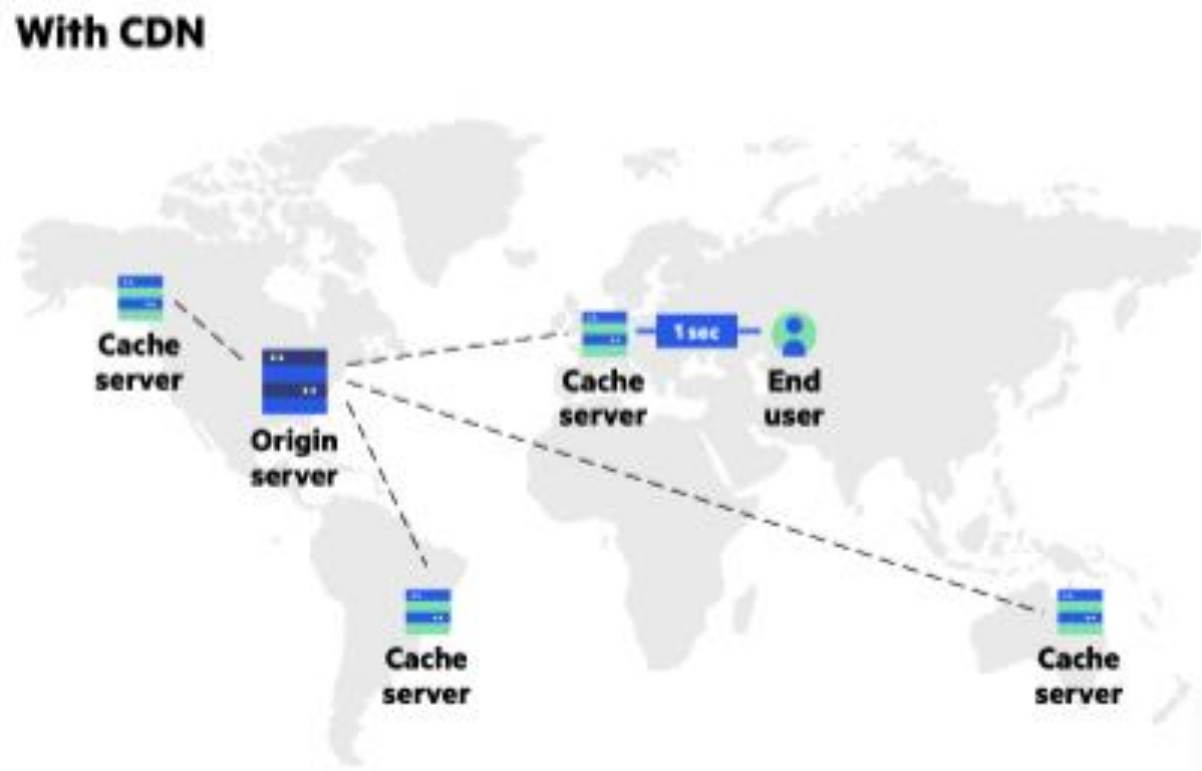
CONTENTS

3.1 Content Delivery Service

Content delivery networks (CDN) are the transparent backbone of the Internet in charge of content delivery. We use CDNs on a daily basis; when reading articles on news sites, shopping online, watching YouTube videos or perusing social media feeds. Think that you are accessing a website through its URL. The moment you hit enter, Does the page get loaded immediately or sometimes, you experience some delay in accessing the content? Whenever the content is delivered to the user, it is called as content delivery. Every text, image, video we see on the screen in only content delivery. For some URLs are experience delay because of physical distance between you and that website's hosting server



The important goal of CDNs is to virtually shorten that physical distance, thereby improve site rendering speed and performance.

- CDN stores the cached content in multiple geographical locations called as Point of Presence (PoP) or Edge Servers/locations
- It is a caching server. Each caching server is responsible for delivering content in its proximity
- CDN puts the content in many places at once, providing superior coverage to your users
- Can have multiple number of edge locations in multiple locations
- Requests are routed to the nearest edge locations

Who uses CDN?

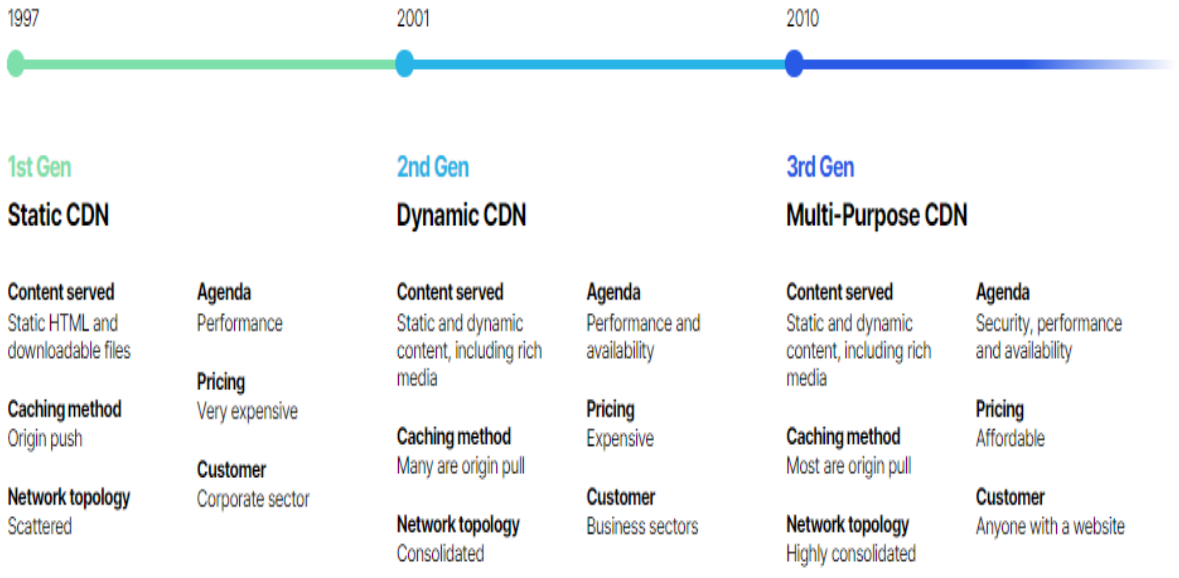
- Advertising
- Media and entertainment
- Online gaming
- E-commerce
- Mobile

- Healthcare
- Higher education
- Government

What CDN does for you?

- Improve page load speed
- Handle high traffic loads
- Block spammers, scrapers and other bad bots
- Localize coverage without the cost
- Reduce bandwidth consumption
- Load balance between multiple servers
- Protect your website from DDoS attacks
- Secure your application

Evolution



3.2 CloudFront

CloudFront is a CDN (Content Delivery Network). It retrieves data from Amazon S3 bucket and distributes it to multiple datacenter locations. It delivers the data through a network of

data centers called **edge locations**. The nearest edge location is routed when the user requests for data, resulting in lowest latency, low network traffic, fast access to data, etc.

Features of CloudFront

Fast − The broad network of edge locations and CloudFront caches copies of content close to the end users that results in lowering latency, high data transfer rates and low network traffic. All these make CloudFront fast.

Simple − It is easy to use.

Can be used with other AWS Services − Amazon CloudFront is designed in such a way that it can be easily integrated with other AWS services, like Amazon S3, Amazon EC2.

Cost-effective − Using Amazon CloudFront, we pay only for the content that you deliver through the network, without any hidden charges and no up-front fees.

Elastic − Using Amazon CloudFront, we need not worry about maintenance. The service automatically responds if any action is needed, in case the demand increases or decreases.

Reliable − Amazon CloudFront is built on Amazon's highly reliable infrastructure, i.e. its edge locations will automatically re-route the end users to the next nearest location, if required in some situations.

Global − Amazon CloudFront uses a global network of edge locations located in most of the regions.

AWS CloudFront delivers the content in the following steps.

Step 1 − The user accesses a website and requests an object to download like an image file.

Step 2 − DNS routes your request to the nearest CloudFront edge location to serve the user request.

Step 3 − At edge location, CloudFront checks its cache for the requested files. If found, then returns it to the user otherwise does the following −

- First CloudFront compares the request with the specifications and forwards it to the applicable origin server for the corresponding file type.

- The origin servers send the files back to the CloudFront edge location.
- As soon as the first byte arrives from the origin, CloudFront starts forwarding it to the user and adds the files to the cache in the edge location for the next time when someone again requests for the same file.

Step 4 – The object is now in an edge cache for 24 hours or for the provided duration in file headers. CloudFront does the following –

- CloudFront forwards the next request for the object to the user's origin to check the edge location version is updated or not.
- If the edge location version is updated, then CloudFront delivers it to the user.
- If the edge location version is not updated, then origin sends the latest version to CloudFront. CloudFront delivers the object to the user and stores the latest version in the cache at that edge location.

Static Content Delivery with S3

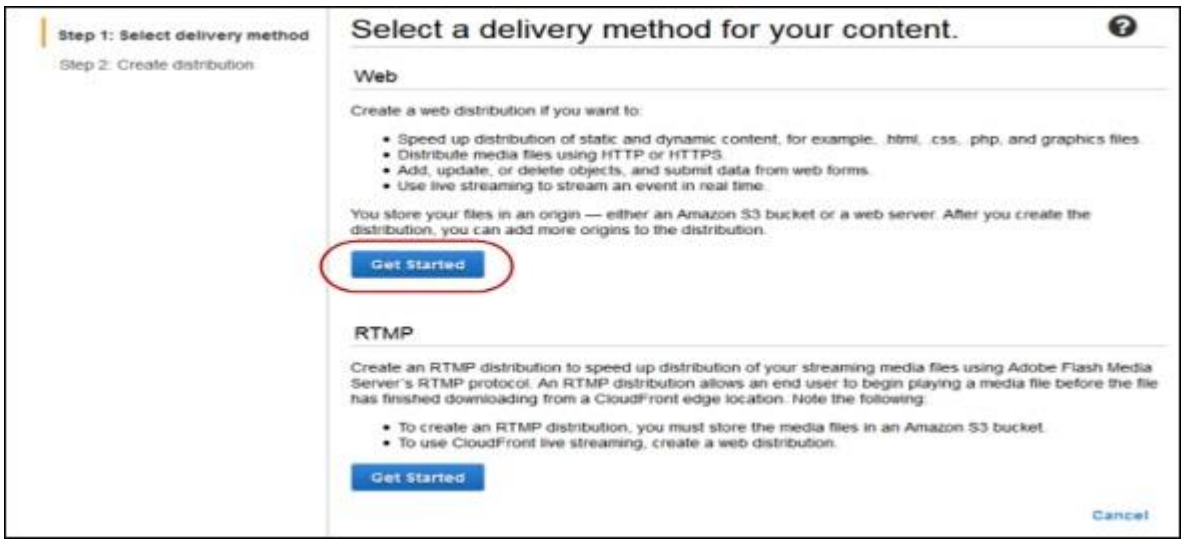
AWS CloudFront can be set up using the following steps.

Step 1 – Sign in to AWS management console using the following link – https://console.aws.amazon.com/

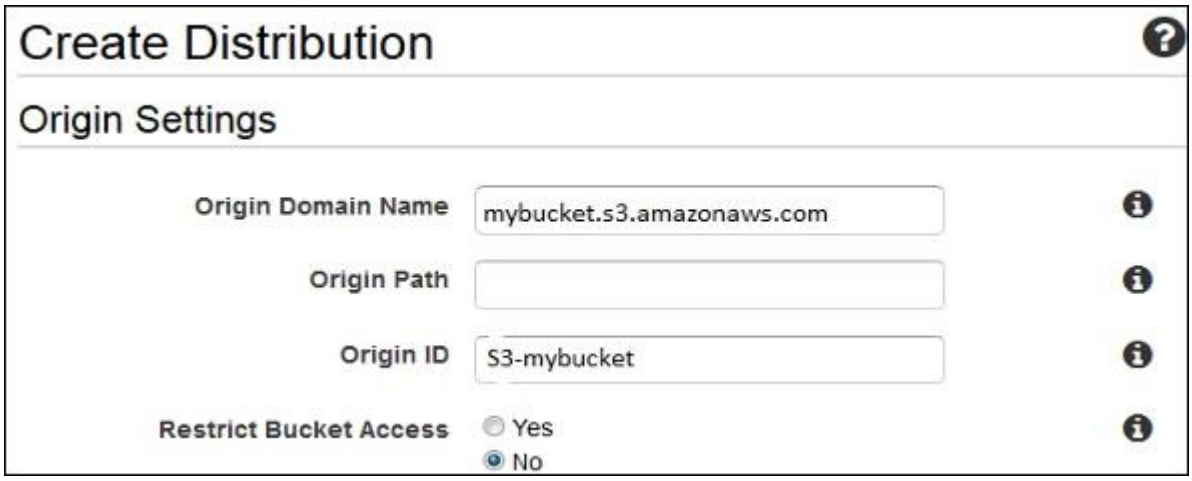
Step 2 – Upload Amazon S3 and choose every permission public.

Step 3 – Create a CloudFront Web Distribution using the following steps.

- Choose the service CloudFront from the services menu of aws console.
- Click the Get Started button in the web section of Select a delivery method for your content page.



- **Create Distribution** page opens. Choose the Amazon S3 bucket created in the Origin Domain Name and leave the remaining fields as default.



- Default Cache Behavior Settings page opens. Keep the values as default and move to the next page.
- A Distribution settings page opens. Fill the details as per your requirement and click the Create Distribution button.
- The Status column changes from In Progress to Deployed. Enable your distribution by selecting the Enable option. It will take around 15 minutes for the domain name to be available in the Distributions list.

Test the Links

After creating distribution, CloudFront knows the location of Amazon S3 server and the user knows the domain name associated with the distribution. However, we can also

create a link to Amazon S3 bucket content with that domain name and have CloudFront serve it. This helps save a lot of time.

