# **Refresher Homework**

**Samik Biswas**

**014500889**

# **Direct Link to HTML App:** <https://youtu.be/2JOYcE7nyXE>

**Direct Link to JavaScript App:** https://youtu.be/pxY6XCnhvxI

# JavaScript (ES6)

## Introduction

I have created an application where the we will have different students and we want to get the weather for each one of them. Through the development of this application I have made use of different JS features that include object oriented principles, inheritance, property assignment, Promises, async await, API calls and many other functionalities that are part of today’s JS based web development process.

## Problem Statement

We will receive different student objects as part of the **input** variable and then if the location of those students is provided, we will need to fetch the weather for them and show them in the console. We are also saving the last saved weather in a file. If we do not get the location details of the students, we display the weather of the place from where the application is being accessed.

### 1. Object and Classes with Object.assign demonstrating inheritance using subclasses in

### JavaScript with method overriding, class static method, use of require, strict mode, let

I have created a general class called person having 2 arguments. Later the specialized class Student inherits Person class. In the Student subclass I have also override the print method to print the name of the student who got initialized. The Student class also contains a static method to be called once the processing of the Student objects is complete. Once student1 and student2 were initialized, I have used the Object.assign method to assign extra properties to the objects.

"use strict"

const{*getMyWeather*}=*require*('./getWeather')

//*parent class*

class Person {

    constructor(age, address) {

        this*.*age = age;

        this*.*address = address

    }

    //*general method in parent class to be overridden by the child class*

*print* = () => {

*console.log*('Person class initiated')

    }

}

//*child class demonstrating inheritance*

class Student *extends* Person {

    constructor(firstName, lastName, age, address) {

        super(age, address)

        this*.*firstName = firstName;

        this*.*lastName = lastName

    }

    //*method overriding*

*print* = () => {

*console.log*(`Student ${this*.*firstName} initiated`)

*console.log*(this)

    }

*// class static method*

*static* *finish* = () => {

*console.log*('The processing is done')

    }

}

*// use of let keywork since we will be assigning new properties. These variables have local scope*

letstudent1=new*Student*('Samik','Biswas',29,'101 E San Fernando');

letstudent2=new*Student*('Arpan','Debnath',27,'9 Aghore Sarani');

//*adding new properties using the Object.assign*

student1 = Object*.assign*(student1, { Concentration: 'Web Development' })

student2 = Object*.assign*(student2, { Concentration: 'Safety Analysis' })

*student1.print*()

*student2.print*()

Output:



### 2. call, apply, bind, Use of arrow functions, split, slice, includes, typeof, regular expressions, rest operator, closure, array destructuring, const and var

We have a main function called process that takes in an argument input and process it further. The argument input contains a ‘,’ separated string about the details of the input, the next argument metadata contains the length number of objects and then the same number of objects follow the input array. The array split function was first used on the first argument of the input and then checks if this is an input of student objects. Also, regular expression check has been implemented to check for any errors in the first argument. Later, array destructuring was used to get the type and metadata from the array. Using the length from metadata, we slice the array to get all the student objects in the students array. For each student we check if that is of type object and an instance of Student class. If the condition satisfies, we call the getWeather function using both call and apply methods and passing extra arguments appropriately. We have also made use of closure to maintain a global counter called processCount that maintains the number of times the process function was called. Later we extract this count and display it on the console.

//*Processing function*

/\*\*

*\* 1. Write a function that calls the getWeather function*

\*/

*// use of rest operator for n number of arguments*

var*getWeatherForPerson*=function(...*args*){

*getMyWeather*(*this.firstName*,*this.lastName*,*this.age*,*this.address*,args)

}

//*input received as input. Var keyword used to give this a global scope*

varinput=['Student, length, objects',{*length*:2},student2,student1]

//*closures, use of arrow function*

const*process*=((*input*)=>{

varprocessCount=0;

    //*get rid of the extra data from the input. Use of Split function*

input[0]=input[0]*.split*(',')[0]

    //*check if the input contains any special characters and the word Student. Use of Regular Expression*

if(/[*~`!#$%\^&\*+=\-\[\]\\';,/{}|\\":<>\?*]/g*.test*(input[0])&&!*input.includes*('Student')){

*console.log*('Invalid Input')

*return*

}

*// Array Destructuring*

const[*type*,*metadata*]=input

    //*further validations before starting the processing*

if(*metadata.length*>=0){

        //*Getting the students object from the input using the Array.slice() functrion*

const*students*=*input.slice*(2,2+*metadata.length*)

*students.forEach*((*student*)=>{

            //*checking if the received input is an object of Student class using typeOf and instanceOf*

if(typeofstudent==='object'&&studentinstanceofStudent){

*console.log*('Sending for weather')

*getWeatherForPerson.call*(student,'Call argument 1','Call argument 2')

*getWeatherForPerson.apply*(student,['Apply argument 1','Apply argument 2'])

}else{

*return*

}

})

        //*calling static method after the process is finished*

*Student.finish*()

}

*return*function(){processCount+=1;*return*processCount}

})(input)

//*start the processing*

*process*(input)

*process*(input)

