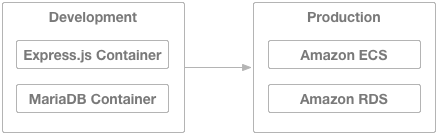
Take Containers From Development To Amazon ECS

Introduction

It's a lot of work to set up your app to use containers and you need a lot of expertise to do it right. It's beneficial to use containers though, due to their lightweight footprint and the ease with which you can run the same code across different environments such as development and production. Bitnami makes it easy to use containers in development and production by offering open-source containers for commonly used languages and application frameworks.

In this tutorial, you will work through the full product development lifecycle for a web application.

[](https://docs.bitnami.com/images/img/how_to_guides/ecs-rds-tutorial/process-bff7fcb8.png)

You will learn the following:

How to use containers from Bitnami to create an Express.js web app

How to run an Express.js app in a production environment consisting of Amazon Relational Database Service (RDS) and Amazon EC2 Container Service (ECS)

How to carry out production deployments of new versions of your Express.js app

How to troubleshoot production issues

How to cleanly delete AWS resources

Requirements

You will need [Docker Hub](https://hub.docker.com/) and [AWS](https://aws.amazon.com/) accounts for this tutorial. Both services allow free sign up.

Step 1: Create Express.js App In Development Environment

One of the easiest ways to set up a development environment with Express.js is to run the Docker compose file available in [Bitnami's GitHub repo](https://github.com/bitnami/bitnami-docker-express) . A Docker compose file describes the containers corresponding to the components of your application such as the database and the application framework and allows you to run an instance of your app. The Express.js Docker compose file that we will use in this tutorial references a MariaDB container and an Express.js container.

Step 1.1: Generate Express.js App Scaffolding

Get the Docker compose file using the following commands:

mkdir express-app cd express-app curl -L -o docker-compose.yml https://raw.githubusercontent.com/bitnami/bitnami-docker-express/master/docker-compose-mariadb.yml

Install the [Docker service](https://www.docker.com/products/overview) if you haven't already and ensure it is running.

Ensure you have the latest container images for our app:

docker-compose pull

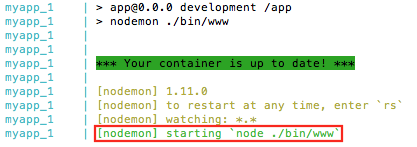
Bring up our dev environment by running the following command. This will start the MariaDB and Express.js containers and create the scaffolding for the Express.js app.

docker-compose up -d

Tail the logs to see the dev environment initialization progress:

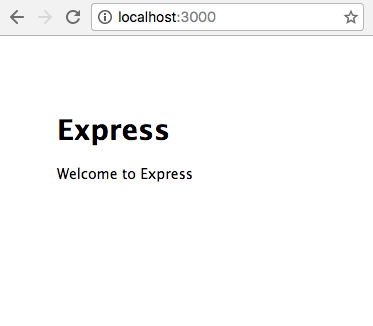
docker-compose logs -f

Wait several seconds for the Node.js invocation to appear. Now our Express.js app scaffolding has been created.

[](https://docs.bitnami.com/images/img/how_to_guides/ecs-rds-tutorial/scaffolding-progress-6e8e0bc1.png)

Type *Ctrl+C* to stop tailing the logs

Navigate to [http://localhost:3000](http://localhost:3000/) to see the default Express.js welcome page.

[](https://docs.bitnami.com/images/img/how_to_guides/ecs-rds-tutorial/initial-homepage-on-dev-stack-0df17326.png)

Step 1.2: Connect App To Database

Connect to our database with the *mysql* client using the following command. This runs the *mysql* command inside the *mariadb* container and connects to the *myapp* database.

docker-compose exec mariadb mysql myapp

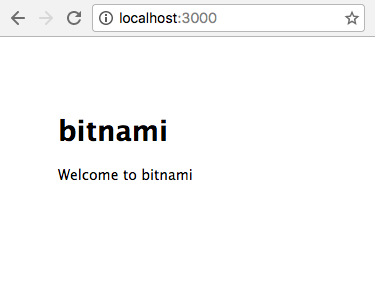
Once you're in the *mysql* shell, create our database schema and add data to it.

mysql> create table names (name varchar(15)); mysql> insert into names values ('bitnami'); mysql> exit

Use a text editor to include database query logic in *routes/index.js* so that it looks like the following. *process.env.DATABASE\_URL* contains the database connection string and was defined in the Docker compose file. App code changes are automatically picked up by the dev environment so we now have the app working end-to-end.

var express = require('express'); var router = express.Router(); var mysql = require('mysql'); // GET homepage router.get('/', function(req, res, next) { // establish database connection var connection = mysql.createConnection(process.env.DATABASE\_URL); connection.connect(); // get name to display from database and render it connection.query('select name from names', function(err, rows, fields) { if (err) throw err; res.render('index', {title: rows[0].name}); }); // close database connection connection.end(); }); module.exports = router;

Navigate to [http://localhost:3000](http://localhost:3000/) to see the app displaying the name it's pulling from the database.

[](https://docs.bitnami.com/images/img/how_to_guides/ecs-rds-tutorial/final-homepage-on-dev-stack-b34f16e2.png)

We now have an application that's ready to be deployed to prod.

Step 2: Run Express.js App On Production Environment

To run our application on prod, we need to first package our app into a container image. Then we need to set up Amazon RDS for our prod database. Finally, we need to set up Amazon ECS for our prod web application and deploy our container image to it.

Step 2.1: Package App For Deployment

Let's begin by packaging our app into a container image and then including the container image in a registry so that Amazon ECS can access it for the prod deployment. A *Dockerfile* was generated when you ran *docker-compose* to create the dev environment and this will be used to package the app.

Run the following docker build command to package the app. Make sure to replace YOUR\_DOCKER\_HUB\_USER with your [Docker Hub](https://hub.docker.com/) username.

docker build -t <YOUR\_DOCKER\_HUB\_USER>/express-app .

|  |
| --- |
| TIP: Don't forget the period at the end of the docker build command. |

The Docker build step creates a container image that is stored on your filesystem as a JSON configuration file and a directory that holds the filesystem for your container. It only contains the code for your Express.js application and the Node.js runtime. MariaDB and the data in it are not included in the container image and will need to be updated on prod independently of the app update process. The steps to do so are described later in this tutorial.

Docker Hub is a commonly used container registry so let's expose our container image through there. By default, images on Docker Hub are publicly available. You can change this setting in Docker Hub.

Push the container image for your app to Docker Hub:

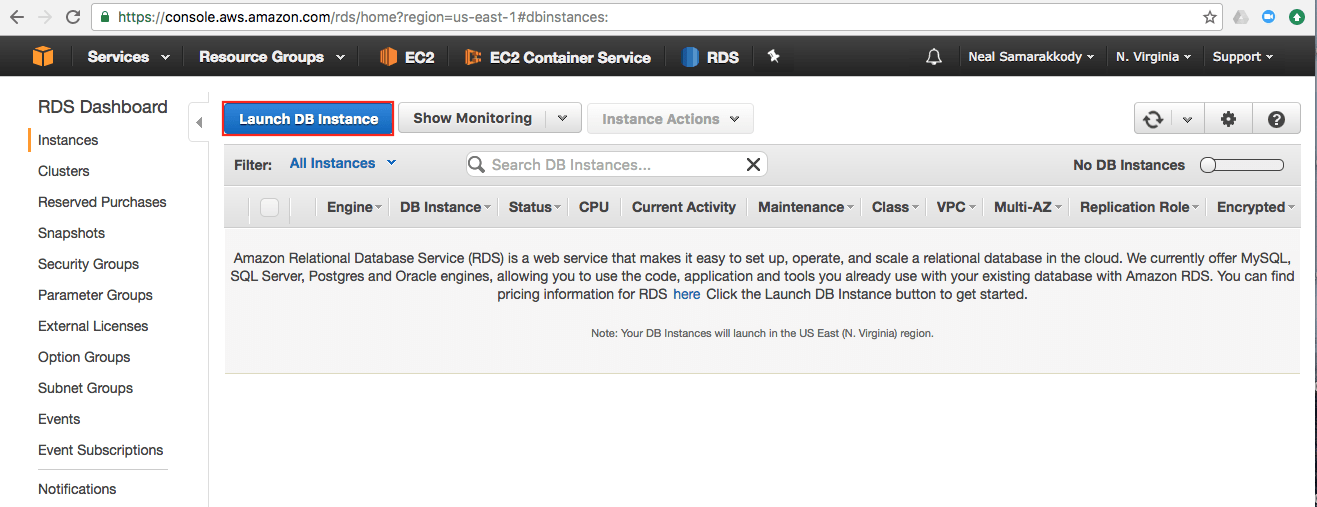
docker login Username: <YOUR\_DOCKER\_HUB\_USER> Password: <YOUR\_DOCKER\_HUB\_PASSWORD> docker push <YOUR\_DOCKER\_HUB\_USER>/express-app

Step 2.2: Set Up Amazon RDS

Amazon RDS can help you get going quickly without worrying about the low-level details of database administration. Set up your prod database with the following steps.

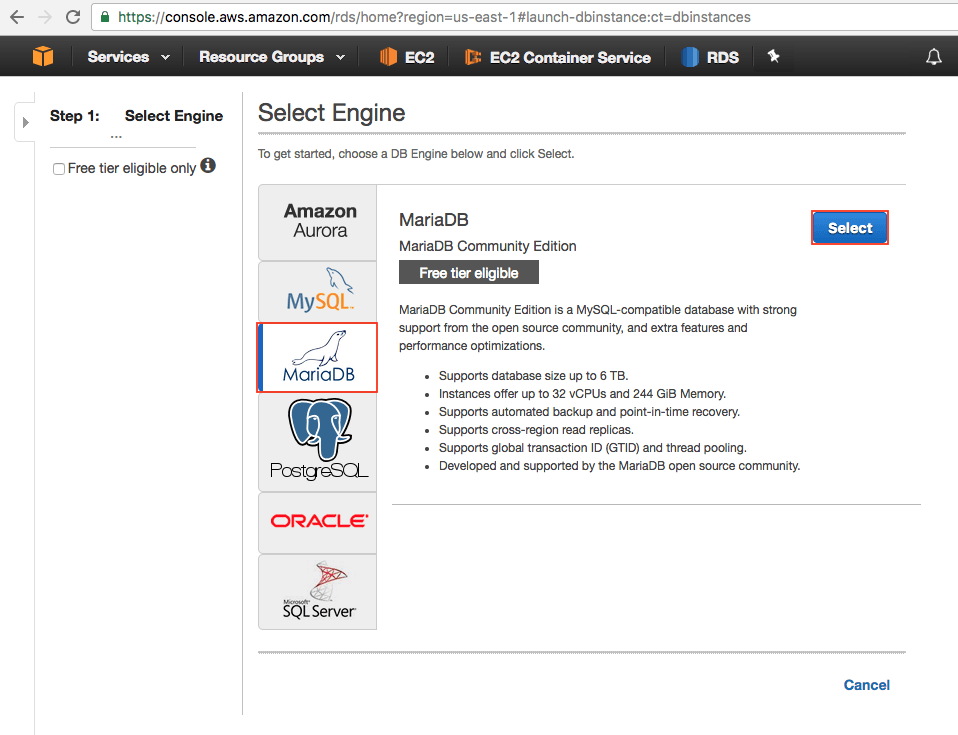
Go to <https://console.aws.amazon.com/rds/home?#dbinstances:>

Click *Launch DB instance*

[](https://docs.bitnami.com/images/img/how_to_guides/ecs-rds-tutorial/rds-dashboard-04ade7e3.png)

