Django Deployments Cookbook Documentation

Agiliq and Contributors

Feb 20, 2019

Table of Contents:

CHAPTER 1

Using Zappa deploy in Lambda & use Aurora Serverless

We will see how to *deploy* a Django application onto **AWS Lambda** using **Zappa** and use **AWS Aurora-Serverless** as the DB.

AWS Lambda is a serverless computing platform by amazon, which is completely event driven and it automatically manages the computing resources. It scales automatically when needed, depending upon the requests the application gets.

Zappa is a python framework used for deploying python applications onto AWS-Lambda. Zappa handles all of the configuration and deployment automatically for us.

And Aurora Serverless is an on-demand, auto-scaling Relational Database System by Amazon AWS(presently compatible with only MySQL). It automatically starts up & shuts down the DB depending on the requirement.

1.1 Install and Configure the Environment

1.1.1 Configure AWS Credentials

First, before using AWS, we have to make sure we have a valid AWS account and have the aws environment variables(access-keys).

then, create a folder at the root level

```
$ mkdir .aws
```

Now, create a file called credentials and store the `aws_access_key_id` and `aws_secret_access_key`. To find these access credentials

- Go to IAM dashboard in AWS console
- Click on Users
- Click on your User name
- Then, go to Security credentials tab

- Go down to Access keys
- Note down the `access_key_id`. `secret_access_key` is only visible when you are creating new user or when creating a new access key, so you need to note down both the access_key_id and secret_access_key at the time of user creation only or create a new access key so that we can get both the keys.

```
###~/.aws/credentials
[default]
aws_access_key_id= XXXXXXXXXXXXXXXXXXXX
aws_secret_access_key=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

1.1.2 Go to Django app

After setting up the aws credentials file, now let us go to the django project, here we used *Pollsapi* (https://github.com/agiliq/building-api-django) as the django project. Now go inside the *pollsapi* app in this repo.

Create a virtual env for the project and do `$ pip install -r requirements.txt`.

1.1.3 Install & Configure Zappa

Next install zappa

```
$ pip install zappa
```

After installing Zappa, let us initilise zappa

```
$ zappa init
```

which will ask us for the following:

- Name of environment - default 'dev'
- S3 bucket for deployments. If the bucket does not exist, zappa will create it for us. Zappa uses this bucket to hold the zappa package temporarily while it is being transferred to AWS lambda, which is then deleted after deployment.

(Its better to create an S3 bucket, which we will later also use to host the static files of our application)

- Project's settings - (which will take the 'pollsapi.settings')

Zappa will automatically find the correct Django settings file and the python runtime version

```
$ zappa init








Welcome to Zappa!

Zappa is a system for running server-less Python web applications on AWS Lambda and␣
↪AWS API Gateway.
This `init` command will help you create and configure your new Zappa deployment.
Let's get started!
```

(continues on next page)

(continued from previous page)

```

Your Zappa configuration can support multiple production stages, like 'dev', 'staging
↪', and 'production'.
What do you want to call this environment (default 'dev'):

AWS Lambda and API Gateway are only available in certain regions. Let's check to make␣
↪sure you have a profile set up inone that will work.
Okay, using profile default!

Your Zappa deployments will need to be uploaded to a private S3 bucket.
If you don't have a bucket yet, we'll create one for you too.
What do you want to call your bucket? (default 'zappa-xpxpcmpap'):zappa-
↪staticfiles1234

It looks like this is a Django application!
What is the module path to your projects's Django settings?
We discovered: pollsapi.settings
Where are your project's settings? (default 'pollsapi.settings'):

You can optionally deploy to all available regions in order to provide fast global␣
↪service.
If you are using Zappa for the first time, you probably don't want to do this!
Would you like to deploy this application globally? (default 'n') [y/n/(p)rimary]: n

Okay, here's your zappa_settings.json:

{
    "dev": {
        "django_settings": "pollsapi.settings",
        "profile_name": "default",
        "project_name": "pollsapi",
        "runtime": "python3.6",
        "s3_bucket": "zappa-staticfiles1234"
    }
}

Does this look okay? (default 'y') [y/n]: y
```

After accepting the info. A file zappa_settings.json gets created which looks like

```
{
  "dev": {
    "django_settings": "pollsapi.settings",
    "profile_name": "default",
    "project_name": "pollsapi",
    "runtime": "python3.6",
    "s3_bucket": "zappa-staticfiles1234"
  }
}
```

Now, before deploying we have to mention the aws_region(where we want ot deploy the django app). Make sure that you have the s3_bucket and aws_region in the same region.

```
{
  "dev": {
    "django_settings": "pollsapi.settings",
```

(continues on next page)

(continued from previous page)

```
    "profile_name": "default",
    "project_name": "pollsapi",
    "runtime": "python3.6",
    "s3_bucket": "zappa-staticfiles1234",

    "aws_region": "us-east-2" // aws_region
  }
}
```

Now let us deploy the app

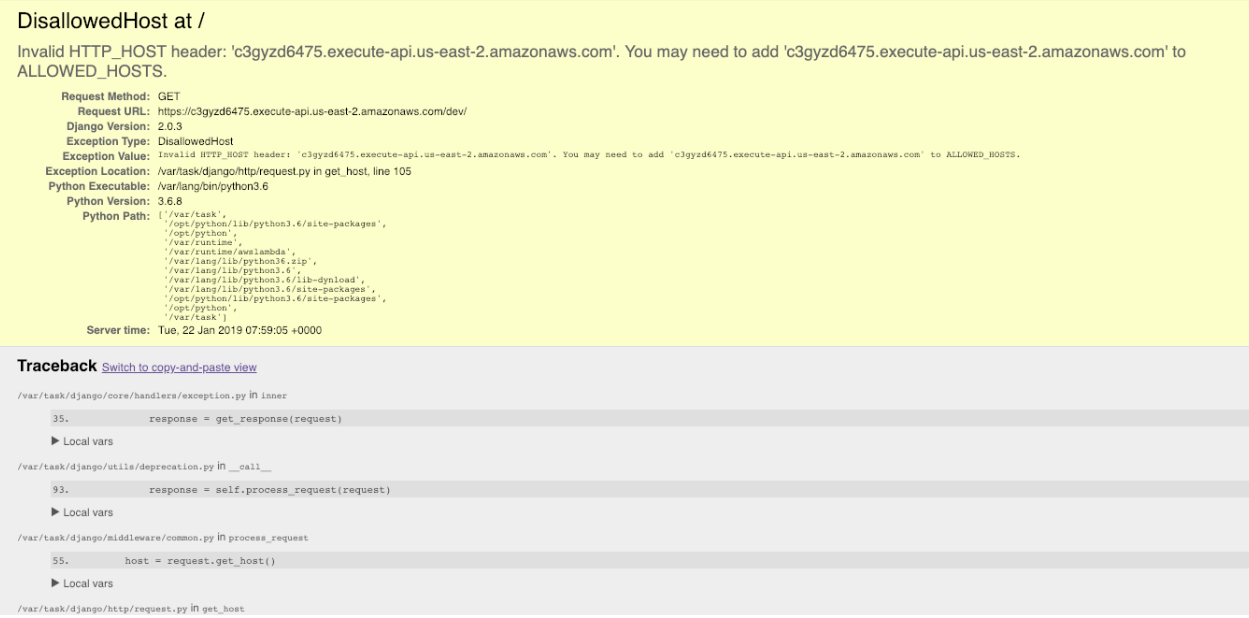
```
$ zappa deploy dev
```

which will show us

```
$ zappa deploy dev

Calling deploy for stage dev..
Downloading and installing dependencies..
 - markupsafe==1.1.0: Using locally cached manylinux wheel
 - sqlite==python36: Using precompiled lambda package
Packaging project as zip.
Uploading pollsapi-dev-1548143620.zip (36.2MiB)..
100%|| 37.9M/37.9M [00:14<00:00, 2.69MB/s]
Scheduling..
Scheduled pollsapi-dev-zappa-keep-warm-handler.keep_warm_callback with expression rate(4 minutes)!
Uploading pollsapi-dev-template-1548143703.json (1.6KiB)..
100%|| 1.61K/1.61K [00:00<00:00, 3.40KB/s]
Waiting for stack pollsapi-dev to create (this can take a bit)..
100%|| 4/4 [00:10<00:00,  2.72s/res]
Deploying API Gateway..
Deployment complete!: https://1astmowyfc.execute-api.us-east-2.amazonaws.com/dev
```

Now, when we click on the link we will see this



DisallowedHost at /

Invalid HTTP_HOST header: 'c3gyzd6475.execute-api.us-east-2.amazonaws.com'. You may need to add 'c3gyzd6475.execute-api.us-east-2.amazonaws.com' to ALLOWED_HOSTS.

```
   Request Method: GET
      Request URL: https://c3gyzd6475.execute-api.us-east-2.amazonaws.com/dev/
   Django Version: 2.0.3
   Exception Type: DisallowedHost
  Exception Value: Invalid HTTP_HOST header: 'c3gyzd6475.execute-api.us-east-2.amazonaws.com'. You may need to add 'c3gyzd6475.execute-api.us-east-2.amazonaws.com' to ALLOWED_HOSTS.
Exception Location: /var/task/django/http/request.py in get_host, line 105
Python Executable: /var/lang/bin/python3.6
   Python Version: 3.6.8
      Python Path: ['/var/task',
                    '/opt/python/lib/python3.6/site-packages',
                    '/opt/python',
                    '/var/runtime',
                    '/var/runtime/awslambda',
                    '/var/lang/lib/python36.zip',
                    '/var/lang/lib/python3.6',
                    '/var/lang/lib/python3.6/lib-dynload',
                    '/var/lang/lib/python3.6/site-packages',
                    '/opt/python/lib/python3.6/site-packages',
                    '/opt/python',
                    '/var/task']
      Server time: Tue, 22 Jan 2019 07:59:05 +0000
```

Traceback Switch to copy-and-paste view

```
/var/task/django/core/handlers/exception.py in inner

    35.             response = get_response(request)

    ▶ Local vars

/var/task/django/utils/deprecation.py in __call__

    93.             response = self.process_request(request)

    ▶ Local vars

/var/task/django/middleware/common.py in process_request

    55.         host = request.get_host()

    ▶ Local vars

/var/task/django/http/request.py in get_host
```

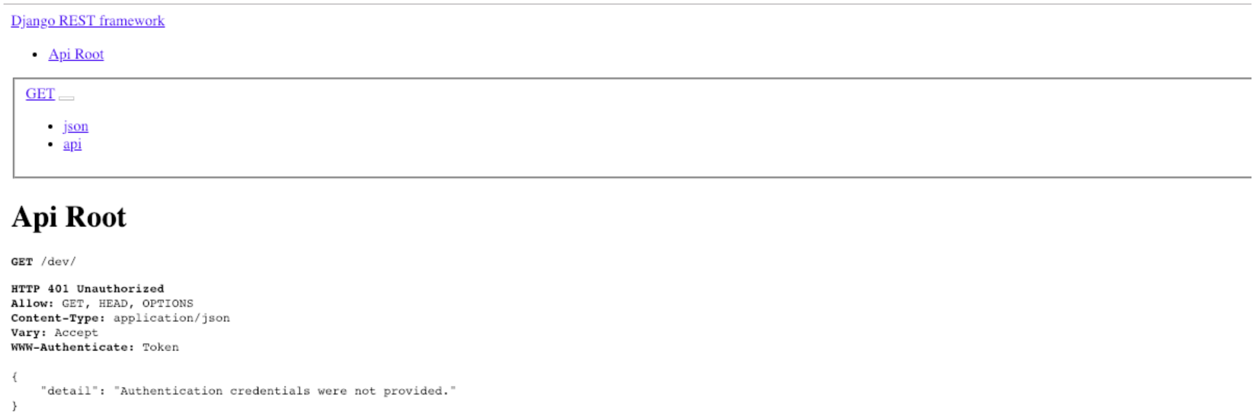
So, we will add the host to our to our ALLOWED_HOSTS in `pollsapi/settings.py`

```
ALLOWED_HOSTS = [ '127.0.0.1', '1astmowyfc.execute-api.us-east-2.amazonaws.com', ]
```

After this, we have update zappa,

```
$ zappa update dev
```

and after updating the app when we refresh the page we see,



Api Root

```
GET /dev/

HTTP 401 Unauthorized
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept
WWW-Authenticate: Token

{
    "detail": "Authentication credentials were not provided."
}
```

The Static files are not available !!

1.2 Serving Static Files

For serving static files we use S3 bucket(which we have created earlier).

We have to enable **CORS** for the S3 bucket, which enables browsers to get resources/files from different urls. Go to S3 Bucket properties and then to Permissions, and click `CORS Configuration`, and paste these lines

```
 <CORSConfiguration>
        <CORSRule>
            <AllowedOrigin>*</AllowedOrigin>
            <AllowedMethod>GET</AllowedMethod>
            <MaxAgeSeconds>3000</MaxAgeSeconds>
            <AllowedHeader>Authorization</AllowedHeader>
        </CORSRule>
</CORSConfiguration>
```

1.2.1 Configure Django for S3

```
$ pip install django-s3-storage
```

and also add it in the `requirements.txt` file.

```
...
django-s3-storage==0.12.4
...
```

Now update the *settings.py* file to add *'django*s3_storage'_ to INSTALLED_APPS

```
INSTALLED_APPS = (
            ...,
            'django_s3_storage',
      )
```

and also add these lines at the bottom

```
S3_BUCKET = "zappa-staticfiles1234"

STATICFILES_STORAGE = "django_s3_storage.storage.StaticS3Storage"

AWS_S3_BUCKET_NAME_STATIC = S3_BUCKET

STATIC_URL = "https://%s.s3.amazonaws.com/" % S3_BUCKET
```

Push the static files to the cloud

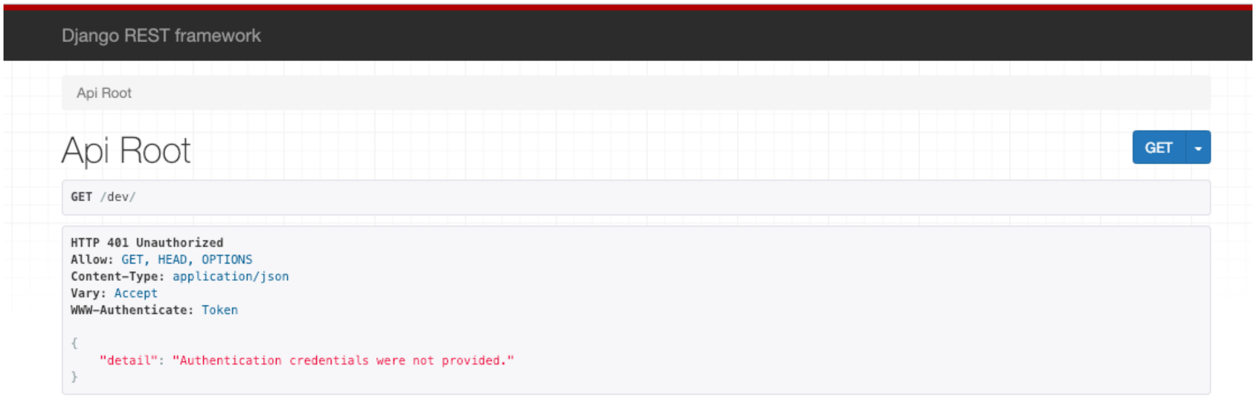
we can push the static files by

```
$ python manage.py collectstatic --noinput
```

and do

```
$ zappa update dev
```

and after updating zappa, let us check by refreshing the page



```
GET /dev/
```

```
HTTP 401 Unauthorized
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept
WWW-Authenticate: Token