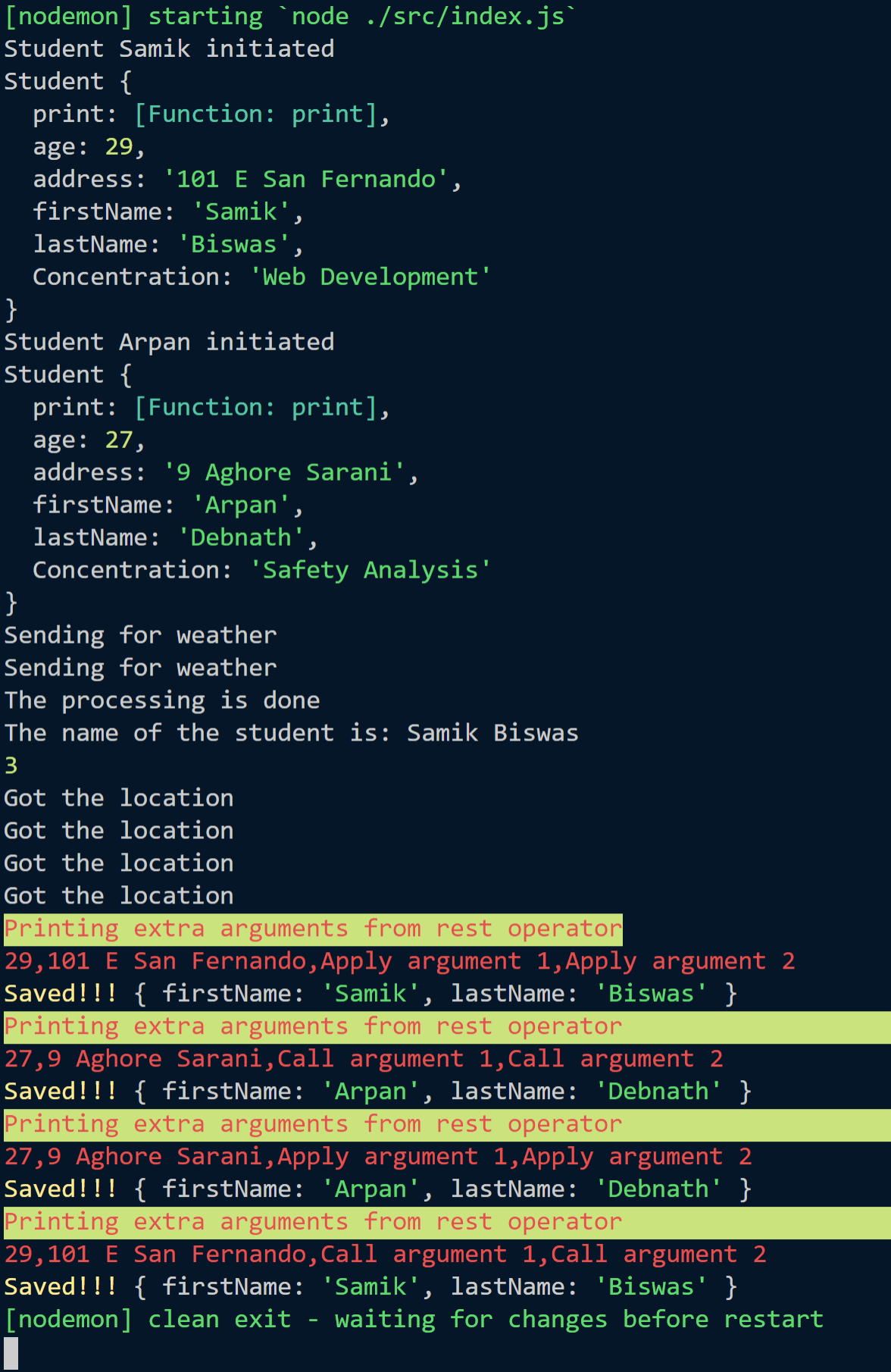
//*getting the final process count as part of closures demo.*

//*Since processing was run 3 times, processCount should be 3*

const*processCount*=*process*(input)

*console.log*(processCount)

Output:



### 3. Callback functions, async await, default arguments, JSON.stringify, making external API call using axios

In this function, we take in the location and name of the user and few other additional arguments. Using the location of the user we fetch their weather and then save it to a file. In this function we make use of async await feature of Javascript, where we wait till we have the weather for the location provided and then use the JavaScript fs module to write it to a file after converting the JSON response to a JSON string representation using the JSON.stringify function. We have also used default arguments in the function parameters, in case the caller does not provide any arguments these defaults will be used. The rest operator in the function parameter also ensures that we can accept n number of extra arguments and we later print them to test the functionality.

const*axios*=*require*('axios')

const*chalk*=*require*('chalk')

const*fs*=*require*('fs')

//*arrow function, default arguments, rest operator, async function*

const*getWeather*=*async*(*latitudes*='37.8267',*longitudes*='-122.4233',*user*={*firstName*:'John',*lastName*:'Doe'},...*args*)=>{

*try*{

        //*use of await*

const*res*=(*await axios.get*(`*https://api.darksky.net/forecast/93973d2448bd32f2d0e84834772a7d62/*${latitudes}*,*${longitudes}`))*.data*

        //*object destructuring*

        //*use of const - the values of const cannot be changed*

const{*latitude*,*longitude*,*daily*}=res

        //*spread, JSON.stringify,callback function*

*fs.writeFile*('weather.txt',*JSON.stringify*({latitude,longitude, …daily,...user}),(*error*)=>{

if(error)*throw*error;

*console.log*(*chalk.yellowBright*('Saved!!!'),user);

});

        //*use of let - The value will change with each extra argument and has local scope*

*console.log*(*chalk.bgYellow*(*chalk.redBright*('Printing extra arguments')))

*for*(letargofargs)*console.log*(*chalk.redBright*(arg))

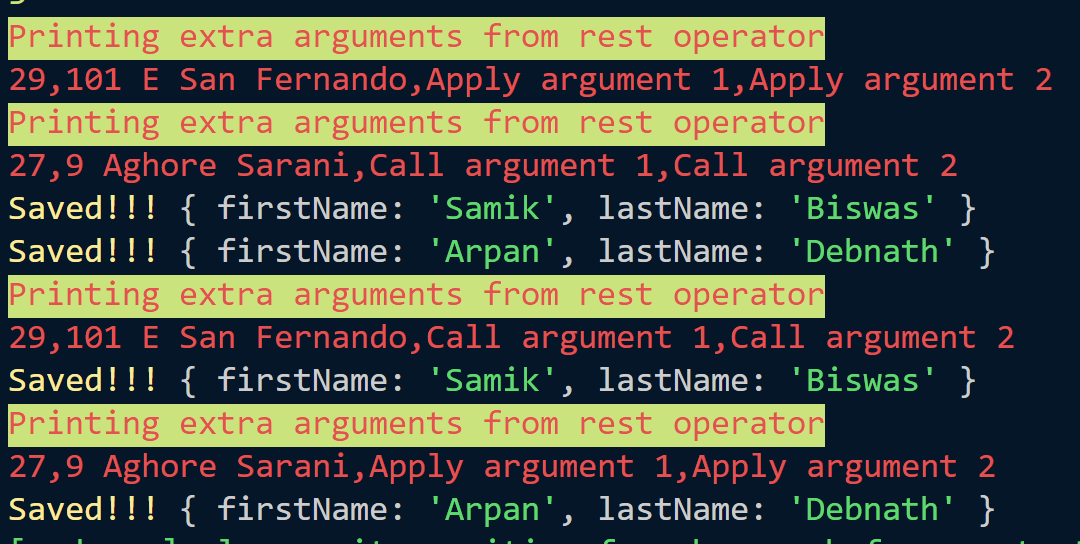
}*catch*(error){

*console.log*(*chalk.redBright*(error))