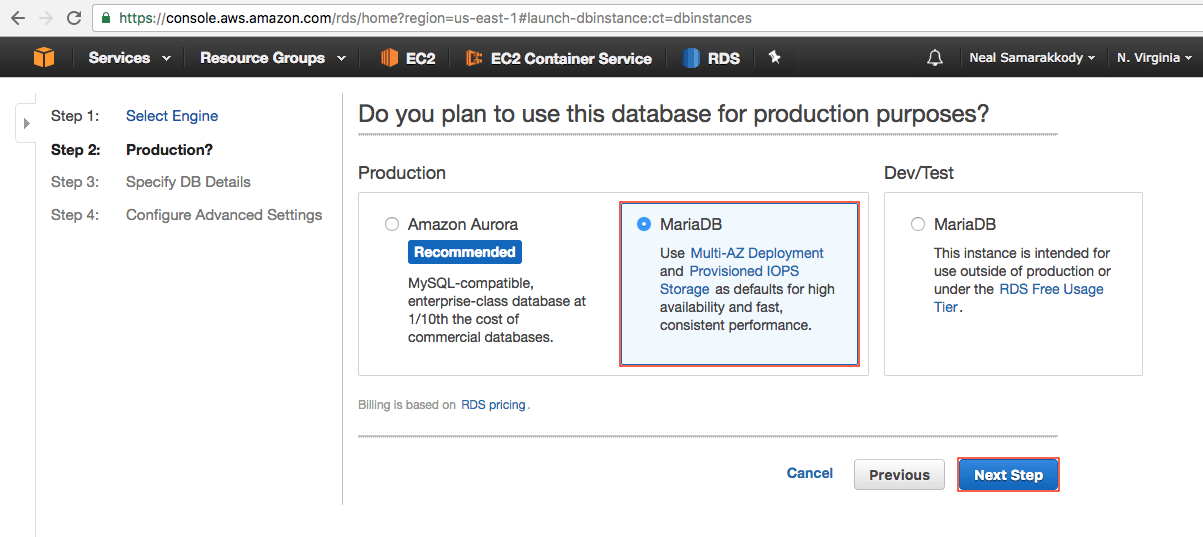
Click *MariaDB*

Click *Select*

[](https://docs.bitnami.com/images/img/how_to_guides/ecs-rds-tutorial/select-prod-db-aff30ba5.png)

Click *MariaDB* in the *Production* box

Click *Next Step*

[](https://docs.bitnami.com/images/img/how_to_guides/ecs-rds-tutorial/select-prod-db-purpose-fd1f2e2e.png)

Fill in the *Settings* section:

*DB Instance Identifier* : express-app

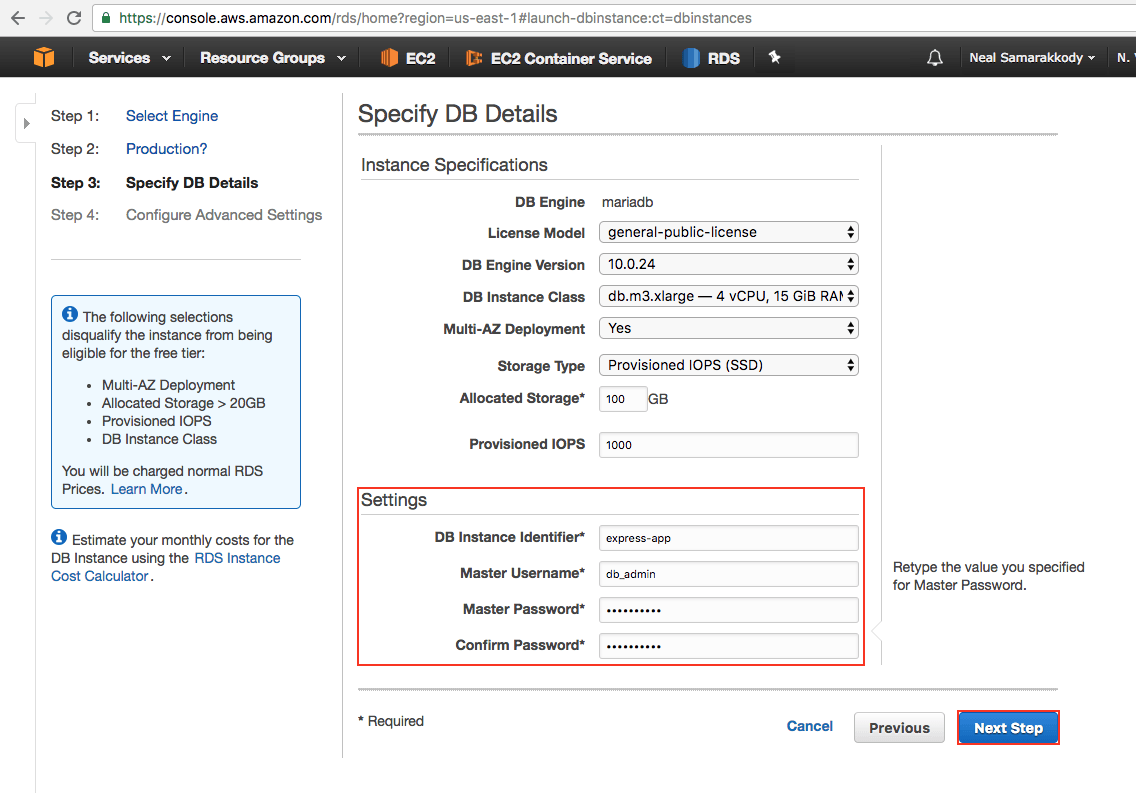
*Master Username* : db\_admin

*Master Password* : YOUR\_DB\_PASSWORD

*Confirm Password* : YOUR\_DB\_PASSWORD

|  |
| --- |
| NOTE: Record YOUR\_DB\_PASSWORD for use in subsequent steps of this tutorial. |

Click *Next Step*

[](https://docs.bitnami.com/images/img/how_to_guides/ecs-rds-tutorial/create-prod-db-1-9a57664b.png)

Fill in the *Network & Security* section:

VPC: Create new VPC

*Subnet Group* : Create new DB Subnet Group

*Publicly Accessible* : No

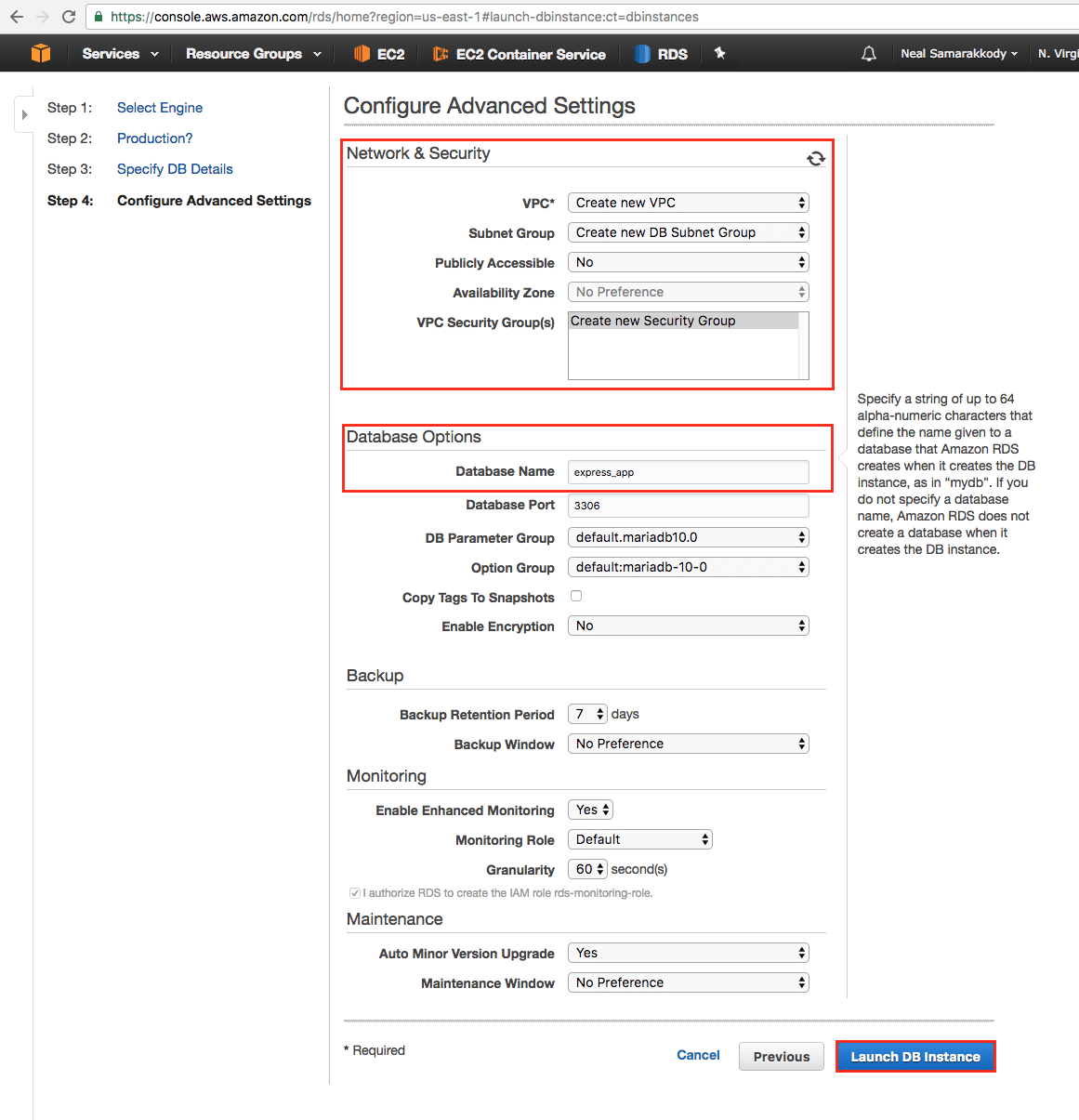
*Availability Zone* : No Preference

*VPC Security Group* : Create new Security Group

Fill in the *Database Options* section:

*Database Name* : express\_app

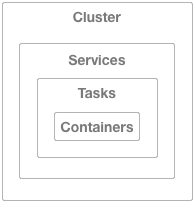
Click *Launch DB Instance*

[](https://docs.bitnami.com/images/img/how_to_guides/ecs-rds-tutorial/create-prod-db-2-df60f698.png)

It takes a few minutes for our database to start up. While that's happening, let's go set up our ECS cluster.

Step 2.3: Set Up Amazon ECS

The second part of our prod infrastructure requiring setup is the cluster that will run our containers. There are multiple layers to this cluster, as illustrated below. Each cluster is comprised of one or more services. Each service is comprised of one or more tasks. And each task is comprised of one or more containers.

[](https://docs.bitnami.com/images/img/how_to_guides/ecs-rds-tutorial/cluster-780465f6.png)

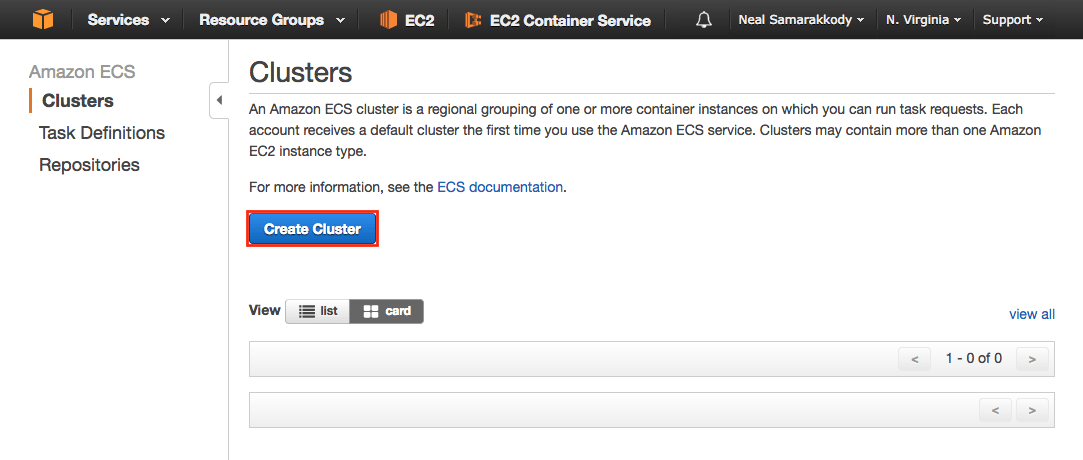
This structure supports a wide variety of application architectures. For example, a cluster could be your application, a service could be a microservice within that application, and a task could correspond to the database tier or the business logic tier of the microservice.

For our app, we already have the persistence layer set up in Amazon RDS so the only tier we need to run in Amazon ECS is the Express.js app. We will create a task defintion to spin up a single container for the app. This task will be encapsulated within a single service that allows us to stop and start the app. This service will run inside a cluster that corresponds to the EC2 host running the containers.