{
    "detail": "Authentication credentials were not provided."
}
```

1.3 Setup Serverless MySQL Database

Let us create an AWS Aurora MySQL serverless.

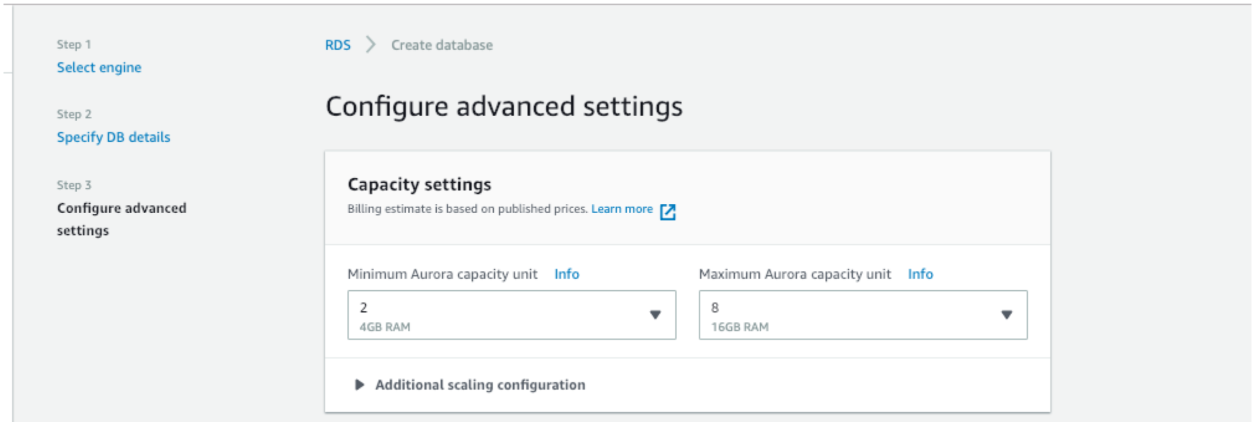
Go to AWS console and go to RDS and create a new Database

select *Amazon Aurora* and choose the edition which is *Aurora serverless* and click *next*

Select the *Serverless* radio button.

And in **DB cluster identifier** enter *MyClusterName*

Set the *Master username* and *password* and remember them for later use. And click *Next*.



Step 1
Select engine

Step 2
Specify DB details

Step 3
Configure advanced settings

RDS > Create database

Configure advanced settings

Capacity settings

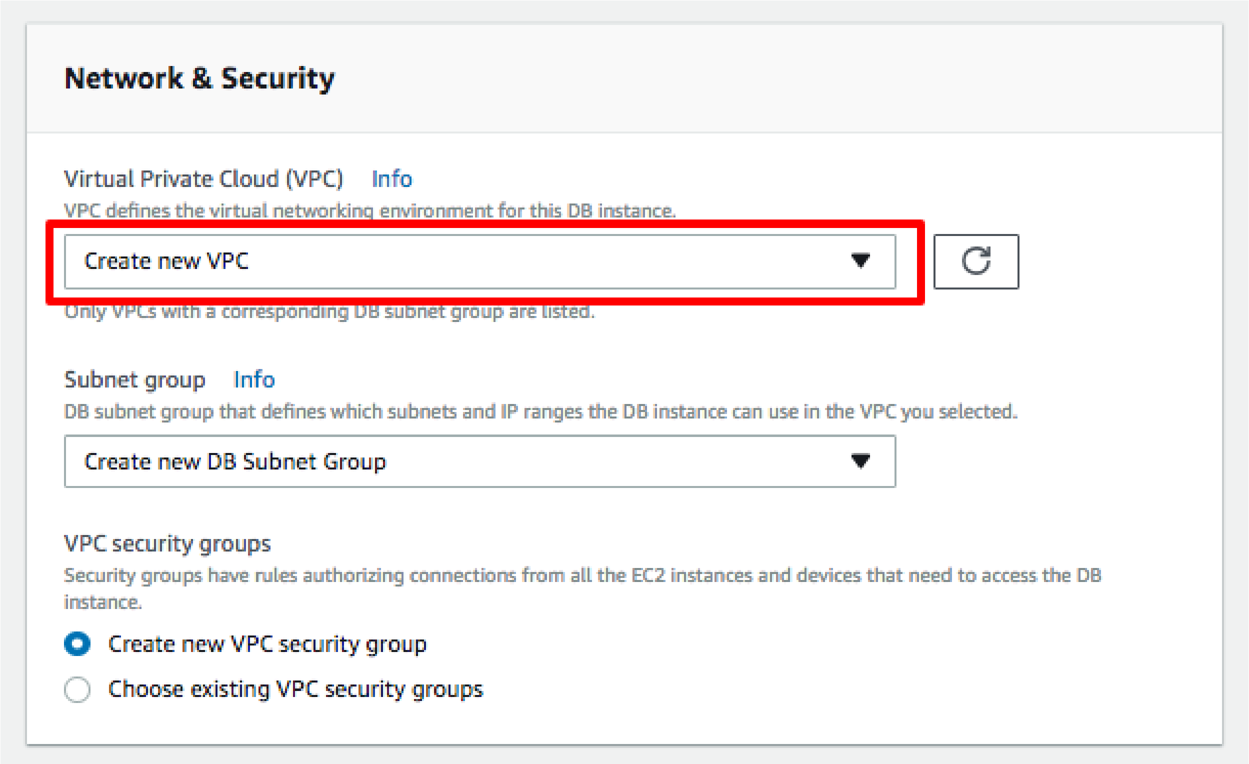
Billing estimate is based on published prices. Learn more

Minimum Aurora capacity unit Info: 2 (4GB RAM)

Maximum Aurora capacity unit Info: 8 (16GB RAM)

▶ Additional scaling configuration

In next page, *Configure advanced settings* , in **Capacity setting** section, select the Minimum & Maximum Aurora capacity units.



Network & Security

Virtual Private Cloud (VPC) Info

VPC defines the virtual networking environment for this DB instance.

Create new VPC

Only VPCs with a corresponding DB subnet group are listed.

Subnet group Info

DB subnet group that defines which subnets and IP ranges the DB instance can use in the VPC you selected.

Create new DB Subnet Group

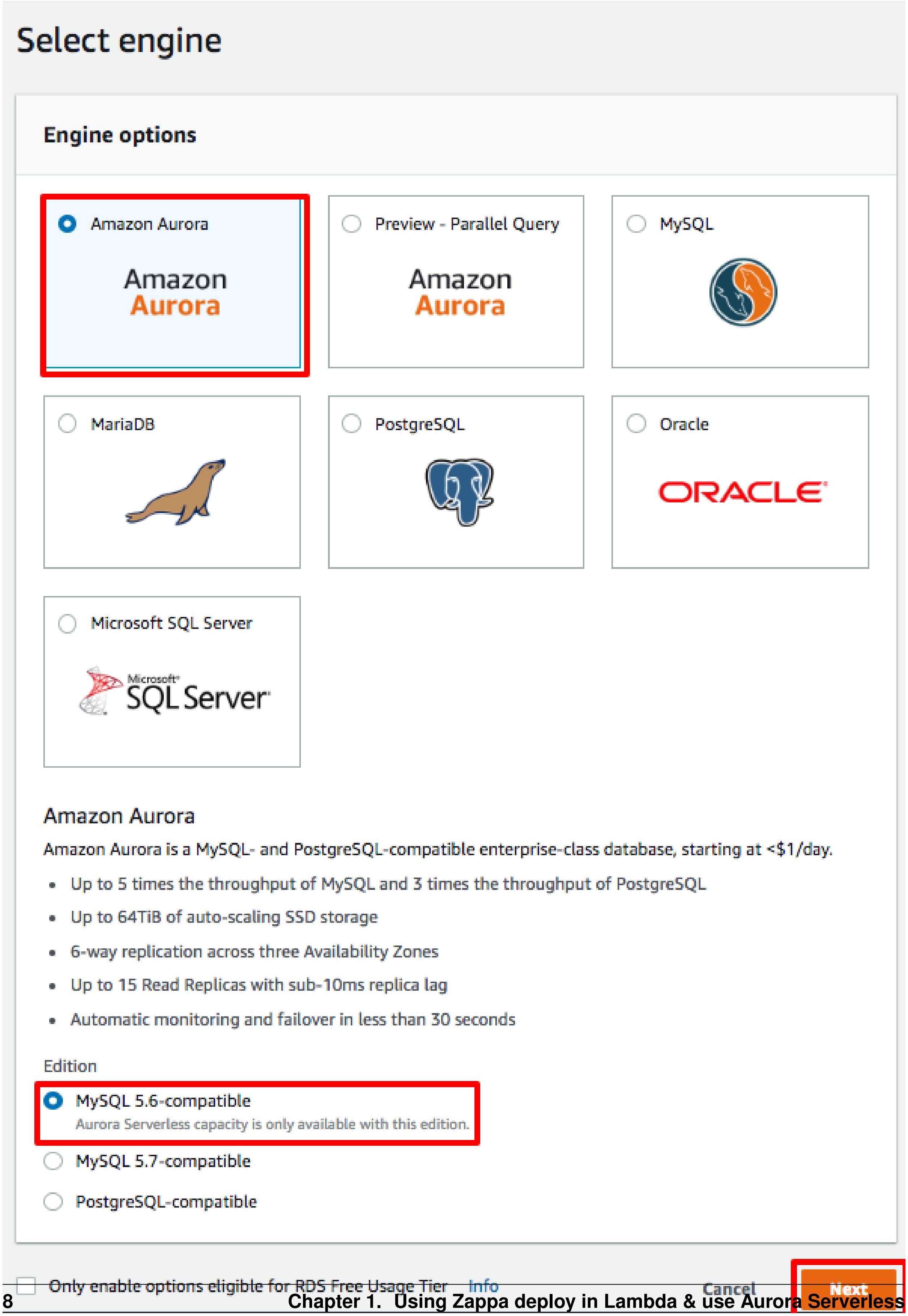
VPC security groups

Security groups have rules authorizing connections from all the EC2 instances and devices that need to access the DB instance.

● Create new VPC security group

○ Choose existing VPC security groups

And in *Network & Security* section, under **Virtual Private Cloud (VPC)** list, select *Create new VPC*. Under **Subnet group** list, select *Create new DB Subnet Group*. Under **VPC security groups** list, select *Create new VPC security*



Select engine

Engine options

● Amazon Aurora

○ Preview - Parallel Query

○ MySQL

○ MariaDB

○ PostgreSQL

○ Oracle

○ Microsoft SQL Server

Amazon Aurora

Amazon Aurora is a MySQL- and PostgreSQL-compatible enterprise-class database, starting at <$1/day.

- Up to 5 times the throughput of MySQL and 3 times the throughput of PostgreSQL
- Up to 64TiB of auto-scaling SSD storage
- 6-way replication across three Availability Zones
- Up to 15 Read Replicas with sub-10ms replica lag
- Automatic monitoring and failover in less than 30 seconds

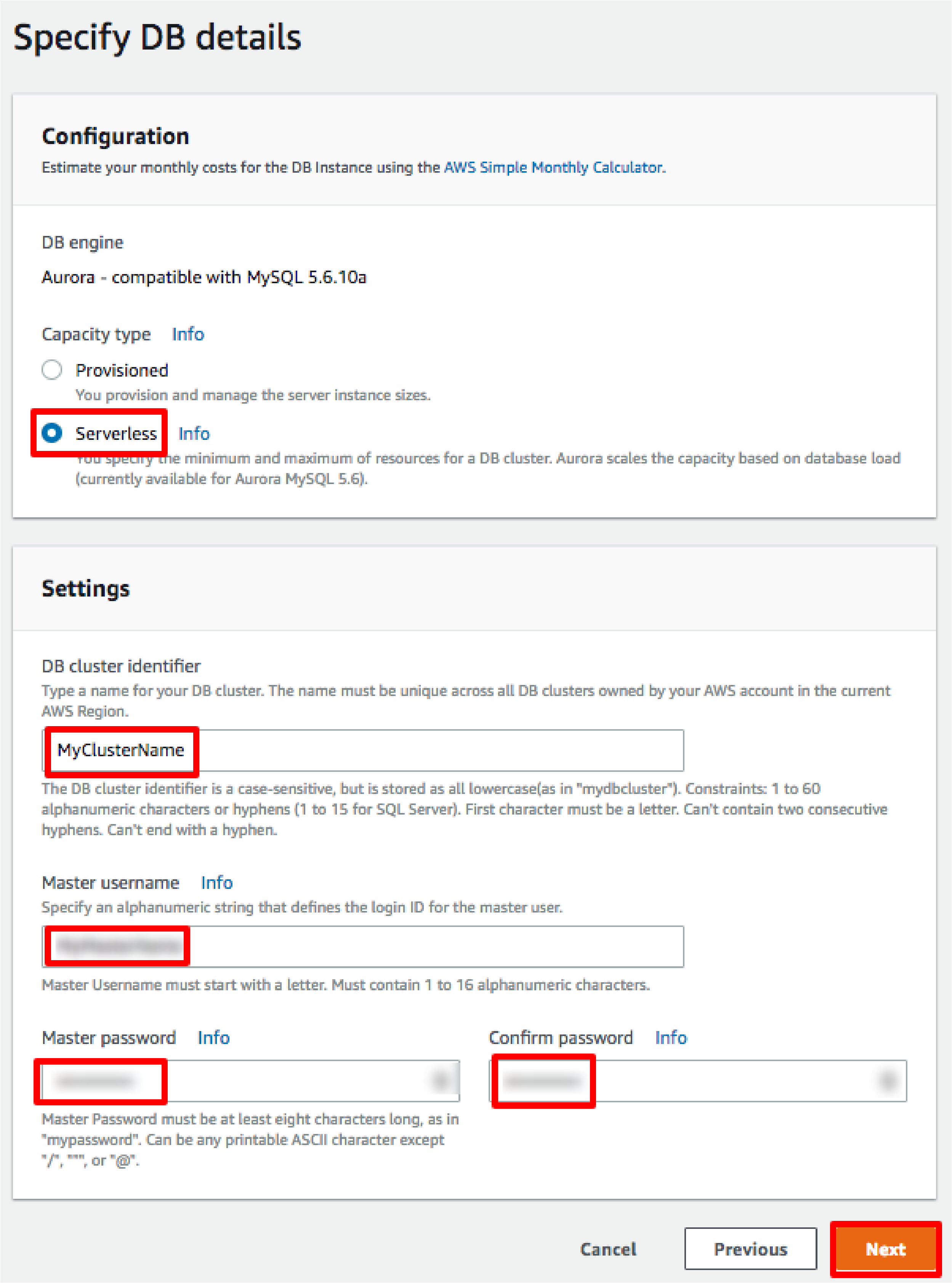
Edition

● MySQL 5.6-compatible
Aurora Serverless capacity is only available with this edition.

○ MySQL 5.7-compatible

○ PostgreSQL-compatible

□ Only enable options eligible for RDS Free Usage Tier Info



Specify DB details

Configuration

Estimate your monthly costs for the DB Instance using the AWS Simple Monthly Calculator.

DB engine

Aurora - compatible with MySQL 5.6.10a

Capacity type Info

○ Provisioned
You provision and manage the server instance sizes.

● Serverless Info
You specify the minimum and maximum of resources for a DB cluster. Aurora scales the capacity based on database load (currently available for Aurora MySQL 5.6).

Settings

DB cluster identifier
Type a name for your DB cluster. The name must be unique across all DB clusters owned by your AWS account in the current AWS Region.

MyClusterName

The DB cluster identifier is a case-sensitive, but is stored as all lowercase(as in "mydbcluster"). Constraints: 1 to 60 alphanumeric characters or hyphens (1 to 15 for SQL Server). First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.

Master username Info
Specify an alphanumeric string that defines the login ID for the master user.

[redacted]

Master Username must start with a letter. Must contain 1 to 16 alphanumeric characters.

Master password Info

[redacted]

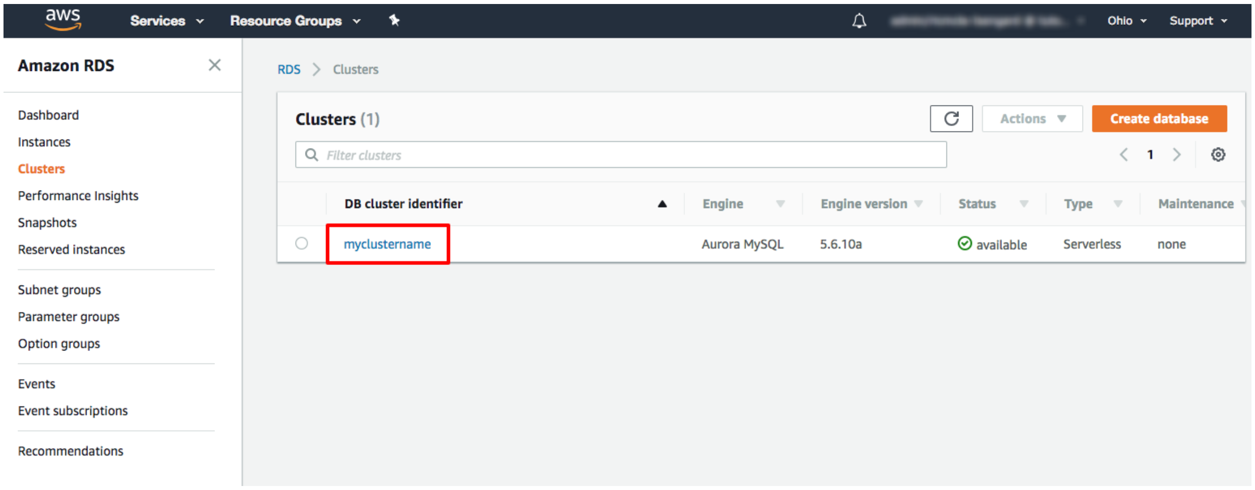
Master Password must be at least eight characters long, as in "mypassword". Can be any printable ASCII character except "/", """, or "@".

Confirm password Info

[redacted]

group.

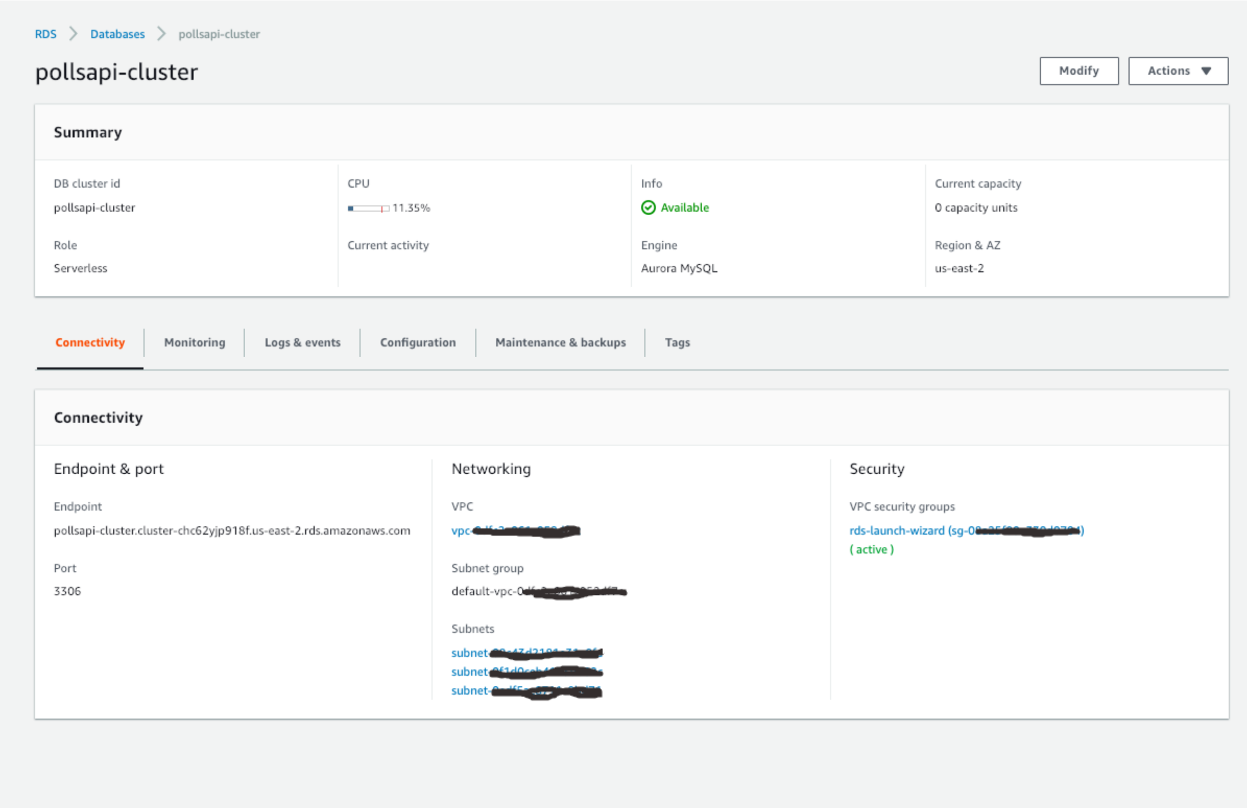
And Click **Create database**



Clusters (1)

	DB cluster identifier	Engine	Engine version	Status	Type	Maintenance
○	myclustername	Aurora MySQL	5.6.10a	available	Serverless	none

Now our Serverless Database is created, click on the *db-cluster* name to see the details



pollsapi-cluster

Summary

DB cluster id: pollsapi-cluster
CPU: 11.35%
Info: Available
Current capacity: 0 capacity units
Role: Serverless
Current activity
Engine: Aurora MySQL
Region & AZ: us-east-2

Connectivity

Endpoint & port
Endpoint: pollsapi-cluster.cluster-chc62yjp918f.us-east-2.rds.amazonaws.com
Port: 3306

Networking
VPC: vpc-[redacted]
Subnet group: default-vpc-0[redacted]
Subnets: subnet-[redacted] subnet-[redacted] subnet-[redacted]

Security
VPC security groups: rds-launch-wizard (sg-0[redacted]) (active)

We will use the *VPC*, *Subnet Ids* and the *security-group* later.

1.4 Connect Django to MySQL DB

Now our MySQL db is created, we have to link it to our app.

We use `mysqlclient` to connect django to the MySQl Database Server.