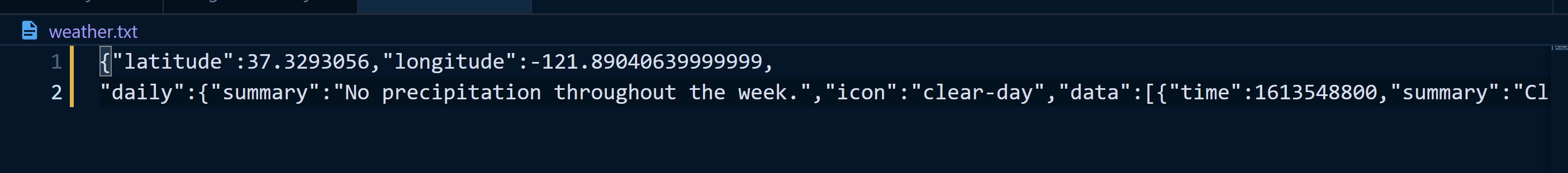
}

}

Output:



Weather File:



### 4. Promises using then catch

This function will get the current location of the user and then call the getWeather function. This function uses JavaScript promises to get the location data and when the location data is received, the callback function provided in the then block will execute and send the location of the user to get the weather for the same. Since we cannot move forward without getting the location of the user, Promises will be the best way forward. In case there is an error fetching the location, the catch block will be activated and the getLastWeather function will be called.

//*capturing promises using then catch block*

const*getMyWeather*=(*firstName*,*lastName*,...*args*)=>{

    //*JS promises*

*axios.post*('https://www.googleapis.com/geolocation/v1/geolocate?key=AIzaSyCodTFr6QoE\_rfpdpfUszzwPSrLhTjnNgc')*.then*((*data*)=>{

const{*lat*,*lng*}=*data.data.location*

*console.log*('Got the location')

*getWeather*(lat,lng,{firstName,lastName},args)

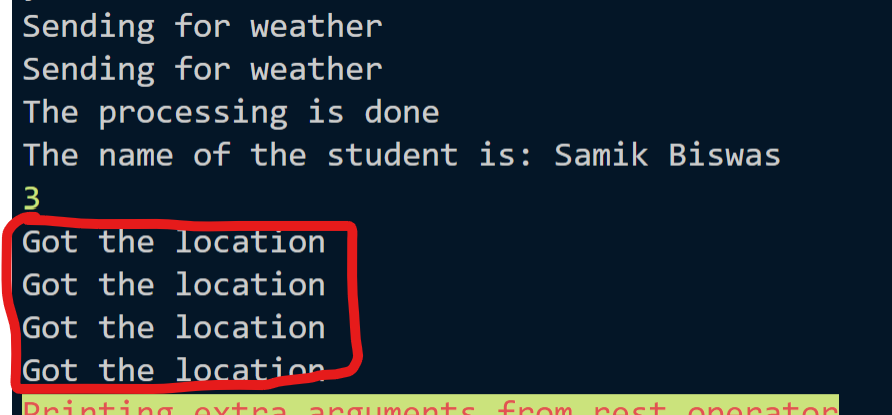
})*.catch*((*error*)=>{

*getLastWeather*()

})

}

Output:



### 5. Json.parse(), object destructuring, module.exports

In case there was some error getting the weather for the locations provided, we return the last weather searched. The last weather searched is already saved in the file weather.txt in JSON string format. We retrieve the details and then convert it into JSON object using JSON.parse() function and then display on the console. Finally the function getWeather was exported from this file using the module.exports functionality.

//*JSON.parse*

const*getLastWeather*=()=>{

*try*{

*fs.readFile*('weather.txt',(*error*,*data*)=>{

if(error)*throw*error;

const*jsonData*=*JSON.parse*(data)

const{*latitude*,*longitude*,*firstName*,*lastName*}=jsonData

*console.log*(`*The location was searched by*${*chalk.blueBright*(firstName)}${*chalk.blueBright*(lastName)}*for latitude:*${*chalk.cyanBright*(latitude)}*and longitude:*${*chalk.cyanBright*(longitude)}`)

});

}*catch*(error){

*console.log*(error)

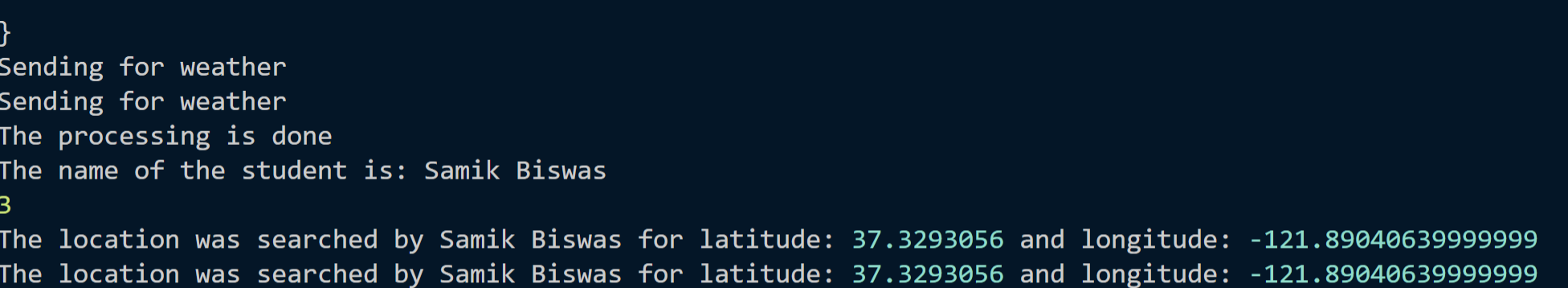
}

}

//*exports*

module*.*exports = { getMyWeather }

Output:



### 6. bind

In the below code snippet I have used the javascript bind functionality to create an function called getName that is not part of any object. I have used the JS bind functionality to bind this function to a student object and store it in getNameBound function. Later when I called getNameBound function, I get the name from the object that it was bound to.