Let's begin by creating the top-level:

Navigate to <https://console.aws.amazon.com/ecs/home#/clusters>

Click *Create Cluster*

[](https://docs.bitnami.com/images/img/how_to_guides/ecs-rds-tutorial/ecs-dashboard-1-0c97dd29.png)

Fill in the cluster configuration:

*Cluster name* : express-app

*Key pair* : YOUR\_KEY\_PAIR

|  |
| --- |
| NOTE: You must have the PEM file for the selected key pair or you won't be able to complete the tutorial. If you don't have the PEM file, create a key pair through the EC2 console at <https://console.aws.amazon.com/ec2/v2/home#KeyPairs:sort=keyName>and save the resulting PEM file. |

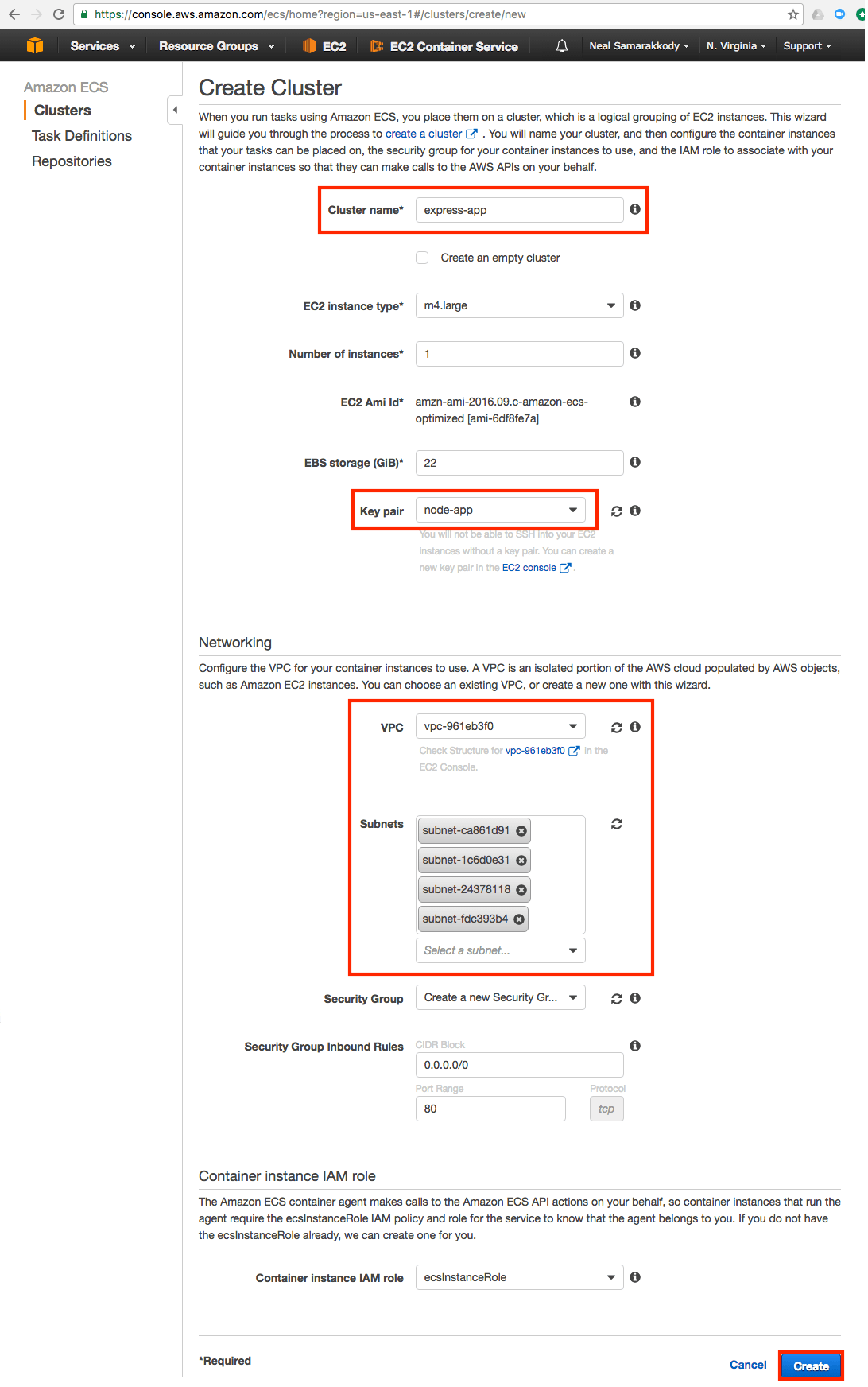
Fill in the *Networking* section:

VPC: YOUR\_DB\_VPC

|  |
| --- |
| NOTE: To get YOUR\_DB\_VPC, open <https://console.aws.amazon.com/rds/home?#dbinstances:>in a different browser tab, select the *express-app* DB Instance, click *Instance Actions* , click *See Details* , and find the VPC field in the *Security and Network* section. |

*Subnets* : Select all subnets

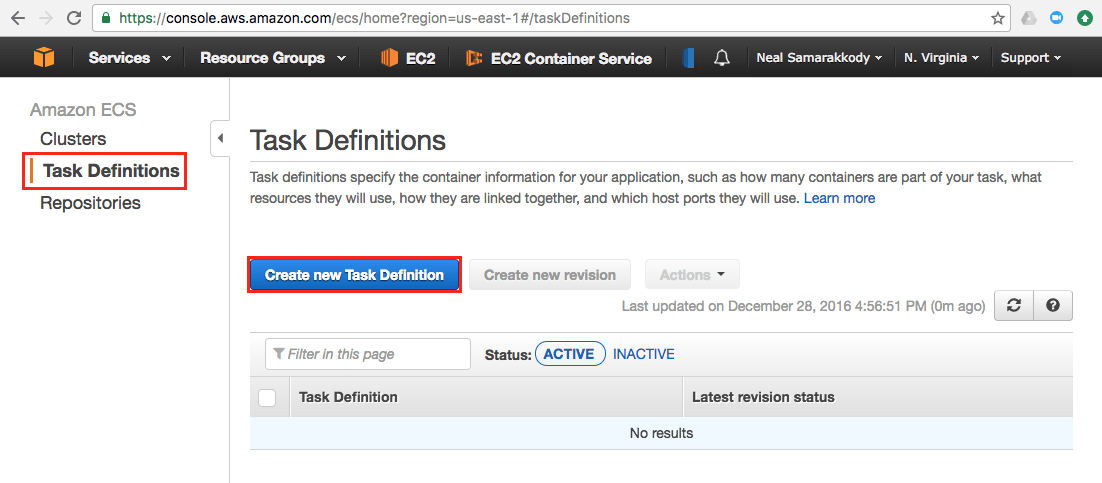
Click *Create*

[](https://docs.bitnami.com/images/img/how_to_guides/ecs-rds-tutorial/create-cluster-8e28fd19.png)

Now that the cluster exists, let's define a container for it to run.

Click *Task Definitions*

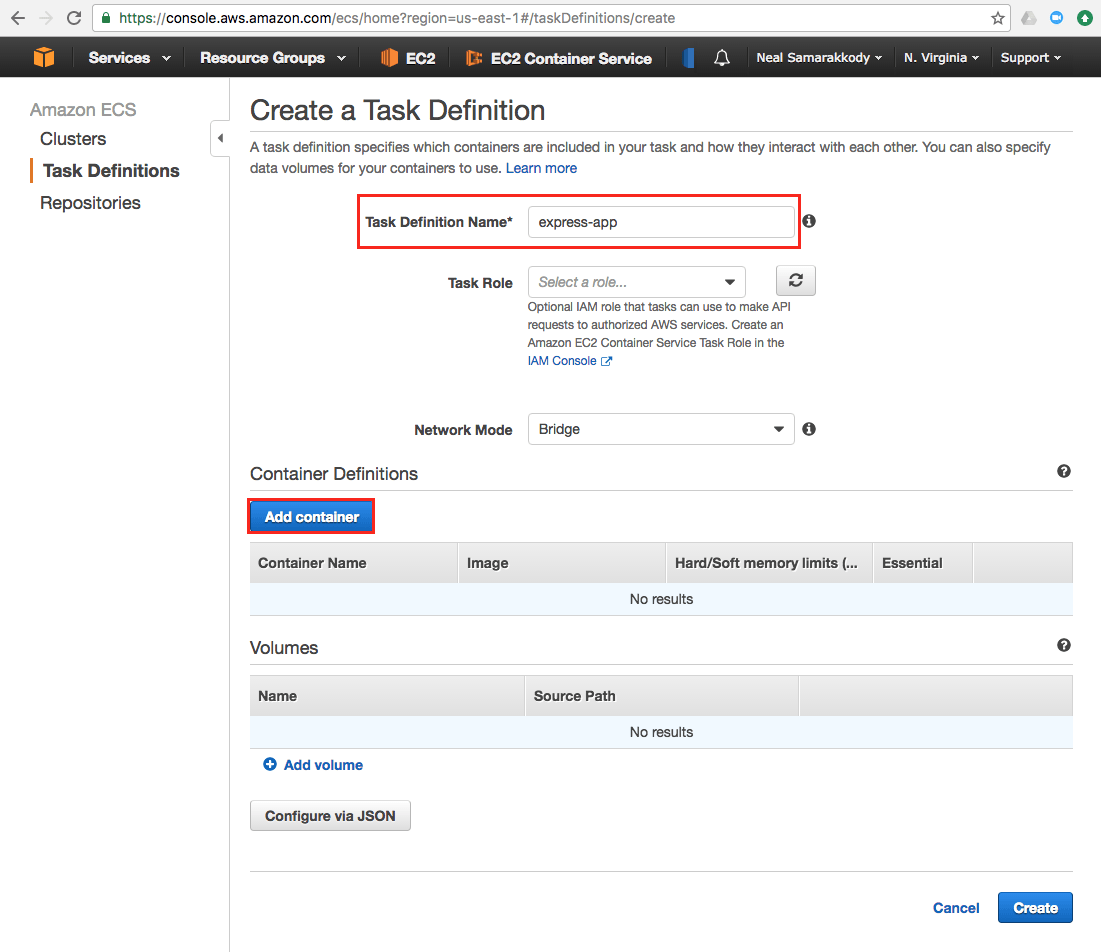
Click *Create new Task Definition*

[](https://docs.bitnami.com/images/img/how_to_guides/ecs-rds-tutorial/task-definitions-7f873d1c.png)

Fill in the task configuration:

*Task Definition Name* : express-app

Click *Add container*

[](https://docs.bitnami.com/images/img/how_to_guides/ecs-rds-tutorial/create-task-1-4e706fcc.png)

Fill in the container configuration:

*Container name* : express-app

*Image* : YOUR\_DOCKER\_HUB\_USER/express-app

*Memory Limits* : Soft limit - 300

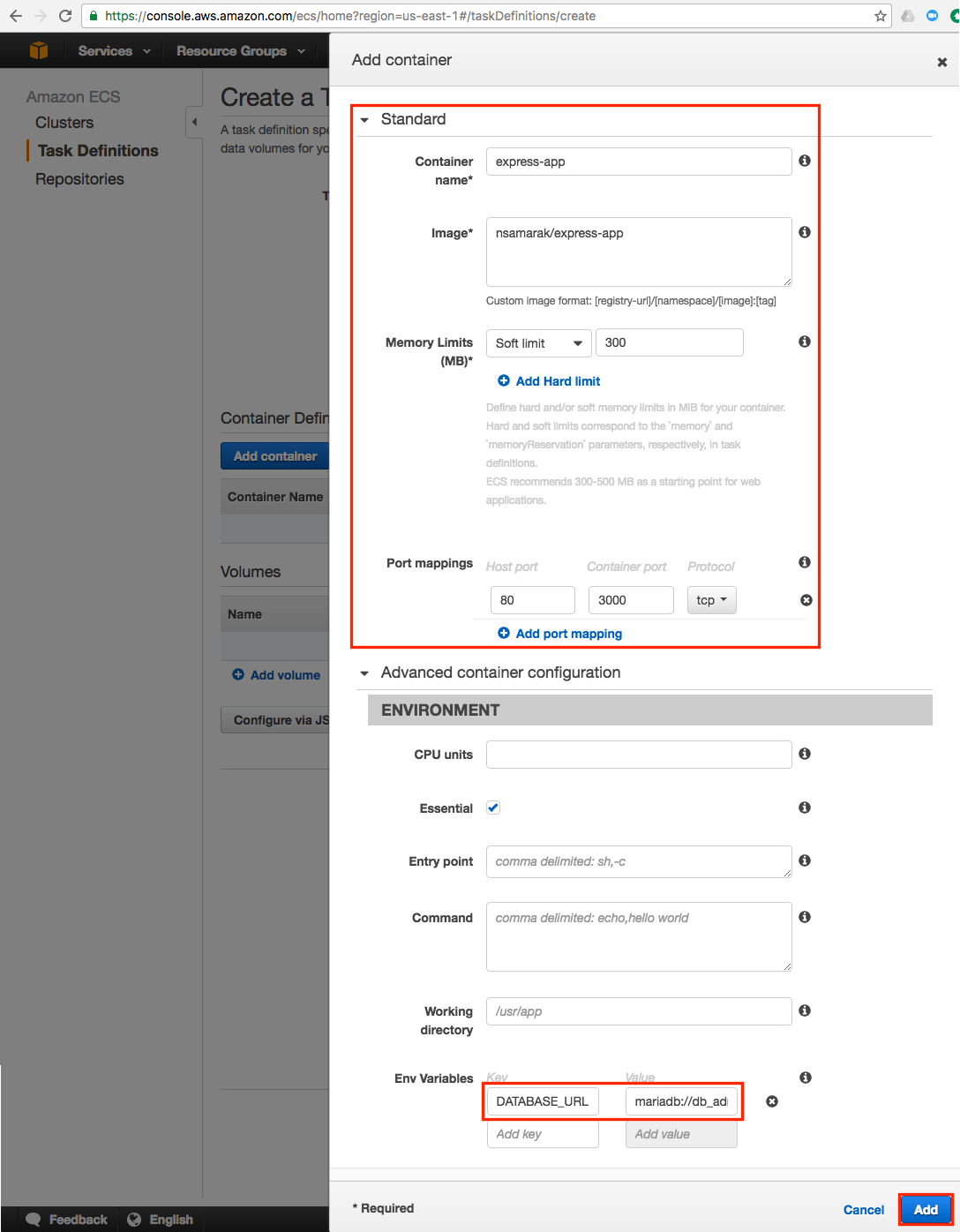
*Port mappings* : host port - 80, container port - 3000, protocol - tcp

Add an env variable in the *Advanced container configuration* section:

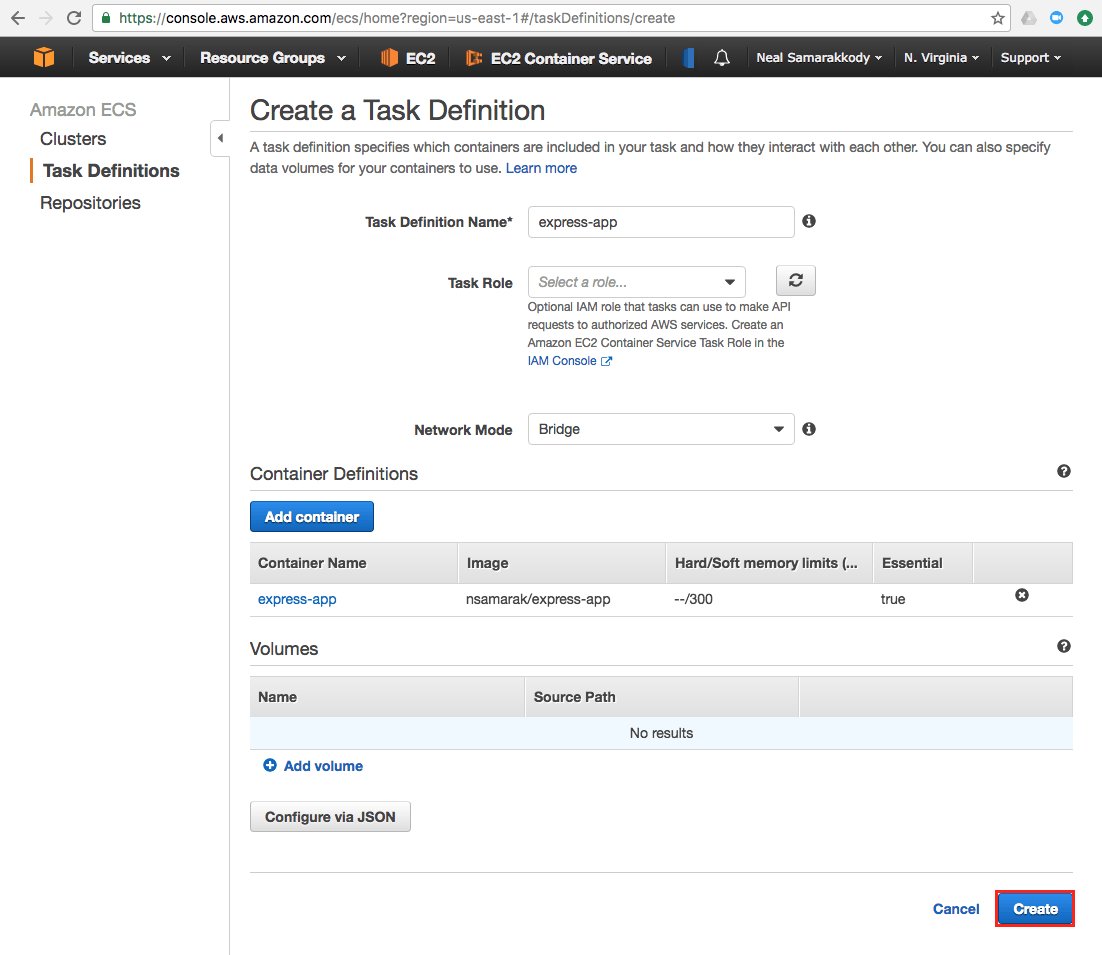
DATABASE\_URL: mariadb://db\_admin:YOUR\_DB\_PASSWORD@YOUR\_DB\_HOST:3306/express\_app?ssl=Amazon+RDS

|  |
| --- |
| NOTE: To get YOUR\_DB\_HOST, open <https://console.aws.amazon.com/rds/home?#dbinstances:>in a different browser tab, select the *express-app* DB Instance, click *Instance Actions* , click *See Details* , and find the *Publicly Accessible Endpoint* field in the *Security and Network* section. Save the hostname so you don't have to look it up again later in the tutorial. |

Click *Add*

[](https://docs.bitnami.com/images/img/how_to_guides/ecs-rds-tutorial/add-container-ecf44870.png)

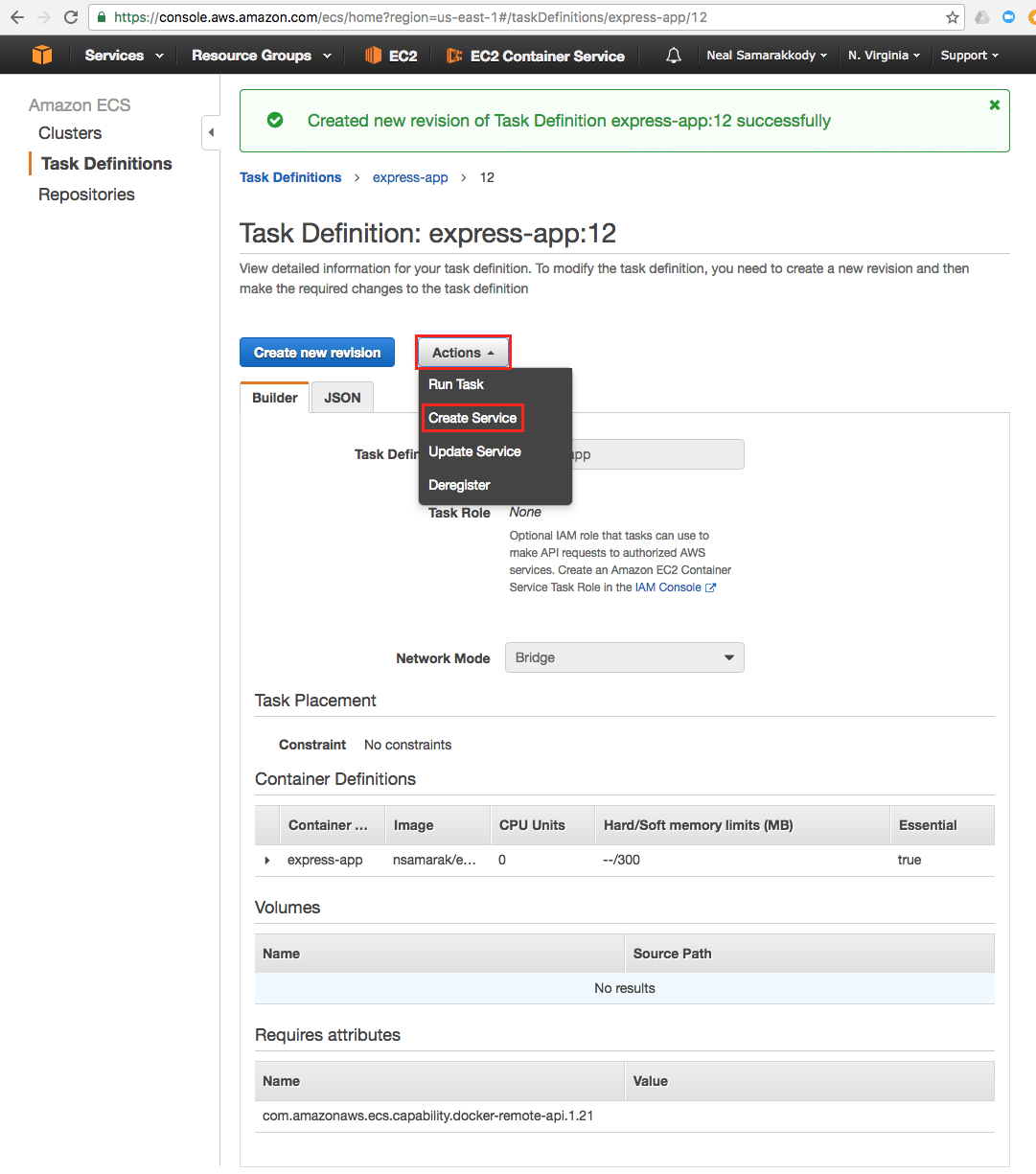
Click *Create*

[](https://docs.bitnami.com/images/img/how_to_guides/ecs-rds-tutorial/create-task-2-2b622052.png)

We need a service to aggregate containers and allow them to be run in a cluster. So let's do that next.