```
$ pip install mysqlclient
```

and add it to the `requirements.txt` file

```
# requirements.txt
...
mysqlclient==1.3.14
...
```

Now we need to update `pollsapi/settings.py` file,

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'pollsdb', # dbname
        'USER': 'polls_admin', # master username
        'PASSWORD': 'pollsadmin', # master password
        'HOST': 'pollsapi-cluster.cluster-chcxxxxx.us-east-2.rds.amazonaws.com', #␣
↪Endpoint
        'PORT': '3306',
    }
}
```

1.4.1 Configure Zappa Settings for RDS

Now go to **Lambda Management console** and click on **functions** and click on our lambda function(*pollsapi*)

Then we will go to the **configuration page**, Under the **Network** section, in **Virtual Private Cloud (VPC)**

select the same VPC as in Aurora DB

As Aurora Serverless DB clusters do not have publically accessible endpoints, our MyClusterName RDS can only be accessed from within the same VPC.

Then in **Subnets** select all the subnets as in Aurora DB

and for **Security groups** select a different security group than the one on Aurora DB.

Update Security Group Endpoint

Now we have to update the security group Inbound endpoint.

In the RDS console, go to databases section and click on our DB name, which will take us to

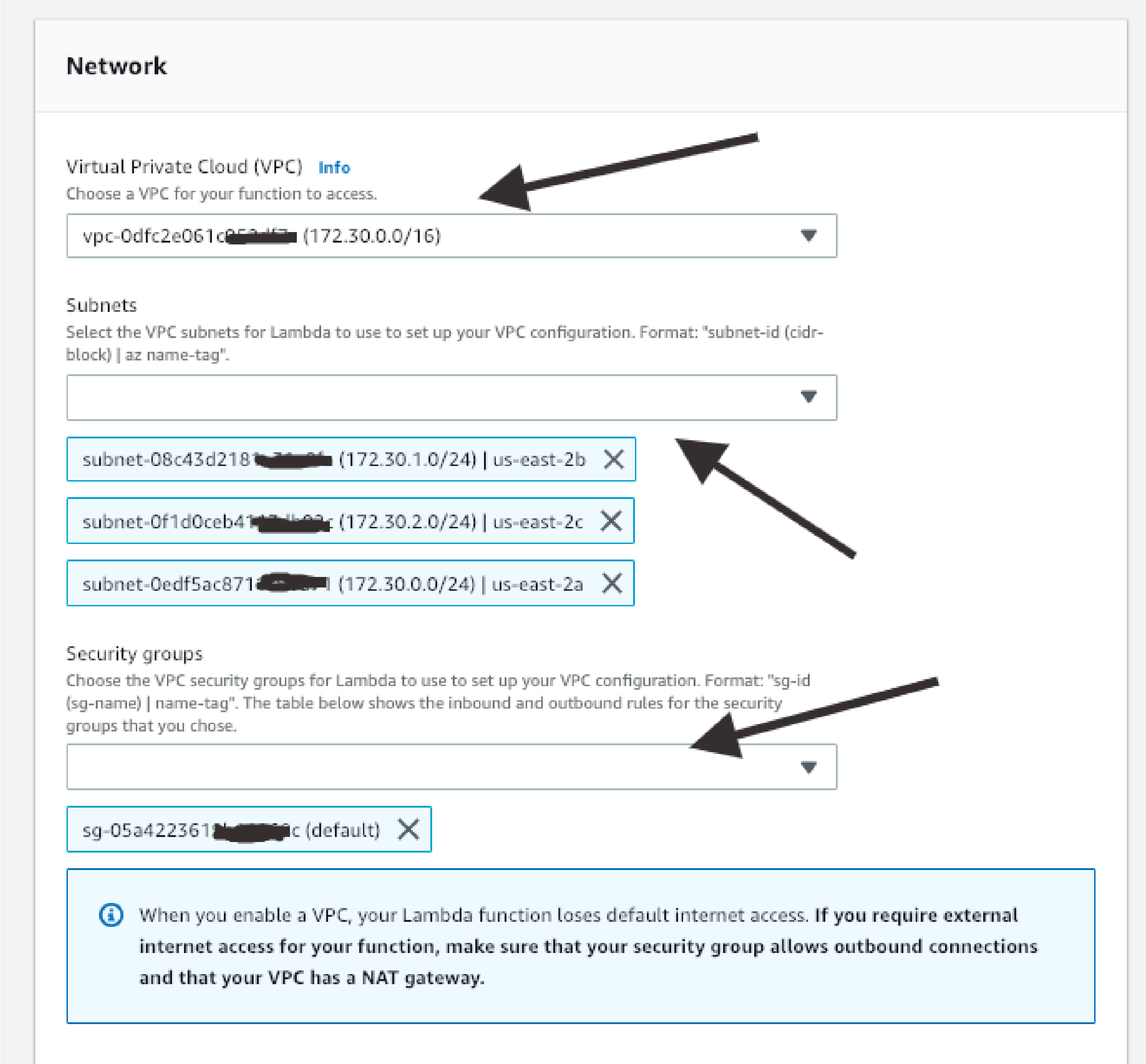
Now click on the security group and we will be taken to the Security Group page

Go to **Inbound** tab in the bottom and click on the **edit** button

Here click on **Add Rule** and enter **Type** as **MYSQL/Aurora** & in **Source** enter the **Security Group Id of the Lambda function** and save it.

Setup the Database

Now let us create a management command our polls app



Network

Virtual Private Cloud (VPC)

Choose a VPC for your function to access.

vpc-0dfc2e061c[redacted] (172.30.0.0/16)

Subnets

Select the VPC subnets for Lambda to use to set up your VPC configuration. Format: "subnet-id (cidr-block) | az name-tag".

subnet-08c43d218[redacted] (172.30.1.0/24) | us-east-2b

subnet-0f1d0ceb41[redacted]c (172.30.2.0/24) | us-east-2c

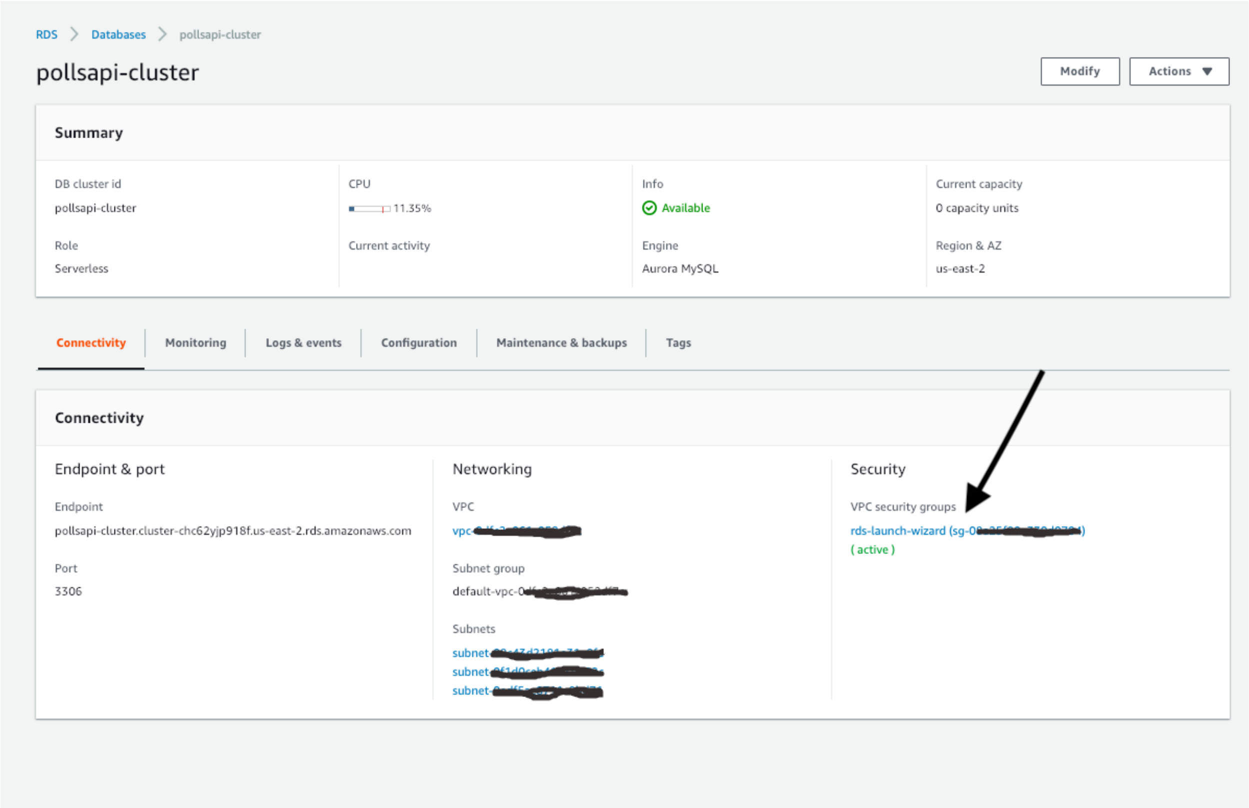
subnet-0edf5ac871[redacted] (172.30.0.0/24) | us-east-2a

Security groups

Choose the VPC security groups for Lambda to use to set up your VPC configuration. Format: "sg-id (sg-name) | name-tag". The table below shows the inbound and outbound rules for the security groups that you chose.

sg-05a42236 1[redacted]c (default)

When you enable a VPC, your Lambda function loses default internet access. **If you require external internet access for your function, make sure that your security group allows outbound connections and that your VPC has a NAT gateway.**



pollsapi-cluster

Summary

DB cluster id	CPU	Info	Current capacity
pollsapi-cluster	11.35%	Available	0 capacity units
Role	**Current activity**	**Engine**	**Region & AZ**
Serverless		Aurora MySQL	us-east-2

Connectivity

Endpoint & port

Endpoint
pollsapi-cluster.cluster-chc62yjp918f.us-east-2.rds.amazonaws.com

Port
3306

Networking

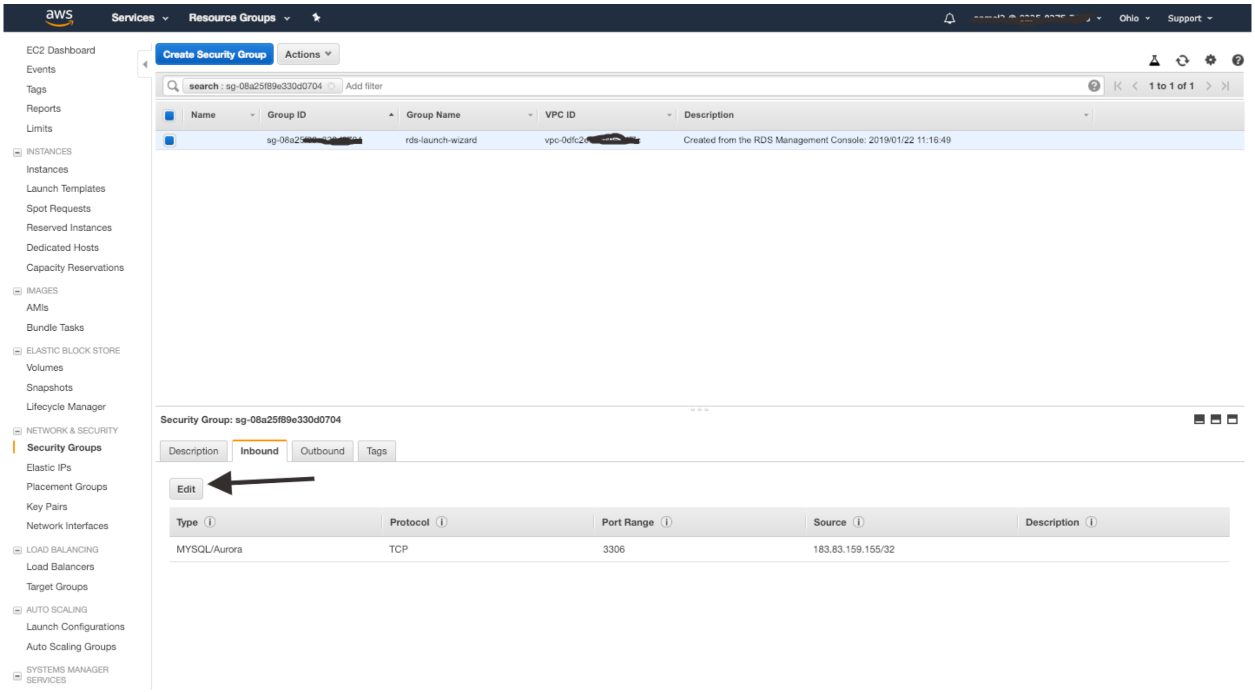
VPC
vpc[redacted]

Subnet group
default-vpc-0[redacted]

Subnets
subnet[redacted]
subnet[redacted]
subnet[redacted]

Security

VPC security groups
rds-launch-wizard (sg-0[redacted])
(active)

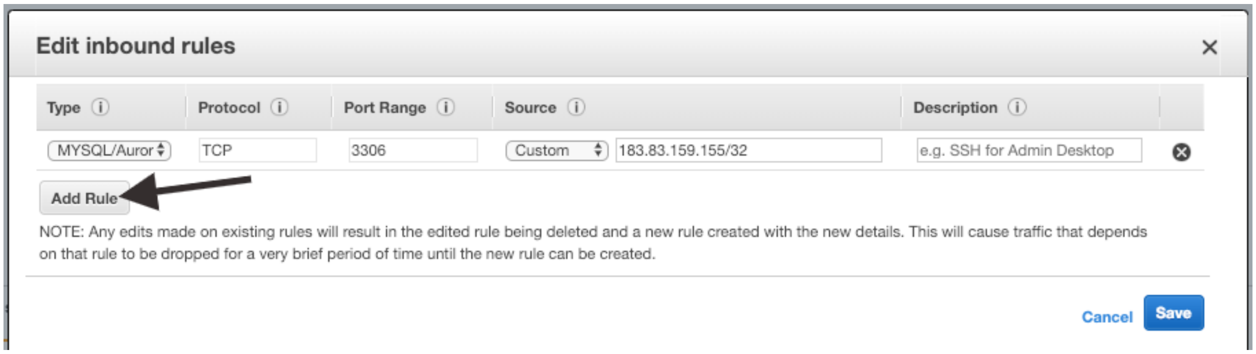


search : sg-08a25f89e330d0704

Name	Group ID	Group Name	VPC ID	Description
	sg-08a25[redacted]	rds-launch-wizard	vpc-0dfc2[redacted]	Created from the RDS Management Console: 2019/01/22 11:16:49

Security Group: sg-08a25f89e330d0704

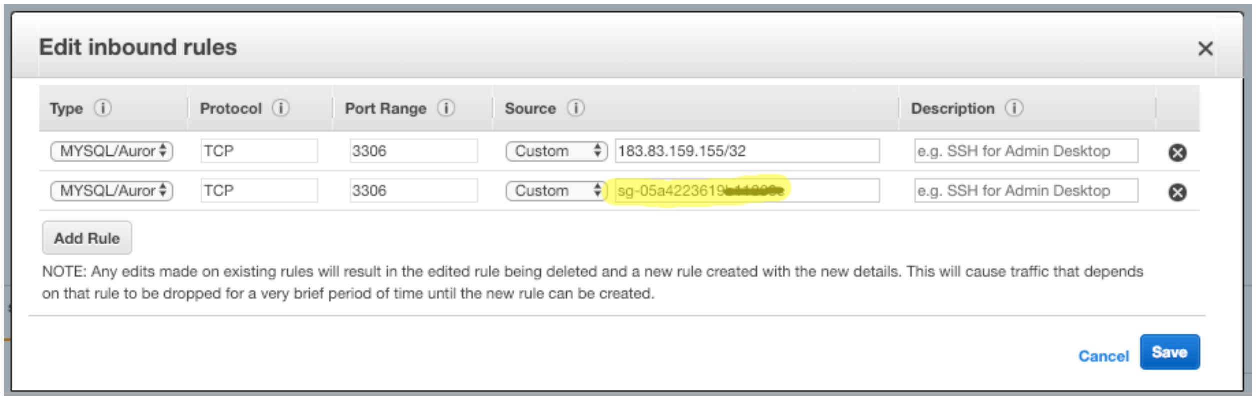
Type	Protocol	Port Range	Source	Description
MYSQL/Aurora	TCP	3306	183.83.159.155/32	



Edit inbound rules

Type	Protocol	Port Range	Source	Description
MYSQL/Auror	TCP	3306	Custom 183.83.159.155/32	e.g. SSH for Admin Desktop

NOTE: Any edits made on existing rules will result in the edited rule being deleted and a new rule created with the new details. This will cause traffic that depends on that rule to be dropped for a very brief period of time until the new rule can be created.