Following are the steps to link an object –

Step 1 – Copy the following HTML code to a new file and write the domain-name that CloudFront assigned to the distribution in the place of domain name. Write a file name of Amazon S3 bucket in the place of object-name.

```
<html>
  <head>CloudFront Testing link</head>
  <body>
    <p>My Cludfront.</p>
    <p><img src = "http://domain-name/object-name" alt = "test image"/>
  </body>
</html>
```

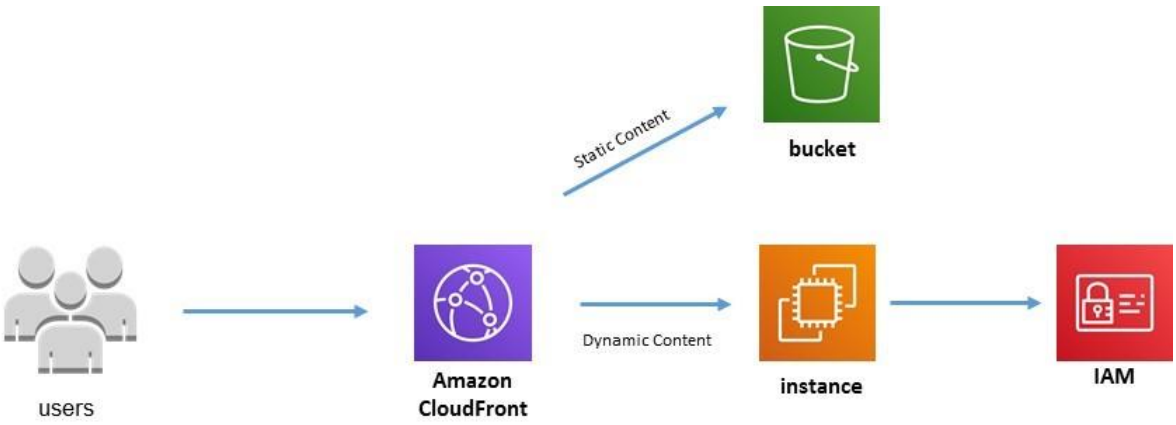
Step 2 – Save the text in a file with **.html** extension.

Step 3 – Open the web page in a browser to test the links to see if they are working correctly. If not, then crosscheck the settings.

Dynamic Content Delivery with EC2

Many websites and web applications serve a combination of static content—HTML, CSS, JPG, or other files that all end viewers can see—and dynamic content, which is personalized for each end viewer. Amazon CloudFront can serve both types of content, to reduce latency, protect your architecture, and optimize costs. Here we will see how to use CloudFront to deliver both static and dynamic content using a single distribution, for dynamic and static websites and web applications. We will learn how to implement this with an CloudFront distribution connected to a custom origin for dynamic content (in this case, an Amazon EC2 web server instance) and an Amazon S3 bucket for static content.

To set up an origin that delivers dynamic content, you first create an Amazon S3 bucket, and then assign a policy that allows CloudFront to access the data from Amazon S3. Next, you create a webserver on an Amazon EC2 instance that includes your dynamic content, and create an Amazon IAM Role that the Amazon EC2 instance will assume. Finally, you create a CloudFront web distribution with two origins, to securely deliver the dynamic and static content to the users from the two origins. Using this template as a starting point, you can also easily build your own customized workflow.



Following are the steps to be followed for the dynamic content delivery through ec2.

1. Go to the AWS Console
2. Create Amazon EC2 instances
3. Create an Application Load Balancer
4. Create target groups with EC2 instances
5. Create a CloudFront distribution
6. configure your origin
7. Configure default cache behavior
8. Configure set cache based on selected request headers to "all“
9. Save distribution
10. Test your CloudFront distribution

3.3 Deployment and Management Service

- Cloud-based deployment & management services allow you to easily deploy and manage applications in the cloud.
- These services automatically handle deployment tasks such as capacity provisioning, load balancing, auto-scaling, and application health monitoring

- It should be possible that the service should provide different deploying environments for various range of applications
- The underlying infrastructure that is necessary for deployment should be automatically provisioned
- The applications that should react for http requests should also be taken care
- The versioning of the applications should be possible with ease

Steps:

- A load balancer, Web server, and database server appliances should be selected from a library of preconfigured virtual machine images.
- Configuring each component to make a custom image should be made. Load balancer is configured accordingly; web server should be populated with the static contents by uploading them to the storage cloud where as the database servers are populated with the dynamic content of the site.
- The developer then feeds the custom code in to the new architecture making components meet their specific requirements.
- The developer chooses a pattern that takes the images for each layer and deploys them, handling networking, security, and scalability issues.

The secure, high-availability Web application is up and running. When the application needs to be updated, the virtual machine images can be updated, copied across the development chain, and the entire infrastructure can be redeployed. In this example, a standard set of components can be used to quickly deploy an application. With this model, enterprise business needs can be met quickly, without the need for the time-consuming, manual purchase, installation, cabling, and configuration of servers, storage, and network infrastructure.

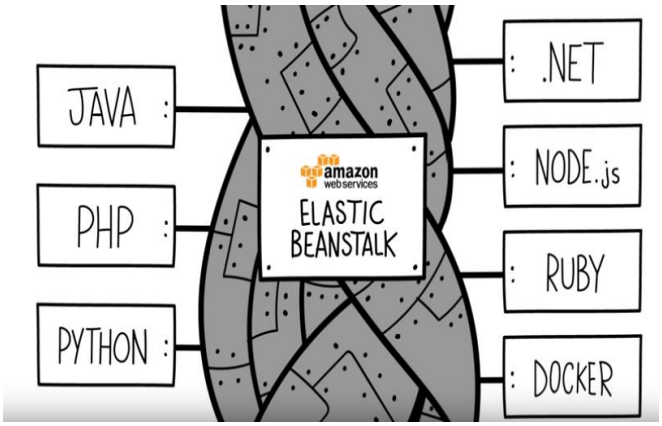
Amazon Elastic Beanstalk

Whenever we want to host an application, we have configure system, manage servers, databases, scaling, load balancing etc. Most of the time the programmers have to concentrate on these things.

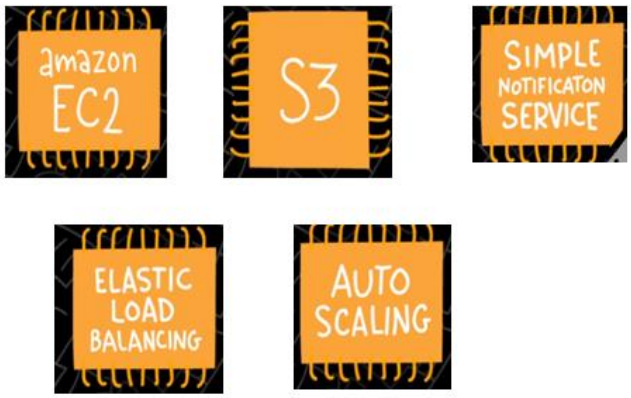
Beanstalk helps in reducing all these activities and helps to host/deploy your application very fast. Beanstalk is a service that deploys, manages and scales the web application for the users. AWS provides PaaS solution called Elastic Beanstalk. It is mature PaaS than other PaaS service provider. Developers need not worry about infrastructure to launch an web app. Developers just upload the application as binary file such as projectname.war. Developers can use application versioning to switch between previous version of application by using

swap URL environment option. It is possible to use an existing instance instead of provisioning a new environment. You can create AMI of your existing instance. You can also use new AMI to create an instance.

Elastic Beanstalk supports Java, PHP, .NET, Node.js, Python, and Ruby applications.



With Elastic Beanstalk you just need to upload the application and specify configuration settings in a simple wizard and the service automatically handles instance provisioning, server configuration, load balancing and monitoring. It manages different servers such as Apache, Tomcat, NginX, IIS. It can communicate with other AWS services.



Amazon CloudFormation

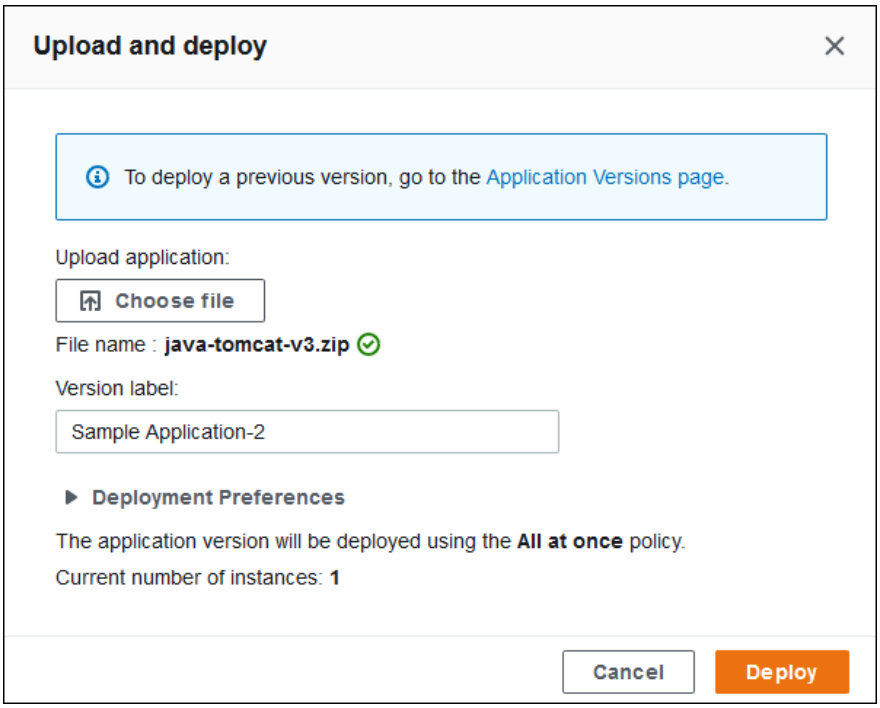
Amazon CloudFormation is a deployment management service from Amazon. With CloudFormation you can create deployments from a collection of AWS resources such as Amazon Elastic Compute Cloud, Amazon Elastic Block Store, Amazon Simple Notification Service, Elastic Load Balancing and Auto Scaling. A collection of AWS resources that you want to manage together are organized into a stack.

3.4 Bean Stalk

Deploying a Sample Application

Deploy

- Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
- In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
- On the environment overview page, choose **Upload and deploy**.
- Choose **Choose file**, and then upload the sample application source bundle



- The console automatically fills in the **Version label** with a new unique label. If you type in your own version label, ensure that it's unique.
- Choose **Deploy**

Configure

- In the navigation pane, choose **Configuration**
- In the Capacity configuration category, choose **Edit**

- In the Auto Scaling group section, change Environment type to Load balanced
- In the Instances row, change Max to 4, and then change Min to 2
- Choose **Apply**.
- A warning tells you that this update replaces all of your current instances. Choose **Confirm**.

Clean up

Delete all application versions.

- In the navigation pane, choose **Applications**, and then choose **getting-started-app**
- In the navigation pane, find your application's name and choose **Application versions**
- On the **Application versions** page, select all application versions that you want to delete
- Choose **Actions**, and then choose **Delete**
- Turn on **Delete versions from Amazon S3**
- Choose **Delete**, and then choose **Done**

Terminate the environment

- In the navigation pane, choose **getting-started-app**, and then choose **GettingStartedApp-env** in the environment list
- Choose **Environment actions**, and then choose **Terminate Environment**
- Confirm that you want to terminate **GettingStartedApp-env** by typing the environment name, and then choose **Terminate**

Delete the getting-started-app application

- In the navigation pane, choose the **getting-started-app**
- Choose **Actions**, and then choose **Delete application**
- Confirm that you want to delete **getting-started-app** by typing the application name, and then choose **Delete**.

Deploying Database Application

1. Launch a DB instance in Amazon RDS
2. Create an Elastic Beanstalk environment
3. Configure security groups, environment properties, and scaling

4. Deploy the sample application
5. Cleanup

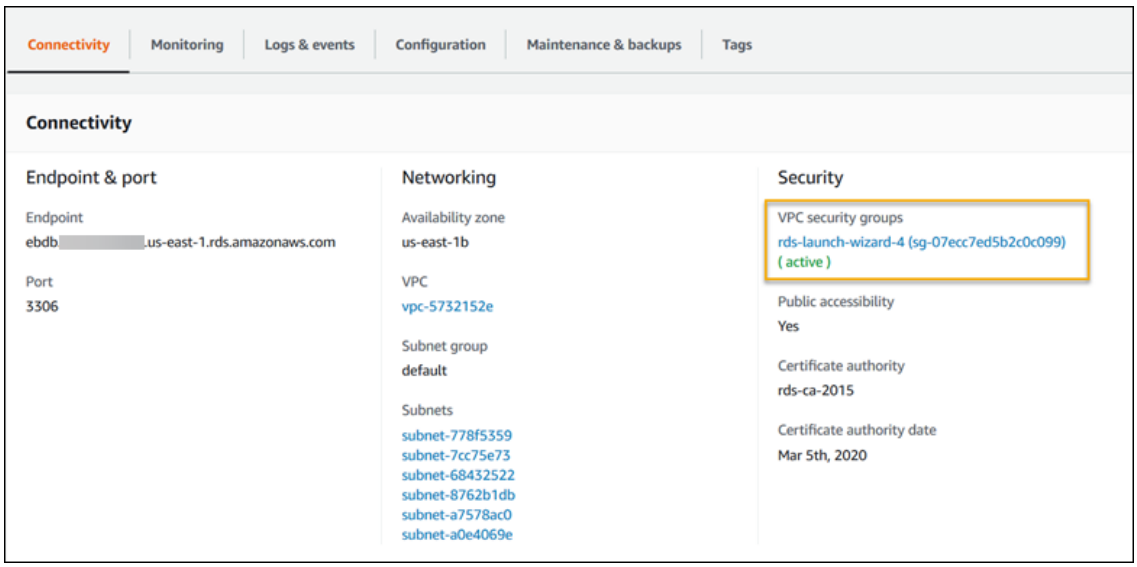
To launch an RDS DB instance in a default VPC

1. Open the RDS console.
2. Choose **Databases** in the navigation pane.
3. Choose **Create database**.
4. Choose **Standard Create**.
5. Under **Additional configuration**, for **Initial database name**, type **ebdb**.
6. Review the default settings carefully and adjust as necessary. Pay attention to the following options:
    - **DB instance class** – Choose an instance size that has an appropriate amount of memory and CPU power for your workload.
    - **Multi-AZ deployment** – For high availability, set this to **Create an Aurora Replica/Reader node in a different AZ**.
    - **Master username** and **Master password** – The database username and password. Make a note of these settings because you'll use them later.
7. Verify the default settings for the remaining options, and then choose **Create database**.