Click *Actions*

Click *Create Service*

[](https://docs.bitnami.com/images/img/how_to_guides/ecs-rds-tutorial/task-actions-62386b29.png)

Enter the following service configuration:

*Task Definition* : express-app

*Cluster* : express-app

*Service name* : express-app

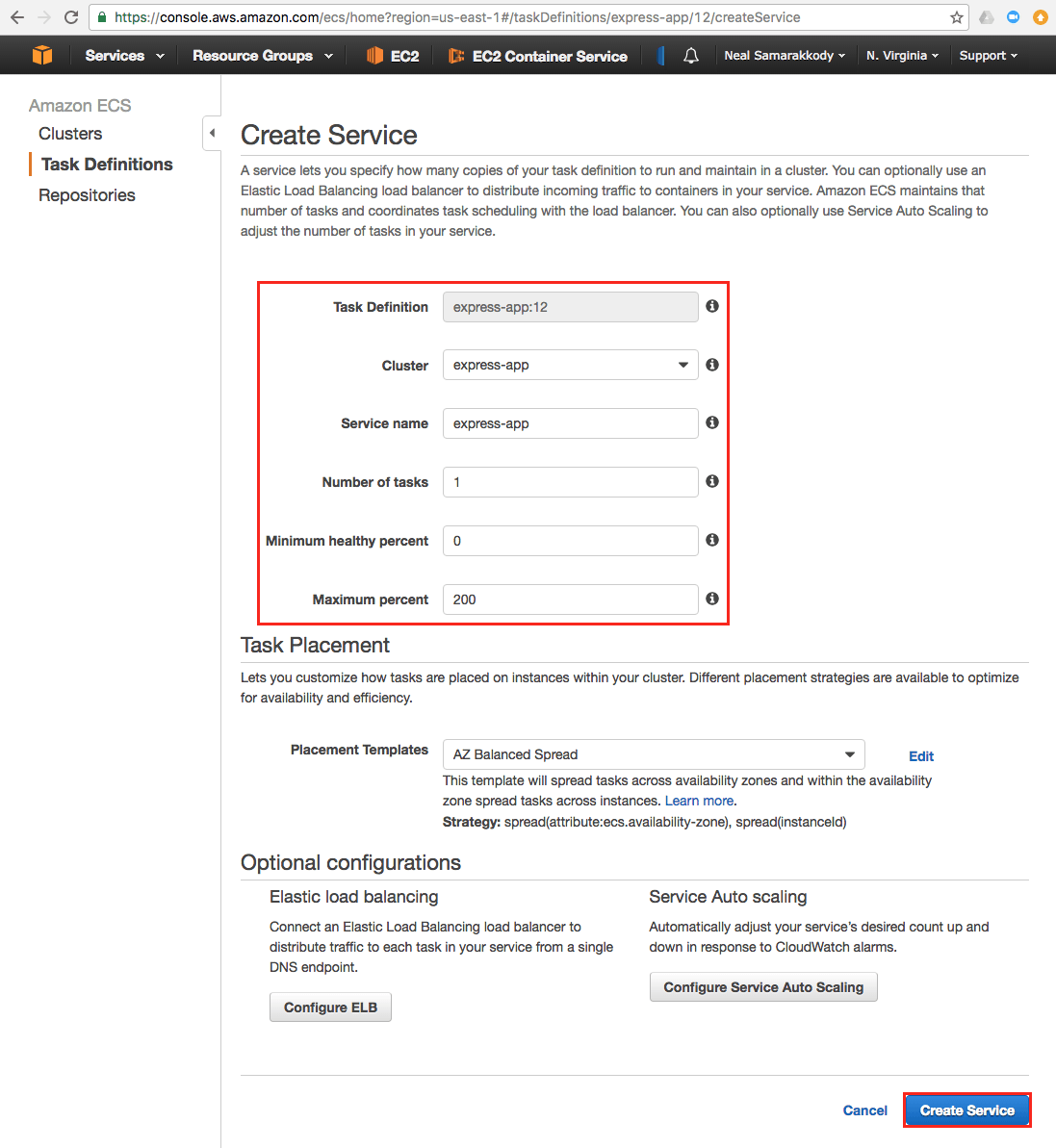
*Number of tasks* : 1

*Minimum healthy percent* : 0

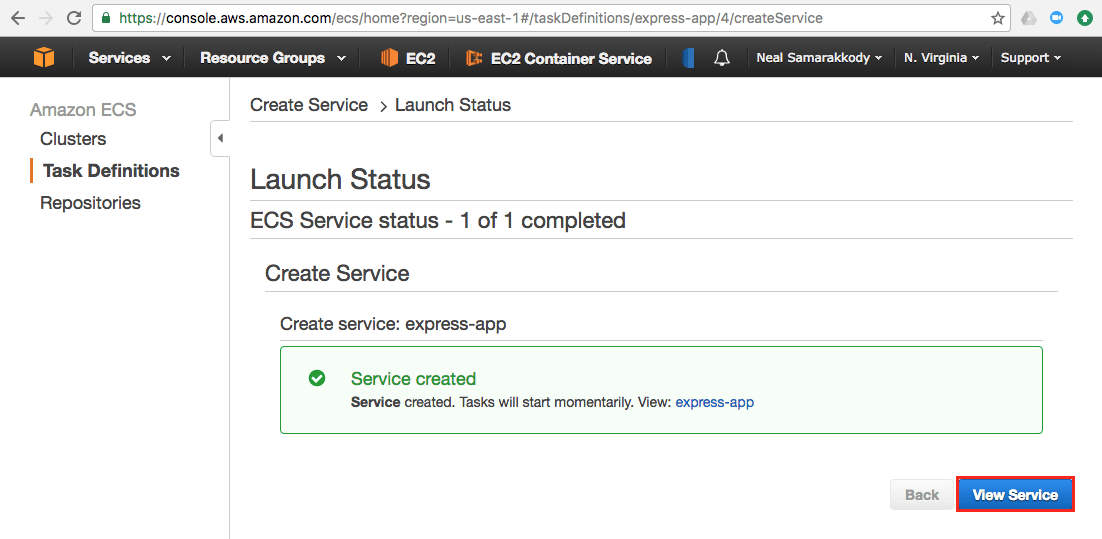
*Maximum percent* : 200

|  |
| --- |
| NOTE: The *Minimum healthy percent* should be low enough and the *Maximum percent* high enough that the cluster is able to do a rolling update of containers while maintaining the configured \*Number of tasks\_. Otherwise, you won't be able to update your app on prod. |

Click *Create Service*

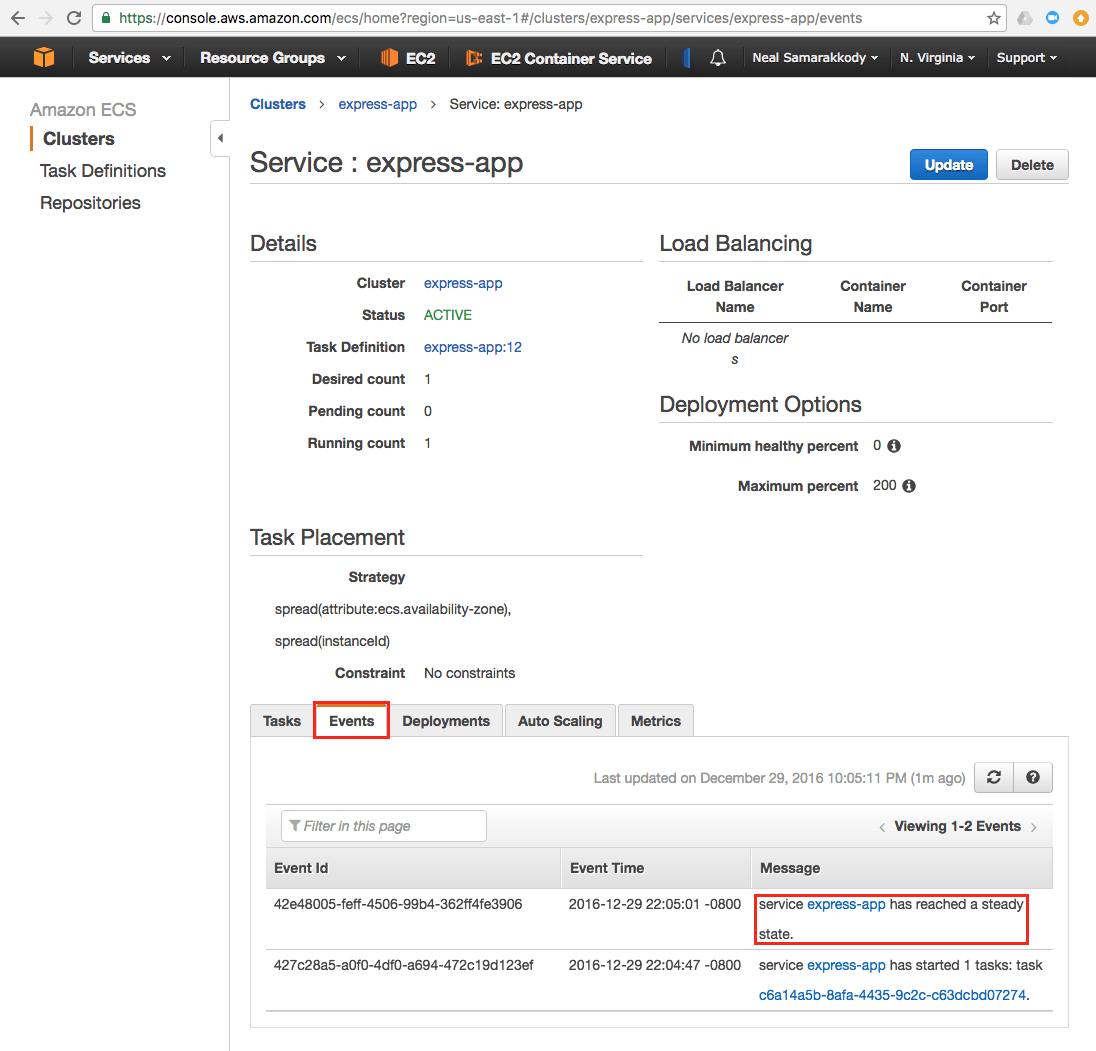
[](https://docs.bitnami.com/images/img/how_to_guides/ecs-rds-tutorial/create-service-c76ae90a.png)

Click *View Service*

[](https://docs.bitnami.com/images/img/how_to_guides/ecs-rds-tutorial/create-service-confirmation-6fb4fbf7.png)

Click the *Events* tab

Refresh periodically until you see message: "service express-app has reached a steady state." Now our cluster is running.

[](https://docs.bitnami.com/images/img/how_to_guides/ecs-rds-tutorial/service-events-6a2e2fa6.png)

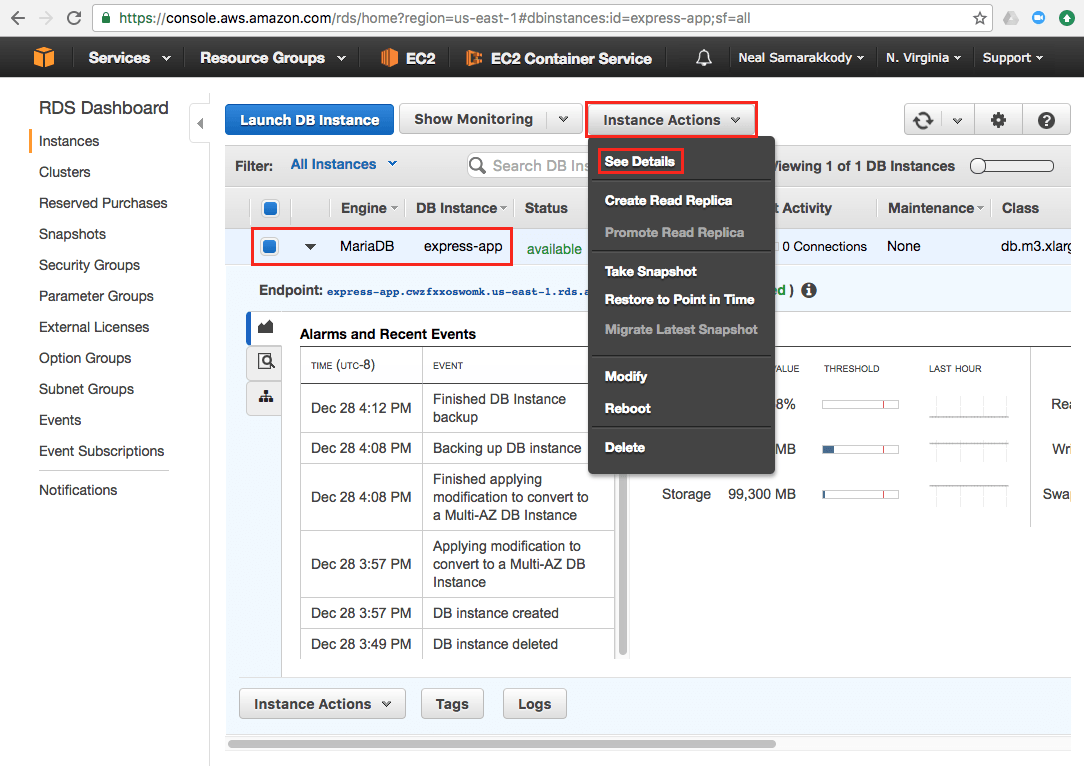
Next we need to ensure there is a network route that allows our prod cluster to talk to our prod database.