Edit inbound rules

Type	Protocol	Port Range	Source	Description
MYSQL/Auror	TCP	3306	Custom 183.83.159.155/32	e.g. SSH for Admin Desktop
MYSQL/Auror	TCP	3306	Custom sg-05a4223619[redacted]	e.g. SSH for Admin Desktop

NOTE: Any edits made on existing rules will result in the edited rule being deleted and a new rule created with the new details. This will cause traffic that depends on that rule to be dropped for a very brief period of time until the new rule can be created.

```
$ cd polls
$ mkdir management
$ cd management
$ touch __init__.py
$ mkdir commands
$ cd commands
$ touch __init__.py
$ touch create_db.py
```

```python
# polls/management/commands/create_db.py
import sys
import logging
import MySQLdb

from django.core.management.base import BaseCommand, CommandError
from django.conf import settings

rds_host = 'pollsapi-cluster.cluster-chc62yjp918f.us-east-2.rds.amazonaws.com'
db_name = 'pollsdb'
user_name = 'polls_admin'
password = 'pollsadmin'
port = 3306

logger = logging.getLogger()
logger.setLevel(logging.INFO)


class Command(BaseCommand):
    help = 'Creates the initial database'
```

(continues on next page)

(continued from previous page)

```
    def handle(self, *args, **options):
        print('Starting db creation')
        try:
            db = MySQLdb.connect(host=rds_host, user=user_name,
                                 password=password, db="mysql", connect_timeout=5)
            c = db.cursor()
            print("connected to db server")
            c.execute("""CREATE DATABASE pollsdb;""")
            c.execute(
                """GRANT ALL PRIVILEGES ON db_name.* TO 'polls_admin' IDENTIFIED BY
↪'pollsadmin';""")
            c.close()
            print("closed db connection")
        except:
            logger.error(
                "ERROR: Unexpected error: Could not connect to MySql instance.")
            sys.exit()
```

Now let us update zappa

```
$ zappa update dev
```

And create the databse using the management command

```
$ zappa manage dev create_db
```

which will show us

```
$ zappa manage dev create_db
[START] RequestId: 5c2de49d-856e-4d75-963d-017a98660XXX Version: $LATEST
[DEBUG] 2019-01-22T14:55:28.387Z 5c2de49d-856e-4d75-963d-017a98660XXX Zappa Event: {
↪'manage': 'create_db'}
Starting db creation
connected to db server
closed db connection
[END] RequestId: 5c2de49d-856e-4d75-963d-017a98660XXX
[REPORT] RequestId: 5c2de49d-856e-4d75-963d-017a98660XXX
Duration: 218.58 ms
Billed Duration: 300 ms
Memory Size: 512 MB
Max Memory Used: 83 MB
```

We have to migrate now

```
$ zappa manage dev migrate
```

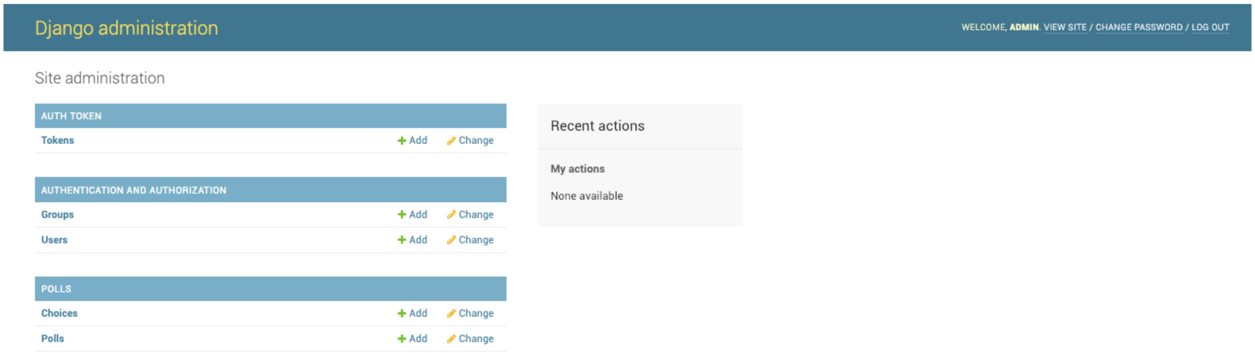
Now let us create the admin user

```
$ zappa invoke --raw dev "from django.contrib.auth.models import User; User.objects.
↪create_superuser('admin', 'anmol@agiliq.com', 'somerandompassword')"
```

Now let us check by logging in the admin page

NOW OUR DJANGO APP IS COMPLETELY SERVERLESS !!

We can check the lambda logs by `zappa dev tail`



Django administration

WELCOME, ADMIN. VIEW SITE / CHANGE PASSWORD / LOG OUT

Site administration

AUTH TOKEN		
Tokens	+ Add	✎ Change

AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	✎ Change
Users	+ Add	✎ Change

POLLS		
Choices	+ Add	✎ Change
Polls	+ Add	✎ Change

Recent actions

My actions

None available

CHAPTER 2

Using Apex-Up deploy in Lambda and use Aurora Serverless

We will try to deploy a basic django app onto **AWS Lambda** using **Apex Up**.

AWS Lambda is a serverless computing platform by amazon, which is completely event driven and it automatically manages the computing resources. It scales automatically when needed, depending upon the requests the application gets.

Apex Up is a Open Source framework used for deploying serverless applications onto AWS-Lambda. Up currently supports Node.js, Golang, Python, Java, Crystal, and static sites out of the box. Up is platform-agnostic, supporting AWS Lambda and API Gateway.

Note :

- **Apex-Up currently supports only Node.js lambda environment**, but we can use python 2.7 and 3.4 in it.
- **We have to use Django 2.0 as it is the only latest version which supports python3.4**

2.1 Install and Configure the Environment

First configure the AWS credentials

https://books.agiliq.com/projects/django-deployments-cookbook/en/latest/using_zappa_lambda_aurora.html#configure-aws-credentials.

2.1.1 Install Apex Up

Currently *Up* has only binary form releases and can be installed by

```
$ curl -sf https://up.apex.sh/install | sh
```

this installs *Up* in `/usr/local/bin` by default.

We can verify the installation by

```
$ up version

# or

$ up --help
```

2.1.2 Go to Django app

We will use *Pollsapi* (https://github.com/agiliq/building-api-django) as the django project.

Note: **We cannot see the django error messages in the url(even if we have DEBUG=True), we can see them in the apex-up logs only**

Now go inside the *pollsapi* app in this repo.

Next create a virtualenv with python34 and install `requirements.txt`

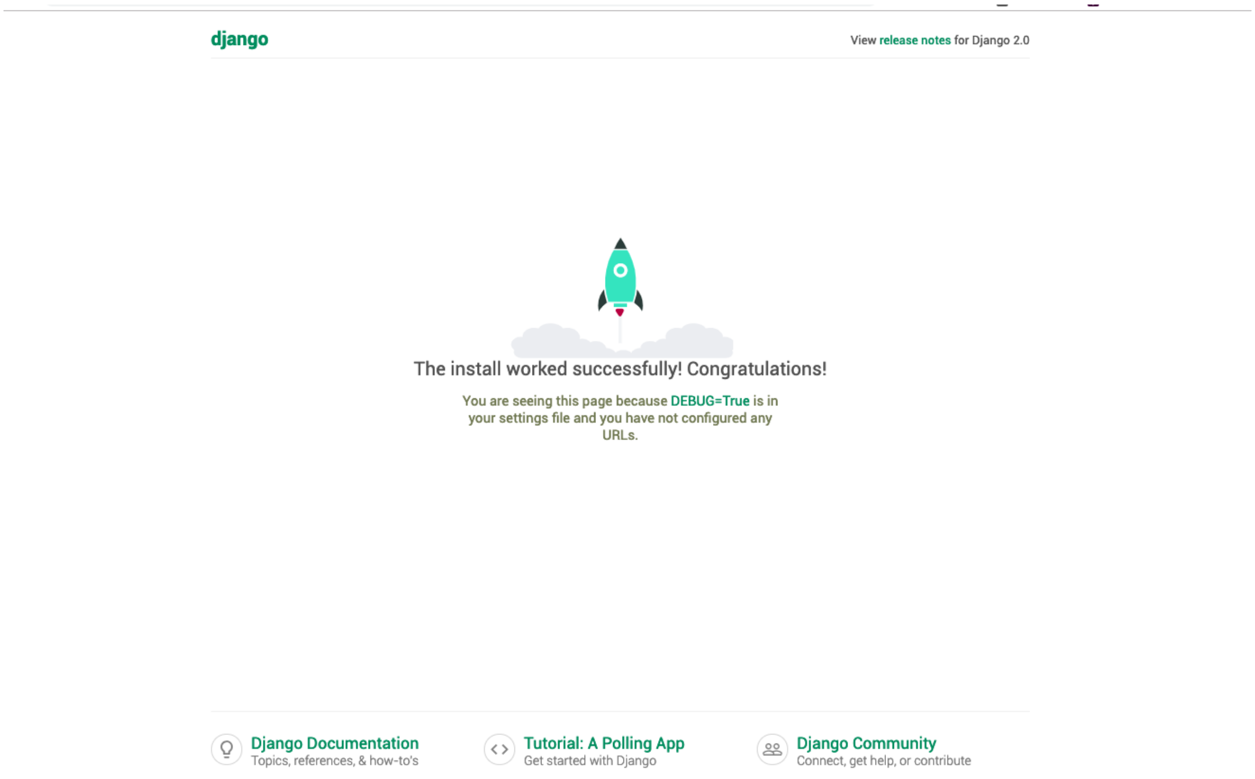
```
$ pip install -r requirements.txt
```

```
$ django-admin --version        # check the django version
2.0.3
```

Now rename the `manage.py` to `app.py` for *apex-up* to work.

```
$ python  app.py runserver
```

which will show us



The install worked successfully! Congratulations!

You are seeing this page because DEBUG=True is in your settings file and you have not configured any URLs.

django
View release notes for Django 2.0
Django Documentation
Topics, references, & how-to's
Tutorial: A Polling App
Get started with Django
Django Community
Connect, get help, or contribute

and in `polls/settings.py` add aws subdomain to the 'ALLOWED_HOSTS'

```
...
ALLOWED_HOSTS = [".amazonaws.com", "127.0.0.1"]  # lambda subdomain and localhost
...
```

2.1.3 Serving Static Files

To configure static files in django https://www.agiliq.com/blog/2019/01/complete-serverless-django/#serving-static-files

2.1.4 Setup Serverless MySQL Database

To set up Aurora serverless DB follow https://www.agiliq.com/blog/2019/01/complete-serverless-django/#setup-serverless-mysql-database

2.1.5 Connect Our App to MySQL DB

To connect our Django App to aurora db, follow https://www.agiliq.com/blog/2019/01/complete-serverless-django/#connect-django-to-mysql-db

After configuring our `settings.py` file should have a similar database config

```
...

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'pollsdb', # dbname
        'USER': 'polls_admin', # master username
        'PASSWORD': 'pollsadmin', # master password
        'HOST': 'pollsapi-cluster.cluster-chcxxxxx.us-east-2.rds.amazonaws.com', #␣
↪Endpoint
        'PORT': '3306',
    }
}
...
```

Now create a file in the same level as the `app.py` file named **‘‘up.json‘‘** and add the following lines

```
{
  "name": "pollsapi",
  "profile": "default",
  "regions": [
    "us-east-2"
  ],
  "proxy": {
    "command": "python3 app.py runserver 0.0.0.0:$PORT"
  }
}
```

here `name` is the name of the project to be deployed

`profile` is the aws credentials profile name

`region` is the region of the lambda function `proxy` acts as a reverse proxy in front of our server, which provides features like CORS, redirection, script injection and middleware style features.

We have to include the following configuration to our proxy object

Add `command` Command run through the shell to start our server (Default ./server)

In the proxy command we have to give the command to start the django server ie *runserver* .

As presently *Up* supports only Node.js lambda runtime environment, but we can use python 2.7 and 3.4 in it. So we can use python3 by mentioning the command as `python3 app.py runserver 0.0.0.0:$PORT` where the `$PORT` is the port where our app runs(which is generated dynamically).

for more configuration settings like using custom domains, secrets, deploying to multiple AWS regions or multiple stages(test/staging/prod etc) check the docs

Now let us test the app by deploying it,

```
$ up
# or
$ up deploy
# or
$ up -v         # verbose
```

```
$ up

     build: 4,752 files, 16 MB (9.463s)
     deploy: staging (commit 3asdfjj) (17.103s)
     stack: complete (26.324s)
     endpoint: https://Xpiix0c1.execute-api.us-east-2.amazonaws.com/staging/

     Please consider subscribing to Up Pro for additional features and to help keep the project alive!
     Visit https://github.com/apex/up#pro-features for details.
```

to get the url of the application

```
$ up url
# or
$ up url --open
```

Now when we open the url, we get

The logs can be checked by these commands

```
$ up logs
# or
$ up logs -f            # for live logs
```

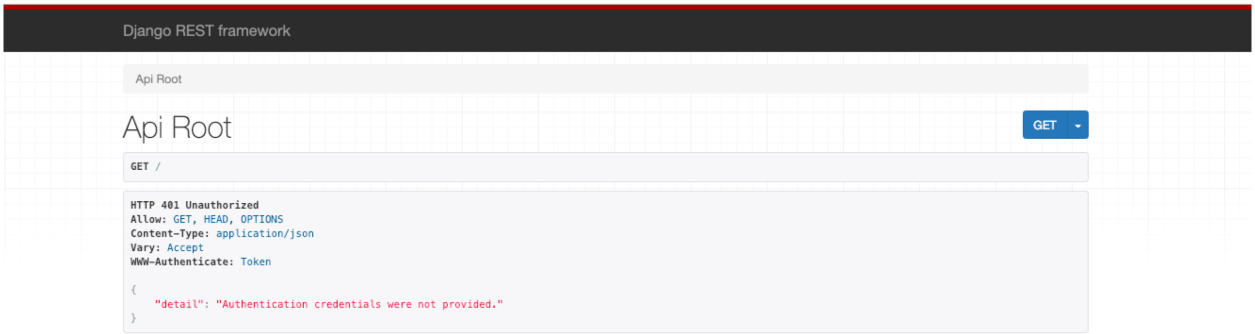
Up also sends our logs to AWS cloudwatch, so we can search for the logs there also.

2.1.6 To run Django Migrations

We have to add the migrate command to the `proxy.command` in the *up.json* file.

```
{
  "name": "pollsapi",
  "profile": "default",
```

(continues on next page)



```
GET /
```

```
HTTP 401 Unauthorized
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept
WWW-Authenticate: Token

{
    "detail": "Authentication credentials were not provided."
}
```

(continued from previous page)

```
  "regions": [
    "us-east-2"
  ],
  "proxy": {
    "command": "python3 app.py migrate && python3 app.py runserver 0.0.0.0:$PORT"
  }
}
```

2.2 Troubleshooting

We should note that we cannot see the django error messages in the url(even if we have DEBUG=True), we can see them in the apex-up logs

We can check for the errors by

```
$ up logs error                 # Shows error logs.

$ up logs 'error or fatal'      # Shows error and fatal logs.

$ up logs 'status >= 400'       # Shows 4xx and 5xx responses.
```

To delete the deployment

```
$ up stack delete   # delete the deployment
```

We have to note that we have only python 2.7 and python 3.4 versions available at present in Apex-Up

CHAPTER 3

Using Zeit-Now & use RDS Postgres

We will see how to deploy a Django application using ***Zeit Now*** and use ***RDS Postgres*** as the DB.

‘Zeit Now <https://zeit.co/now>‘__ is a serverless deployment platform with its own CLI and a desktop app.

‘RDS Postgres <https://aws.amazon.com/rds/postgresql/>‘__ is the open source relational database for Postgres by AWS.

3.1 Get Zeit Now

1. First we have to create an account in Zeit.
2. Then we have to install the *Now* CLI or the *Now Desktop App(which includes CLI)* .

we can download the Now Desktop which does not require Node.js. *Now Desktop* comes with *Now CLI (our command line interface)*

or we can install Now Cli using npm

```
$ npm install -g now
```

To check if *Now* CLI has been installed

```
$ now --version
```

3.2 Go to Django app

After installing *Zeit Now*, let us set up our django project, here we used *Pollsapi* (https://github.com/agiliq/building-api-django) as the django project.

