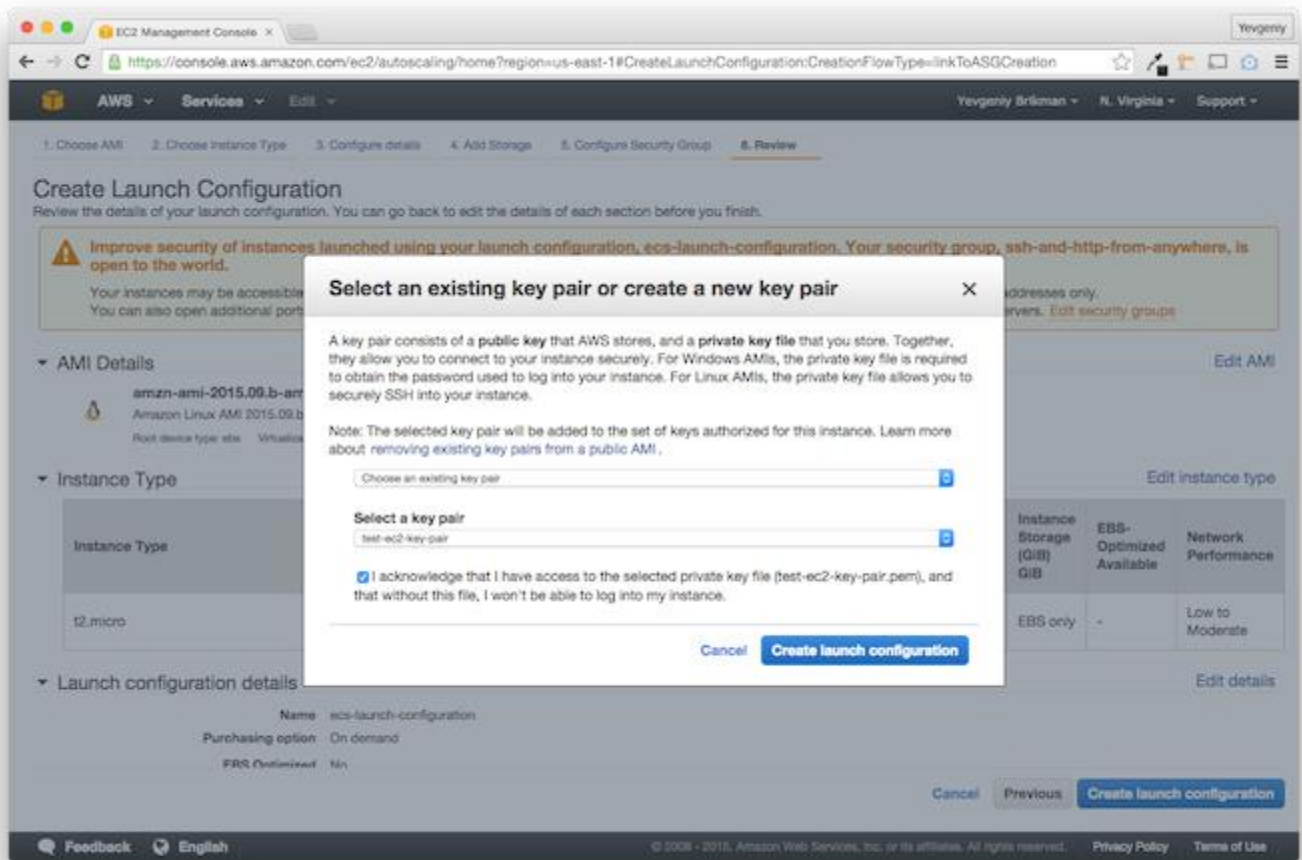
Click the gray “Next: Add Storage” button, leave all the default Storage configuration options, and click the gray “Next: Configure Security Group button”. On the next page, click the

“Select an existing security group” radio button, click the checkbox next to the Security Group you created earlier (`ssh-and-http-from-anywhere`), and click the blue “Review” button:



Use the Security Group you created earlier

Now click the blue “Create launch configuration button” and a modal will pop up asking you to select a Key Pair. Select “Choose an existing key pair” from the first drop-down box, select the Key Pair you created earlier (`my-ec2-key-pair`) from the second drop-down, click the “I acknowledge that I have access to the selected private key file...” checkbox, and click the blue “Create launch configuration” button:



Use the Key Pair you created earlier

Now that you've created a Launch Configuration, AWS should take you to a screen that is prompting you to create an Auto Scaling Group from that Launch Configuration. Give the Auto Scaling Group a name, such as `ecs-auto-scaling-group` and specify a Group size of 5, which will tell the Auto Scaling Group to initially launch 5 EC2 Instances.

Next, you need to pick what Subnet(s) to use. A [Subnet](#) is a range of IP addresses used to segregate AWS Resources (such as your EC2 Instances) from each other or from the public Internet. For example, you might put your front-end App Servers in a *Public Subnet*, with rules that make it accessible from the public Internet and you might put your databases in a *Private Subnet*, with rules that make it *only* accessible from the App Servers but not the public Internet or anywhere else. AWS also uses different Subnets for different Availability Zones. AWS has data centers in different locations all over the world. Each location is composed of different [Regions and Availability Zones](#), where a Region represents a geographical area

(e.g., `us-east-1` and `eu-west-1`) and an Availability Zone is an isolated location within a Region (e.g., `us-east-1a`, `us-east-1b`, `us-east-1c`).

Subnets, Regions, and Availability Zones are all large topics of their own, so I won't cover them here, but you may want to read the [AWS VPC](#) and [Regions and Availability Zones](#) documentation for more info. For now, pick any of the default Subnets from the drop-down list, and the Auto Scaling Group will deploy your EC2 Instances across them. In the “Advanced Details” section, click the “Receive traffic from Elastic Load Balancer(s)” check box, and select the ELB you created earlier (`ecs-load-balancer`). The page should look something like this:

EC2 Management Console

← → ↺ https://console.aws.amazon.com/ec2/autoscaling/home?region=us-east-1#Create

AWS

Services

Edit

1. Configure Auto Scaling group details

2. Configure scaling policies

3. Configure Notifications

4. Co

Create Auto Scaling Group

Launch Configuration ⓘ

ecs-launch-configuration

Group name ⓘ

ecs-auto-scaling-group

Group size ⓘ

Start with 5 instances

Network ⓘ

vpc-ec05a688 (172.31.0.0/16) (default)

Subnet ⓘ

subnet-3a4b6111(172.31.48.0/20) | Default in us-east-1b ×

subnet-3a6cb307(172.31.32.0/20) | Default in us-east-1e ×

Each instance in this Auto Scaling group will be assigned

▼ Advanced Details

Load Balancing ⓘ

☒ Receive traffic from Elastic Load Balancer(s)

ecs-load-balancer ×

Health Check Type ⓘ

☐ ELB ☒ EC2

Health Check Grace Period ⓘ

300 seconds

Monitoring ⓘ

Amazon EC2 Detailed Monitoring metrics, which are provided at 1-minute frequency, are not enabled for the launch configuration ec2-launch-configuration-1. Instances launched from it will use Basic Monitoring metrics.