Navigate to <https://console.aws.amazon.com/rds/home?#dbinstances:>

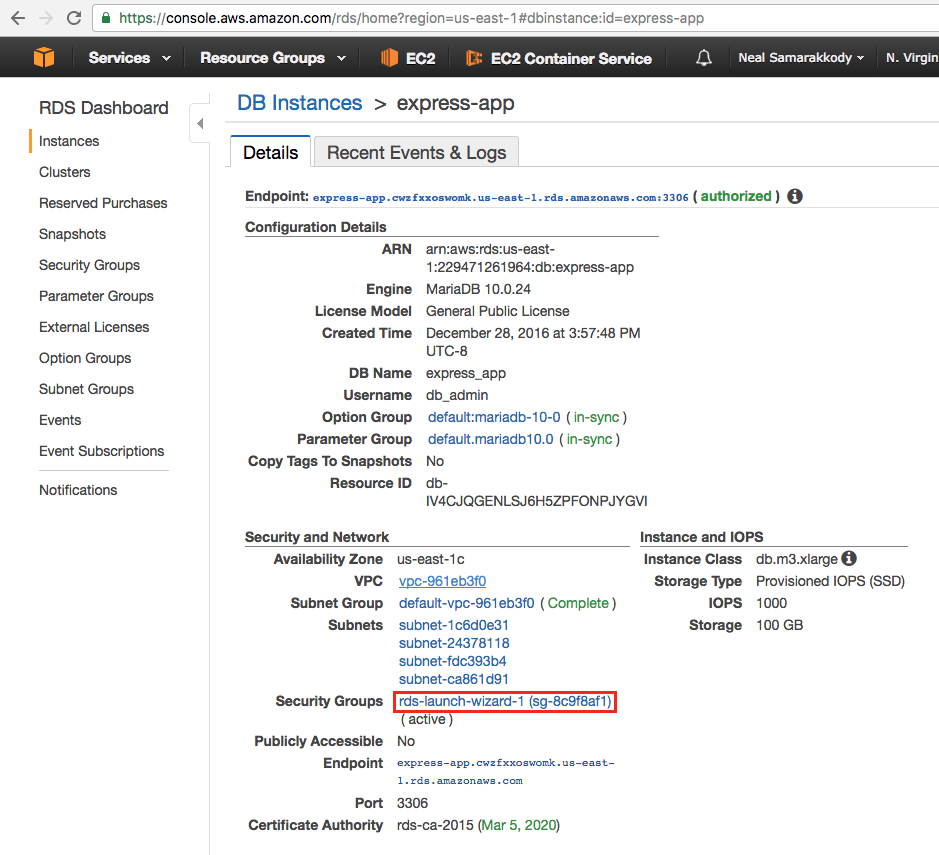
Select the *express-app* DB Instance

Click *Instance Actions*

Click *See Details*

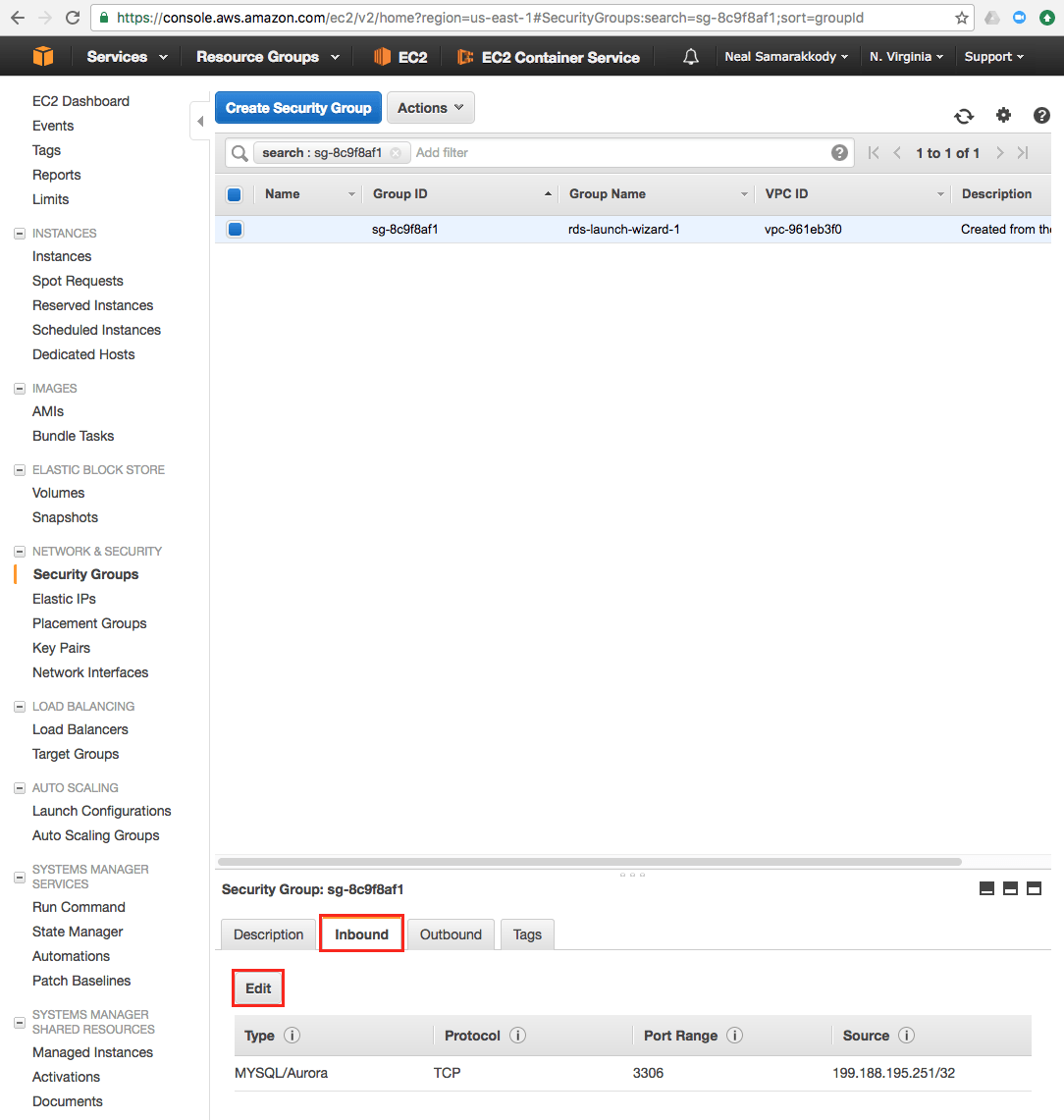
[](https://docs.bitnami.com/images/img/how_to_guides/ecs-rds-tutorial/prod-db-instance-actions-491f612d.png)

Click on the security group ID under *Security and Network*

[](https://docs.bitnami.com/images/img/how_to_guides/ecs-rds-tutorial/prod-db-details-d67ddf12.png)

Click *Inbound*

Click *Edit*

[](https://docs.bitnami.com/images/img/how_to_guides/ecs-rds-tutorial/prod-db-security-group-cee926c6.png)

Ensure there is only 1 inbound rule and that it is:

*Type* : MYSQL/Aurora

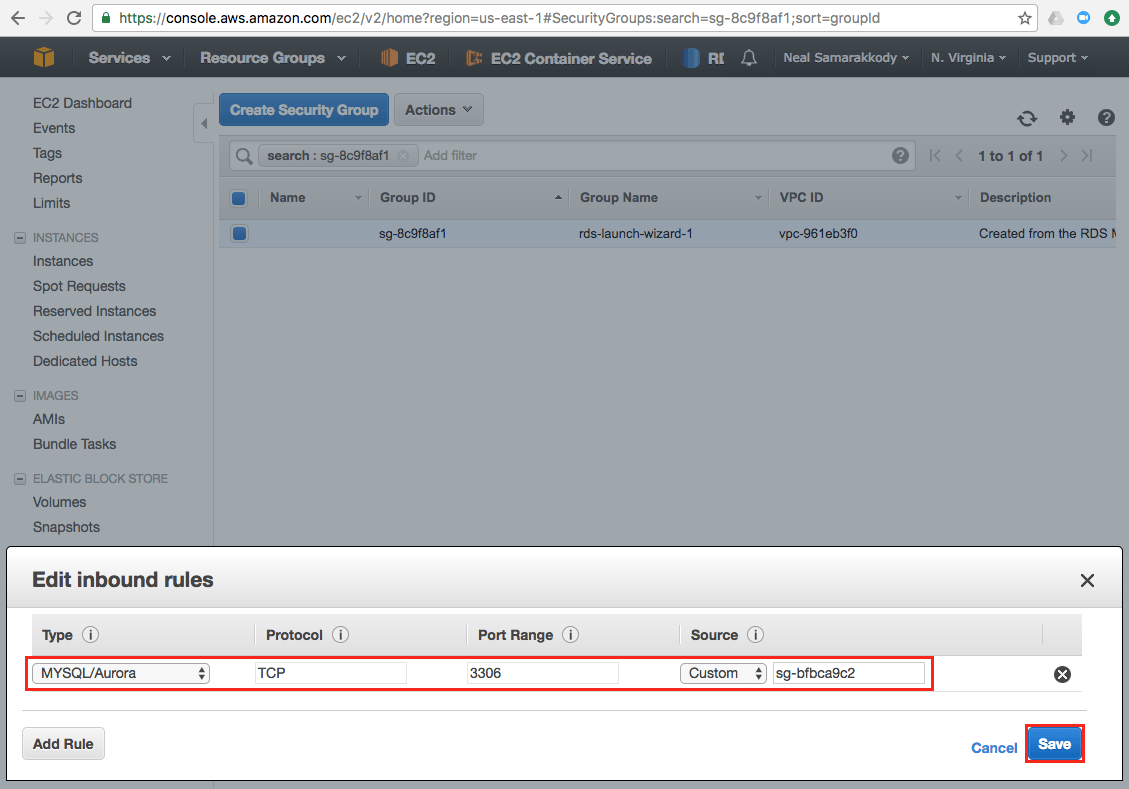
*Protocol* : TCP

*Port Range* : 3306

*Source* : Custom / YOUR\_ECS\_SECURITY\_GROUP

|  |
| --- |
| NOTE: To enter YOUR\_ECS\_SECURITY\_GROUP, start typing *express-app* into that field and then select the auto-completion option that pops up. |

Click *Save*

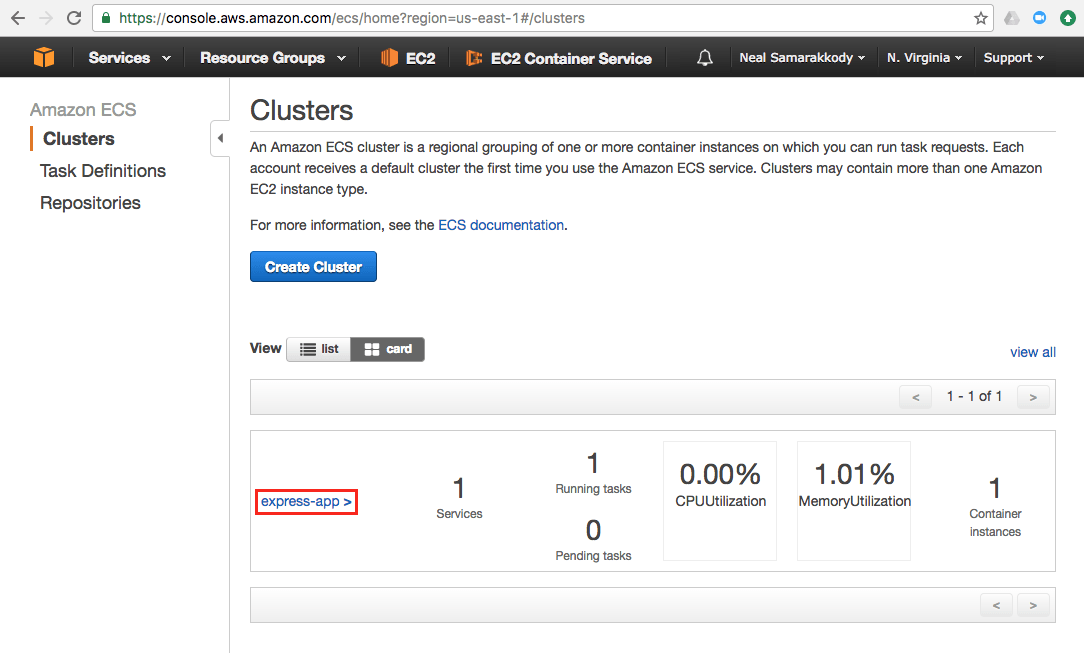
[](https://docs.bitnami.com/images/img/how_to_guides/ecs-rds-tutorial/prod-db-inbound-rules-f6eca1d5.png)

In order for our app to work, we need to populate our prod database so we have data to query from our app. You can do this by SSH'ing to our ECS cluster and running the *mysql* command-line client. We need to open up the firewall on the ECS cluster to allow SSH access and install the *mysql* client beforehand.