3.2.1 Configure Django Settings

We have to add our host to the ALLOWED_HOSTS in the setting.py file

```
...
ALLOWED_HOSTS = [".now.sh"]  # add this subdomain
```

3.2.2 Configure Django for S3

We will use AWS S3 bucket to serve our static files, so let us configure Django for S3

```
$ pip install django-s3-storage
```

and also add it in the requirements.txt file.

```
...
django-s3-storage==0.12.4
...
```

Now update the settings.py file to add 'django_s3_storage' to INSTALLED_APPS

```
INSTALLED_APPS = (
          ...,
          'django_s3_storage',
     )
```

and also add these lines at the bottom

```
S3_BUCKET = "now-staticfiles1234"

STATICFILES_STORAGE = "django_s3_storage.storage.StaticS3Storage"

AWS_S3_BUCKET_NAME_STATIC = S3_BUCKET

STATIC_URL = "https://%s.s3.amazonaws.com/" % S3_BUCKET
```

Push the static files to the cloud

```
$ python manage.py collectstatic
```

3.2.3 Setup now.json

Now go inside the *pollsapi* folder in this repo, and create a file named now.json, and add the following:

```
{
  "version": 2,
  "name": "django-pollsapi",
  "builds": [
    {
      "src": "index.py",
      "use": "@contextualist/python-wsgi",
      "config": { "maxLambdaSize": "60mb" }
```

(continues on next page)

(continued from previous page)

```
    }
  ],
  "routes": [{ "src": "/.*", "dest": "/" }]
}
```

- "version" Specifies the Now Platform version the deployment should use and to work with. Type is String.
- "name" is used to organise the deployment into a project. Is is also used as the perfix for all new deployment instances. Type is Number.
- **“”builds”“** Builders are modules that take a deployment’s source and return an output, consisting of either static files or dynamic Lambdas.

  The builds property is an array of objects where each object is a build step, including a src and a use property, at least. If our project has source files that require transformation to be served to users, Builders enable this ability when deploying.

  Builds object consists of:

  - "src" (String): A glob expression or pathname. If more than one file is resolved, one build will be created per matched file. It can include _* and **_.
  - "use" (String): A npm module to be installed by the build process. It can include a semver compatible version (e.g.: @org/proj@1).
  - "config" (Object): Optionally, an object including arbitrary metadata(like maxLambdaSize etc) to be passed to the Builder.

  We are using builder - "@contextualist/python-wsgi" as we want python with wsgi.
- "routes" consists of a list of route definitions.
  - "src": A regular expression that matches each incoming pathname (excluding querystring).
  - "dest": A destination pathname or full URL, including querystring, with the ability to embed capture groups

Let us create a file named index.py, and copy all lines from wsgi.py to this file

```
import os
from django.core.wsgi import get_wsgi_application

os.environ.setdefault("DJANGO_SETTINGS_MODULE", "pollsapi.settings")
app = get_wsgi_application()  # application = get_wsgi_application()
```

Now we have to rename *application to app*, as the builder will search for the app to run.

After this add these lines to the the index.py file

```
...
os.system("python manage.py migrate")
os.system("python manage.py runserver")
```

At present we cannot change the python version of the Zeit Now environment(which is python 3.4), but this feature will be added in the future.

Now deploy the app

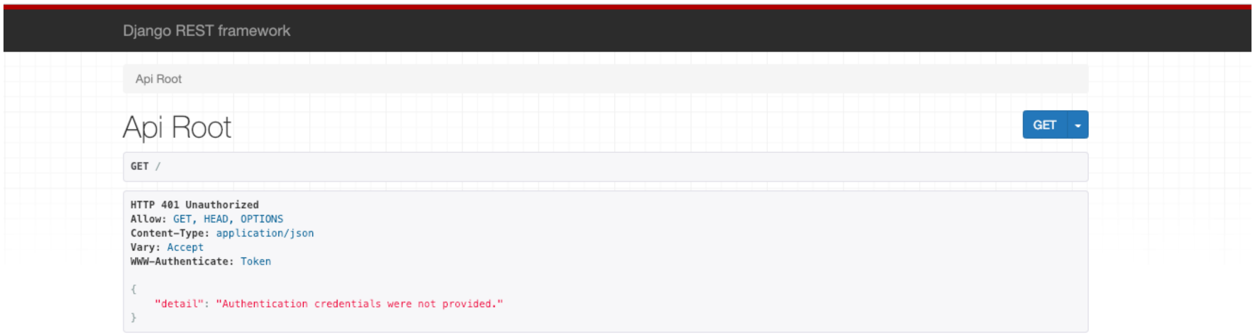
```
$ now
> Deploying ~/building-api-django/pollsapi under anmol@agiliq.com
> Using project django-pollsapi
```

(continues on next page)

(continued from previous page)

```
> Synced 1 file (234B) [1s]
> https://django-pollsapi-4l2pyh2um.now.sh [v2] [in clipboard] [2s]
 index.py         Ready               [1m]
└── λ index.py (20.53MB) [sfo1]
> Success! Deployment ready [1m]
```

Now go to the url, we will see that our project is running



```
GET /
```

```
HTTP 401 Unauthorized
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept
WWW-Authenticate: Token

{
    "detail": "Authentication credentials were not provided."
}
```

Now we have to link it with the Database

3.3 Linking with RDS Postgres

We are using AWS RDS Postgres as our Database.

So first **create an RDS postgres instance** (which also comes in Free tier) and copy the *endpoint* (which we will use to link in the `DATABASES` in settings.py file)

so let us add postgres adapter to our `requirements.txt` file

```
psycopg2==2.7.7
```

and change the `settings.py` file for postgres

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'nowdb', # dbname
        'USER': 'now_admin', # master username
        'PASSWORD': 'nowadmin', # master password
        'HOST': 'nowdb.chc62yjp9.us-east-2.rds.amazonaws.com', # Endpoint
        'PORT': '5432',
    }
}
```

But before using postgres in our Django App, we have to first **download a custom compiled psycopg2 C-library for Python** from https://github.com/jkehler/awslambda-psycopg2

Using psycopg2 via requirements.txt will not sufficient for lambda, as psycopg2 C library for Python is missing in default lambda.

As Zeit Now uses AWS Lambda to deploy our project, we need to use this custom pre-compiled library to use postgres.

First we have to download the repository and copy the folder psycopg2-3.6 to our project and in the same level as our now.json and rename the folder from psycopg2-3.6 to psycopg2.

this will make our app work with the Postgres-DB

After this we have to create an admin-user for our django-app so that we can access the admin

```
$ cd polls
$ mkdir management
$ cd management
$ touch __init__.py
$ mkdir commands
$ cd commands
$ touch __init__.py
$ touch create_admin_user.py
```

```
# polls/management/commands/create_admin_user.py
import sys
import logging

from django.core.management.base import BaseCommand, CommandError
from django.contrib.auth.models import User
from django.conf import settings



class Command(BaseCommand):
    help = 'Creates the initial admin user'

    def handle(self, *args, **options):
        if User.objects.filter(username="admin").exists():
            print("admin exists")
        else:
            u = User(username='admin')
            u.set_password('adminpass')
            u.is_superuser = True
            u.is_staff = True
            u.save()
            print("admin created")
        sys.exit()
```

this command will create the admin user if it does not exists

let us update the index.py by adding the command to create the admin user below the migrate command

```
...
os.system("python manage.py migrate")
os.system("python manage.py create_admin_user")      # add this line
os.system("python manage.py runserver")
```

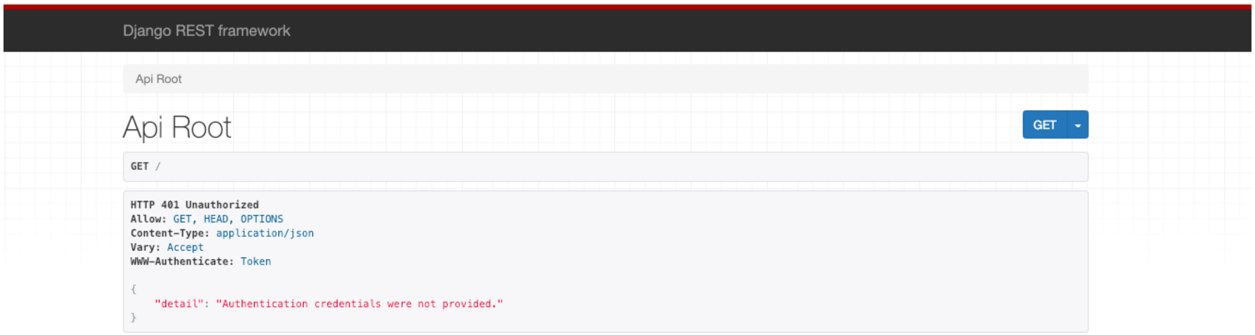
Now let us deploy the app with the updated database settings and the custom postgres library

```
$ now
> Deploying ~/building-api-django/pollsapi under anmol@agiliq.com
> Using project django-pollsapi
> Synced 1 file (234B) [1s]
> https://django-pollsapi-1asdsdfum.now.sh [v2] [in clipboard] [2s]
 index.py         Ready               [1m]
└── λ index.py (20.53MB) [sfo1]
> Success! Deployment ready [1m]
```

we can check the logs of the deployment by adding `/_logs` after our url like https://django-pollsapi-1asdsdfum.now.sh/_logs

Let us check the url

https://django-pollsapi-1asdsdfum.now.sh



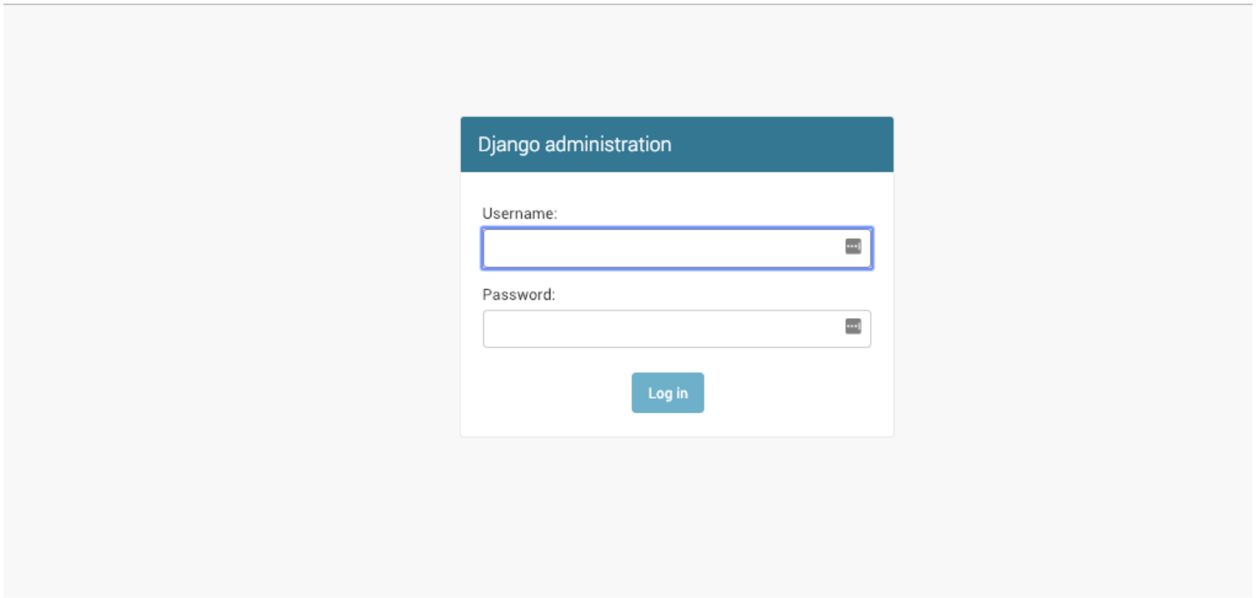
Api Root

GET /

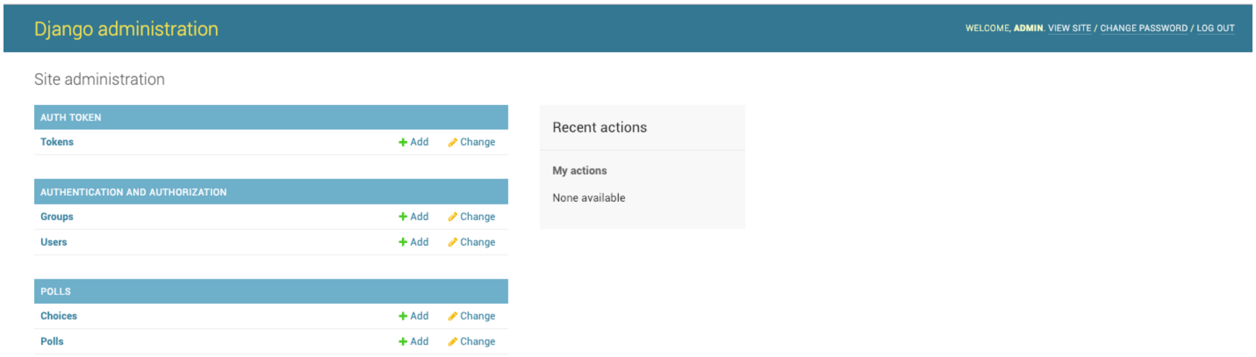
```
HTTP 401 Unauthorized
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept
WWW-Authenticate: Token

{
    "detail": "Authentication credentials were not provided."
}
```

https://django-pollsapi-1asdsdfum.now.sh/admin



Now let us login to our admin



Now our Django app is linked to postgres and deployed using Zeit Now.

CHAPTER 4

Deploy in AWS Fargate

We will deploy a Django app in **AWS Fargate** and use Aurora serverless as the db.

AWS Fargate lets users build and deploy containerized applications without having to manage the underlying servers themselves.

Fargate is a compute engine that allows running containers in Amazon ECS without needing to manage the EC2 servers for cluster. We only deploy our Docker applications and set the scaling rules for it. Fargate is an execution method from ECS.

With *AWS Fargate*, we pay only for the amount of vCPU and memory resources that our containerized application requests ie *We pay only for what we use*.

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow us to package up an application with all of the parts it needs, like libraries and other dependencies, and ship it all out as one package.

And Aurora Serverless is an on-demand, auto-scaling Relational Database System by Amazon AWS(presently compatible with only MySQL). It automatically starts up & shuts down the DB depending on the requirement.

Prerequisites: AWS account and configure the system with aws credentials & aws-cli and Docker in the system.

4.1 Go to Django app

We will use *Pollsapi* (https://github.com/agiliq/building-api-django) as the django project.

Now go inside the *pollsapi* app in this repo.

Let us create a virtual environment and install the requirement.txt

```
$ pip install -r requirements.txt
```

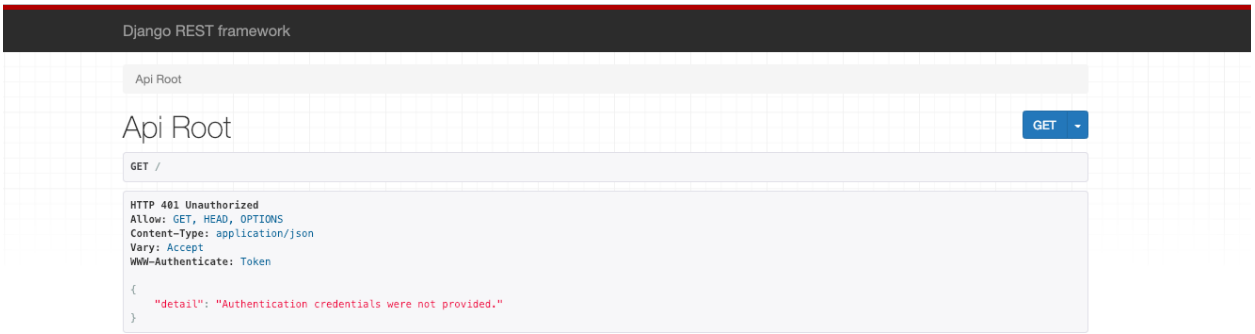
and in `polls/settings.py` add aws subdomain to the ‘ALLOWED_HOSTS’

```
...
ALLOWED_HOSTS = ["*"]  # for all domains - only for development
...
```

And run the application

```
$ ./manage.py runserver
```

which will show us



```
GET /
```

```
HTTP 401 Unauthorized
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept
WWW-Authenticate: Token

{
    "detail": "Authentication credentials were not provided."
}
```

4.2 Build the application using Docker

Now lets now containerize our application using Docker. Let us create a file named `Dockerfile` in the *pollsapi* folder and in the same level as *manage.py* .

```
$ touch Dockerfile
```

and add the following lines

In this Dockerfile, we install Python and our application and then specify how we want to run our application in the container.

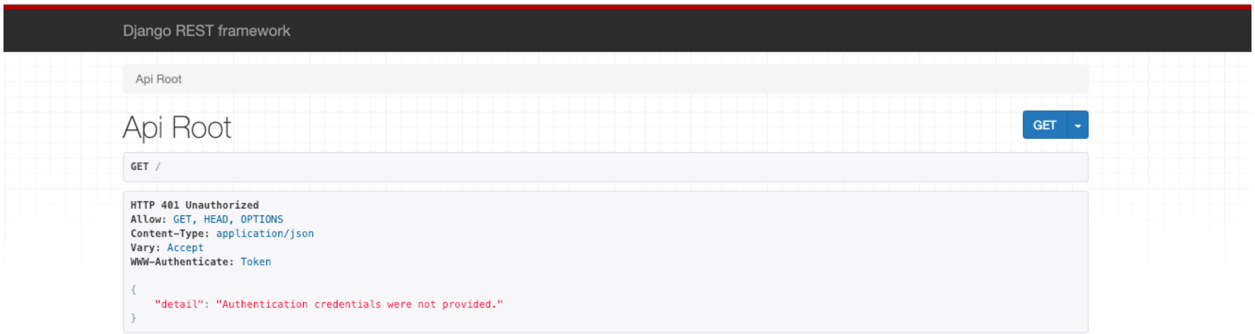
Let us Build the Docker container for our pollsapi app

```
$ docker build -t pollsapi-app .
```

The `docker build` command builds Docker images from a Dockerfile. We will run the container we created in the previous step.