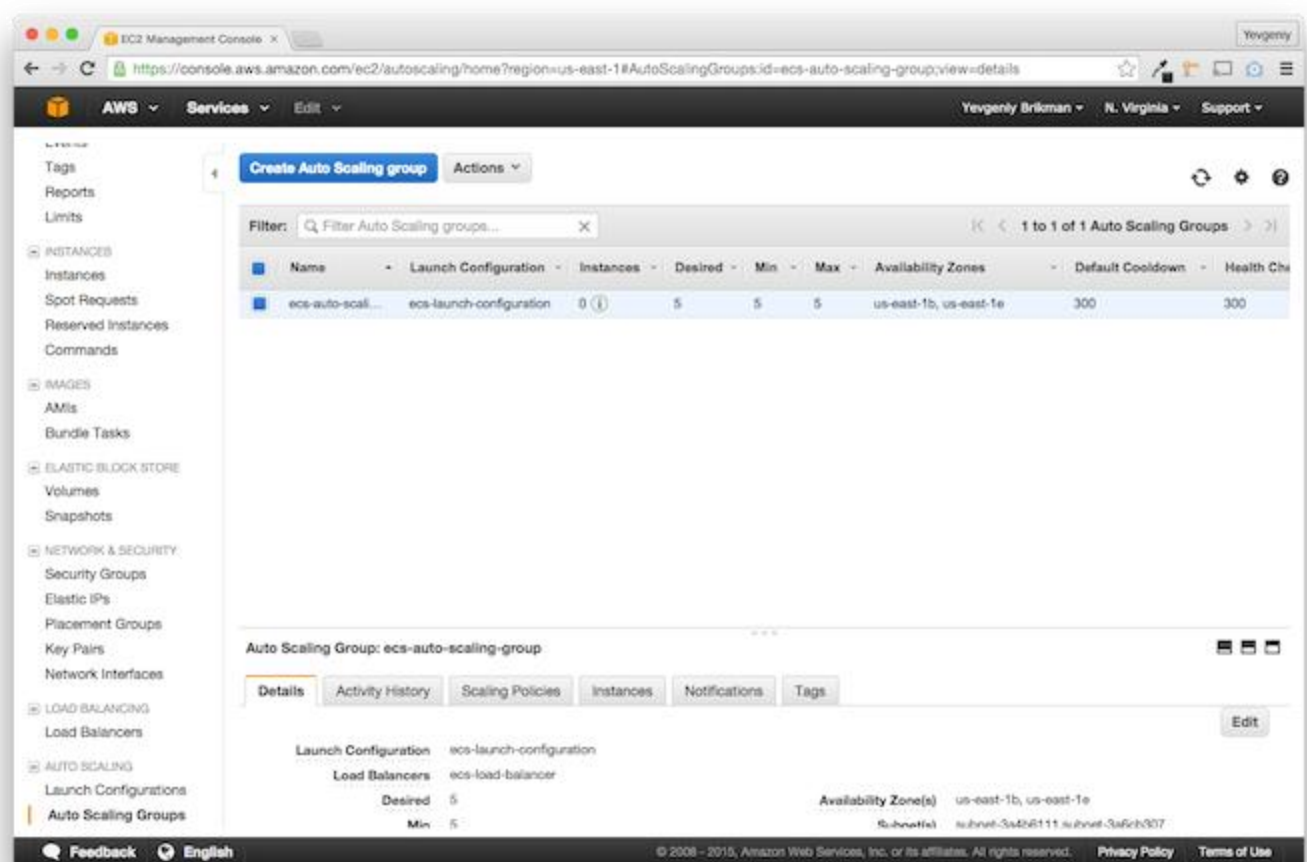
Feedback

English

© 2019 Amazon.com, Inc. or its affiliates. All rights reserved.

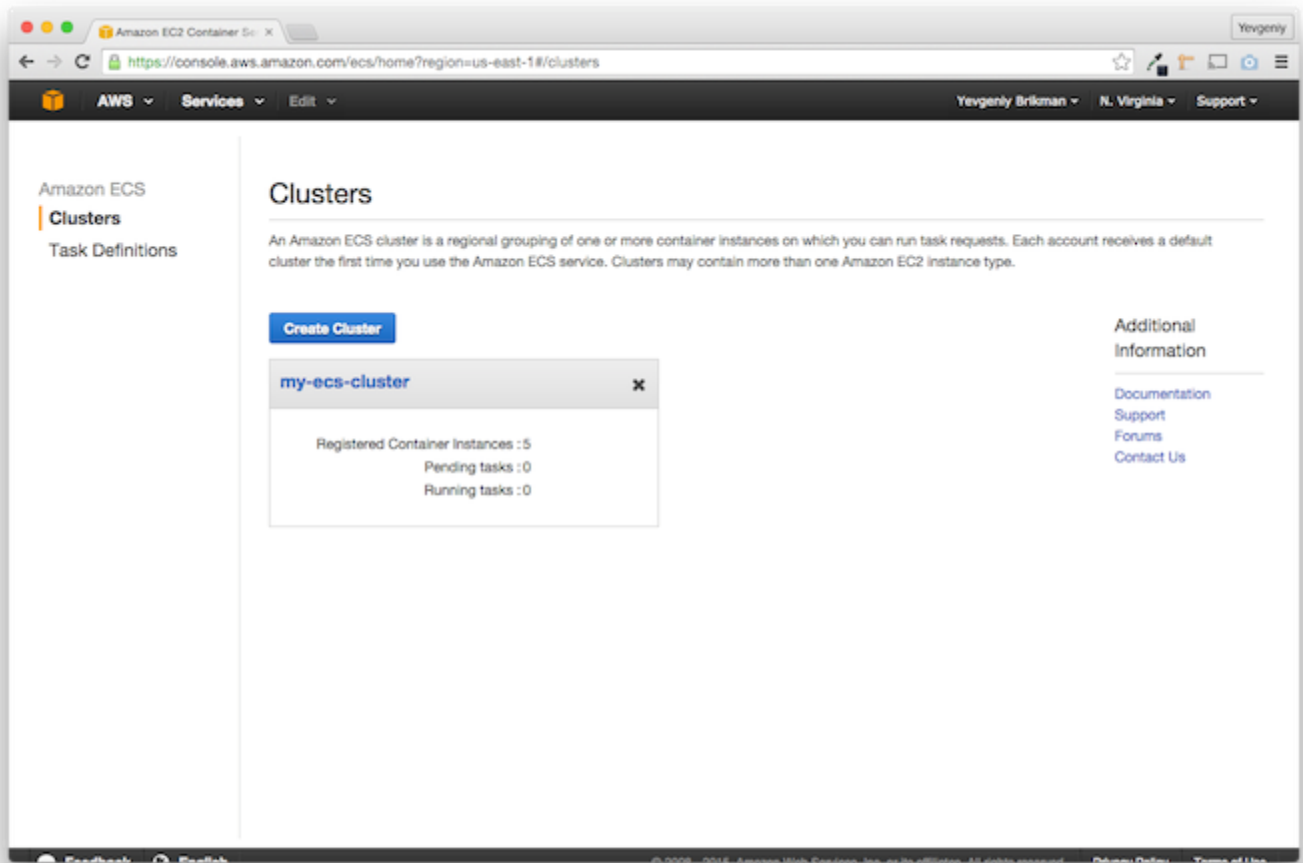
Give the Auto Scaling Group a name, a size of 5, a couple Subnets, select the ELB you created earlier

Click the gray “Next: Configure scaling polices” button. On the next page, you could configure rules for how to change the number of EC2 Instances in the Auto Scaling Group, but for this tutorial, you can leave the group at its initial size of 5, so just skip this section and click the blue “Review” button, followed by the blue “Create Auto Scaling Group” button. Once your Auto Scaling Group has been created, click the blue “Close” button on the confirmation screen and you should see your Auto Scaling Group in the list:



Your newly created Auto Scaling Group

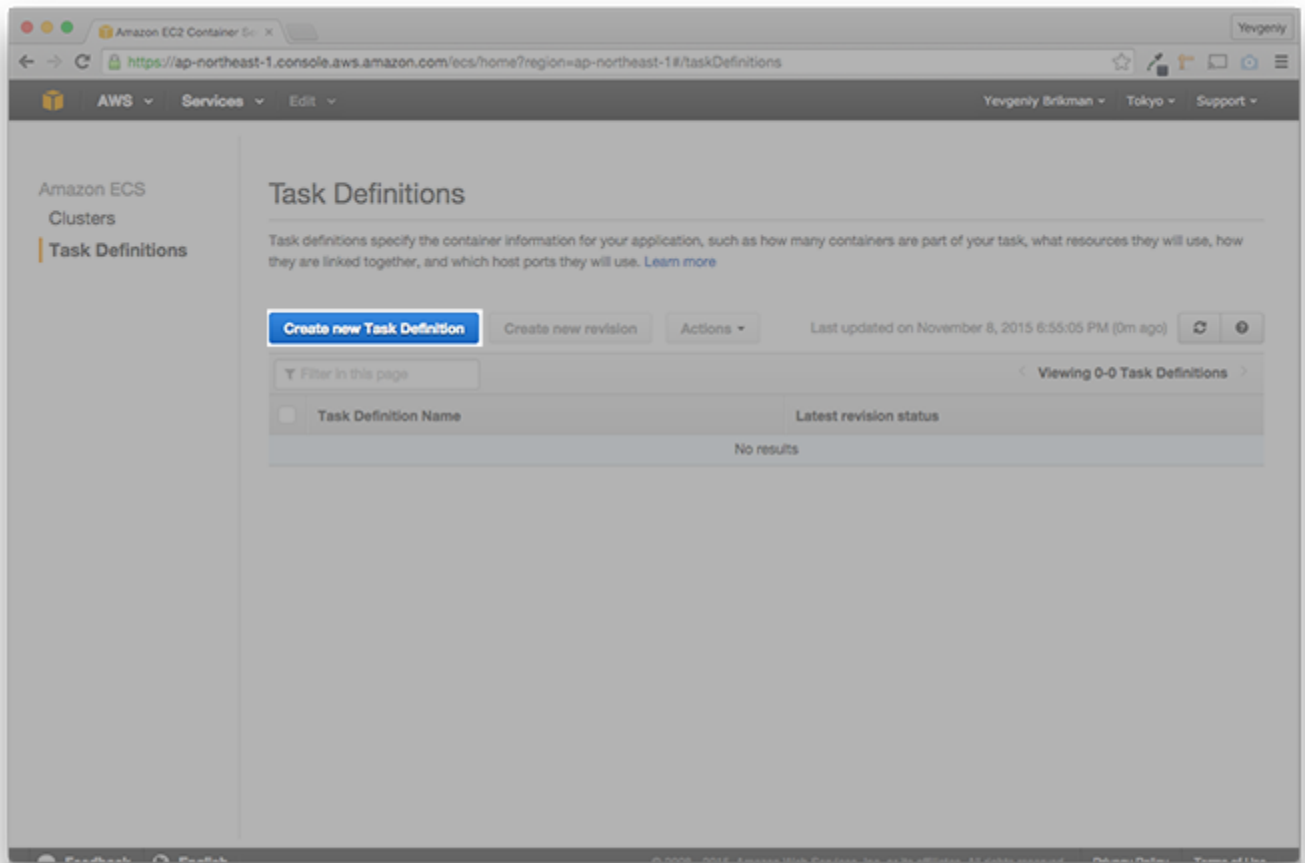
Initially, the Auto Scaling Group will show 5 “Desired Instances”, but 0 actually launched Instances. If you wait a minute and refresh the list, the number of launched Instances will go to 5. Head back to the [ECS Console](#), and you should now see five “Registered Container Instances” in your ECS Cluster:



Your ECS Cluster should now have 5 registered instances

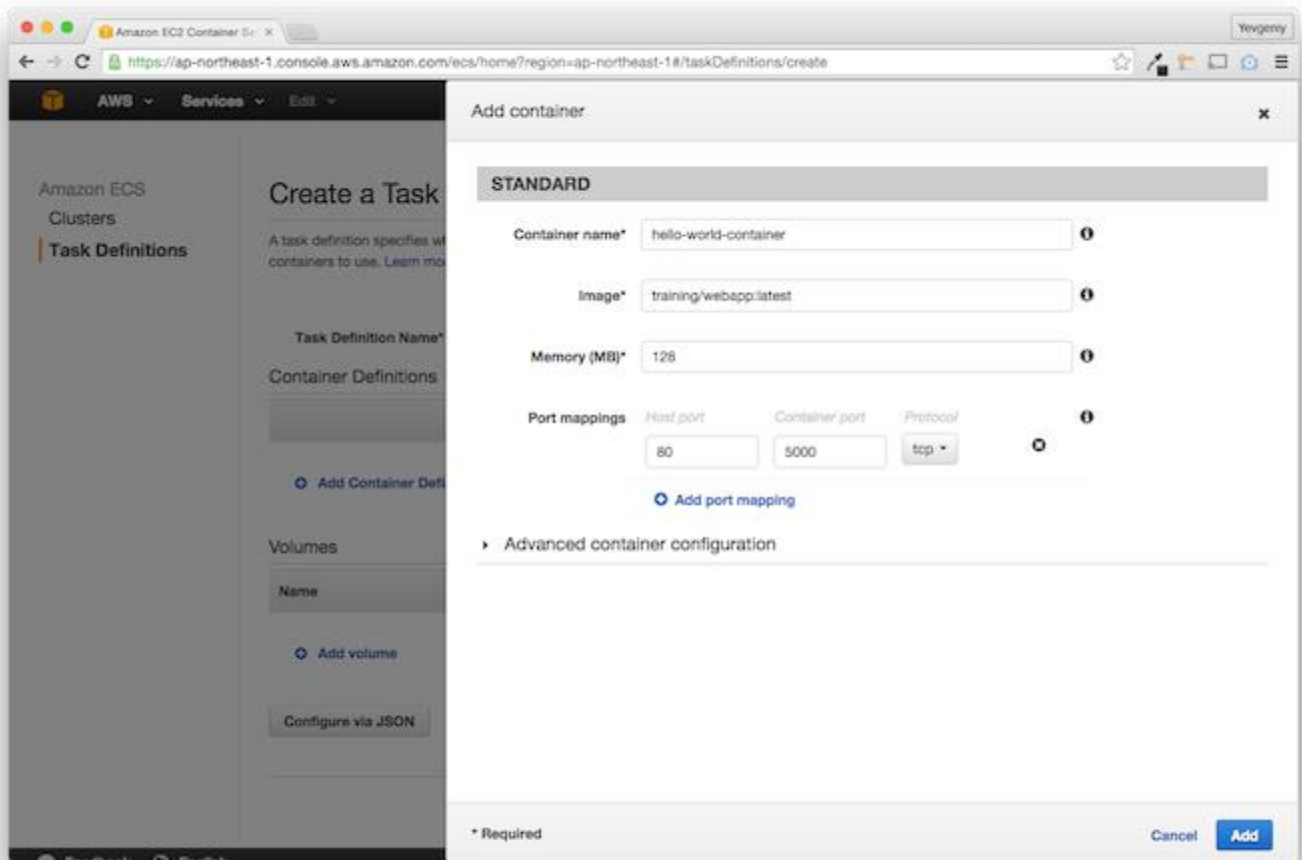
Running Docker containers in your Cluster

Now that you have a working Cluster, you can finally run some Docker containers in it. To do that, you first have to create an [ECS Task](#), which defines the Docker image(s) to run, the resources (CPU, memory, ports) you need, what volumes to mount, etc. Click the “Task Definitions” link on the left and then the blue “Create new Task Definition” button:



Create a new ECS Task

Give the Task a name, such as `hello-world-task` and click the “Add Container Definition” link. In the section that slides out, specify the Container name (e.g., `hello-world-container`), the Docker image to run (`training/webapp:latest`), the amount of memory to allocate (128 is plenty for this tutorial), the port mapping (map port 80 on the host to port 5000 in the Docker container), and click the blue “Add” button:

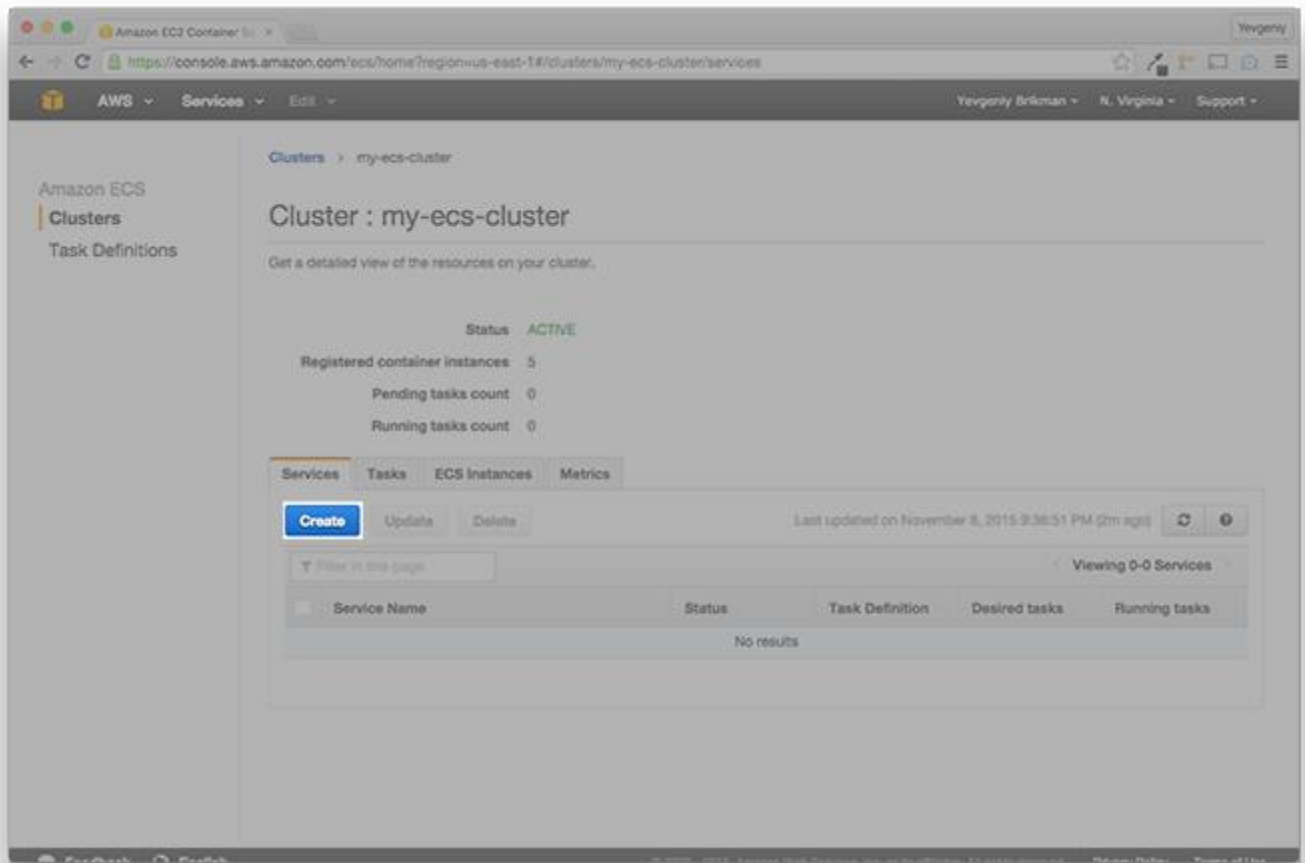


Task definition

Click the blue “Create” button to create the Task. Now it’s time to run the Task in your Cluster. Click on the “Clusters” link in the menu on the left and then click on the name of your Cluster (`my-ecs-cluster`). There are two ways to run Tasks:

1. **One-off tasks.** This is useful for a Task that runs once to completion and exits. See the “Tasks” tab in your Cluster.
2. **Services.** This is useful for Tasks that run continuously, such as a web service. See the “Services” tab in your Cluster.