Next, modify the security group attached to your DB instance to allow inbound traffic on the appropriate port. This is the same security group that you will attach to your Elastic Beanstalk environment later, so the rule that you add will grant ingress permission to other resources in the same security group.

To modify the inbound rules on your RDS instance's security group

1. Open the Amazon RDS console.
2. Choose **Databases**.
3. Choose the name of your DB instance to view its details.
4. In the **Connectivity** section, make a note of the **Subnets**, **Security groups**, and **Endpoint** shown on this page so you can use this information later.
5. Under **Security**, you can see the security group associated with the DB instance. Open the link to view the security group in the Amazon EC2 console.



6. In the security group details, choose **Inbound**.
7. Choose **Edit**.
8. Choose **Add Rule**.
9. For **Type**, choose the DB engine that your application uses.
10. For **Source**, type **sg-** to view a list of available security groups. Choose the security group associated with an Elastic Beanstalk environment's Auto Scaling group to allow Amazon EC2 instances in the environment to have access to the database.

11. Choose **Save**.

Create an Elastic Beanstalk environment

Use the Elastic Beanstalk console to create an Elastic Beanstalk environment. Choose the **PHP** platform and accept the default settings and sample code. After you launch the environment, you can configure the environment to connect to the database, then deploy the sample application that you downloaded from GitHub.

To launch an environment (console)

1. Open the Elastic Beanstalk console using this preconfigured link: console.aws.amazon.com/elasticbeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced
2. For **Platform**, select the platform and platform branch that match the language used by your application.
3. For **Application code**, choose **Sample application**.
4. Choose **Review and launch**.
5. Review the available options. Choose the available option you want to use, and when you're ready, choose **Create app**.

Environment creation takes about 5 minutes and creates the following resources:

1. **EC2 instance** – An Amazon Elastic Compute Cloud (Amazon EC2) virtual machine configured to run web apps on the platform that you choose.
2. Each platform runs a specific set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination of these. Most platforms use either Apache or NGINX as a reverse proxy that sits in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.
3. **Instance security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic isn't allowed on other ports.
4. **Load balancer** – An Elastic Load Balancing load balancer configured to distribute requests to the instances running your application. A load balancer also eliminates the need to expose your instances directly to the internet.
5. **Load balancer security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the internet reach the load balancer. By default, traffic isn't allowed on other ports.
6. **Auto Scaling group** – An Auto Scaling group configured to replace an instance if it is terminated or becomes unavailable.
7. **Amazon S3 bucket** – A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
8. **Amazon CloudWatch alarms** – Two CloudWatch alarms that monitor the load on the instances in your environment and that are triggered if the load is too high or too

low. When an alarm is triggered, your Auto Scaling group scales up or down in response.

9. **AWS CloudFormation stack** – Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the AWS CloudFormation console.
10. **Domain name** – A domain name that routes to your web app in the form *subdomain.region.elasticbeanstalk.com*.

All of these resources are managed by Elastic Beanstalk. When you terminate your environment, Elastic Beanstalk terminates all the resources that it contains. The RDS DB instance that you launched is outside of your environment, so you are responsible for managing its lifecycle.

Configure security groups, environment properties, and scaling

Add the security group of your DB instance to your running environment. This procedure causes Elastic Beanstalk to reprovision all instances in your environment with the additional security group attached.

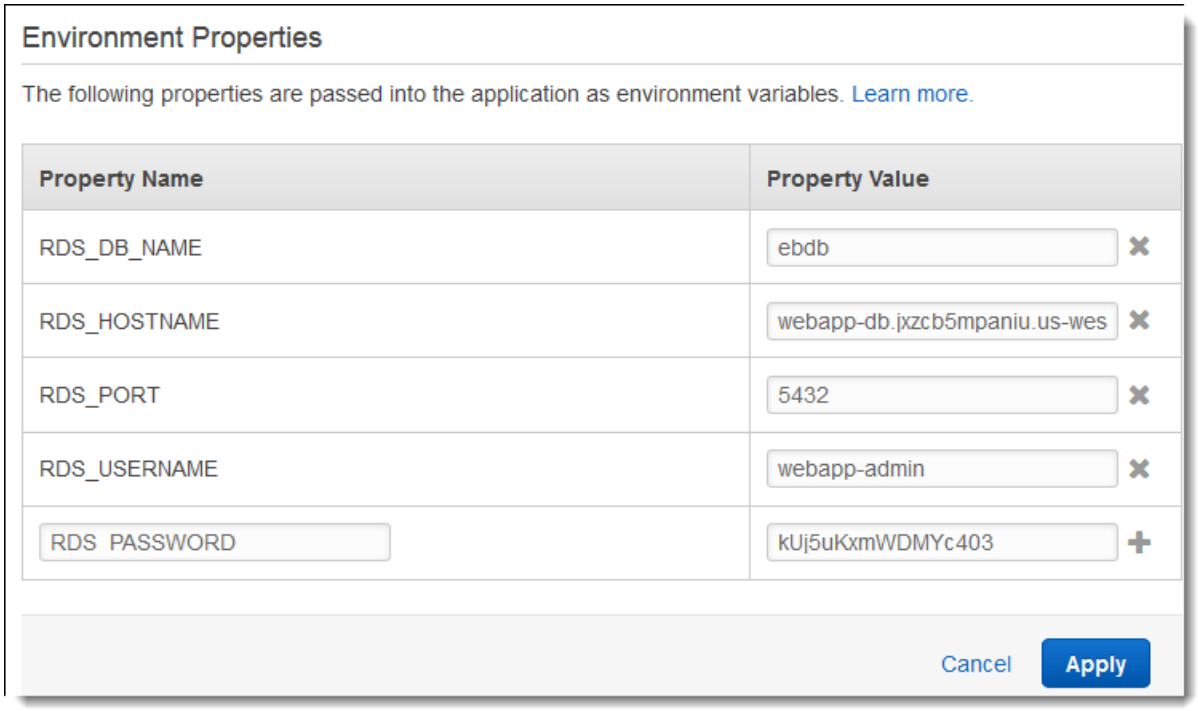
To add a security group to your environment

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Instances** configuration category, choose **Edit**.
5. Under **EC2 security groups**, choose the security group to attach to the instances, in addition to the instance security group that Elastic Beanstalk creates.
6. Choose **Apply**.
7. Read the warning, and then choose **Confirm**.

Next, use environment properties to pass the connection information to your environment. The sample application uses a default set of properties that match the ones that Elastic Beanstalk configures when you provision a database within your environment.

To configure environment properties for an Amazon RDS DB instance

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Software** configuration category, choose **Edit**.
5. In the **Environment properties** section, define the variables that your application reads to construct a connection string. For compatibility with environments that have an integrated RDS DB instance, use the following names and values. You can find all values, except for your password, in the RDS console.



6. Choose **Apply**.

Finally, configure your environment's Auto Scaling group with a higher minimum instance count. Run at least two instances at all times to prevent the web servers in your environment from being a single point of failure, and to allow you to deploy changes without taking your site out of service.

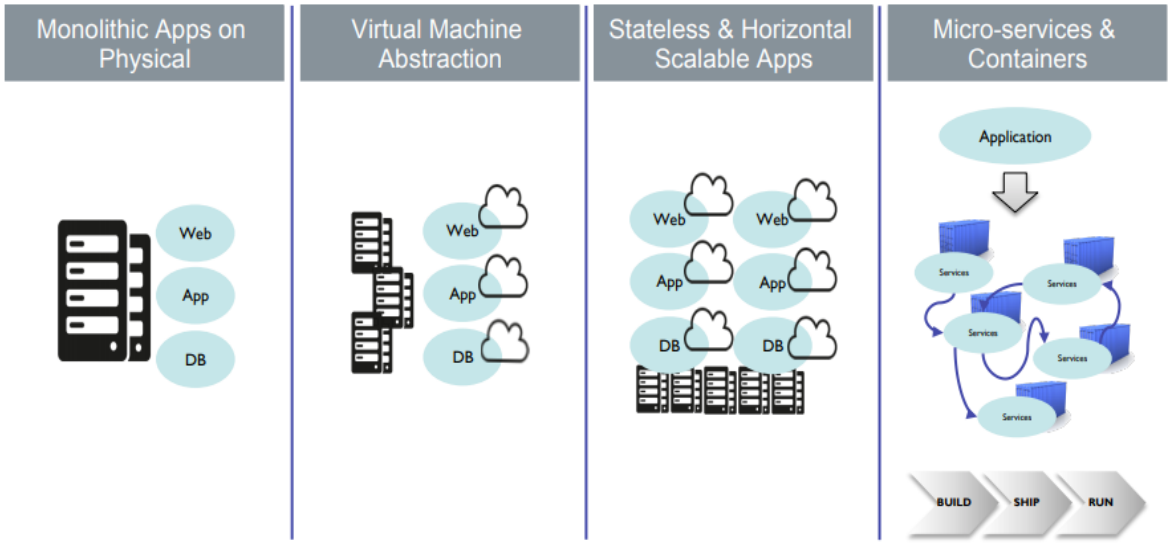
To configure your environment's Auto Scaling group for high availability

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

3. In the navigation pane, choose **Configuration**.
4. In the **Capacity** configuration category, choose **Edit**.
5. In the **Auto Scaling group** section, set **Min instances** to **2**.
6. Choose **Apply**.

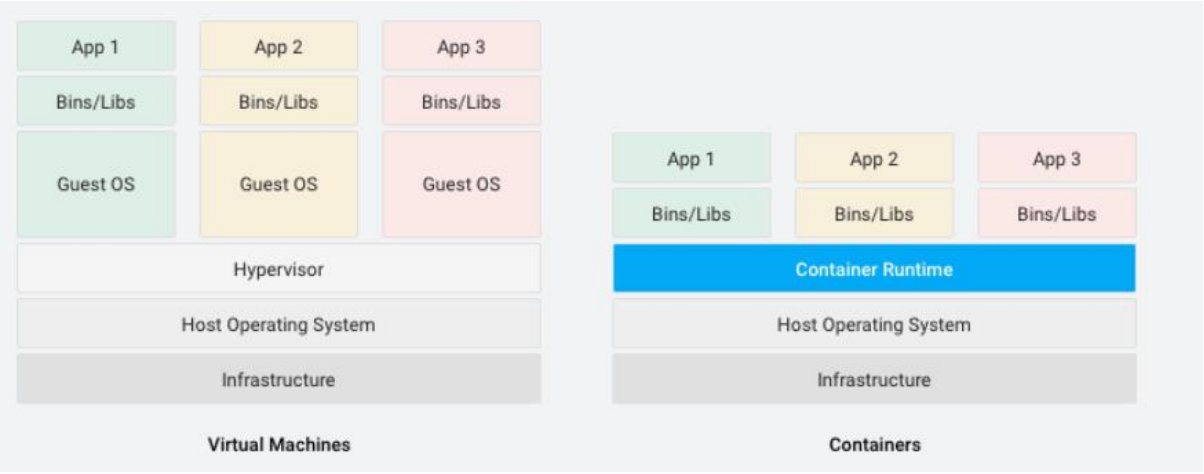
3.5 Container/Runtime Service

Containers offer a logical packaging mechanism in which applications can be abstracted from the environment in which they actually run. This decoupling allows container-based applications to be deployed easily and consistently, regardless of whether the target environment is a private data center, the public cloud, or even a developer's personal laptop. Containerization provides a clean separation of concerns, as developers focus on their application logic and dependencies, while IT operations teams can focus on deployment and management without bothering with application details such as specific software versions and configurations specific to the app.



For those coming from virtualized environments, containers are often compared with virtual machines (VMs). You might already be familiar with VMs: a guest operating system such as Linux or Windows runs on top of a host operating system with virtualized access to the underlying hardware. Like virtual machines, containers allow you to package your application together with libraries and other dependencies, providing isolated environments for running your software services. As you'll see below however, the similarities end here as

containers offer a far more lightweight unit for developers and IT Ops teams to work with, carrying a myriad of benefits.



Why Containers?

Instead of virtualizing the hardware stack as with the virtual machines approach, containers virtualize at the operating system level, with multiple containers running atop the OS kernel directly. This means that containers are far more lightweight: they share the OS kernel, start much faster, and use a fraction of the memory compared to booting an entire OS.

There are many container formats available.

Consistent Environment

Containers give developers the ability to create predictable environments that are isolated from other applications. Containers can also include software dependencies needed by the application, such as specific versions of programming language runtimes and other software libraries. From the developer's perspective, all this is guaranteed to be consistent no matter where the application is ultimately deployed. All this translates to productivity: developers and IT Ops teams spend less time debugging and diagnosing differences in environments, and more time shipping new functionality for users. And it means fewer bugs since developers can now make assumptions in dev and test environments they can be sure will hold true in production.

Run Anywhere

Containers are able to run virtually anywhere, greatly easing development and deployment: on Linux, Windows, and Mac operating systems; on virtual machines or bare metal; on a developer's machine or in data centers on-premises; and of course, in the public cloud. The widespread popularity of the Docker image format for containers further helps with portability. Wherever you want to run your software, you can use containers.

Isolation

Containers virtualize CPU, memory, storage, and network resources at the OS-level, providing developers with a sandboxed view of the OS logically isolated from other applications.

	Container Benefits	Virtual Machine Benefits
Consistent Runtime Environment	✓	✓
Application Sandboxing	✓	✓
Small Size on Disk	✓	
Low Overhead	✓	

From Code to Applications

Containers allow you to package your application and its dependencies together into one succinct manifest that can be version controlled, allowing for easy replication of your application across developers on your team and machines in your cluster.