```
$ docker run -p 8800:8800 -t pollsapi-app
February 19, 2019 - 13:22:46
Django version 2.0.3, using settings 'pollsapi.settings'
Starting development server at http://0.0.0.0:8800/
Quit the server with CONTROL-C.
```

now when we go to the url `0.0.0.0:8800`, we will see



```
GET /
```

```
HTTP 401 Unauthorized
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept
WWW-Authenticate: Token

{
    "detail": "Authentication credentials were not provided."
}
```

4.3 Deploying our application using AWS Fargate

Here, we will deploy our container to Amazon's Elastic Container Repository (ECR) and then launch the application using Fargate.

4.3.1 Create a new repository in ECR

Run the following command to create a new repository for the application:

```
$ aws ecr create-repository --repository-name pollsapi-app --region us-east-1
```

If the command is successful, we should see:

```
{
    "repository": {
        "repositoryArn": "arn:aws:ecr:us-east-1:822502757923:repository/pollsapi-app",
        "registryId": "822502757923",
        "repositoryName": "pollsapi-app",
        "repositoryUri": "822502757923.dkr.ecr.us-east-1.amazonaws.com/pollsapi-app",
        "createdAt": 1550555101.0
    }
}
```

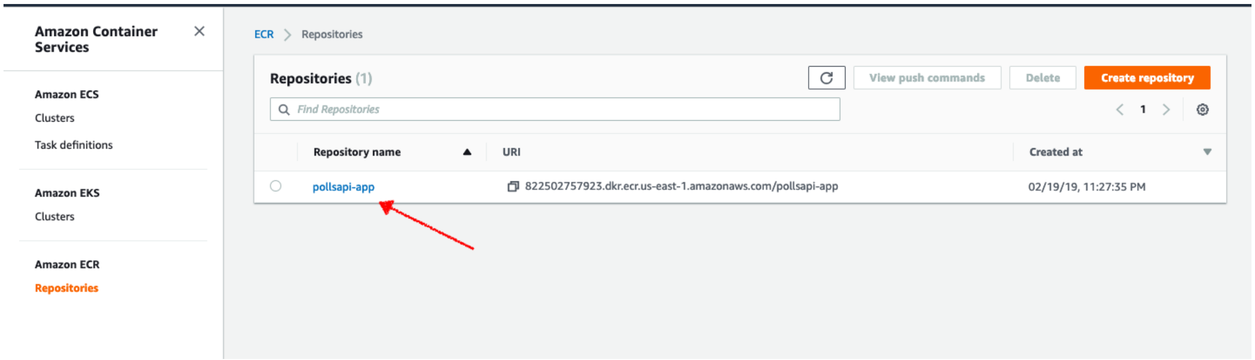
This will create a repository by name `pollsapi-app` in AWS ECR

Now click on the repository name and go inside

we will see that we have no image here, click on `Push Commands` to get a list of commands that we need to run to be able to push our image to ECR. Follow the steps as they are given.

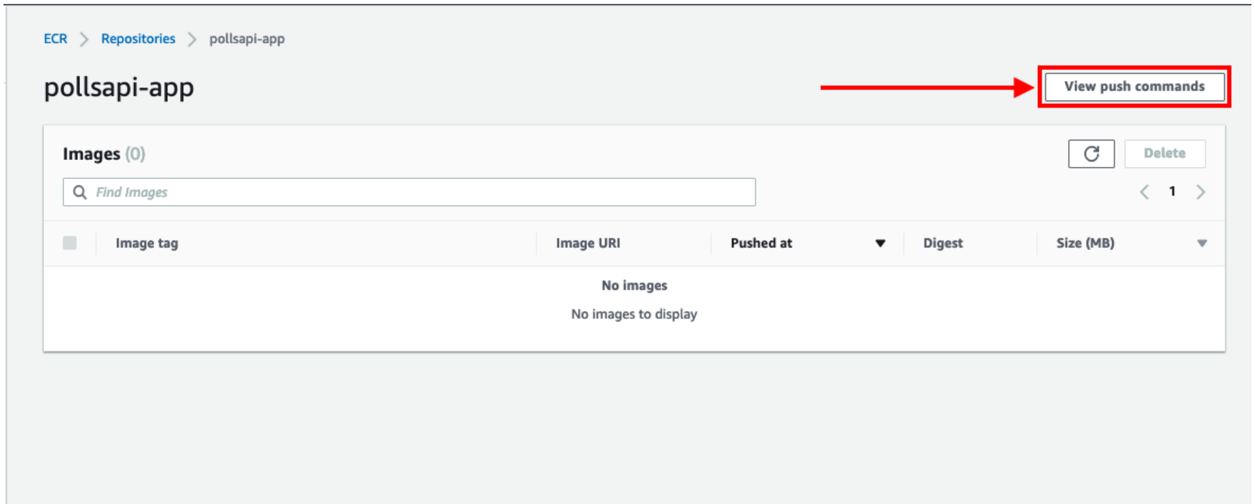
Now we have pushed our image in ECR.

After pushing the image, we can see the image-url



Repositories (1)

Repository name	URI	Created at
pollsapi-app	822502757923.dkr.ecr.us-east-1.amazonaws.com/pollsapi-app	02/19/19, 11:27:35 PM



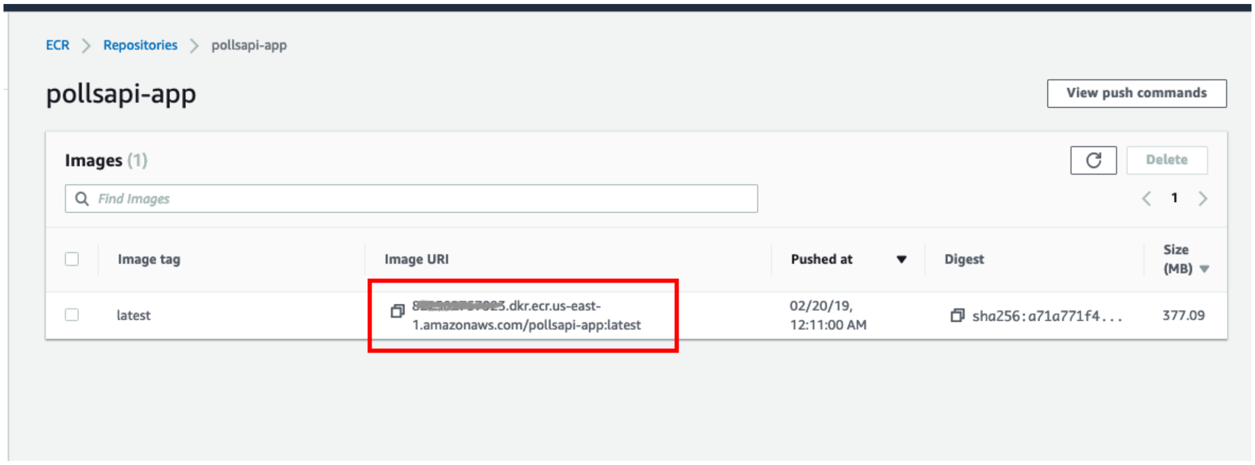
pollsapi-app

Images (0)

Image tag	Image URI	Pushed at	Digest	Size (MB)

No images

No images to display



pollsapi-app

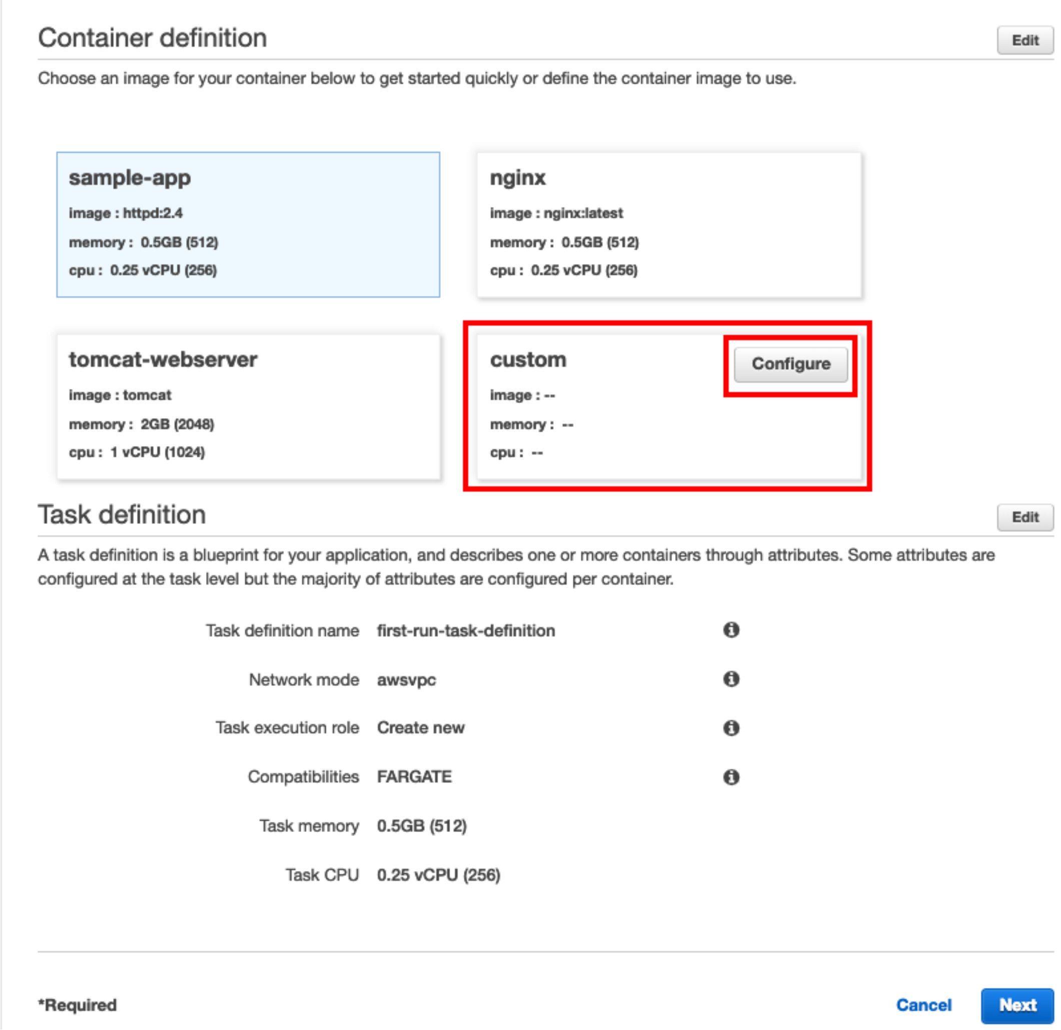
Images (1)

Image tag	Image URI	Pushed at	Digest	Size (MB)
latest	8[redacted]5.dkr.ecr.us-east-1.amazonaws.com/pollsapi-app:latest	02/20/19, 12:11:00 AM	sha256:a71a771f4...	377.09

4.3.2 Create Fargate Application

Now, let us go to the link https://console.aws.amazon.com/ecs/home?region=us-east-1#/getStarted and create a new Fargate Application. Click on *Get Started*.

Now select under the container definition choose *Custom* and click on *Configure*.



Container definition

Choose an image for your container below to get started quickly or define the container image to use.

sample-app
image : httpd:2.4
memory : 0.5GB (512)
cpu : 0.25 vCPU (256)

nginx
image : nginx:latest
memory : 0.5GB (512)
cpu : 0.25 vCPU (256)

tomcat-webserver
image : tomcat
memory : 2GB (2048)
cpu : 1 vCPU (1024)

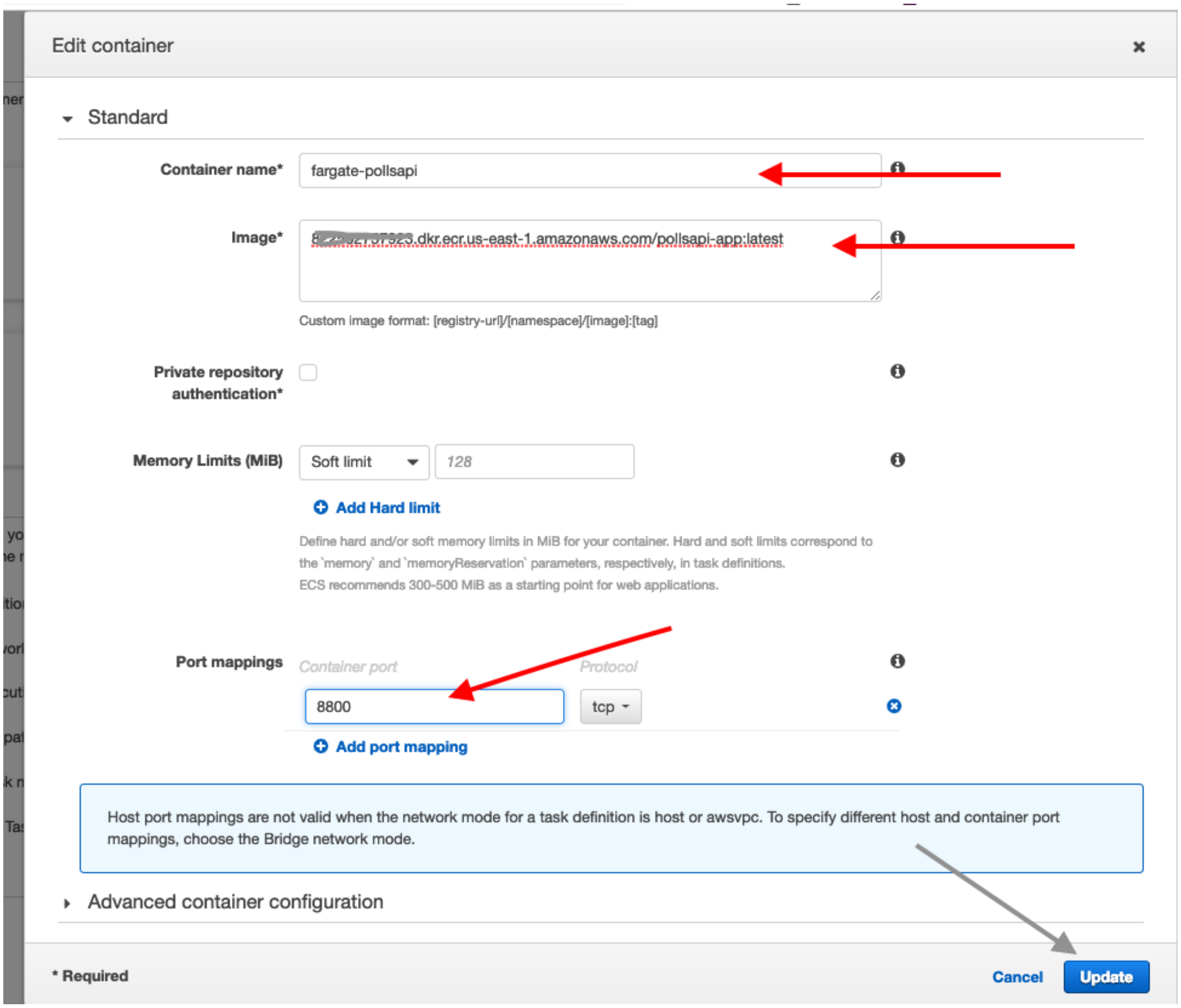
custom
image : --
memory : --
cpu : --

Task definition

A task definition is a blueprint for your application, and describes one or more containers through attributes. Some attributes are configured at the task level but the majority of attributes are configured per container.

Task definition name: first-run-task-definition
Network mode: awsvpc
Task execution role: Create new
Compatibilities: FARGATE
Task memory: 0.5GB (512)
Task CPU: 0.25 vCPU (256)

In the popup, enter a name for the container and add the URL to the container image. We should be able to get the URL from ECR. The format of the URL should be similar to the one listed below.



Edit container

Standard

Container name* fargate-pollsapi

Image* 8[redacted].dkr.ecr.us-east-1.amazonaws.com/pollsapi-app:latest

Custom image format: [registry-url]/[namespace]/[image]:[tag]

Private repository authentication* □

Memory Limits (MiB) Soft limit 128

Add Hard limit

Define hard and/or soft memory limits in MiB for your container. Hard and soft limits correspond to the `memory` and `memoryReservation` parameters, respectively, in task definitions.
ECS recommends 300-500 MiB as a starting point for web applications.