To enable SSH access, do the following:

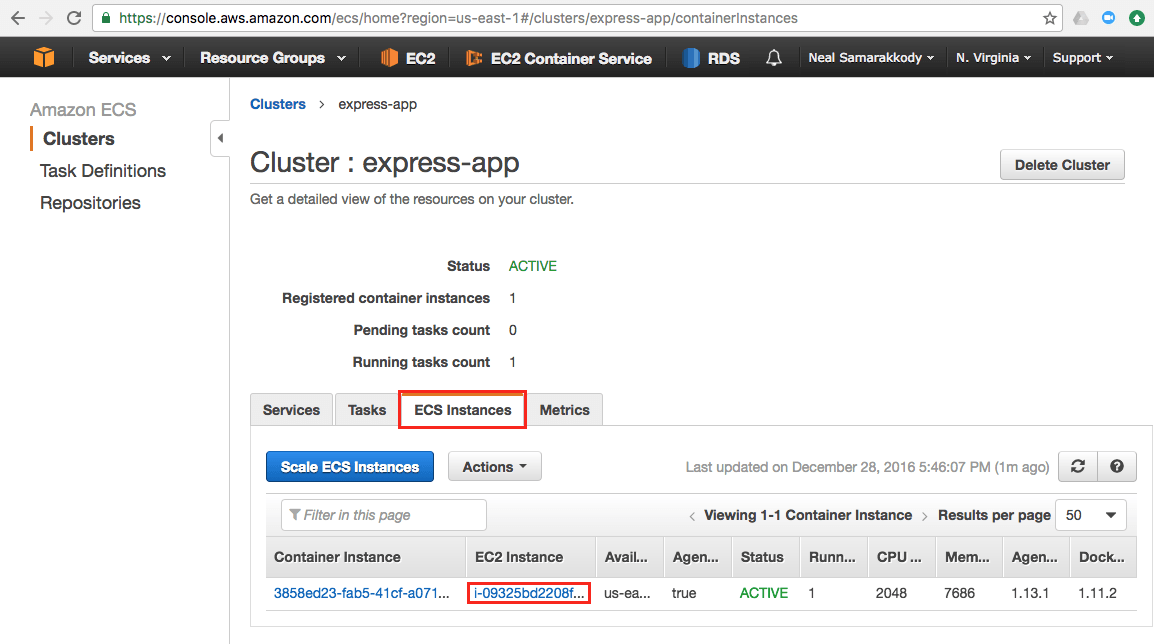
Navigate to <https://console.aws.amazon.com/ecs/home#/clusters>

Select the *express-app* cluster

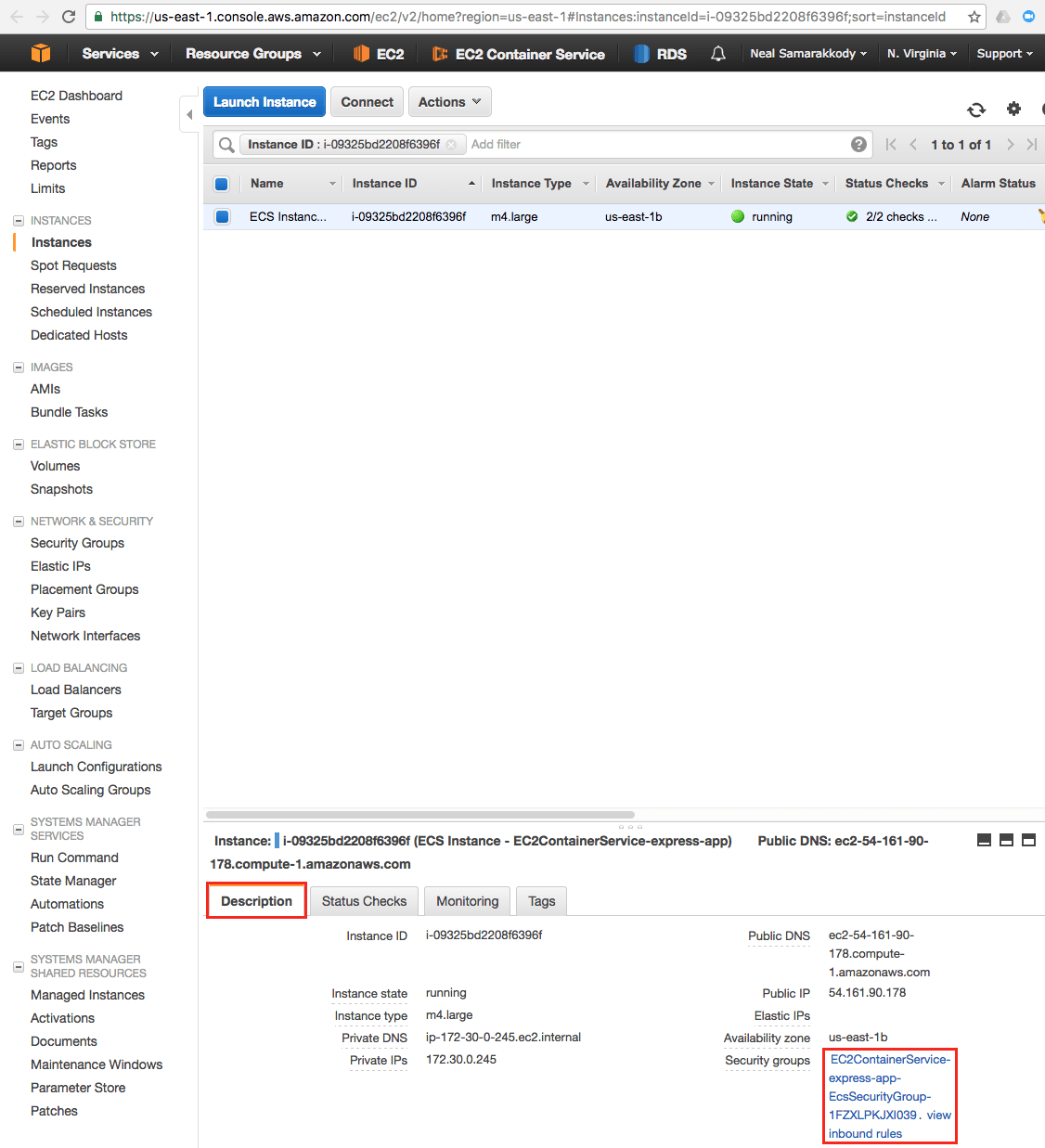
[](https://docs.bitnami.com/images/img/how_to_guides/ecs-rds-tutorial/ecs-dashboard-2-5b7147ca.png)

Click on the *ECS Instances* tab

Click on the EC2 instance ID

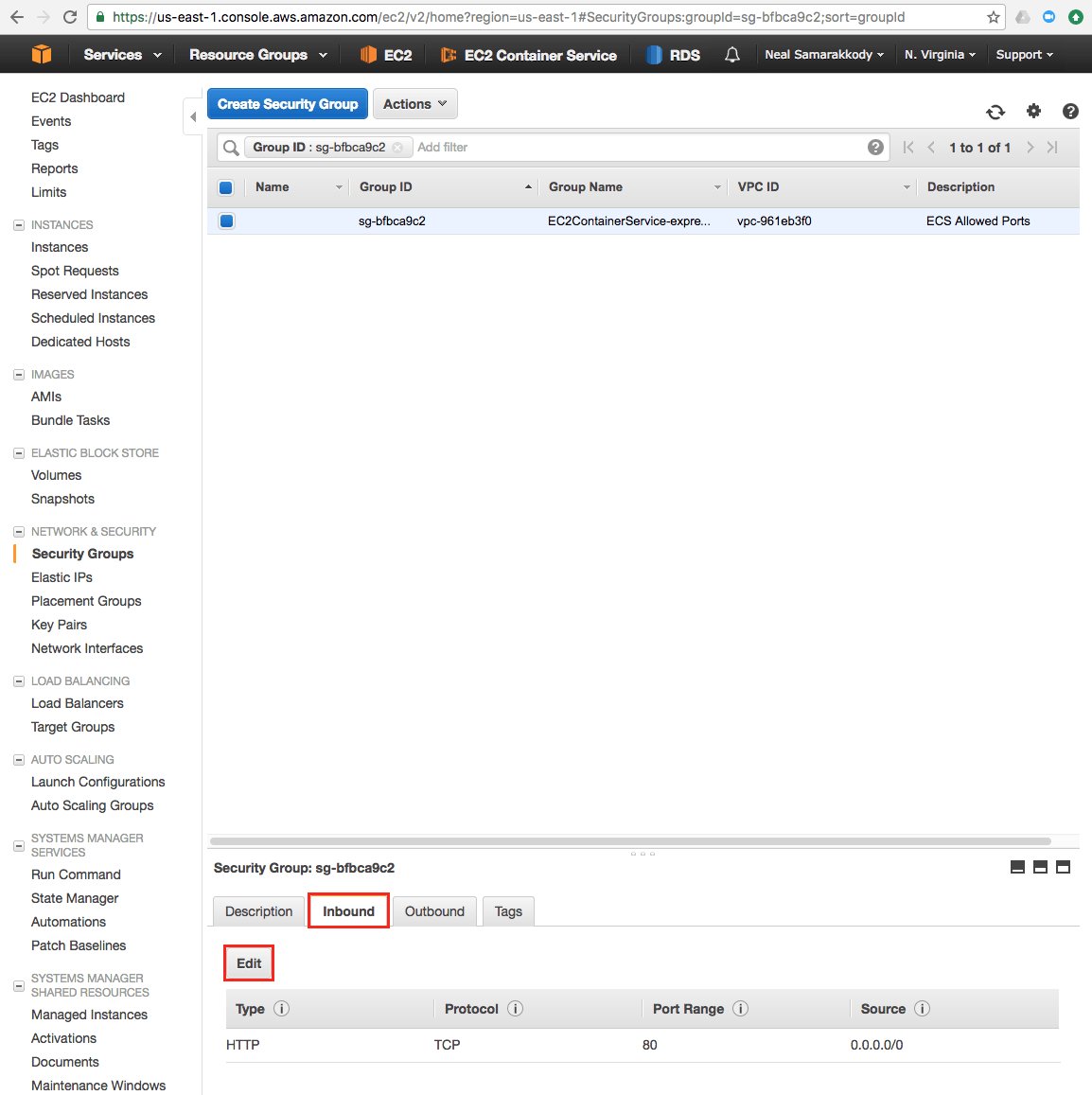
[](https://docs.bitnami.com/images/img/how_to_guides/ecs-rds-tutorial/ecs-instances-in-cluster-d360ac80.png)

Click on the Security groups link in the *Description* tab

[](https://docs.bitnami.com/images/img/how_to_guides/ecs-rds-tutorial/ecs-instance-description-f450fe76.png)

Click *Inbound*

Click *Edit*

[](https://docs.bitnami.com/images/img/how_to_guides/ecs-rds-tutorial/ecs-instance-security-group-ec392e89.png)