Just as how software libraries package bits of code together, allowing developers to abstract away logic like user authentication and session management, containers allow your application as a whole to be packaged, abstracting away the operating system, the machine, and even the code itself. Combined with a service-based architecture, the entire unit that

developers are asked to reason about becomes much smaller, leading to greater agility and productivity. All this eases development, testing, deployment, and overall management of your applications.

Benefits

A container may be only tens of megabytes in size, whereas a virtual machine with its own entire operating system may be several gigabytes in size. Because of this, a single server can host far more containers than virtual machines.

Another major benefit is that virtual machines may take several minutes to boot up their operating systems and begin running the applications they host, while containerized applications can be started almost instantly. That means containers can be instantiated in a "just in time" fashion when they are needed and can disappear when they are no longer required, freeing up resources on their hosts.

A third benefit is that containerization allows for greater modularity. Rather than run an entire complex application inside a single container, the application can be split in to modules (such as the database, the application front end, and so on). This is the so-called microservices approach. Applications built in this way are easier to manage because each module is relatively simple, and changes can be made to modules without having to rebuild the entire application. Because containers are so lightweight, individual modules (or microservices) can be instantiated only when they are needed and are available almost immediately.

3.6 Docker

Amazon Elastic Container Service (Amazon ECS) is the Amazon Web Service you use to run Docker applications on a scalable cluster. In this tutorial, you will learn how to run a Docker-enabled sample application on an Amazon ECS cluster behind a load balancer, test the sample application, and delete your resources to avoid charges.

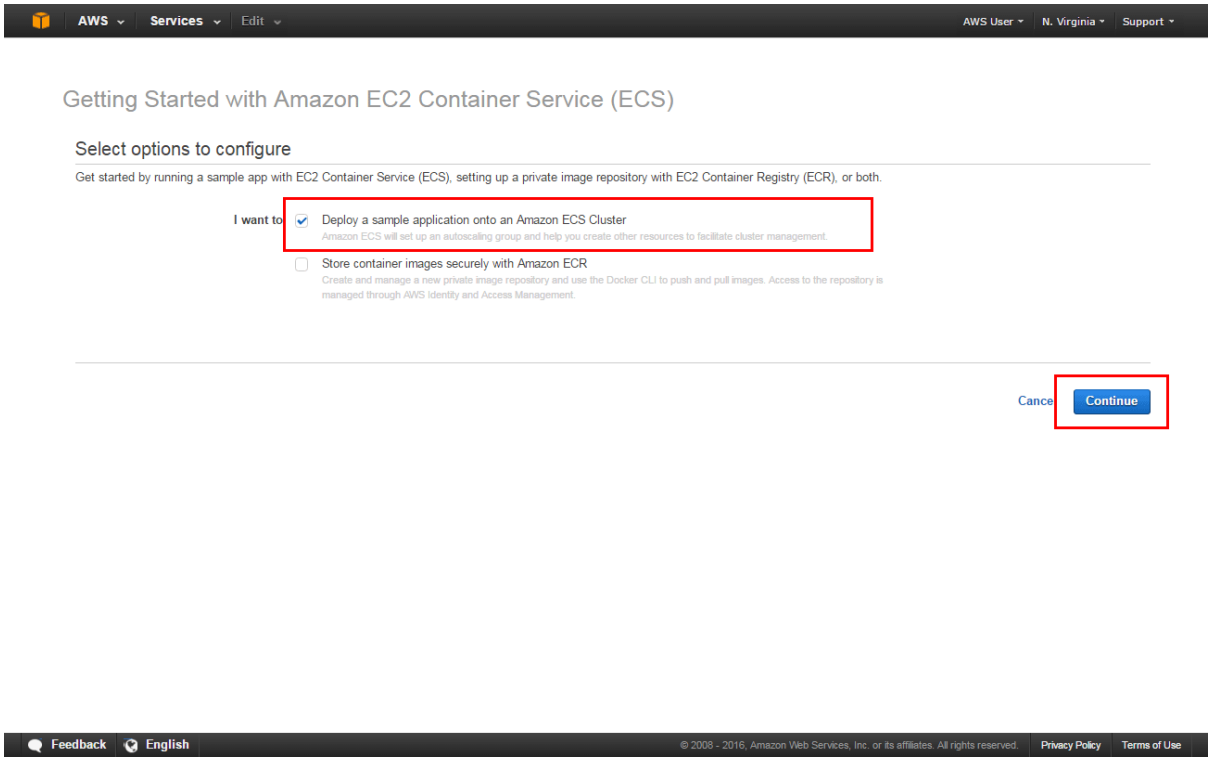
Step 1: Set up your first run with Amazon ECS

The Amazon ECS first run wizard will guide you through creating a cluster and launching a sample web application. In this step, you will enter the Amazon ECS console and launch the wizard.

a. Click here to open the Amazon ECS console first run wizard.

b. With Amazon ECS, you have the option to use Amazon Elastic Container Registry (Amazon ECR) to create an image repository and push an image to it as part of the first run wizard (see the screenshot to the right). This feature is currently available in select regions.

- If you do not have Amazon ECR options, skip to step 2.
- If you have Amazon ECR options, uncheck the box next to *Deploy a sample application onto an Amazon ECS Cluster* and select Continue.

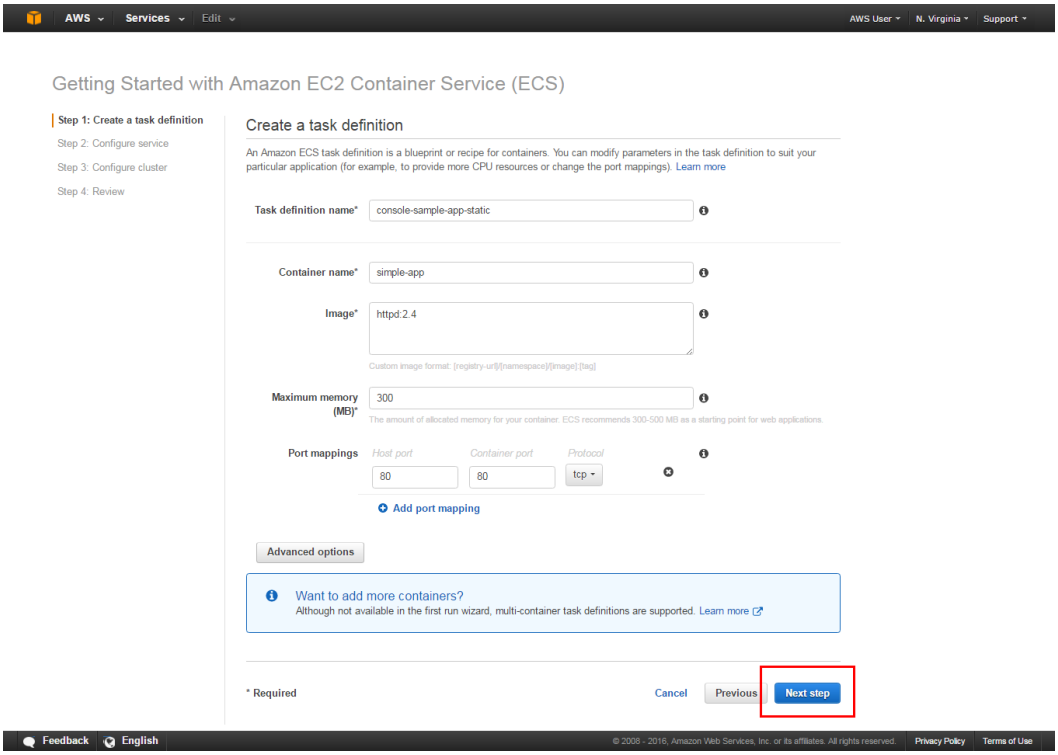


Step 2: Create a task definition

A *task definition* is like a blueprint for your application. In this step, you will specify a task definition so Amazon ECS knows which Docker image to use for containers, how many containers to use in the task, and the resource allocation for each container.

The task definition comes pre-loaded with default configuration values.

- Review the default values and select Next Step.

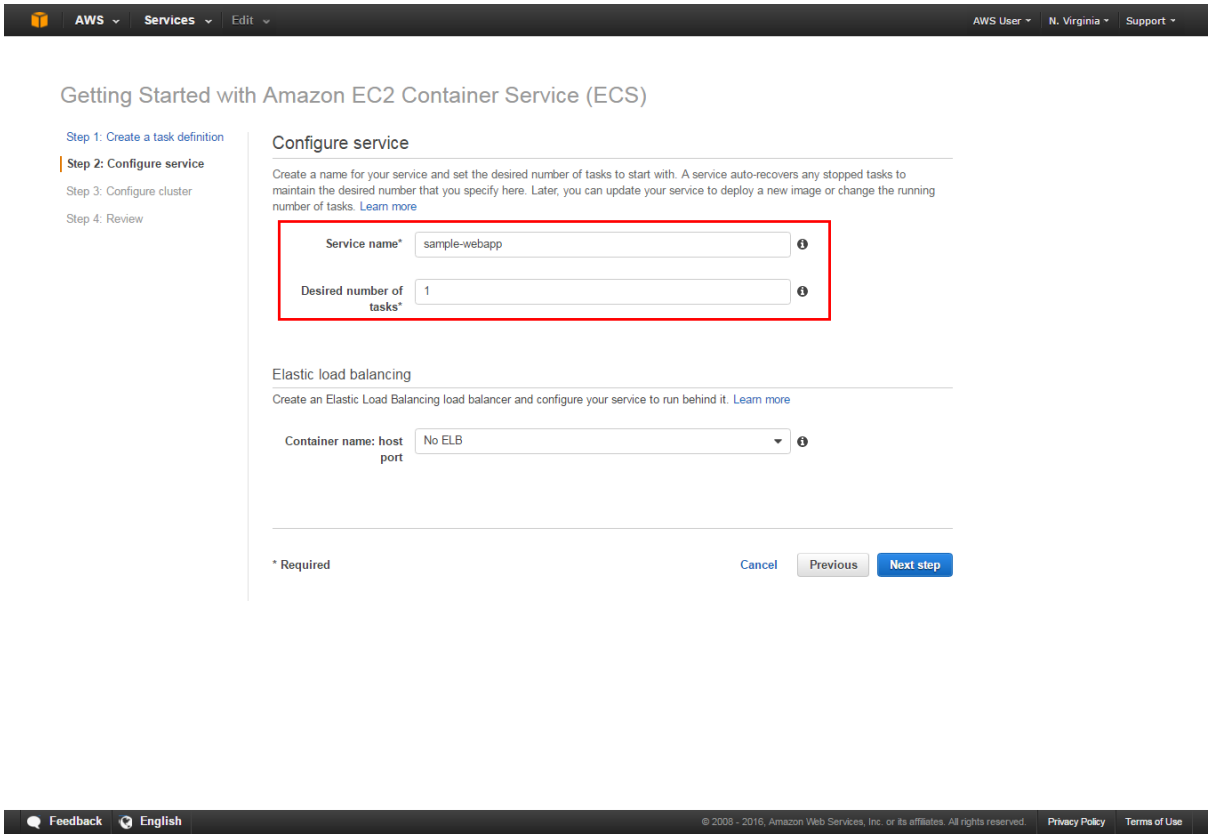


Step 3: Configure your service

Now that you have created a task definition, you will configure the Amazon ECS *service*. A service launches and maintains copies of the task definition in your cluster. For example, by running an application as a service, Amazon ECS will auto-recover any stopped tasks and maintain the number of copies you specify.

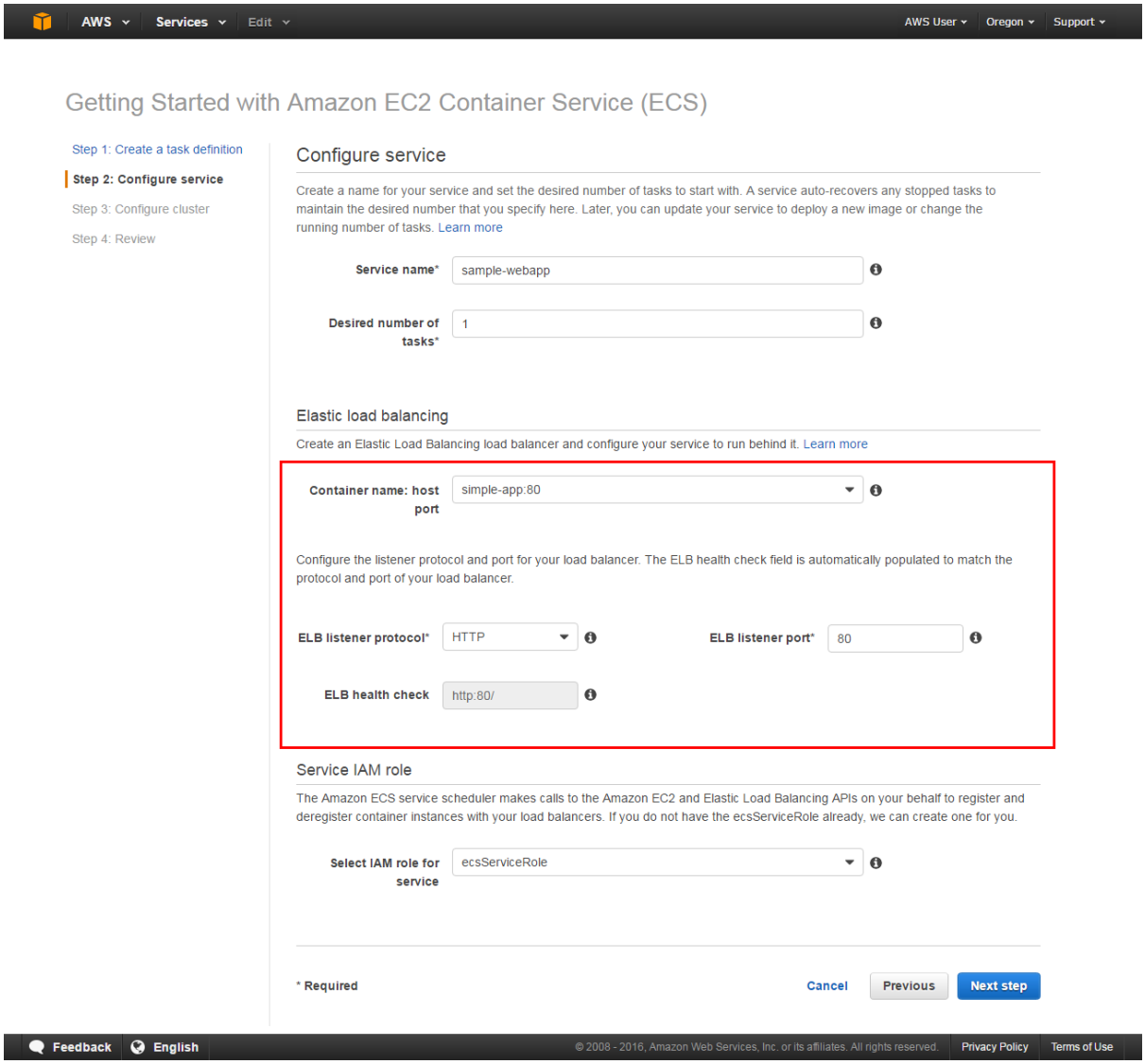
a. Configure service options:

- Service Name: The default *sample-webapp* is a web-based "Hello World" application provided by AWS. It is meant to run indefinitely, so by running it as a service, it will restart if the task becomes unhealthy or unexpectedly stops.
- Desired number of tasks: To stay within the AWS free tier, leave the default value of *1*. This will create 1 copy of your task.