Port mappings

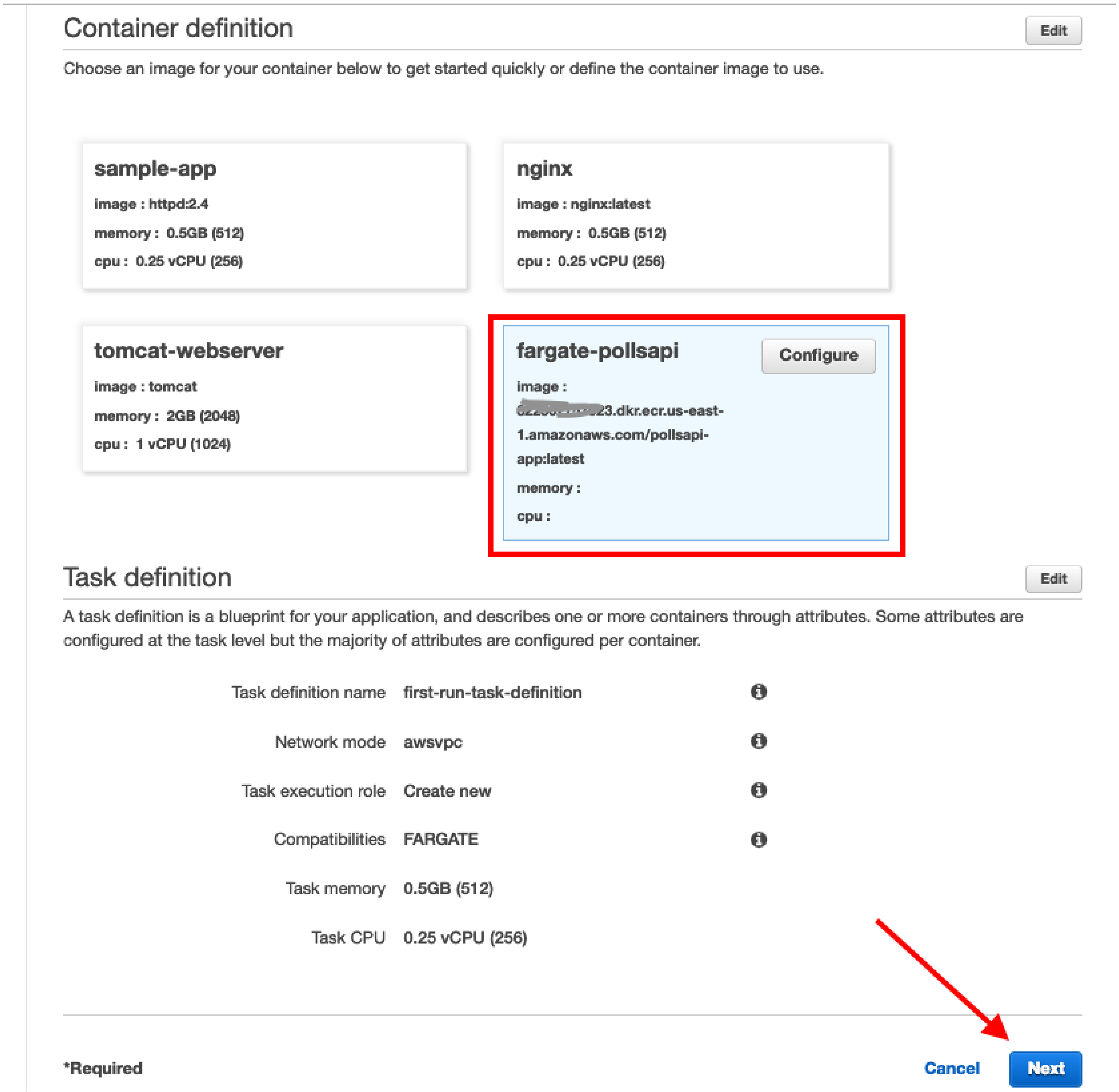
Container port	Protocol
8800	tcp

Add port mapping

Host port mappings are not valid when the network mode for a task definition is host or awsvpc. To specify different host and container port mappings, choose the Bridge network mode.

Advanced container configuration

* Required



Container definition

Choose an image for your container below to get started quickly or define the container image to use.

sample-app

image : httpd:2.4

memory : 0.5GB (512)

cpu : 0.25 vCPU (256)

nginx

image : nginx:latest

memory : 0.5GB (512)

cpu : 0.25 vCPU (256)

tomcat-webserver

image : tomcat

memory : 2GB (2048)

cpu : 1 vCPU (1024)

fargate-pollsapi

image : [redacted]23.dkr.ecr.us-east-1.amazonaws.com/pollsapi-app:latest

memory :

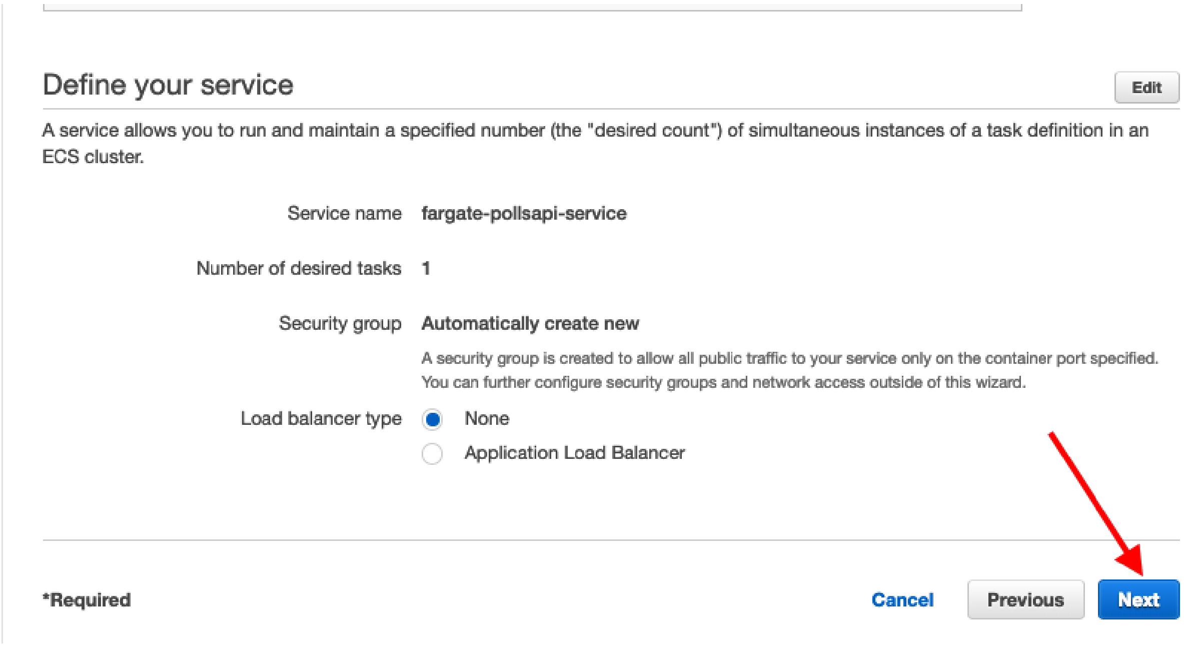
cpu :

Task definition

A task definition is a blueprint for your application, and describes one or more containers through attributes. Some attributes are configured at the task level but the majority of attributes are configured per container.

Task definition name	first-run-task-definition
Network mode	awsvpc
Task execution role	Create new
Compatibilities	FARGATE
Task memory	0.5GB (512)
Task CPU	0.25 vCPU (256)

*Required



Define your service

A service allows you to run and maintain a specified number (the "desired count") of simultaneous instances of a task definition in an ECS cluster.

Service name fargate-pollsapi-service

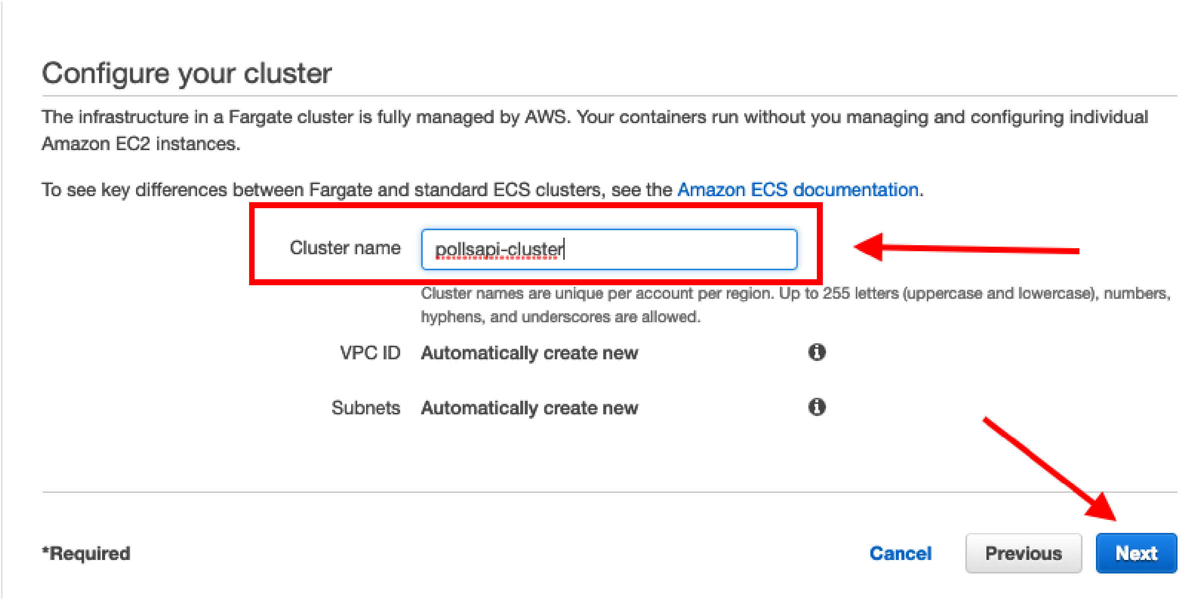
Number of desired tasks 1

Security group Automatically create new

A security group is created to allow all public traffic to your service only on the container port specified. You can further configure security groups and network access outside of this wizard.

Load balancer type ● None ○ Application Load Balancer

*Required



Configure your cluster

The infrastructure in a Fargate cluster is fully managed by AWS. Your containers run without you managing and configuring individual Amazon EC2 instances.

To see key differences between Fargate and standard ECS clusters, see the Amazon ECS documentation.

Cluster name pollsapi-cluster

Cluster names are unique per account per region. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.

VPC ID Automatically create new

Subnets Automatically create new

*Required

In the cluster section, give the cluster name.

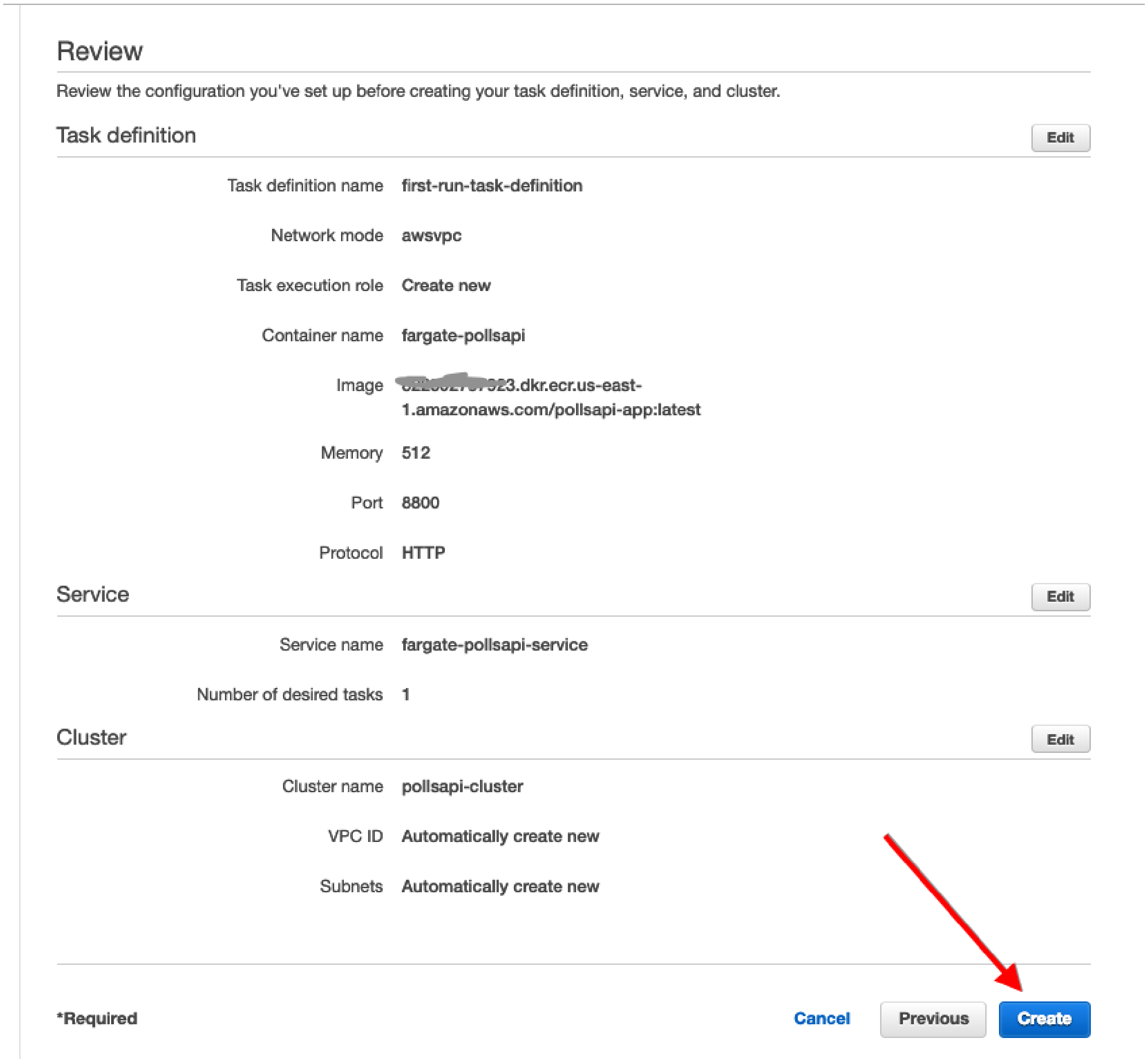
Now we can see the status of the service we just created. Wait for the steps to complete and then click on `View Service`.

Once on the services page, click on the Tasks tab to see the different tasks running for our application. Click on the task id.

Now let us go to the url in the public-ip with the port `http://3.88.173.94:8800`, we can see

to check logs we have to go to the `logs` tab in the services page

Now let us create an Aurora Serverless to link it with



Review

Review the configuration you've set up before creating your task definition, service, and cluster.

Task definition

Task definition name first-run-task-definition

Network mode awsvpc

Task execution role Create new

Container name fargate-pollsapi

Image [redacted]923.dkr.ecr.us-east-1.amazonaws.com/pollsapi-app:latest

Memory 512

Port 8800

Protocol HTTP

Service

Service name fargate-pollsapi-service

Number of desired tasks 1

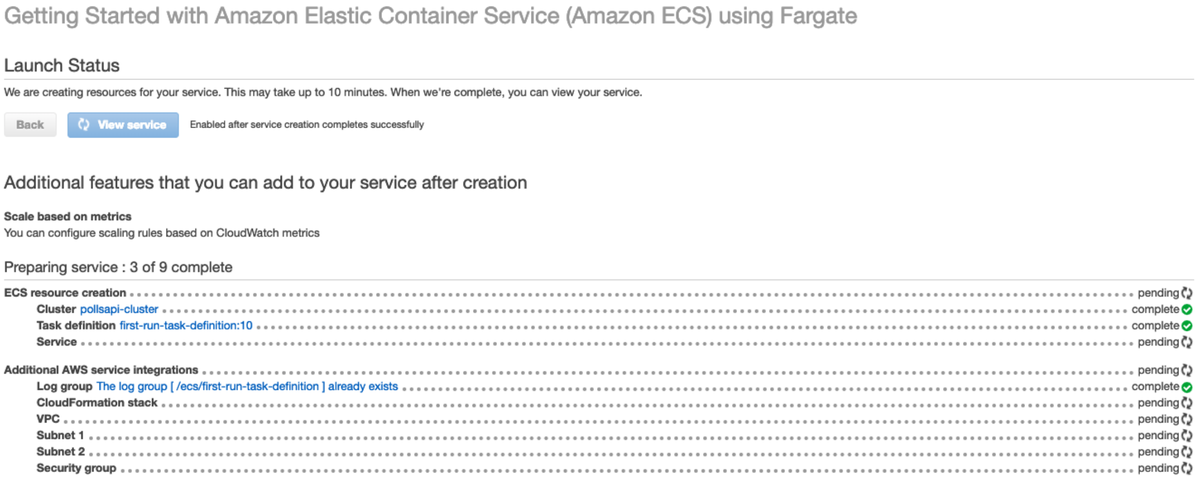
Cluster

Cluster name pollsapi-cluster

VPC ID Automatically create new

Subnets Automatically create new

*Required



Getting Started with Amazon Elastic Container Service (Amazon ECS) using Fargate

Launch Status

We are creating resources for your service. This may take up to 10 minutes. When we're complete, you can view your service.

Enabled after service creation completes successfully

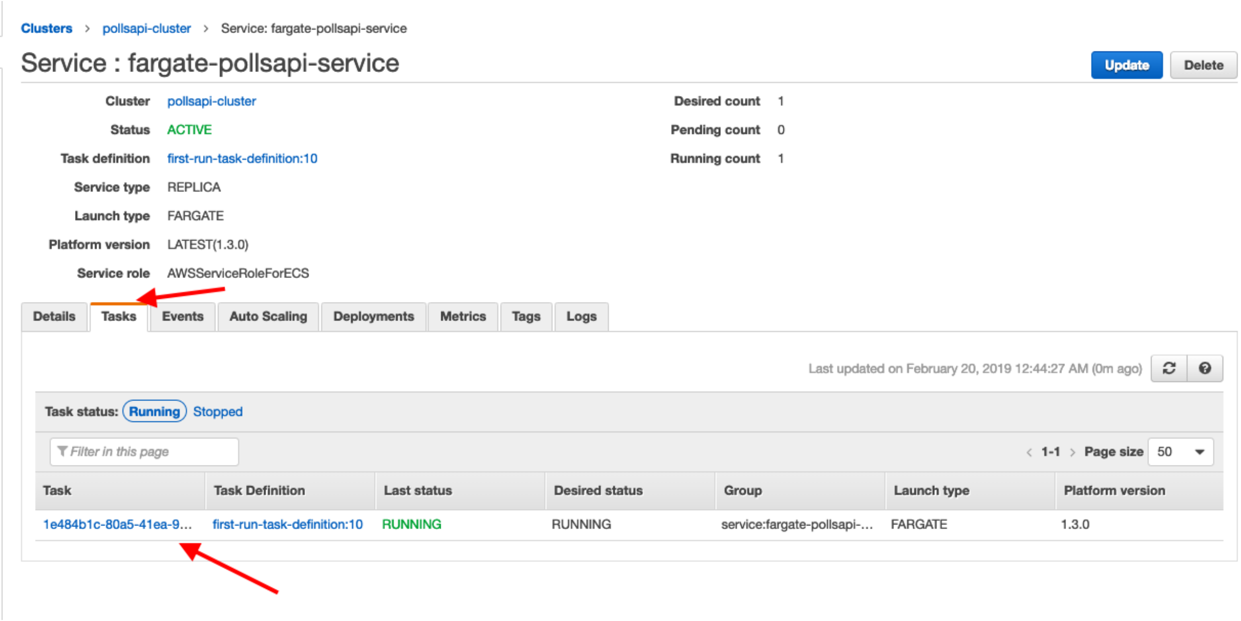
Additional features that you can add to your service after creation

Scale based on metrics
You can configure scaling rules based on CloudWatch metrics

Preparing service : 3 of 9 complete