Since `training/webapp` is a web app, we will run it as a Service. In the Services tab, click the blue “Create button”:



Create an ECS Service

Select the Task you created earlier (`hello-world-task:1`), give the Service a name (e.g., `hello-world-service`), specify that you want 4 tasks (one less than the number of EC2 Instances in your ECS Cluster, as we'll discuss later), select the ELB you created earlier (`ecs-load-balancer`), select the service IAM Role you created earlier (`ecs-service-role`), and click the blue "Create Service" button:

The screenshot shows the 'Create Service' page in the Amazon ECS console. The browser address bar displays the URL: `https://console.aws.amazon.com/ecs/home?region=us-east-1#/clusters/my-ecs-cluster/createService`. The page contains the following configuration fields:

- Task Definition:** A dropdown menu showing 'hello-world-task:1'.
- Cluster:** A dropdown menu showing 'my-ecs-cluster'.
- Service name:** A text input field containing 'hello-world-service'.
- Number of tasks:** A text input field containing '4'.

Below these fields is the **Elastic Load Balancing** section, which includes the text: 'An Elastic Load Balancer distributes incoming application traffic across the tasks running in your service. Select a load balancer or create a new ELB in the [EC2 Console](#).' The configuration fields are:

- Load Balancer : Host Port*:** A dropdown menu showing 'ecs-load-balancer:80'.
- Container Name : Host Port*:** A dropdown menu showing 'hello-world-container:80'.

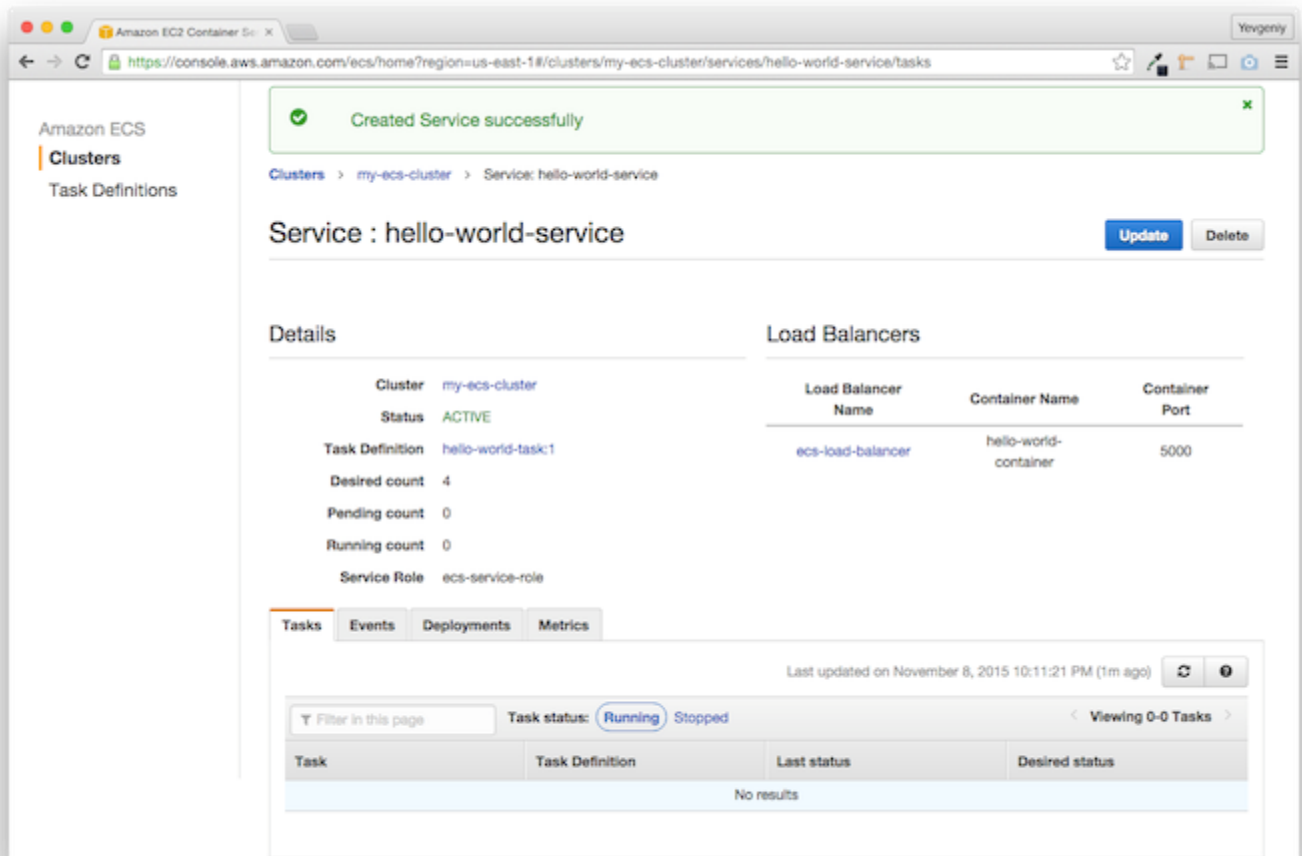
Next is the **Service Role** section, with the text: 'The IAM role for ECS Service grants the necessary permissions for ECS to interact with resources such as Elastic Load Balancing.' The configuration fields are:

- ECS service role*:** A dropdown menu showing 'ecs-service-role', followed by the text 'or' and a button labeled 'Create IAM Role'.

At the bottom right of the page are two buttons: 'Cancel' and 'Create Service'.