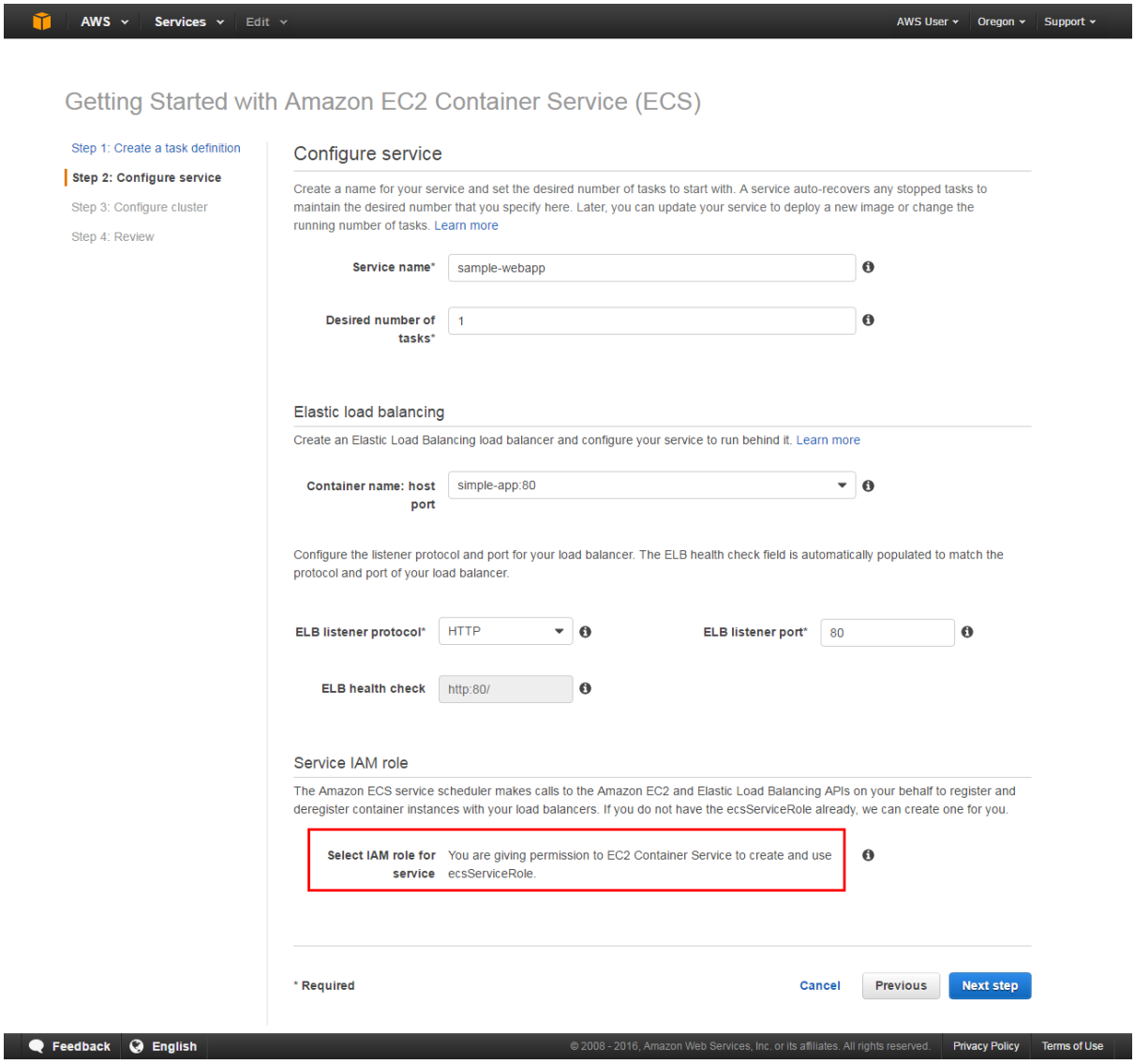
b. Elastic load balancing: You have the option to use a load balancer with your service. Amazon ECS can create an Elastic Load Balancing (ELB) load balancer to distribute the traffic across the container instances your task is launched on.

- Container name: host port: select *Simple-app:80.*
- The default values for *ELB listener protocol, ELB listener port, and ELB health check* are set up for the sample application. For more information on load balancing configuration, see Service Load Balancing.

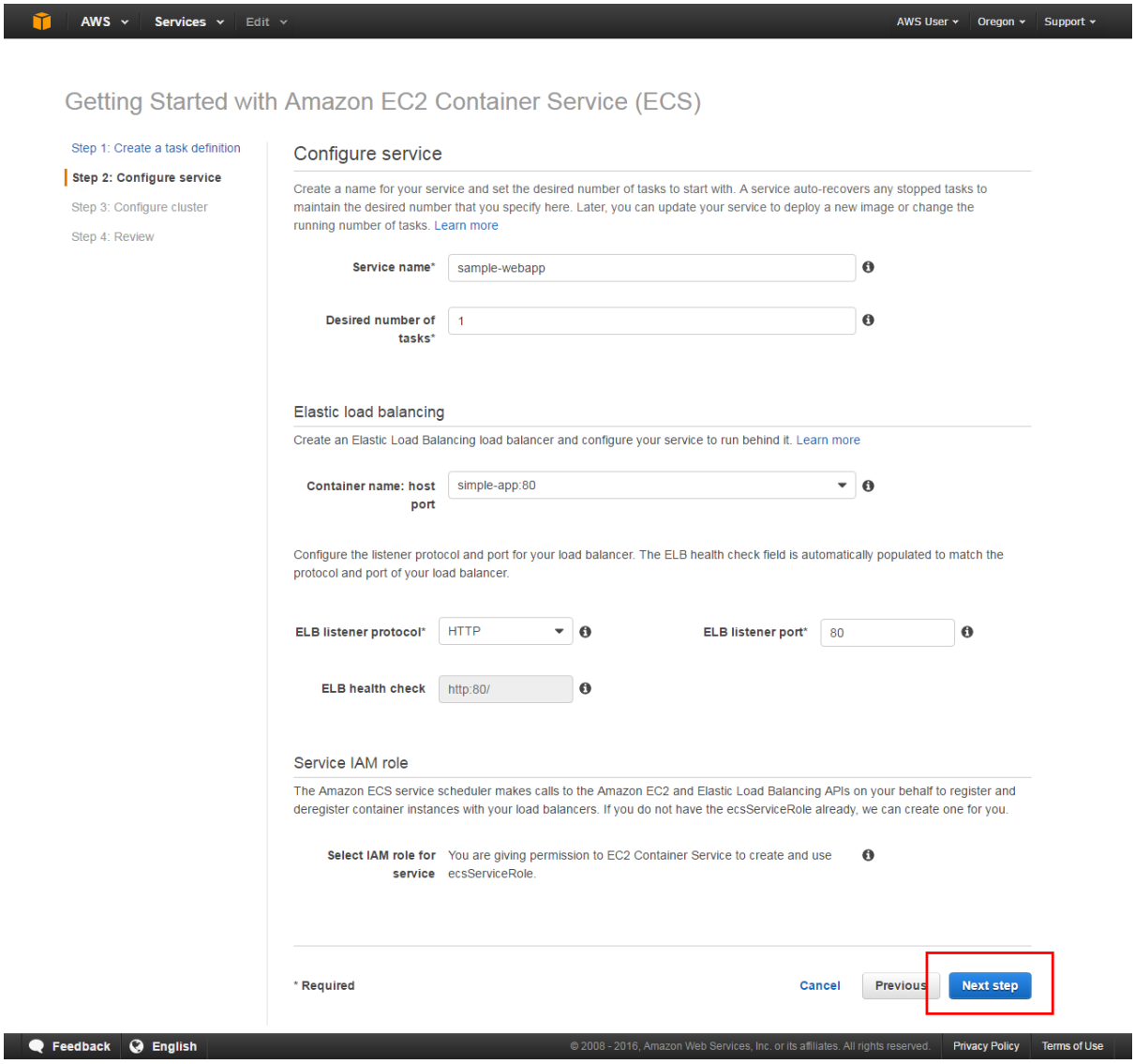


c. Before you can attach a load balancer to an Amazon ECS service, you must create an Identity and Access Management (IAM) role for your services to use. This will allow Amazon ECS to make calls to the Amazon EC2 and Elastic Load Balancing APIs to register and deregister instances with your load balancers.

- If you do not have a Service IAM Role already, Amazon ECS will create one named *ecsServiceRole*.
- If you have an existing Amazon ECS service role, select it from the dropdown.



d. Review your settings and select Next Step.



Step 4: Configure your cluster

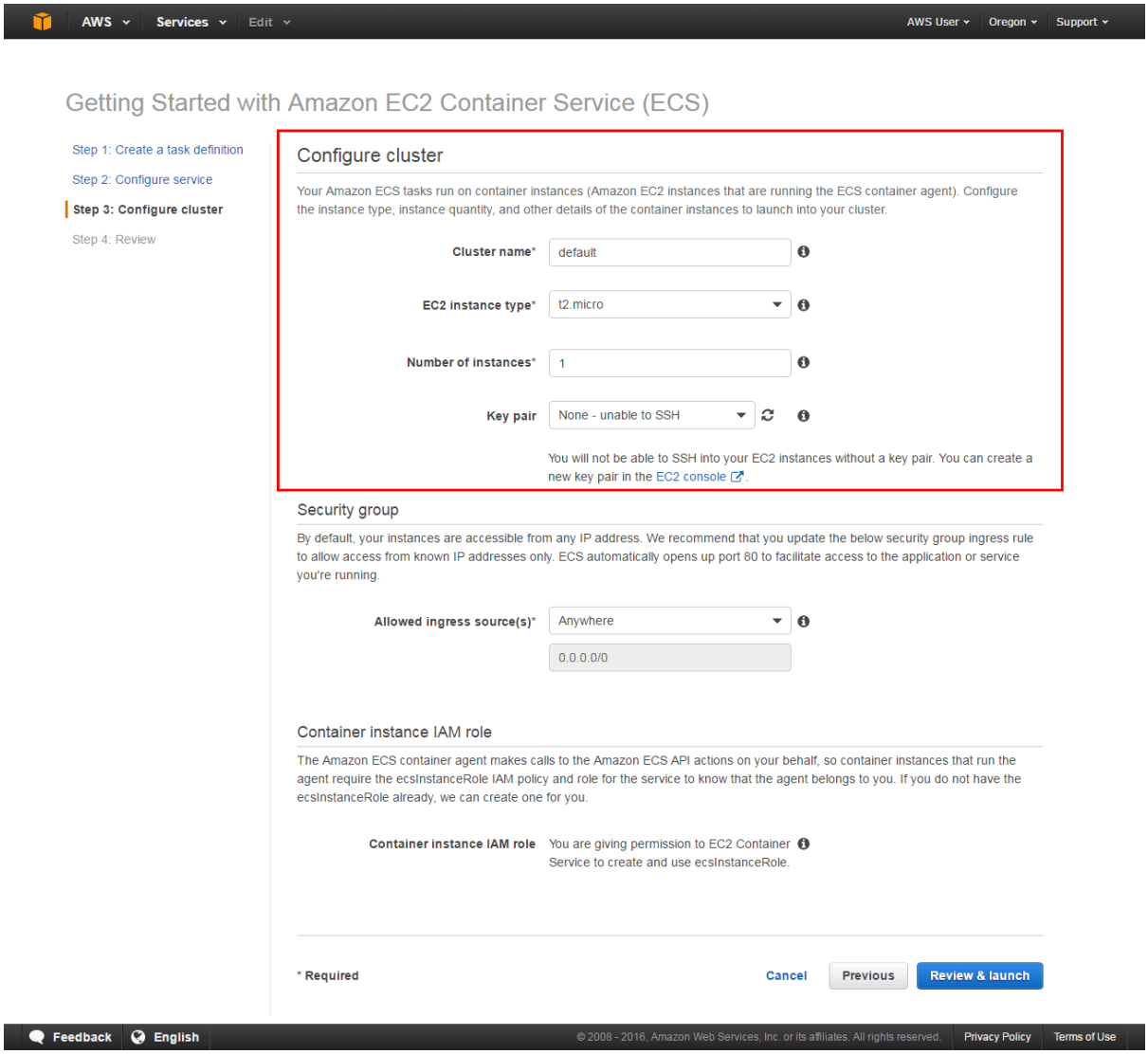
Your Amazon ECS tasks run on a *cluster*, which is the set of container instances running the Amazon ECS container agent. In this step, you will configure the cluster, review security settings, and set IAM roles.