```
ECS resource creation ................................................ pending
    Cluster pollsapi-cluster ......................................... complete
    Task definition first-run-task-definition:10 ..................... complete
    Service .......................................................... pending

Additional AWS service integrations .................................. pending
    Log group The log group [ /ecs/first-run-task-definition ] already exists ... complete
    CloudFormation stack ............................................. pending
    VPC .............................................................. pending
    Subnet 1 ......................................................... pending
    Subnet 2 ......................................................... pending
    Security group ................................................... pending
```



Clusters > pollsapi-cluster > Service: fargate-pollsapi-service

Service : fargate-pollsapi-service

Cluster pollsapi-cluster

Status ACTIVE

Task definition first-run-task-definition:10

Service type REPLICA

Launch type FARGATE

Platform version LATEST(1.3.0)

Service role AWSServiceRoleForECS

Desired count 1

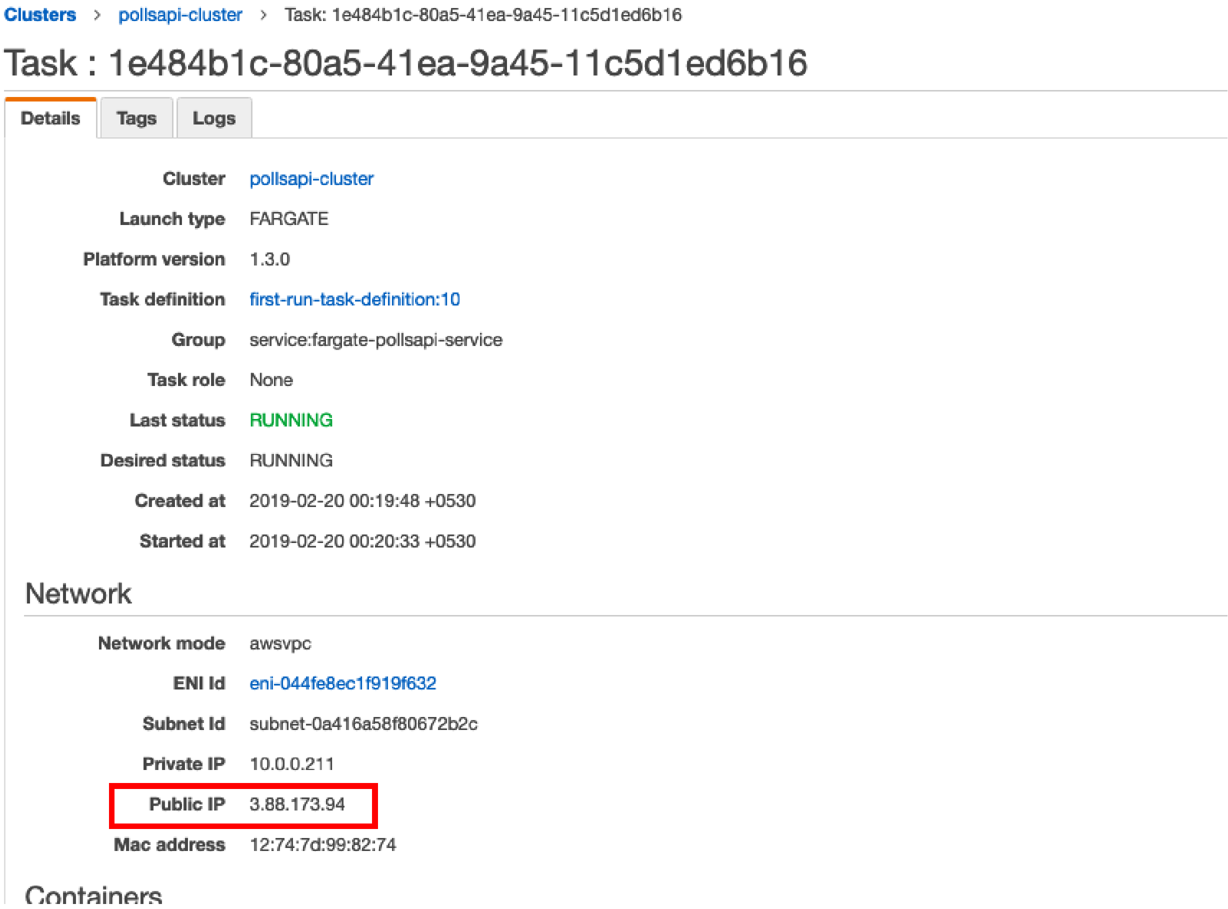
Pending count 0

Running count 1

Last updated on February 20, 2019 12:44:27 AM (0m ago)

Task status: Running Stopped

Task	Task Definition	Last status	Desired status	Group	Launch type	Platform version
1e484b1c-80a5-41ea-9...	first-run-task-definition:10	RUNNING	RUNNING	service:fargate-pollsapi-...	FARGATE	1.3.0



Clusters > pollsapi-cluster > Task: 1e484b1c-80a5-41ea-9a45-11c5d1ed6b16

Task : 1e484b1c-80a5-41ea-9a45-11c5d1ed6b16

Cluster pollsapi-cluster

Launch type FARGATE

Platform version 1.3.0

Task definition first-run-task-definition:10

Group service:fargate-pollsapi-service

Task role None

Last status RUNNING

Desired status RUNNING

Created at 2019-02-20 00:19:48 +0530

Started at 2019-02-20 00:20:33 +0530

Network

Network mode awsvpc

ENI Id eni-044fe8ec1f919f632

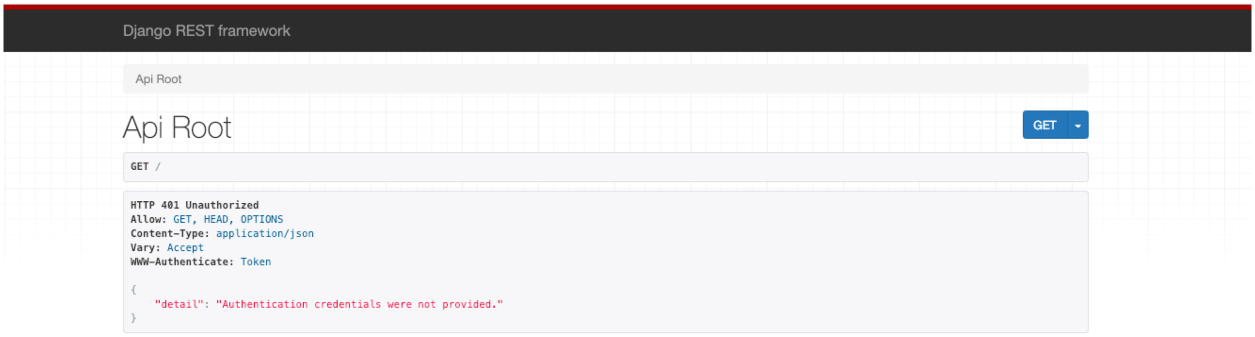
Subnet Id subnet-0a416a58f80672b2c

Private IP 10.0.0.211

Public IP 3.88.173.94

Mac address 12:74:7d:99:82:74

Containers



```
GET /
```

```
HTTP 401 Unauthorized
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept
WWW-Authenticate: Token

{
    "detail": "Authentication credentials were not provided."
}
```

4.4 Setup Serverless MySQL Database

To set up Aurora serverless DB follow https://www.agiliq.com/blog/2019/01/complete-serverless-django/#setup-serverless-mysql-database

4.5 Connect Our App to MySQL DB

While creating Aurora-serverless **make sure that Fargate and Aurora are in same VPC**

To connect our Django App to aurora db, follow https://www.agiliq.com/blog/2019/01/complete-serverless-django/#connect-django-to-mysql-db

After configuring our `settings.py` file should have a similar database config

```
...

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'pollsdb', # dbname
        'USER': 'polls_admin', # master username
        'PASSWORD': 'pollsadmin', # master password
        'HOST': 'pollsapi-cluster.cluster-chcxxxxx.us-east-2.rds.amazonaws.com', #␣
↪Endpoint
        'PORT': '3306',
    }
}
...
```

4.5.1 Update Security Group Endpoint

Update Security Group Endpoint of Aurora and add Security Group of Fargate in the inbound rules, follow https://www.agiliq.com/blog/2019/01/complete-serverless-django/#update-security-group-endpoint

4.5.2 Setup the Database

We will write a command to create the database. To setup the database follow,

```
$ cd polls
$ mkdir management
$ cd management
$ touch __init__.py
$ mkdir commands
$ cd commands
$ touch __init__.py
$ touch create_db.py
```

```
# polls/management/commands/create_db.py
import sys
import logging
import MySQLdb

from django.core.management.base import BaseCommand, CommandError
from django.conf import settings

rds_host = 'pollsapi-cluster.cluster-chc62yjp918f.us-east-2.rds.amazonaws.com'
db_name = 'pollsdb'
user_name = 'polls_admin'
password = 'pollsadmin'
port = 3306

logger = logging.getLogger()
logger.setLevel(logging.INFO)


class Command(BaseCommand):
    help = 'Creates the initial database'

    def handle(self, *args, **options):
        print('Starting db creation')
        try:
            db = MySQLdb.connect(host=rds_host, user=user_name,
                                 password=password, db="mysql", connect_timeout=5)
            c = db.cursor()
            print("connected to db server")
            c.execute("""CREATE DATABASE pollsdb;""")
            c.execute(
                """GRANT ALL PRIVILEGES ON db_name.* TO 'polls_admin' IDENTIFIED BY
↪'pollsadmin';""")
            c.close()
            print("closed db connection")
        except:
            logger.error(
                "ERROR: Unexpected error: Could not connect to MySql instance.")
            sys.exit()
```

Now let us create another command to *create admin*, follow

```
$ cd polls
$ mkdir management
$ cd management
```

(continues on next page)

(continued from previous page)

```
$ touch __init__.py
$ mkdir commands
$ cd commands
$ touch __init__.py
$ touch create_admin_user.py
```

```
# polls/management/commands/create_admin_user.py
import sys
import logging

from django.core.management.base import BaseCommand, CommandError
from django.contrib.auth.models import User
from django.conf import settings


class Command(BaseCommand):
    help = 'Creates the initial admin user'

    def handle(self, *args, **options):
        if User.objects.filter(username="admin").exists():
            print("admin exists")
        else:
            u = User(username='admin')
            u.set_password('adminpass')
            u.is_superuser = True
            u.is_staff = True
            u.save()
            print("admin created")
        sys.exit()
```

this command will create the admin user if it does not exists

Now next create a shell script file with name `start.sh`, and write the following

```
$ touch start.sh
```

```
#!/bin/sh
python manage.py create_db
python manage.py migrate
python manage.py create_admin_user
python manage.py runserver 0.0.0.0:8800
exec "$@"
```

And give it permissions

```
$ chmod +x start.sh
```

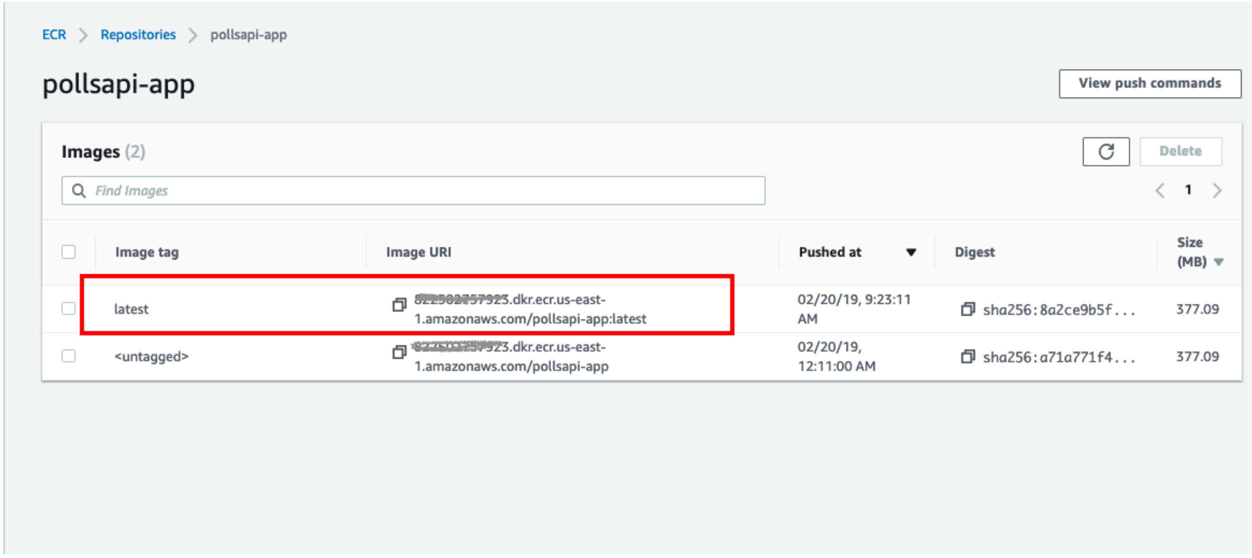
And Now update the `Dockerfile`

Now lets push the updated container image to ECS by following the `Push Commands`.

With Fargate, our containers are always started with the latest ECS image and Docker version.

Let us go to the `http://3.88.173.94:8800/admin`, we can see Now we can see that we can login and that our Database connection is established fine.

Now our Django app is running in AWS Fargate and used Aurora Serverless as the DB.

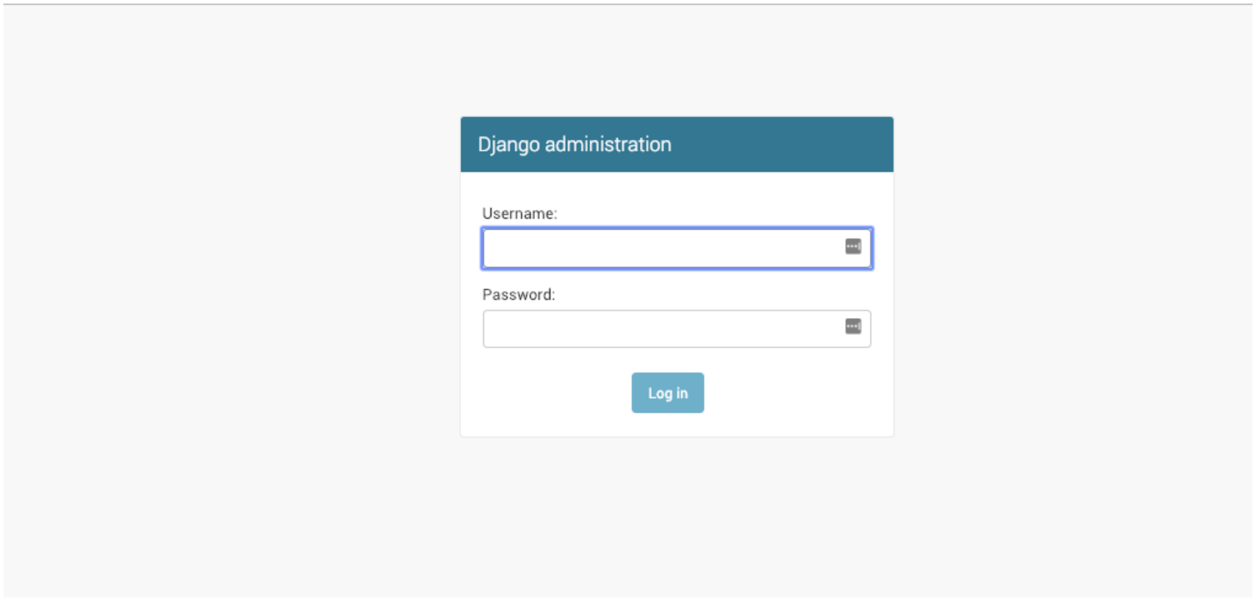


ECR > Repositories > pollsapi-app

pollsapi-app

Images (2)

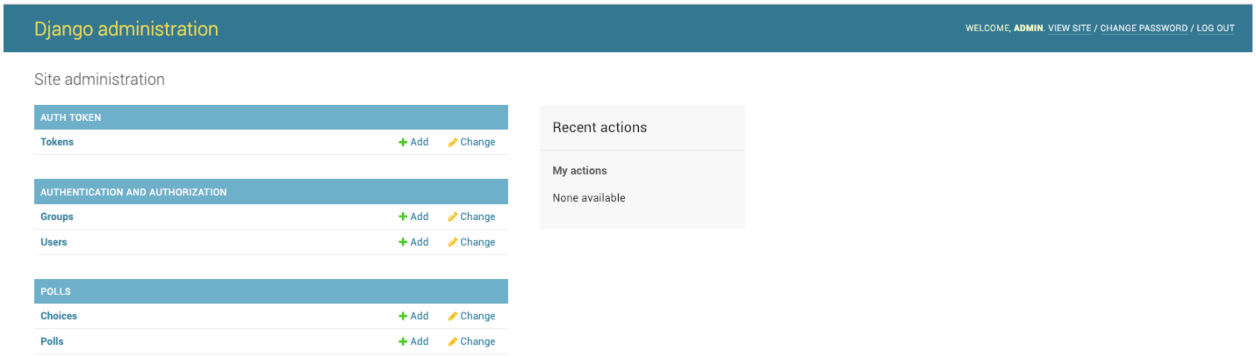
□	Image tag	Image URI	Pushed at ▼	Digest	Size (MB) ▼
□	latest	[redacted].dkr.ecr.us-east-1.amazonaws.com/pollsapi-app:latest	02/20/19, 9:23:11 AM	sha256:8a2ce9b5f...	377.09
□	<untagged>	[redacted].dkr.ecr.us-east-1.amazonaws.com/pollsapi-app	02/20/19, 12:11:00 AM	sha256:a71a771f4...	377.09



Django administration

Username:

Password:



Django administration

Site administration

AUTH TOKEN		
Tokens	+ Add	Change

AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	Change
Users	+ Add	Change

POLLS		
Choices	+ Add	Change
Polls	+ Add	Change

Recent actions

My actions

None available

CHAPTER 5

Indices and tables

- genindex
- modindex
- search