//*define external functions to be later bind with objects*

const*getName*=function(){

*console.log*(`*The name of the student is:*${*this.firstName*}${*this.lastName*}`)

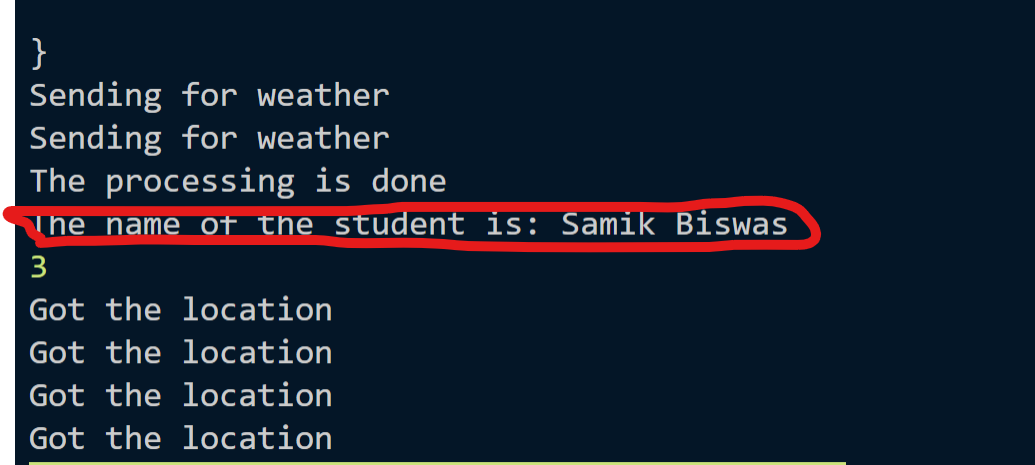
}

//*binding the getName function to the objects*

const*getNameBound*=*getName.bind*(student1)

*getNameBound*()

Output:



# HTML:

## Introduction:

A simple HTML based web app to get you current location.

## Problem Statement:

Once the user has entered his email password and name to log in, we will save the Name and the email in localStorage until explicitly log out and the password is stored in the session storage. Once all the details have been entered, we return the current latitude and longitude of the user using the Geolocation API. The application was finally deployed on AWS ECS after containerizing it using docker.

## Concepts Used:

1. **Events**: These are actions that occur in the system which can be captured and responded appropriately. I my app I have used the onClick and onSubmit events to run different functions accordingly.
2. **localStorage**: We can store different data either from the user or those received from 3rd party API calls. When stored into localStorage these data points will remain stored until explicitly deleted.
3. **sessionStorage**: Data can also be stored as part of sessionStorage but will only be accessible in the current tab only.

Validations have been performed using different HTML validation options such as email, patterns, required etc.

### localStorage and sessionStorage, Geolocation API call

1. <script>
2. //*get the user from localStorage*
3. const*user*=*JSON.parse*(*localStorage.getItem*('user'))
4. if (user) {
5. *document.getElementById*('name')*.*value = *user.*name
6. *document.getElementById*('email')*.*value = *user.*email
7. }
8. //*get the password from sessionStorage*
9. const*password*=*sessionStorage.getItem*('password')
10. if (password) {
11. *document.getElementById*('password')*.*value = JSON*.*parse(password)*.*password
12. }
13. const*handleSubmit*=(*e*)=>{
14. *localStorage.setItem*('user',*JSON.stringify*({
15. 'name':*document.getElementById*('name')*.value*,
16. 'email':*document.getElementById*('email')*.value*
17. }))
18. *sessionStorage.setItem*('password',*JSON.stringify*({
19. 'password':*document.getElementById*('password')*.value*
20. }))
21. //*get the response from the geolocation API*
22. letxhr=new*XMLHttpRequest*();
23. *xhr.open*('POST',"https://www.googleapis.com/geolocation/v1/geolocate?key=AIzaSyCodTFr6QoE\_rfpdpfUszzwPSrLhTjnNgc",*true*);
24. *xhr.send*();
25. *xhr.onreadystatechange*=processRequest;
26. function*processRequest*(*e*){
27. if(*xhr.readyState*==4&&*xhr.status*==200){
28. letresponse=*JSON.parse*(*xhr.responseText*);
29. const*latitude*=*document.getElementById*('latitude')
30. const*longitude*=*document.getElementById*('longitude')
31. //*display the coordinates*
32. *latitude.innerText*=`*Latitude:*${*response.location.lat.toFixed*(2)}`
33. *longitude.innerText*=`*Longitude:*${*response.location.lng.toFixed*(2)}`
34. }
35. }
36. }
37. const*handleLogout*=()=>{
38. *localStorage.removeItem*('user')
39. *sessionStorage.removeItem*('password')
40. *document.getElementById*('name')*.value*=""
41. *document.getElementById*('password')*.value*=""
42. *document.getElementById*('email')*.value*=""
43. *document.getElementById*('latitude')*.innerText*=""
44. *document.getElementById*('longitude')*.innerText*=""
45. }
46. </script>

### HTML based validations, events

<body>

    <div *class*="container">

        <form *onsubmit*="*handleSubmit*(); *return* false;">

            <div *class*="form-group">

                <label *for*="name">Name</label>

                <input *type*="text" *class*="form-control" *id*="name" *aria-describedby*="name" *placeholder*="Enter Name"

*required* *autofocus*>

            </div>

            <div *class*="form-group">

                <label *for*="password">Password</label>

                <input *type*="password" *class*="form-control" *id*="password" *aria-describedby*="password"

*placeholder*="Enter Password" *required* *minlength*="8" *pattern*="^(?=.\*?[A-Z])(?=.\*?[a-z])(?=.\*?[0-9])(?=.\*?[#?!@$%^&\*-]).{8,}$">

            </div>

            <div *class*="form-group">

                <label *for*="email">Email</label>

                <input *type*="text" *class*="form-control" *id*="email" *aria-describedby*="emailHelp"