a. Follow the configuration settings below:

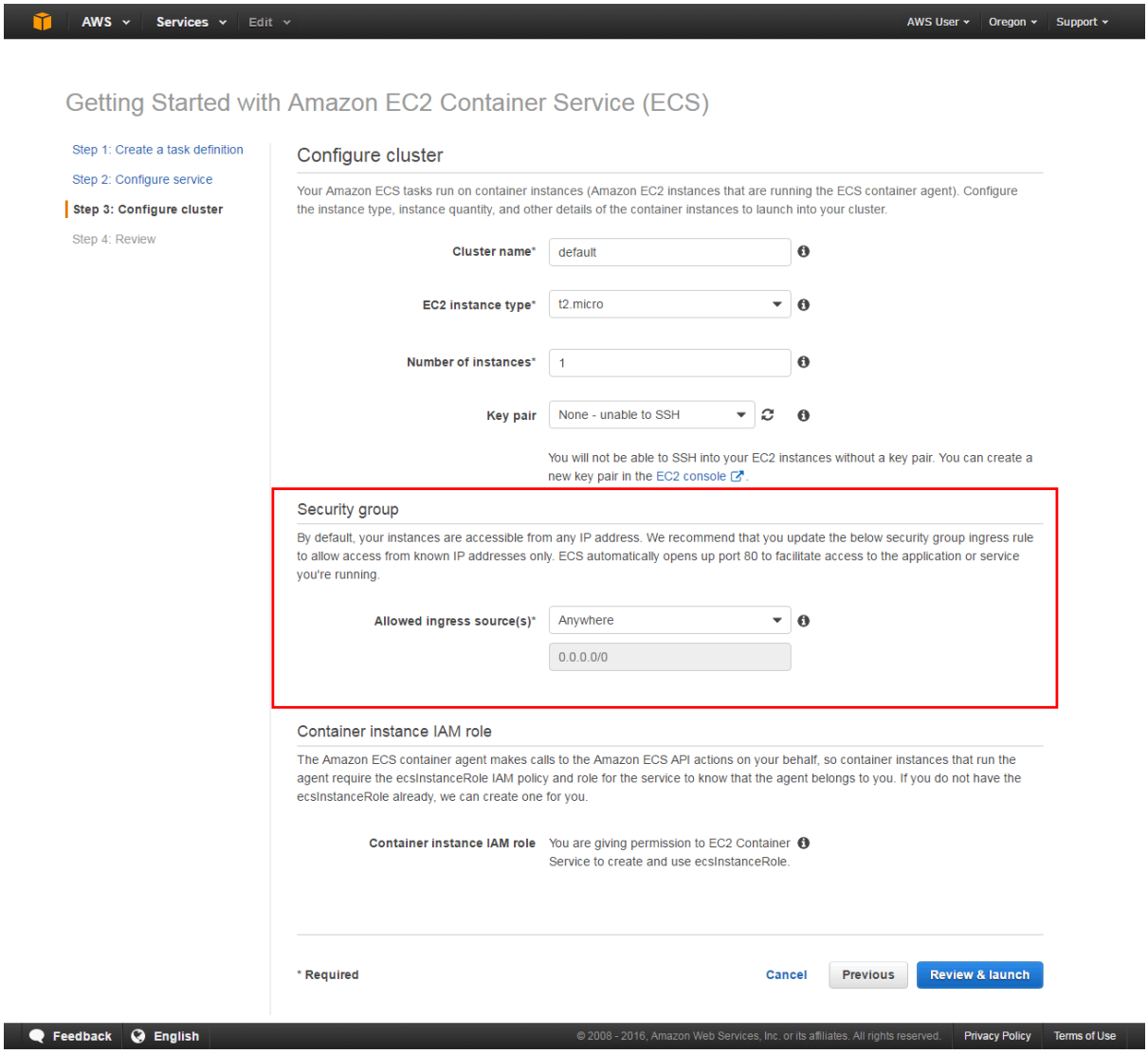
- Cluster name: Enter *sample-cluster.*
- EC2 instance type: The default *t2.micro* instance type will keep you within the free tier. Instance types with more CPU and memory resources can handle more tasks.

For more information on the different instance types, see Amazon EC2 Instance Types.

- Number of instances: Leave the default value of *1* to launch one Amazon EC2 instance to launch into your cluster for tasks to be placed on. The more instances you have in your cluster, the more tasks you can place on them.
- Key pair: A key pair is required to SSH into your instances later on. You can continue by selecting *None - unable to SSH,* selecting an existing key pair, or by creating one in the Amazon EC2 console.

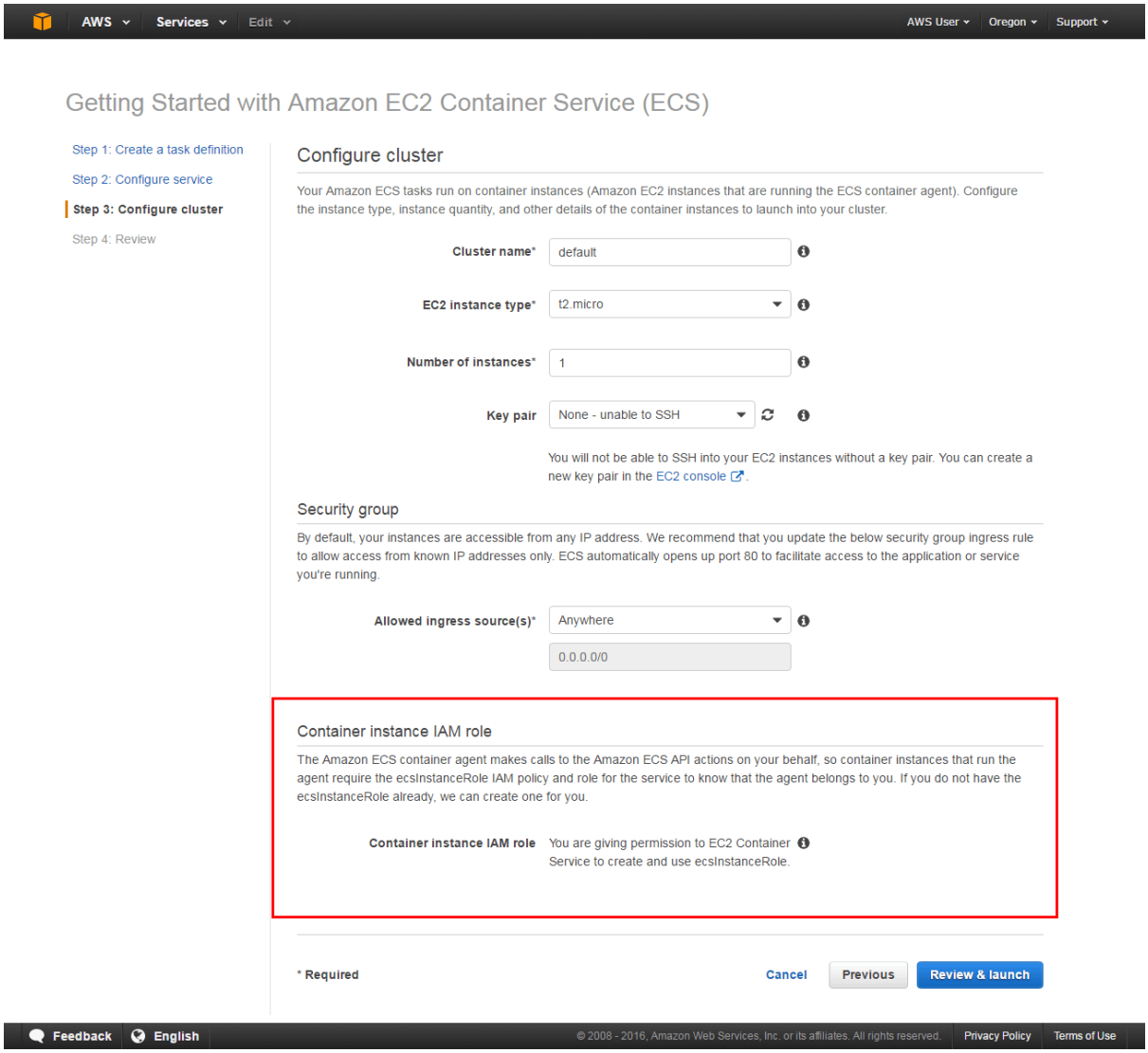


b. (Optional) Security Group: The default value *(Anywhere)* allows access from the entire Internet. You also have the option to choose a CIDR block that restricts access to your instances.

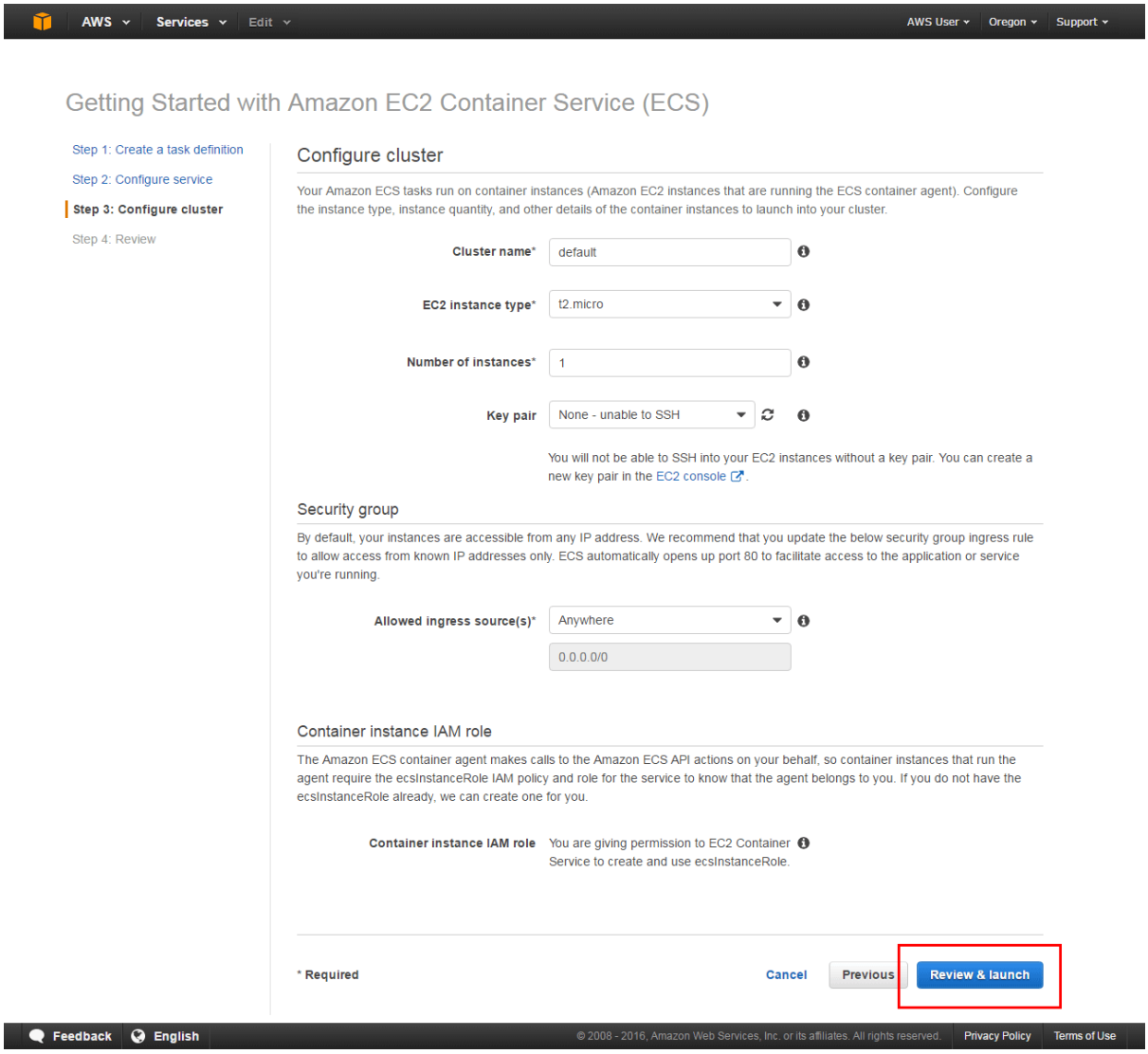


c. Container instance IAM role:

- If you do not have an IAM role, the Amazon ECS wizard will create one for you.
- If you have an existing container instance IAM role, select it from the dropdown list.



c. Select Review and Launch.