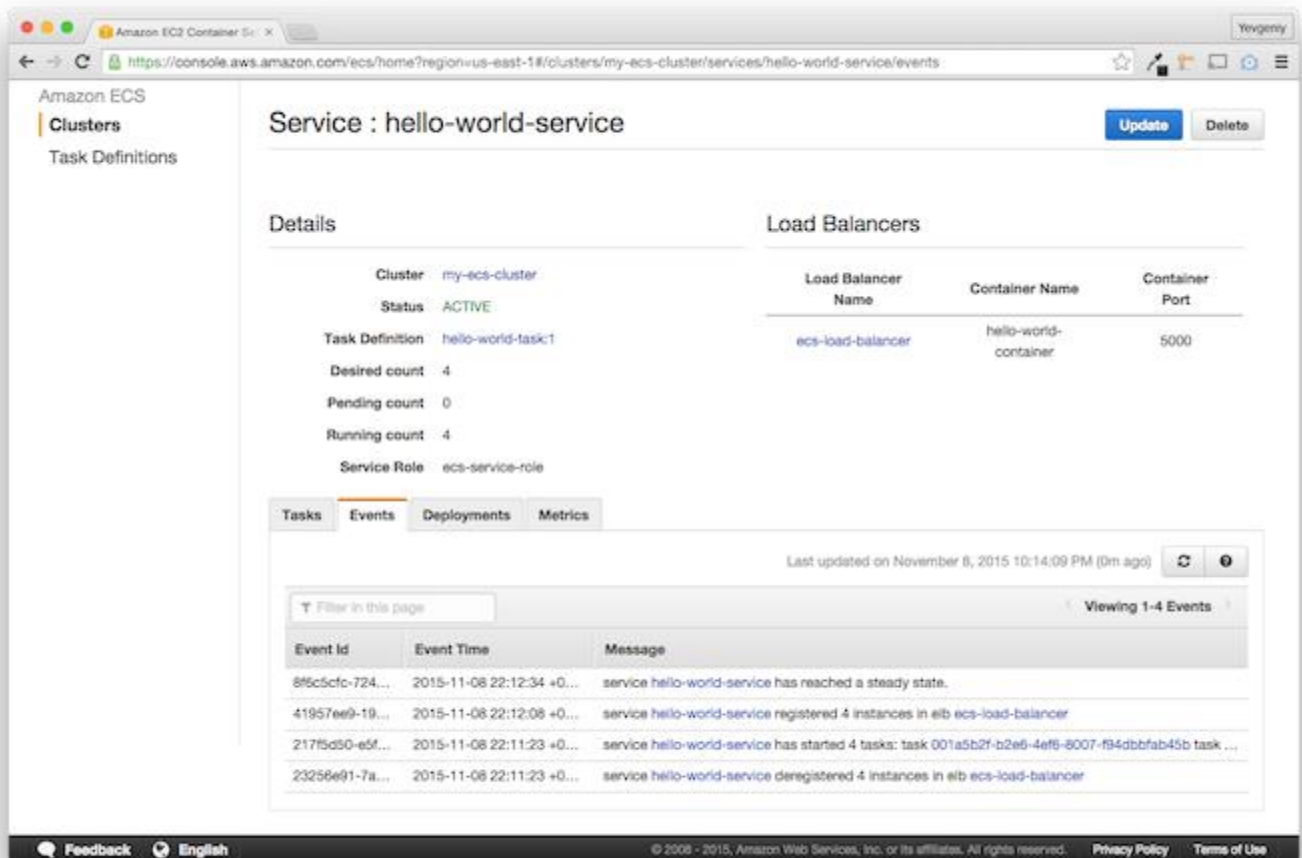
ECS Service Config

You should now see that your ECS Service has been created. Initially, the “desired count” will be at 4 and the “running count” will be at 0:



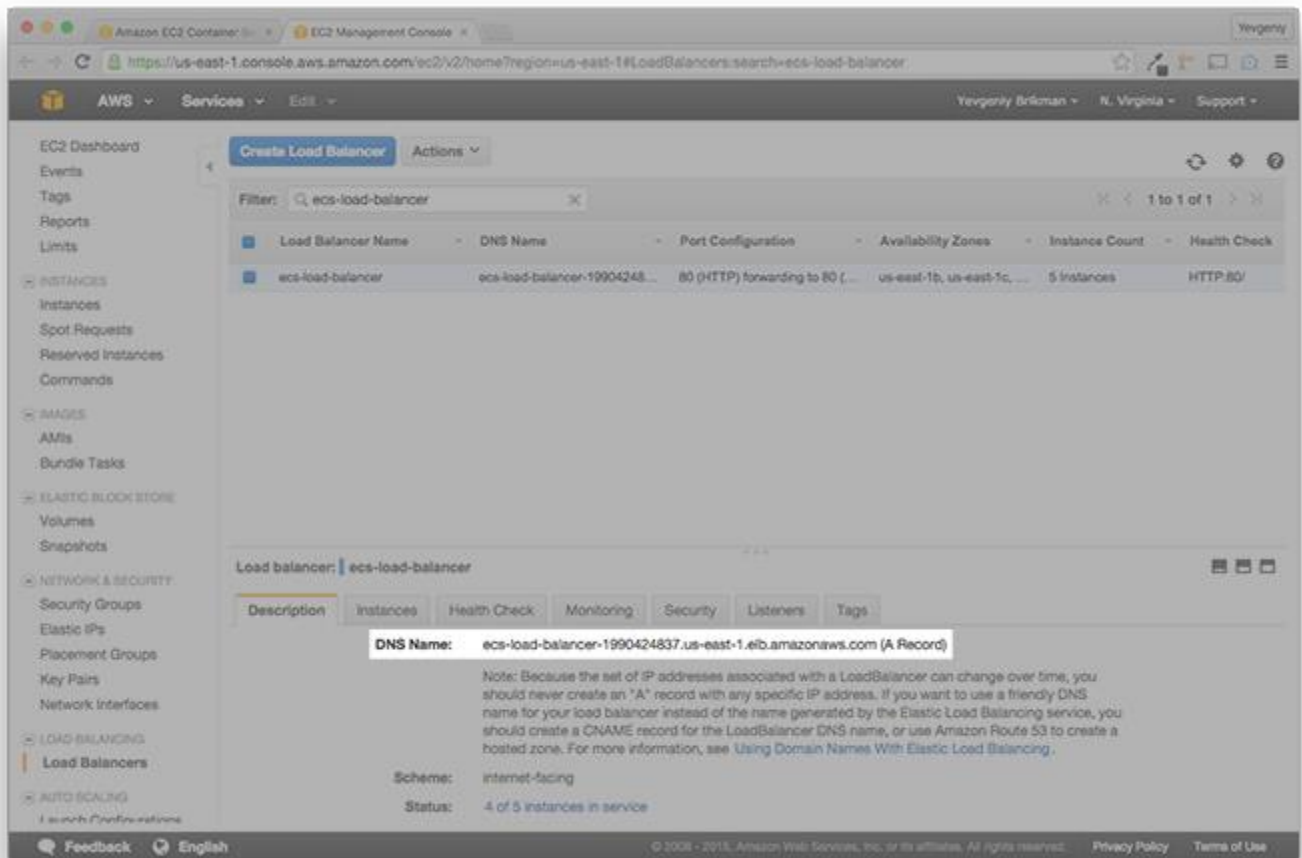
Your new ECS Service

Click the “Events” tab to see the deployment process:



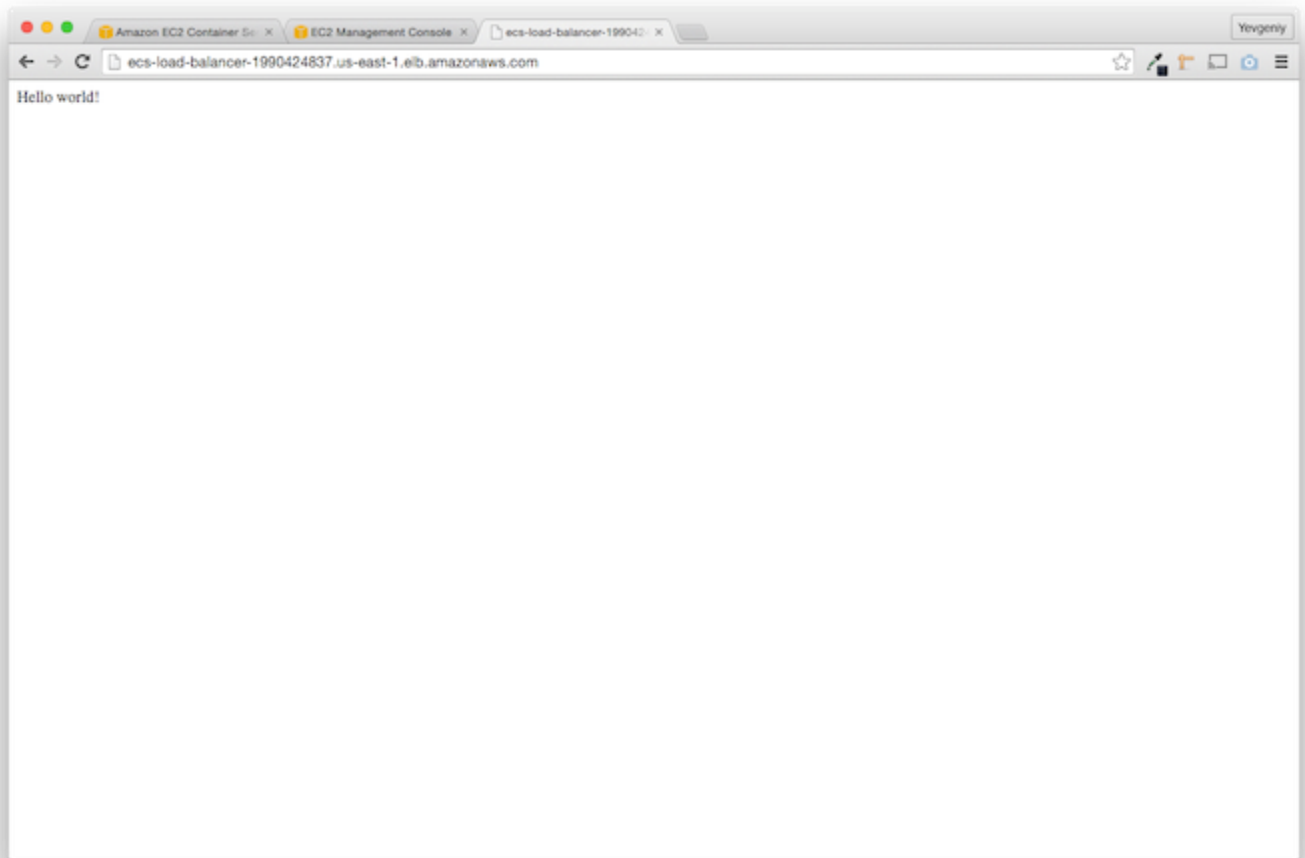
ECS Service deployment events

You may have to refresh a few times, but you should see your ECS Service starting 4 tasks, then registering 4 EC2 Instances in the ELB, and finally, reaching a “steady state”, which means the deployment has completed. That means you now have 4 Docker containers running on 4 EC2 Instances and an ELB distributing load between them. To test it out, click on your ELB (the Events tab should make the ELB’s name, `ecs-load-balancer` a clickable link) to go to the EC2 Console and copy its DNS Name:



ELB DNS

If you open this DNS in your browser, you should see the familiar `Hello world!` text:

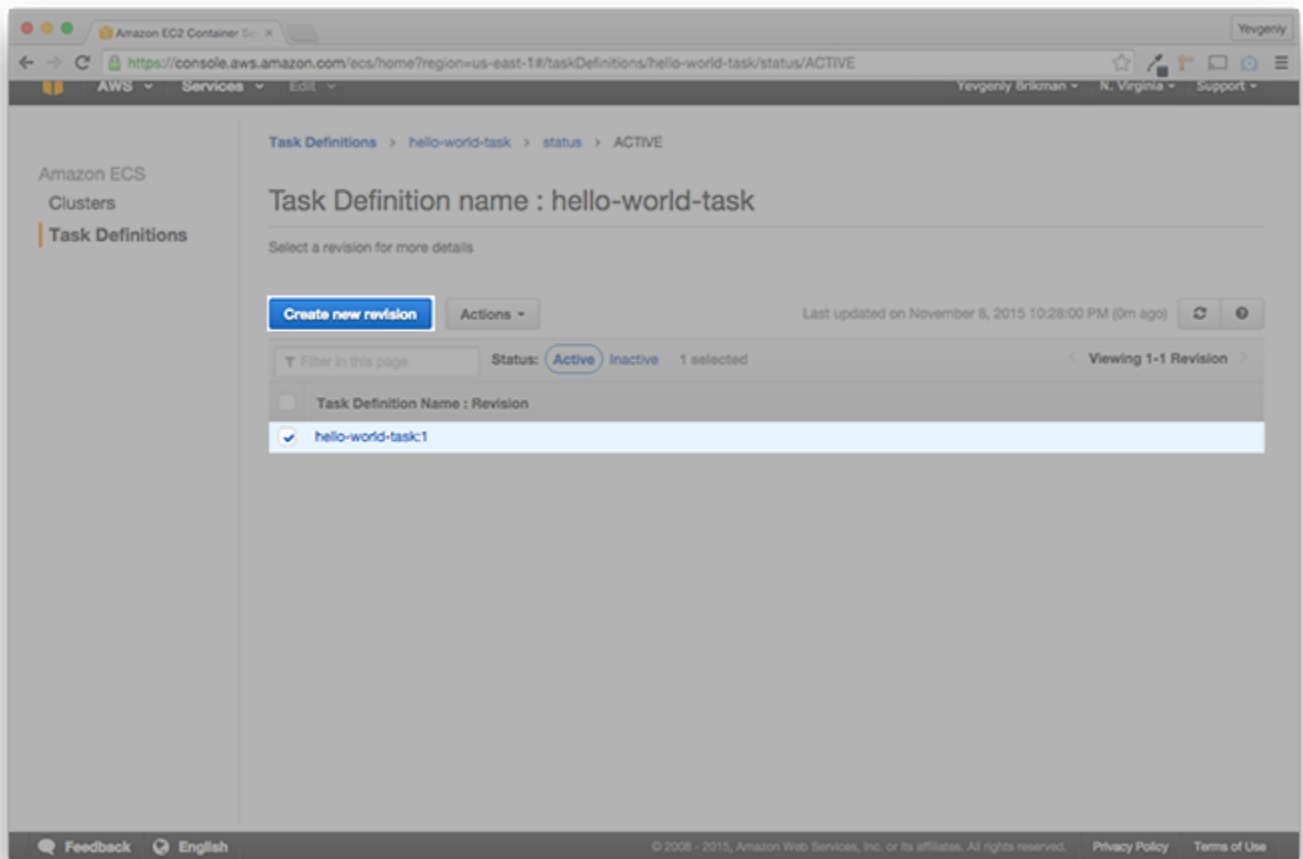


Accessing the app via the ELB

Update Docker containers in your ECS Cluster

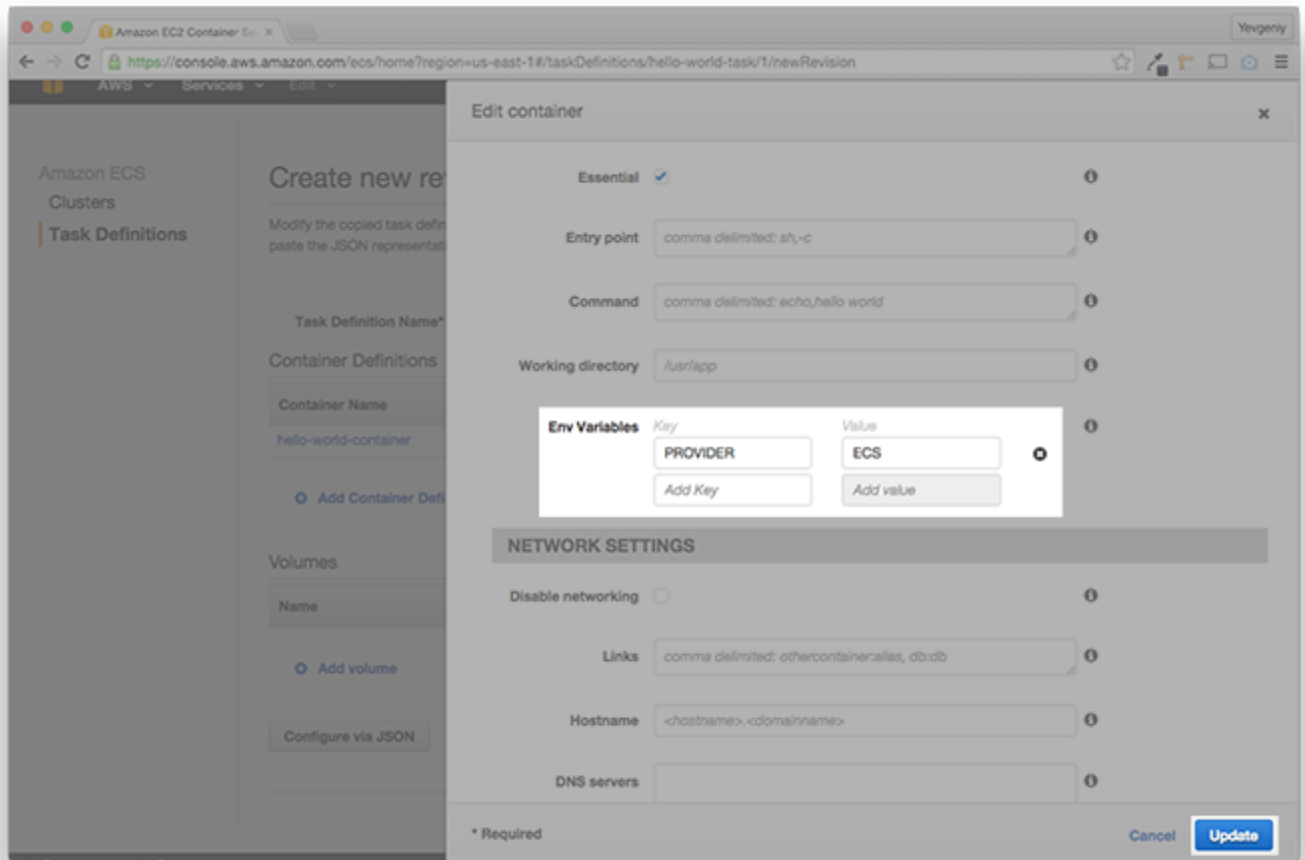
Now that you have your Docker containers running in the ECS Cluster, let's go through an example of how you could update them to a new version. Normally, you would make some changes to your app, check them in, kick off a build, and produce a new Docker image with a new tag. However, for this example, the `training/webapp` Docker image only has the `latest` tag, so we'll update something else. If you look at the [source of the training/webapp Docker container](#), you'll see that it uses the value of the environment variable `PROVIDER` as the second word after "Hello", and only falls back to "world" if `PROVIDER` is not set. Let's modify our ECS Task to set the `PROVIDER` value to `ECS` for our Docker container.