*placeholder*="Enter Email" *required*>

                <small *id*="emailHelp" *class*="form-text text-muted">We'll never share your email with anyone

                    else.</small>

            </div>

            <p *id*="latitude"></p>

            <p *id*="longitude"></p>

            <button *type*="submit" *class*="btn btn-warning" *id*="submit" *name*="submit" *value*="submit">

                Login

            </button>

            <button *class*="btn btn-danger" *id*="logout" *name*="logout" *value*="logout" *onclick*="*handleLogout*(); *return* false;">

                Log Out

            </button>

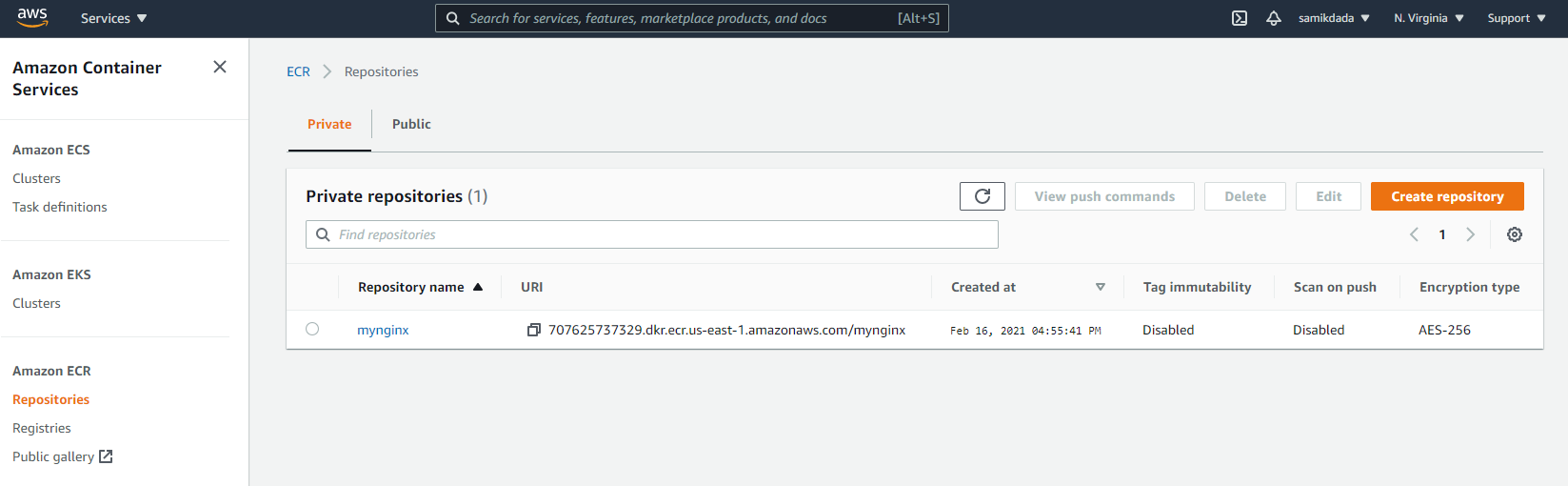
        </form>

    </div>

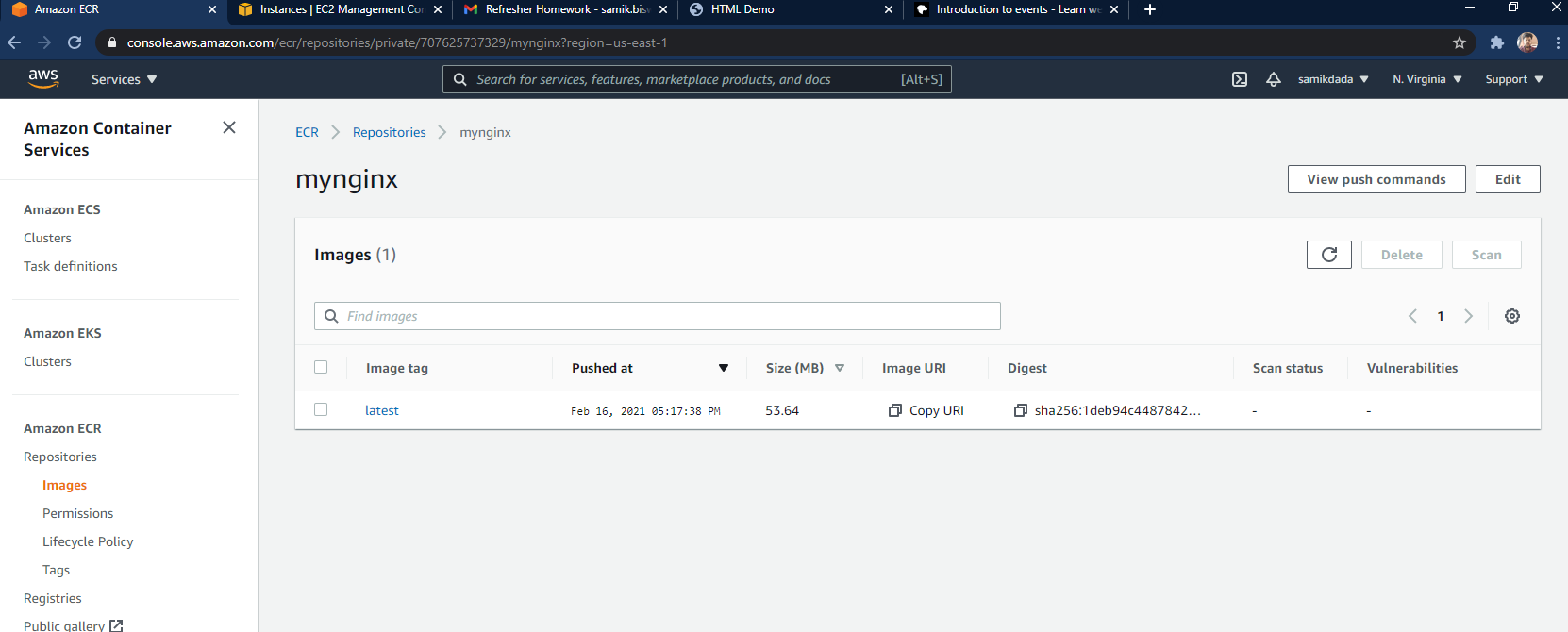
</body>

## AWS Console Screenshots

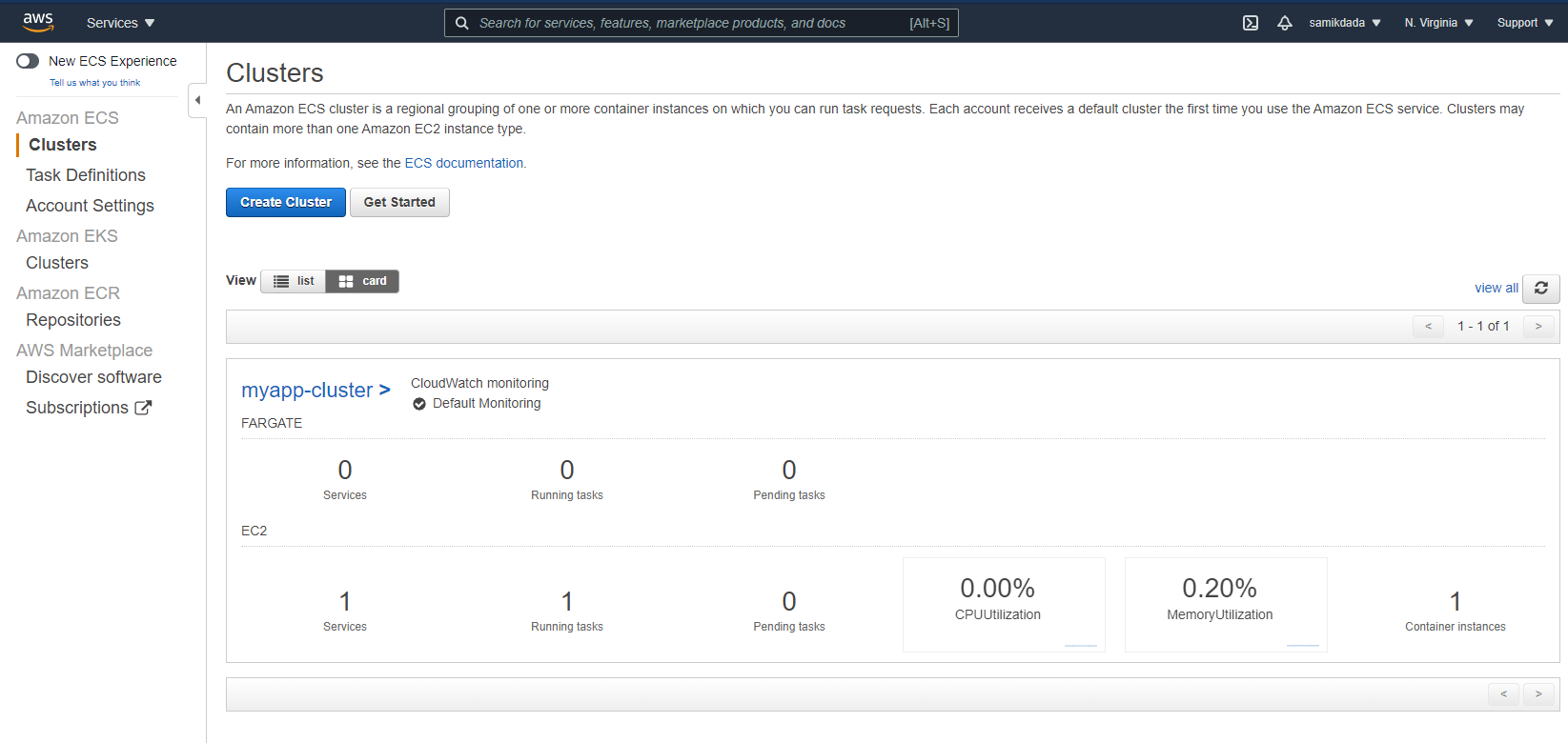
1. **Image Repository**



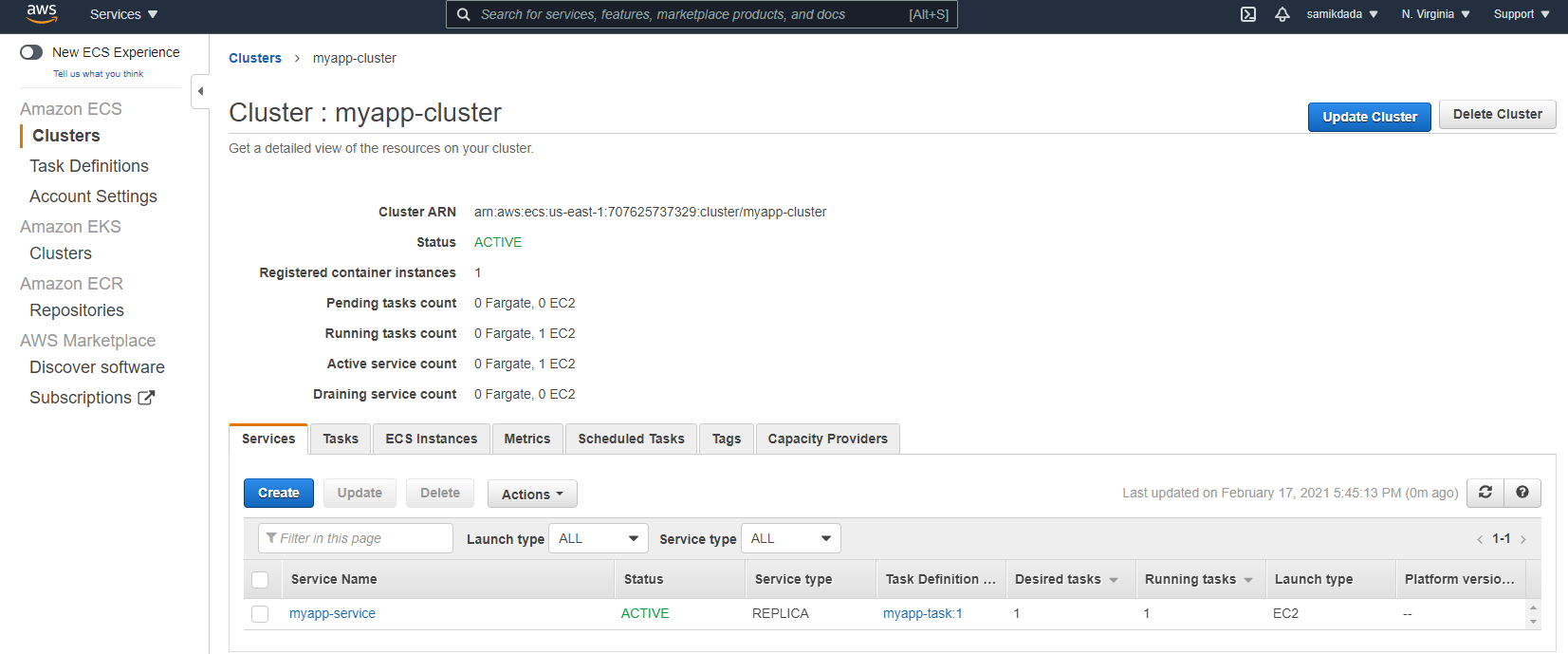
1. Pushed Image