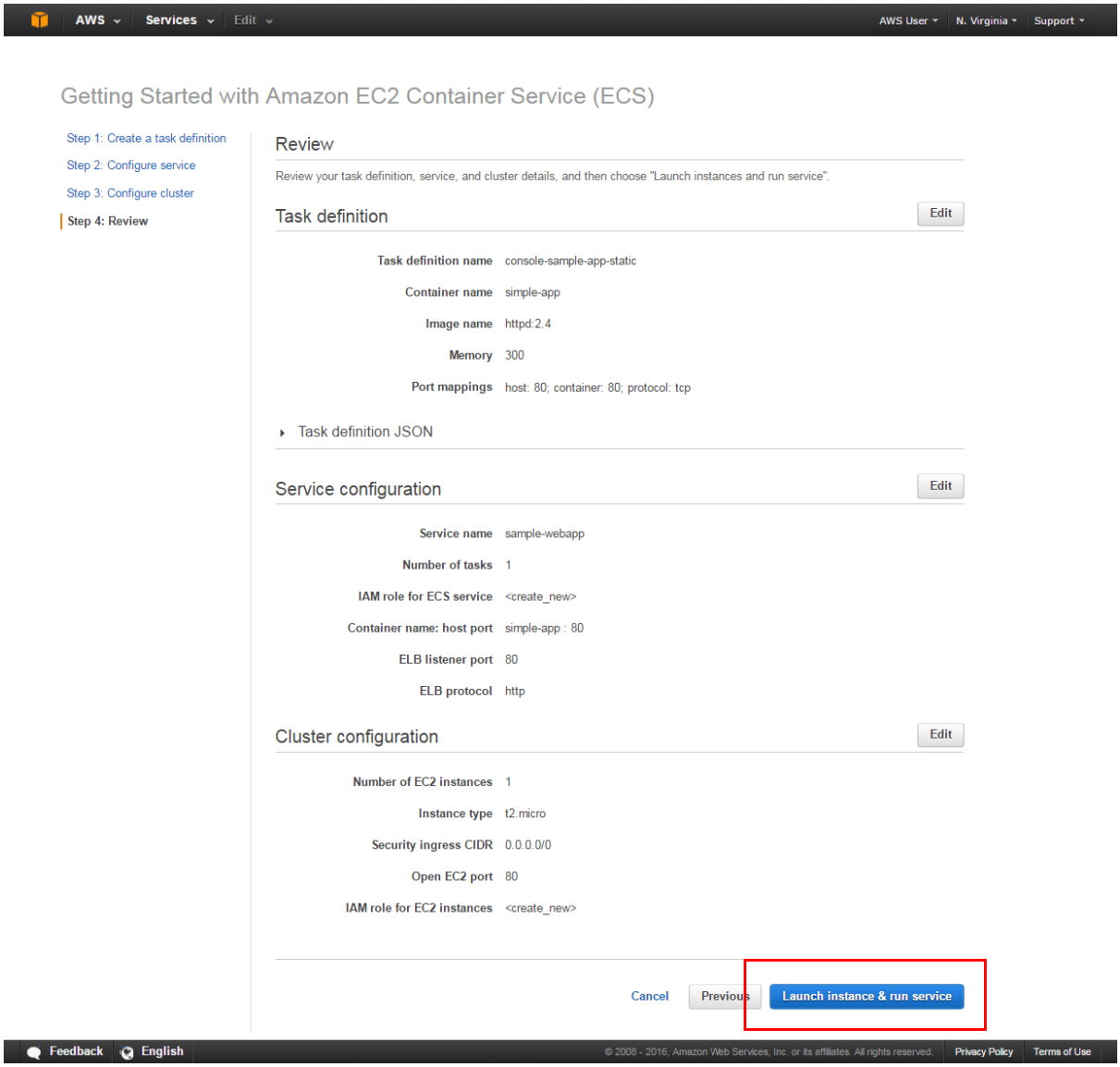


Step 5: Launch and view your resources

In previous steps, you have configured your task definition (which is like an application blueprint), the Amazon ECS service (which launches and maintains copies of your task definitions), and your cluster (which is the set of container instances running the container agent). In this step, you will review, launch, and view the resources you create.

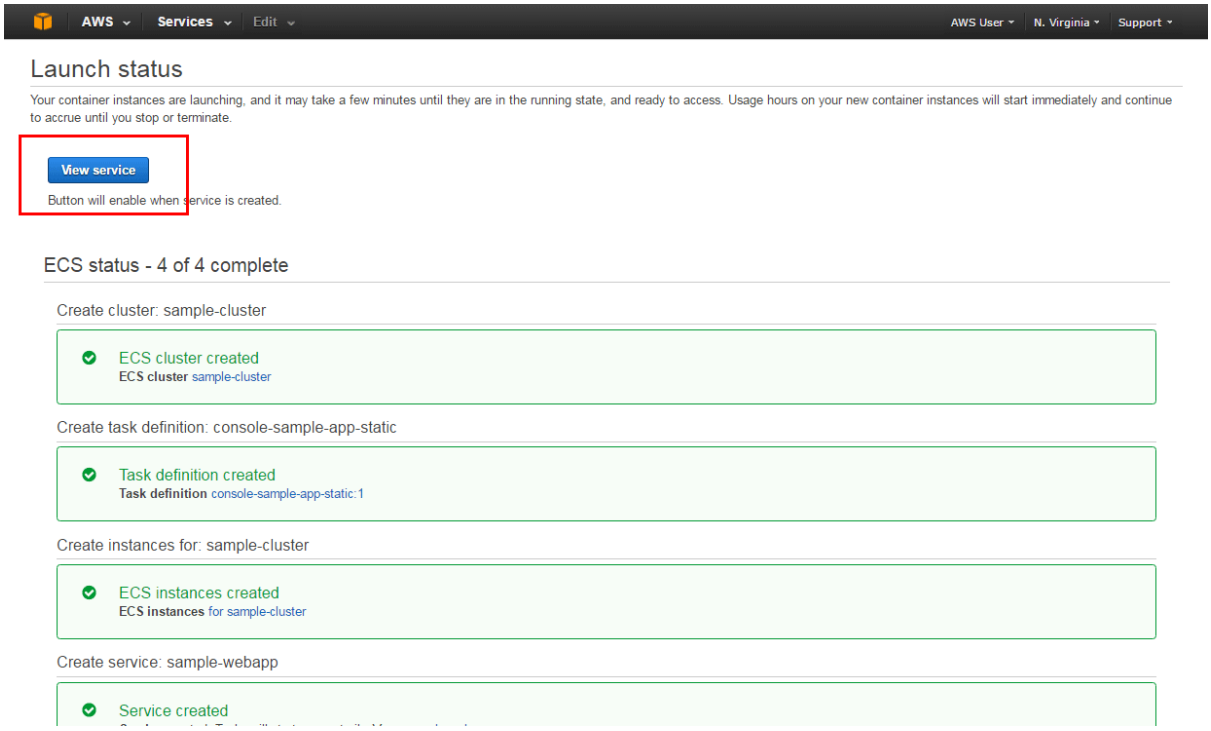
a. You have a final chance to review your task definition, task configuration, and cluster configurations before launching.

- Select Launch instance & run service.



b. You are on a *Launch Status* page that shows the status of your launch and describes each step of the process.

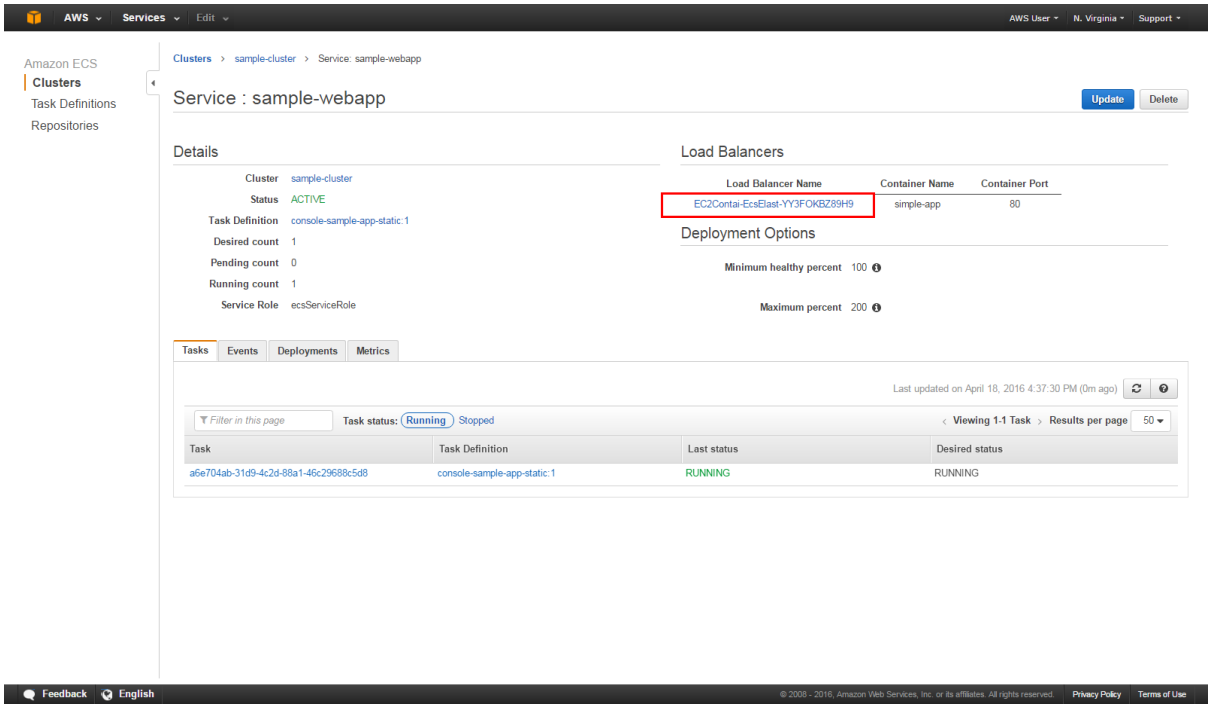
- After the launch is complete, select View service.



Step 6: Open the Sample Application

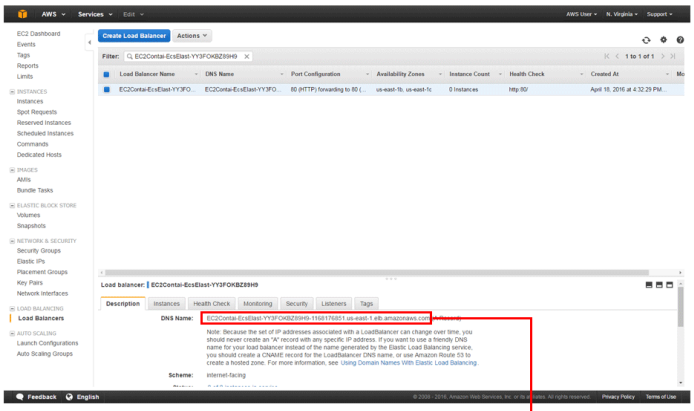
In this step, you will verify that the sample application is up and running by pointing your browser to the load balancer DNS name.

a. On the sample-webapp page, click on your Load Balancer Name.

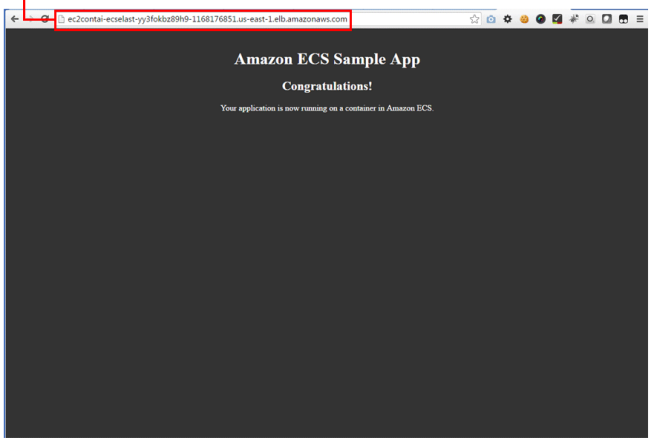


b. You will now test the sample application:

- Copy the ELB DNS name.
- Paste it into a new browser window.
- Hit Enter on your keyboard to view the sample application (in this case, a static webpage).