Note: ECS Tasks are immutable. You can't change the old definition—which is actually a good thing, as it allows you to easily roll back to an older version if you hit a big in a newer one—but you can create a new *Revision* of the Task. To do that, click the “Task Definitions” link in the menu on the left, click on your ECS Task (`hello-world-task`), click the checkbox next to the first Revision of your Task (`hello-world-task:1`), and click the blue “Create new revision” button:



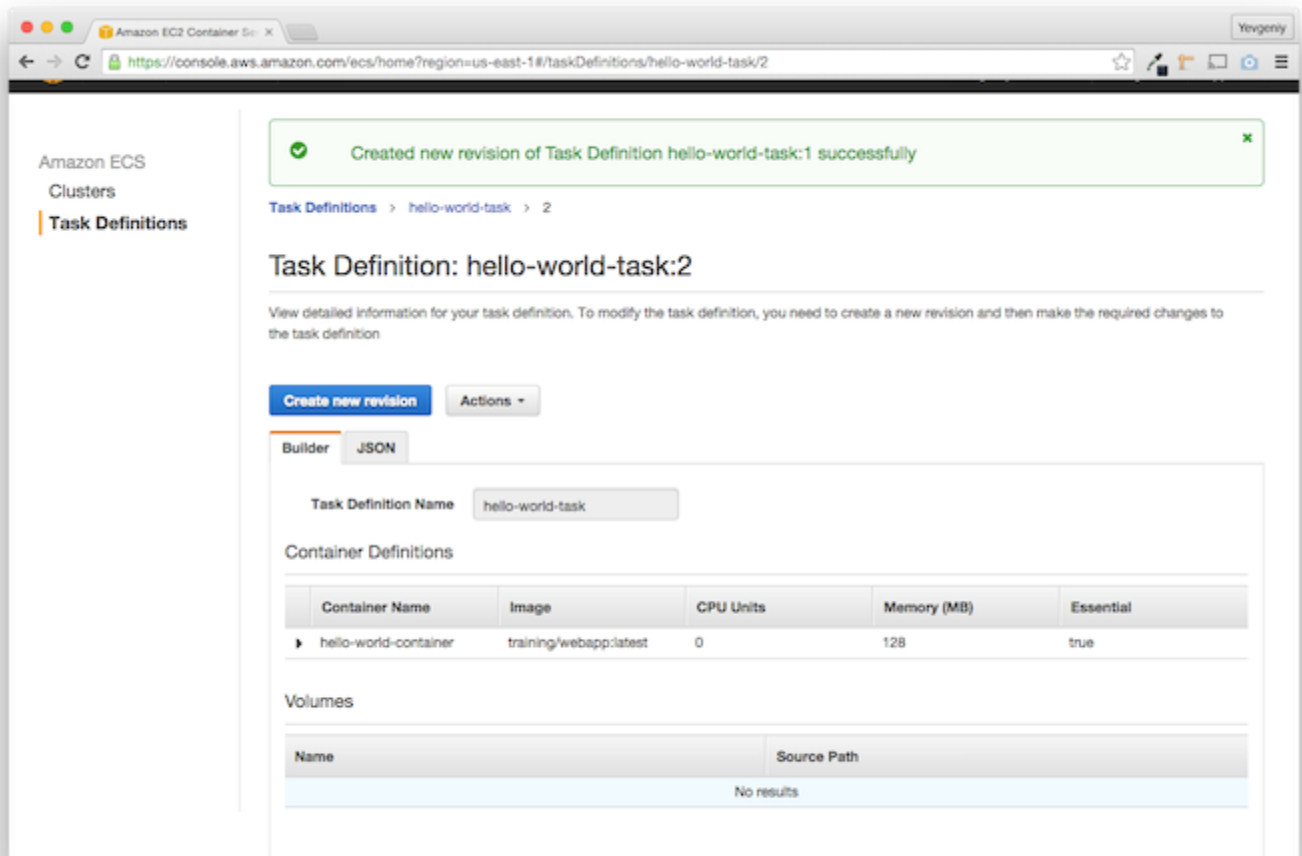
Create a new Revision of your ECS Task

Click on the Container Definition (`hello-world-container`), click the “Advanced container configuration” link, and scroll down to “Env Variables”. Here, enter `PROVIDER` as the Key and `ECS` as the Value:



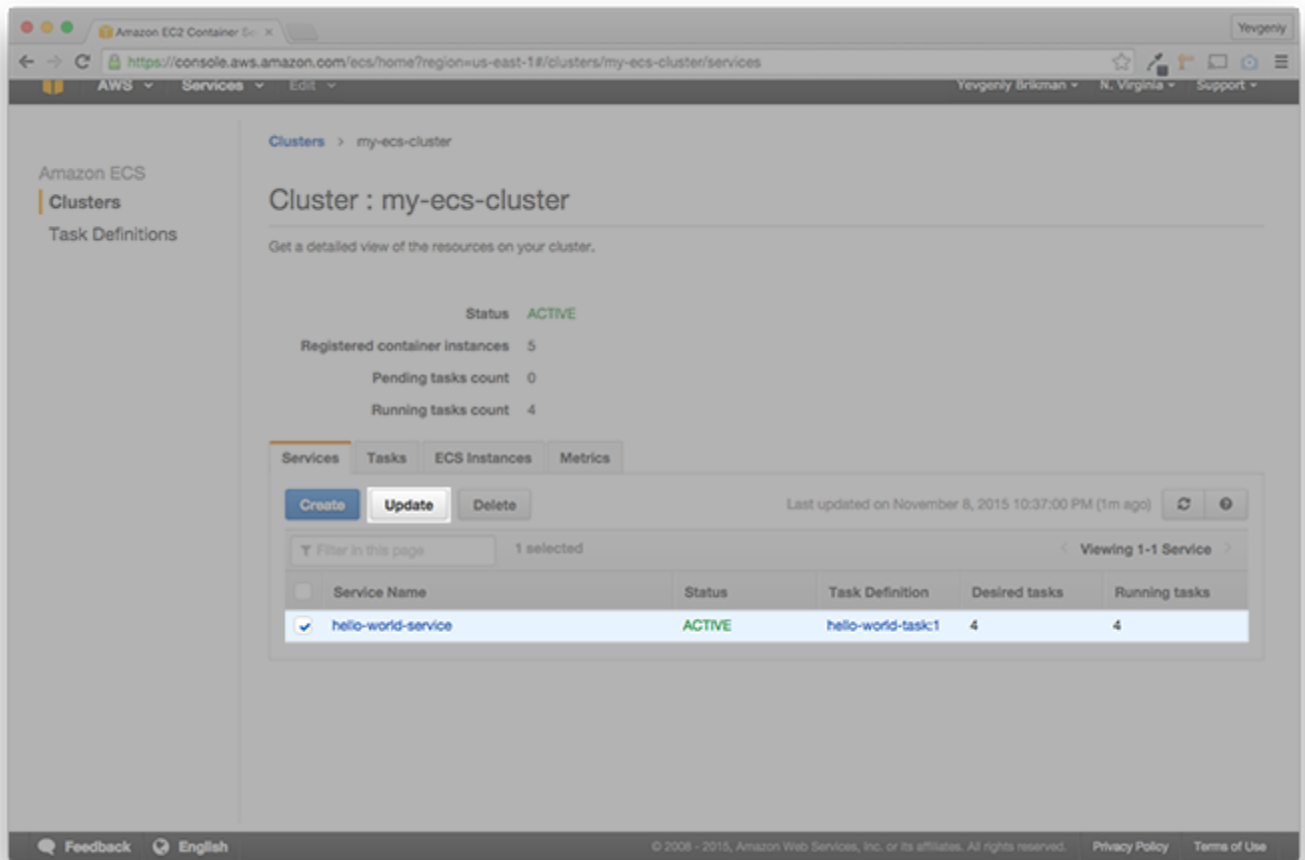
Add an environment variable to the container definition

This would also be the time to enter the new tag of your updated Docker image, but for this tutorial, we're going to stick with `latest`, so just click the blue "Update" button, then the blue "Create" button, and you should now have Revision 2 of your Task:



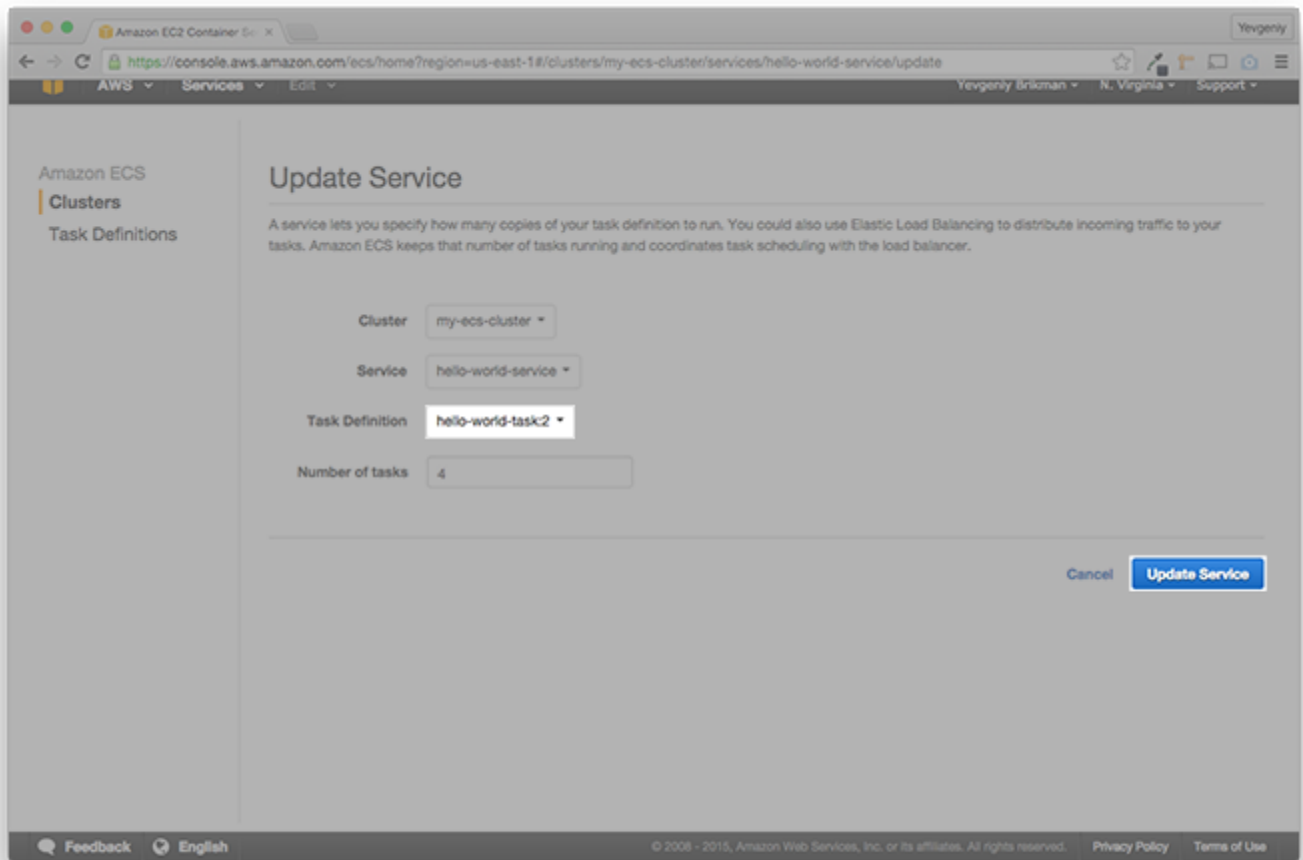
Revision 2 of the ECS Task

Now it's time to deploy the new Revision in the ECS Service. Click the "Clusters" link in the menu on the left, then click on your ECS Cluster (`my-ecs-cluster`), then click the checkbox next to your ECS Service (`hello-world-service`), and click the gray "Update" button:



Update ECS Service

Change the Task Definition from Revision 1 (`hello-world-task:1`) to Revision 2 (`hello-world-task:2`) and click the blue “Update Service” button:



Update the Task Revision in the ECS Service

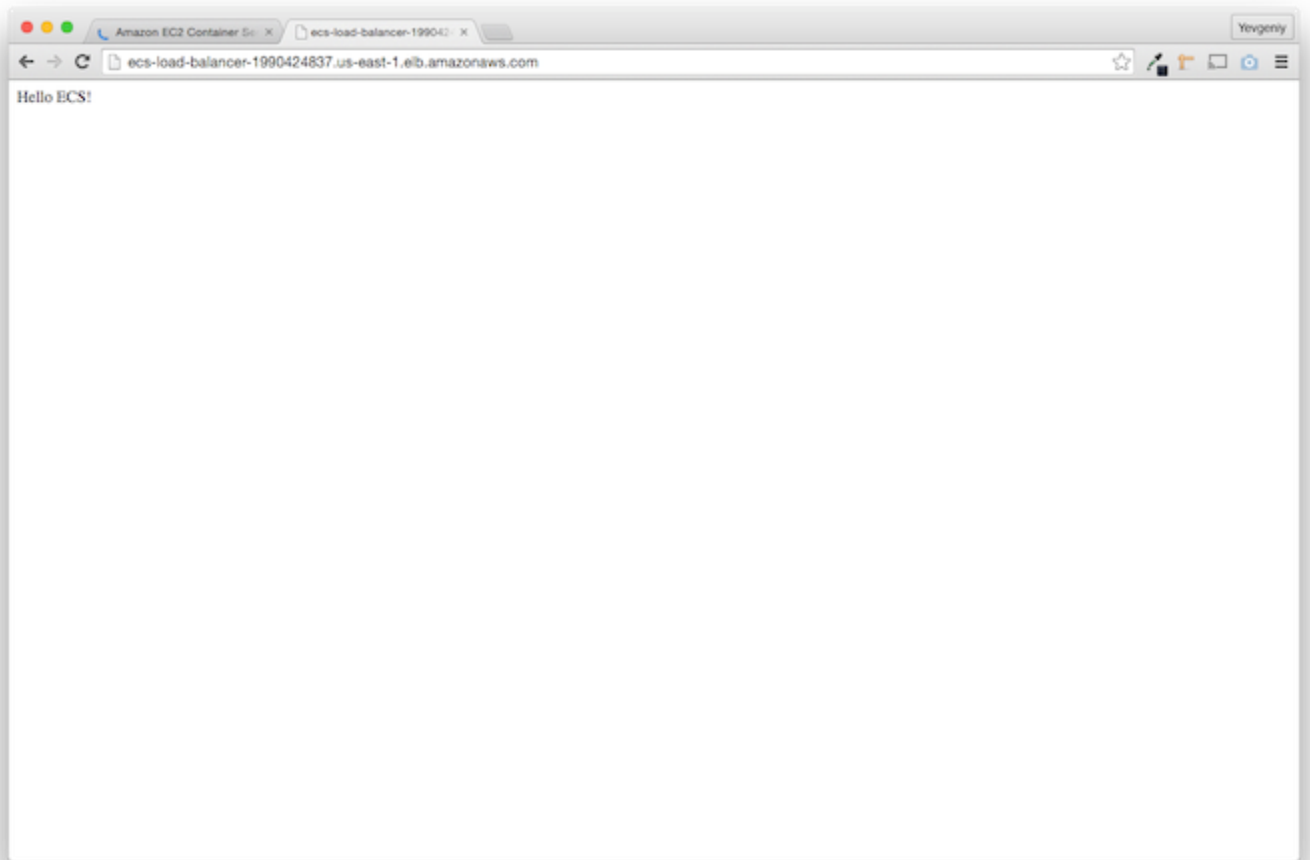
That's it! Now your new Task is deploying. If you click on the "Events" tab, you will see the ECS deployment process, which is roughly the following:

1. Look at the Container Definition in the Task to find out what CPU, memory, and ports it is requesting.
2. Find an EC2 Instance in the ECS Cluster that has the requested CPU, memory, and ports available. Finding the optimal server to run each Task is called *Task Scheduling*. In fact, since Docker containers provide isolation, if an EC2 Instance is not using all of its CPU or memory, a Scheduler could even decide to run multiple Docker containers on the same EC2 Instance, making more efficient usage of your resources.

3. If there is no available EC2 Instance that has the requested resources, show the error “service hello-world-service was unable to place a task because the resources could not be found” and try again in the future.
4. If there is an available EC2 Instance with the requested resources, run the new Task Revision on this Instance.
5. When the new Task is up and running, register it in the ELB.
6. Start [draining connections](#) for an old Revision of the Task.
7. Remove the old Revision of the Task from the ELB.
8. Stop the old Revision of the Task.

Steps #2 above is why you need to have more EC2 Instances in your Cluster (5) than desired Tasks in the ECS Service (4). That way, there is always at least one EC2 Instance available to deploy a new Revision of your Tasks. If you refresh the “Events” tab periodically, you’ll see it go through the same cycle 4 times: it will start a new Task on the spare EC2 Instance, register it in the ELB, deregister an old Task from the ELB, and then stop the old Task. This is essentially a “rolling deployment”, where you are updating one EC2 Instance at a time. If you have twice as many EC2 Instances as Tasks (e.g., 8 EC2 Instances and 4 desired Tasks), ECS will be able to do the update much faster, similar to a [blue-green deployment](#) (however, you’ll have to pay for twice as many EC2 Instances for this luxury).

After a minute or two, if you go to the URL of your ELB, you should see the new `Hello`
`ECS!` text:



The new Task Revision is working