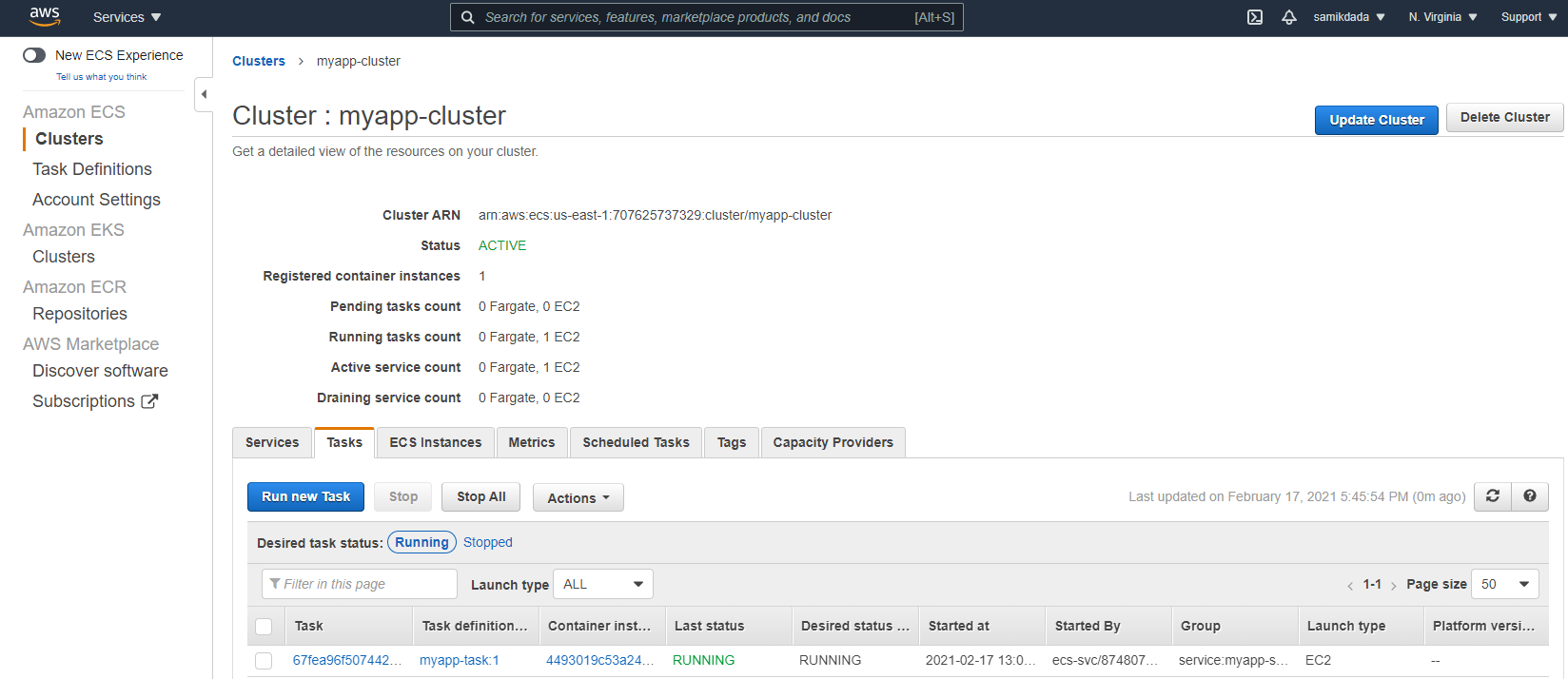
1. **ECS Cluster**



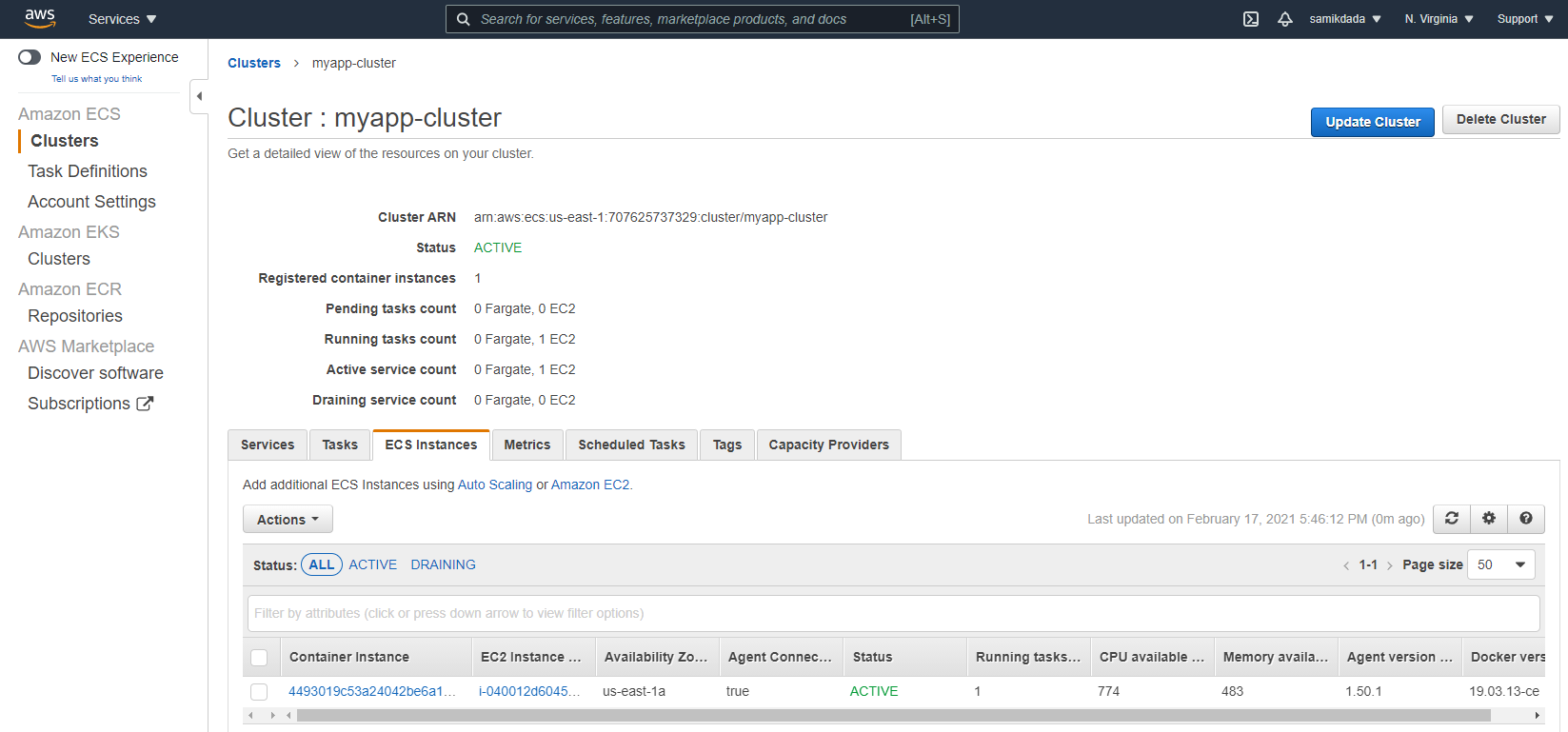
1. **ECS Service**



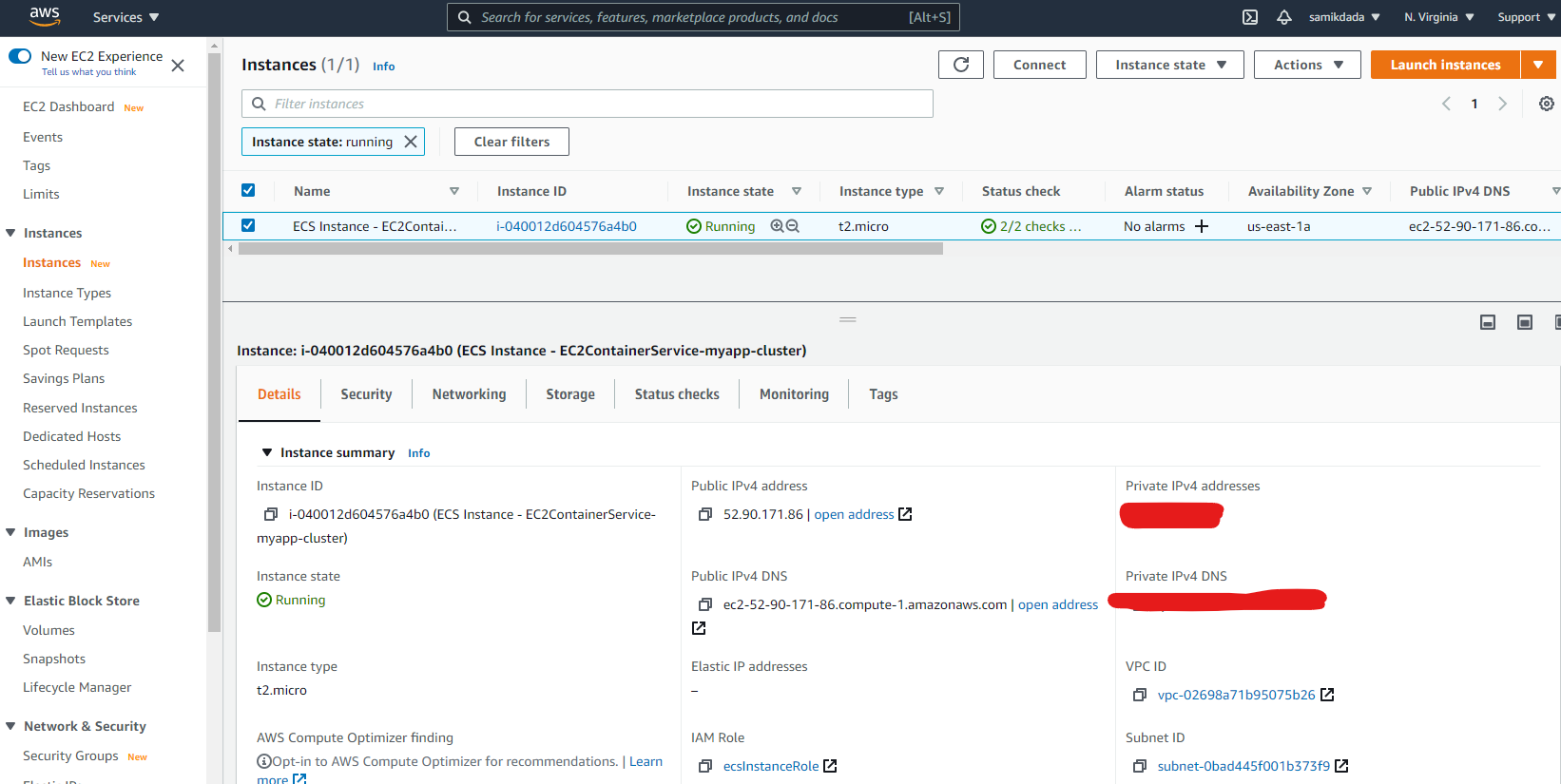
1. **ECS Task**



1. **EC2 instance created by ECS**



1. **EC2 instance details**



1. **Running instance of the app:**