Amazon EC2 Console: Elastic Load Balancer



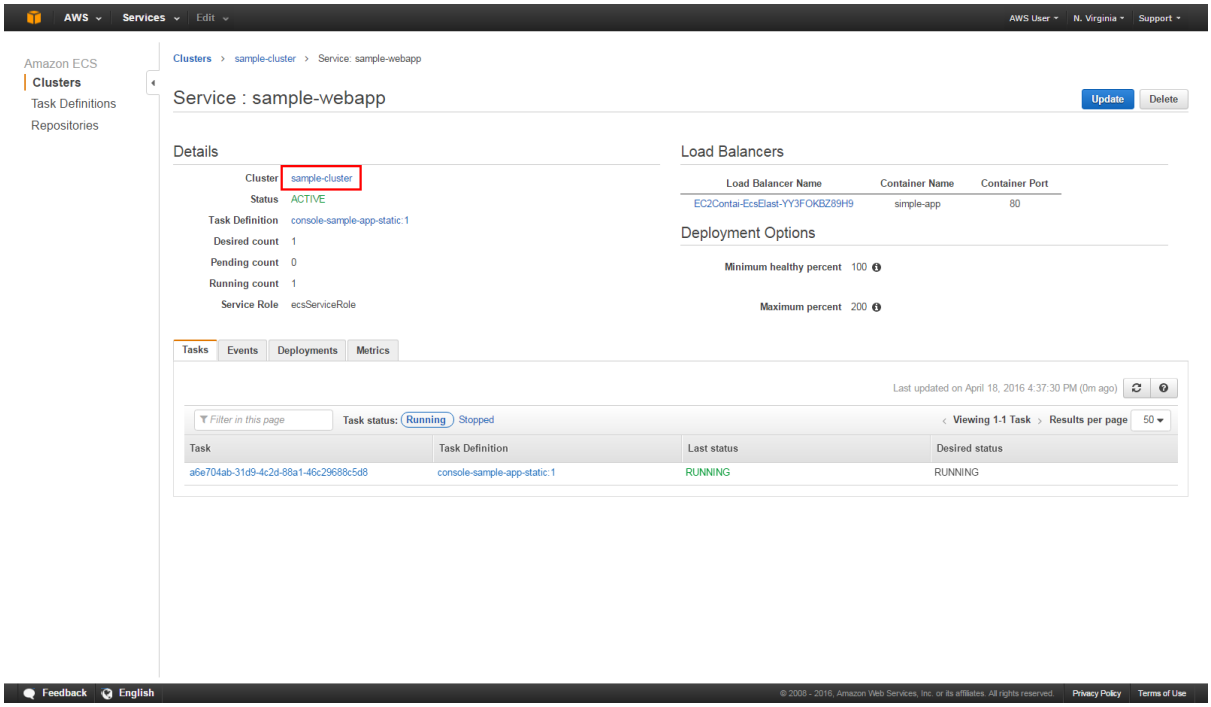
Web Browser

Step 7: Delete Your Resources

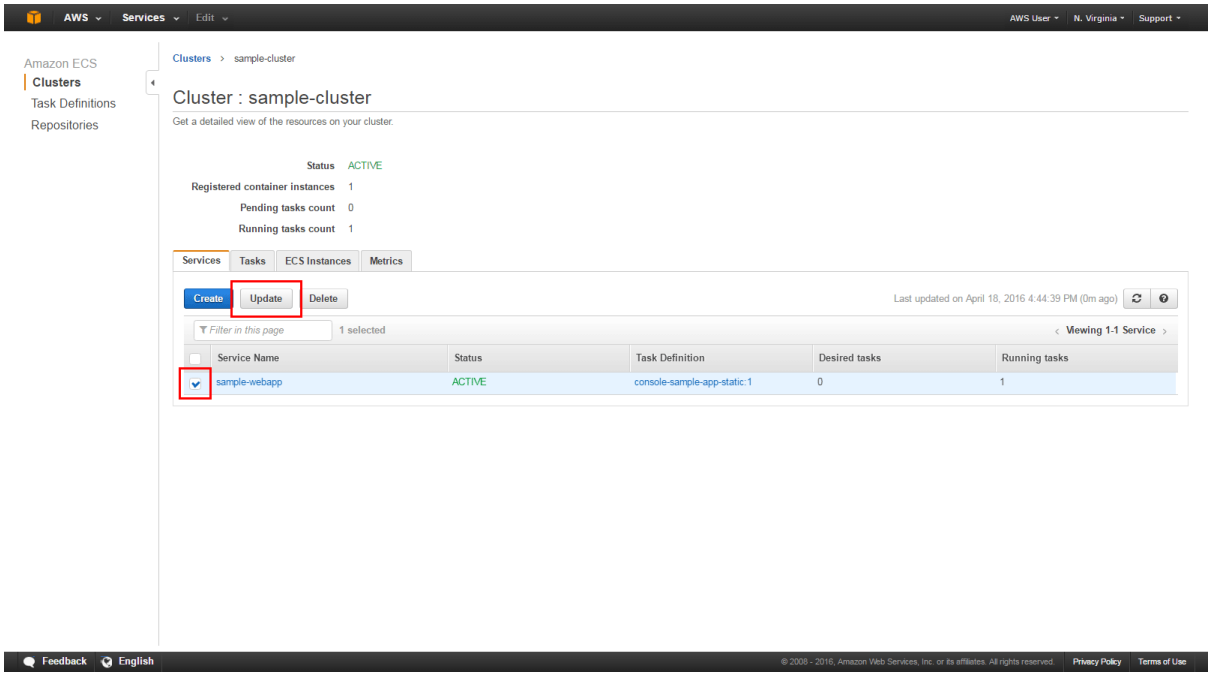
Throughout this tutorial, you've launched three resources: an Amazon ECS cluster, an Amazon EC2 instance, and a load balancer. In this step, you will clean up all your resources to avoid unwanted charges.

a. Navigate back to the Amazon ECS console page

- Click on the cluster name (sample-cluster).

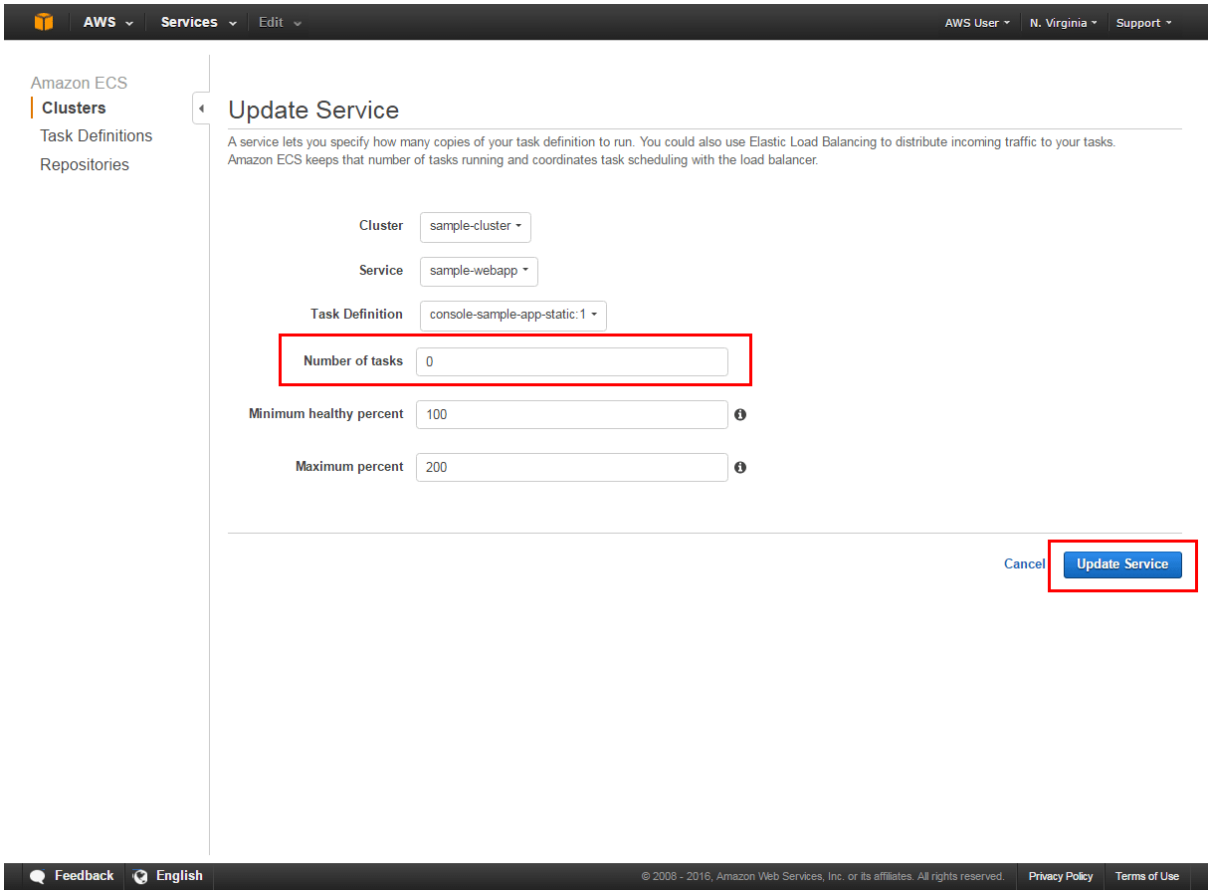


b. Select the checkbox next to *sample-webapp* and click Update.



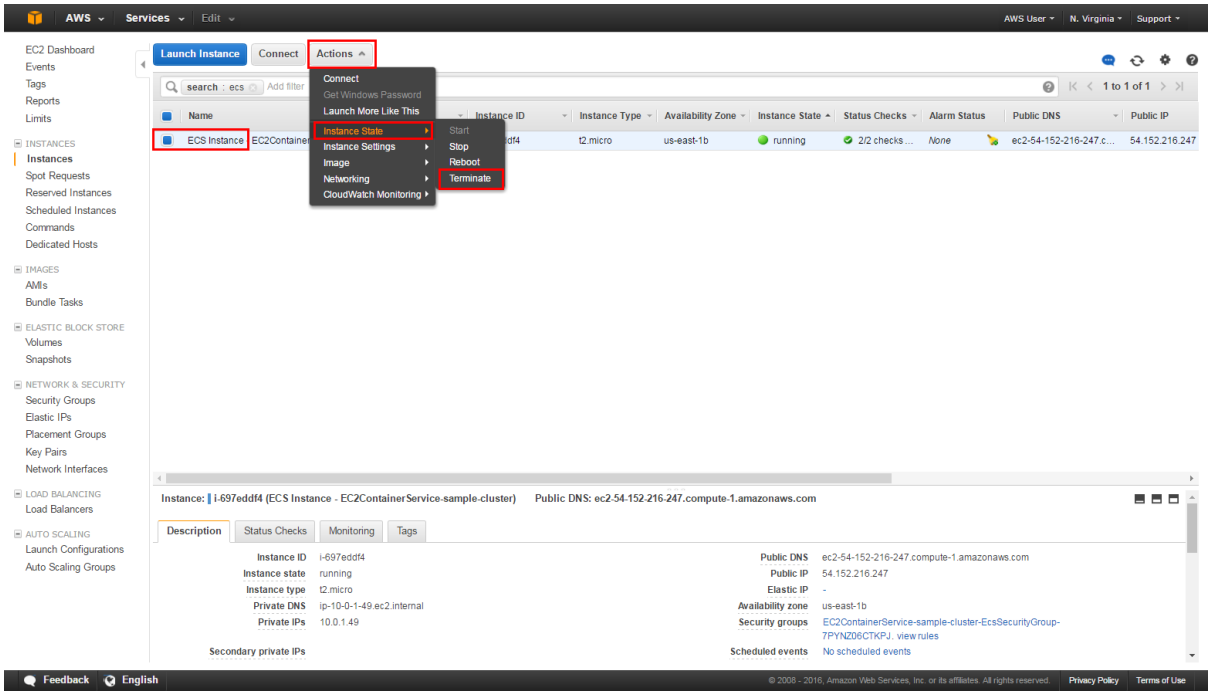
c. To ensure you don't accidentally delete a service with active tasks, you need to stop all tasks before Amazon ECS will delete a service.

- Set the Number of tasks to *0* and select Update Service.
- After you update your service, select Delete.



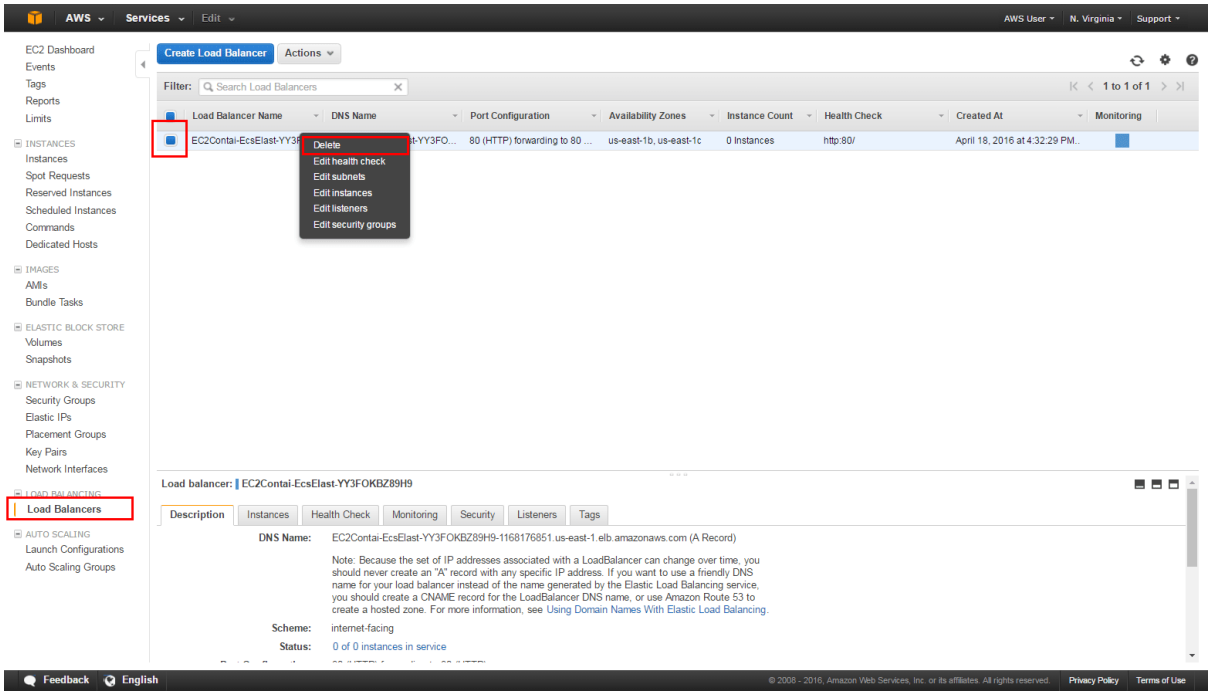
d. Delete the Amazon EC2 instances that were launched with your cluster:

- Enter the Amazon EC2 console
- In the left hand panel, select Instances.
- Select the checkbox next to the instance named *ECS Instance - EC2ContainerService-default.*
- Select Actions > Instance State > Terminate.



e. Delete your load balancers:

- On the left panel, select Load Balancers.
- Select the checkbox next to the load balancer you created for your service (it should start with *EC2Contai-EcsElast*).
- Right click and select Delete.


AWS
Services
Edit
AWS User
N. Virginia
Support
EC2 Dashboard
Events
Tags
Reports
Limits
INSTANCES
Instances
Spot Requests
Reserved Instances
Scheduled Instances
Commands
Dedicated Hosts
IMAGES
AMIs
Bundle Tasks
ELASTIC BLOCK STORE
Volumes
Snapshots
NETWORK & SECURITY
Security Groups
Elastic IPs
Placement Groups
Key Pairs
Network Interfaces
LOAD BALANCING
Load Balancers
AUTO SCALING
Launch Configurations
Auto Scaling Groups
Create Load Balancer
Actions
Filter:
Search Load Balancers
1 to 1 of 1
Load Balancer Name
DNS Name
Port Configuration
Availability Zones
Instance Count
Health Check
Created At
Monitoring
EC2Contai-EcsElast-YY3F
t-YY3FO...
80 (HTTP) forwarding to 80 ...
us-east-1b, us-east-1c
0 Instances
http:80/
April 18, 2016 at 4:32:29 PM..
Delete
Edit health check
Edit subnets
Edit instances
Edit listeners
Edit security groups
Load balancer: EC2Contai-EcsElast-YY3FOKBZ89H9
Description
Instances
Health Check
Monitoring
Security
Listeners
Tags
DNS Name:
EC2Contai-EcsElast-YY3FOKBZ89H9-1168176851.us-east-1.elb.amazonaws.com (A Record)
Note: Because the set of IP addresses associated with a LoadBalancer can change over time, you should never create an "A" record with any specific IP address. If you want to use a friendly DNS name for your load balancer instead of the name generated by the Elastic Load Balancing service, you should create a CNAME record for the LoadBalancer DNS name, or use Amazon Route 53 to create a hosted zone. For more information, see Using Domain Names With Elastic Load Balancing.
Scheme:
internet-facing
Status:
0 of 0 instances in service
Feedback
English
© 2008 - 2016, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use