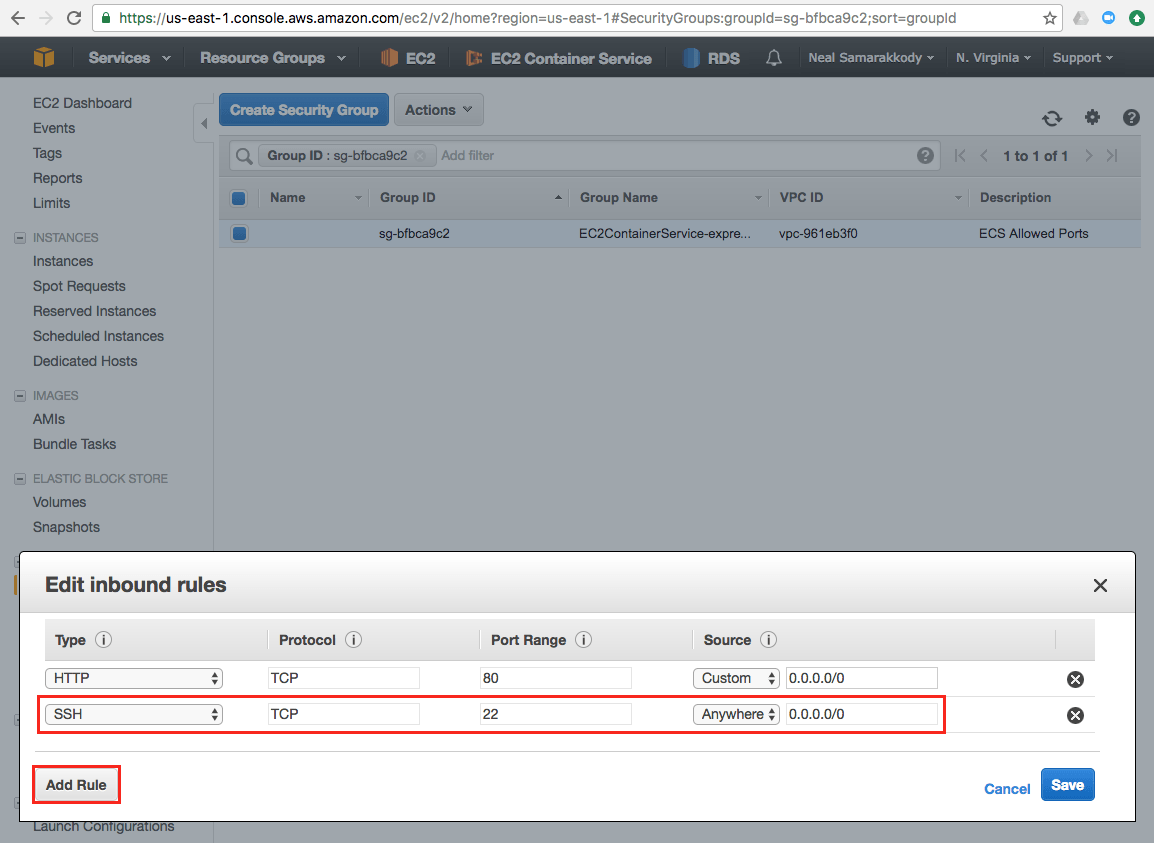
Click *Add Rule*

Fill in rule details:

*Type* : SSH

*Source* : Anywhere

Click *Save*

[](https://docs.bitnami.com/images/img/how_to_guides/ecs-rds-tutorial/ecs-instance-inbound-rules-08d69991.png)

Ensure that only your user has access to the PEM file on your local machine, since SSH won't use the file otherwise. On Linux and OS X, you can enforce permissions by doing:

chmod go-rwx <YOUR\_PEM\_FILE>

YOUR\_PEM\_FILE is the path to the PEM file that corresponds to your key pair.

Now you can SSH into the ECS cluster:

ssh -i <YOUR\_PEM\_FILE> ec2-user@<YOUR\_ECS\_HOST>

|  |
| --- |
| NOTE: To get YOUR\_ECS\_HOST, open <https://console.aws.amazon.com/ecs/home#/clusters>in a different browser tab, select the *express-app* cluster, click on the *ECS Instances* tab, click on the container instance ID, and find the *Public DNS* field. Save the hostname so you don't have to look it up again later in the tutorial. |

Install the *mysql* client.

sudo yum -y install mysql

And run the *mysql* client to modify our database.

mysql -h <YOUR\_DB\_HOST> -u db\_admin -p'<YOUR\_DB\_PASSWORD>' express\_app

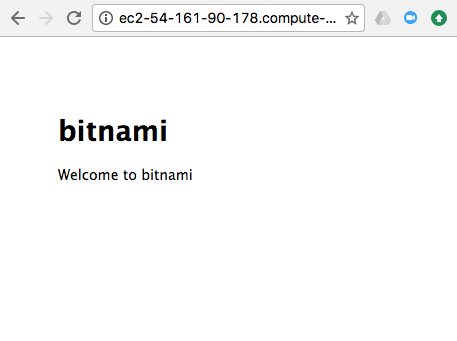
|  |
| --- |
| NOTE: To get YOUR\_DB\_HOST, open <https://console.aws.amazon.com/rds/home?#dbinstances:>in a different browser tab, select the *express-app* DB Instance, click *Instance Actions* , click *See Details* , and find the *Publicly Accessible Endpoint* field in the *Security and Network* section. |
| TIP: Don't forget the single quotes around the password. |

Once you're in the *mysql* shell, populate the database:

mysql> create table names (name varchar(15)); mysql> insert into names values ('bitnami'); mysql> exit

At this point, you should have a fully functional app on prod. Access it by visiting YOUR\_ECS\_HOST in a web browser.

|  |
| --- |
| NOTE: To get YOUR\_ECS\_HOST, open <https://console.aws.amazon.com/ecs/home#/clusters>in a different browser tab, select the *express-app* cluster, click on the *ECS Instances* tab, click on the container instance ID, and find the *Public DNS* field. |

[](https://docs.bitnami.com/images/img/how_to_guides/ecs-rds-tutorial/homepage-on-prod-dd47ca2b.png)

Congratulations! You've completed the tutorial. The remaining sections deal with periodically [updating the app that was just launched](https://docs.bitnami.com/aws/how-to/ecs-rds-tutorial/#appendix-a-update-expressjs-app) , [troubleshooting production issues](https://docs.bitnami.com/aws/how-to/ecs-rds-tutorial/#appendix-b-troubleshoot-expressjs-app-on-prod) , and if necessary, [deleting your AWS resources](https://docs.bitnami.com/aws/how-to/ecs-rds-tutorial/#appendix-c-remove-expressjs-app-from-prod) .

Appendix A: Update Express.js App

Any meaningful production application will need to be updated from time to time. Depending on the nature of the changes, we may need to update our database, application code, or both.

Appendix A.1: Update Database

If you made changes to your database schema or need to add data to your prod database, then those must be enacted on prod separately from an update to the code of the Express.js app.

SSH into the ECS cluster:

ssh -i <YOUR\_PEM\_FILE> ec2-user@<YOUR\_ECS\_HOST>

YOUR\_PEM\_FILE is the path to the PEM file that corresponds to your key pair.

|  |
| --- |
| NOTE: To get YOUR\_ECS\_HOST, open <https://console.aws.amazon.com/ecs/home#/clusters>in a different browser tab, select the *express-app* cluster, click on the *ECS Instances* tab, click on the container instance ID, and find the *Public DNS* field. |

Run the *mysql* client:

mysql -h <YOUR\_DB\_HOST> -u db\_admin -p'<YOUR\_DB\_PASSWORD>' express\_app

|  |
| --- |
| NOTE: To get YOUR\_DB\_HOST, open <https://console.aws.amazon.com/rds/home?#dbinstances:>in a different browser tab, select the *express-app* DB Instance, click *Instance Actions* , click *See Details* , and find the *Publicly Accessible Endpoint* field in the *Security and Network* section. |
| TIP: Don't forget the single quotes around the password. |

Once you're in the *mysql* shell, run DDL and DML statements to update your database.

Appendix A.2: Update Express.js Code

This section assumes you've made code changes to your app and need to test them locally and then deploy to prod.

Restart your dev environment to test your new code:

docker-compose restart

Navigate to [http://localhost:3000](http://localhost:3000/) to see the updated app.

Once you're happy with the changes, it's time to go to prod.

Navigate to the root directory of your app (this is where the *Dockerfile* is) and create an updated container image with the following command.

docker build -t <YOUR\_DOCKER\_HUB\_USER>/express-app .

|  |
| --- |
| TIP: Don't forget the period at the end of the docker build command. |

Push that image to Docker Hub so that Amazon ECS can access it for the prod deployment:

docker login docker push <YOUR\_DOCKER\_HUB\_USER>/express-app

Now we need to have the ECS cluster run the latest container image. Configure a new task with the latest container image and tell our service to use it.

Navigate to <https://console.aws.amazon.com/ecs/home#/taskDefinitions>

Click the checkbox for the express-app task

Click *Create new revision*

Click *Create*

Click *Actions*

Click *Update Service*

Click *Update Service* (again)

Click *View Service*

Click the *Events* tab

Refresh periodically until you see a message of "service express-app has reached a steady state." If you see a message that says "service express-app was unable to place a task", don't worry. This appears to be a side effect of ECS attempting to restart the cluster. Just keep refreshing for a couple of more minutes until you see the steady state message.

Now our cluster has been updated.

Access the updated app by visiting YOUR\_ECS\_HOST in a web browser.

|  |
| --- |
| NOTE: To get YOUR\_ECS\_HOST, open <https://console.aws.amazon.com/ecs/home#/clusters>in a different browser tab, select the *express-app* cluster, click on the *ECS Instances* tab, click on the container instance ID, and find the *Public DNS* field. |

Appendix B: Troubleshoot Express.js App On Prod

Appendix B.1: See If The Container Exited With A Useful Error

The ECS console displays the container's exit message.

Navigate to <https://console.aws.amazon.com/ecs/home#/clusters>

Select the *express-app* cluster

Click on the *Tasks* tab

Click *Stopped*

Click on the task ID for the task that corresponded to the failed container

Expand the containers listed in the *Containers* table

View errors in the *Details* section

Appendix B.2: Look At Error Logs For The Container

We can look at the error logs for the container by connecting to the ECS cluster and using Docker commands.

SSH to the cluster:

ssh -i <YOUR\_PEM\_FILE> ec2-user@<YOUR\_ECS\_HOST>

YOUR\_PEM\_FILE is the path to the PEM file that corresponds to your key pair.

|  |
| --- |
| NOTE: To get YOUR\_ECS\_HOST, open <https://console.aws.amazon.com/ecs/home#/clusters>in a different browser tab, select the *express-app* cluster, click on the *ECS Instances* tab, click on the container instance ID, and find the *Public DNS* field. |

Run the following to get a list of running containers:

docker ps -a

Find the container ID for the YOUR\_DOCKER\_HUB\_USER/express-app image that is having issues and do the following to see the logs for that container:

docker logs <CONTAINER\_ID>

You can find more info on ECS troubleshooting at <http://docs.aws.amazon.com/AmazonECS/latest/developerguide/troubleshooting.html>

Appendix C: Remove Express.js App From Prod

Appendix C.1: Delete Prod Database

Open <https://console.aws.amazon.com/rds/home?#dbinstances:>

Select the *express-app* DB instance

Click *Instance Actions*

Click *Delete*

Answer No for *Create final Snapshot?*

Click the checkbox for *I acknowledge…*

Click *Delete*

Refresh the page until the DB instance is completely deleted

Appendix C.2: Delete Database Snapshots

After your DB instance is deleted, navigate to <https://console.aws.amazon.com/rds/home#db-snapshots:>

Select the snapshots corresponding to the express-app DB instance

Click *Snapshot Actions*

Click *Delete Snapshot*

Click *Delete*

Appendix C.3: Delete Database Subnets

Navigate to <https://console.aws.amazon.com/rds/home#db-subnet-groups:>

Select the DB subnet group corresponding to the express-app DB instance

Click *Delete*

Click *Delete* (again)

Appendix C.4: Delete Database Parameter Groups

Navigate to <https://console.aws.amazon.com/rds/home#parameter-groups:>

Click the checkbox for the *default.mariadb10.0* parameter group

Click *Delete*

Click *Delete* (again)

Ignore the warning about not being able to delete the default parameter groups

Appendix C.5: Delete ECS Cluster

Navigate to <https://console.aws.amazon.com/ecs/home#/clusters>

Select the *express-app* cluster

Click *Delete Cluster*

Click *Delete*

Wait until the ECS cluster is completely deleted

Appendix C.6: Delete Security Groups

Navigate to <https://console.aws.amazon.com/ec2/v2/home#SecurityGroups:>

Search security groups based on your VPC ID

Select all security groups

Click *Actions*

Click *Delete Security Groups*

Click *Yes, Delete*

Ignore the warning about not being able to delete the default security groups

Appendix C.7: Deregister Tasks

Once the ECS cluster is completely deleted, navigate to <https://console.aws.amazon.com/ecs/home#/taskDefinitions>

Click into the *express-app* task

Select all task revisions

Click *Actions*

Click *Deregister*