

Activity: Building an Amazon ECS Service with an ASP.NET Website

Overview

In this hands-on exercise, you will create an Amazon Elastic Container Service (Amazon ECS) service named WeatherSite that consists of an ASP.NET website and a .NET Web API. The API will use an Amazon DynamoDB table to retrieve weather data. The API will be deployed to an Amazon ECS service and the website to another Amazon ECS service.

You can perform this lab in two environments:

- On your local machine using Windows, MacOS, or Linux
- On an [AWS Cloud9](#) environment in the cloud

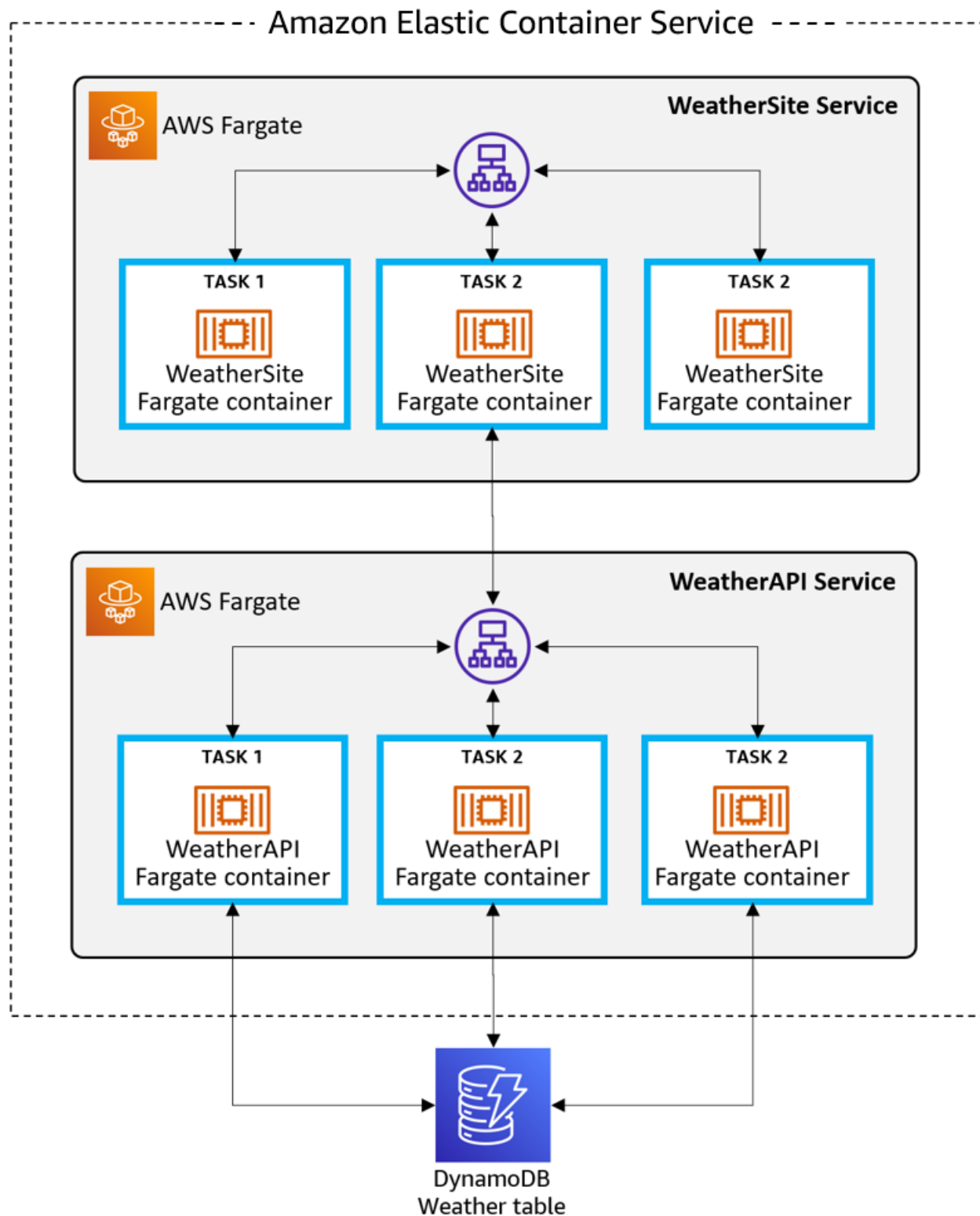
Duration

90 minutes

Here is what you will do

- You will begin by creating a DynamoDB table with some weather data. Next, you will create a .NET Web API project that retrieves data from the table.
- After testing the Web API locally, you will deploy it and create an Amazon ECS service on AWS Fargate using the AWS deployment tool for .NET CLI. Then, you will create an ASP.NET web site that accesses the API.
- After testing the web site locally, you will create another Amazon ECS service on Fargate using the AWS deployment tool for .NET CLI.
- You will test your Amazon ECS service in the cloud by accessing the website, confirming data is retrieved from the API, and monitoring activity in the AWS console.
- You will add data to the DynamoDB table, add another city to the website, and deploy the updates.
- Finally, you will delete your AWS resources.

Activity: Building an Amazon ECS Service with an ASP.NET Website



Activity: Building an Amazon ECS Service with an ASP.NET Website

Section 1: Set up an AWS environment

In this section, you will set up your AWS environment.

1. Obtain an [AWS account](#). Do not use a Production account.
2. Sign in to the [AWS Management Console](#) and choose an AWS Region to work in that [supports Amazon ECS on Fargate](#) and [supports DynamoDB](#).
3. If you are using your local machine for this activity, proceed to **Section 2: Set up the development environment**.
4. Create an AWS Cloud9 environment.
 - a. In the AWS Management Console, navigate to **AWS Cloud9** and choose **Create environment**.
 - b. Name the environment **FargateLab**, add an optional description, choose **New EC2 instance**, and choose **m5.large** for the instance type. Keep the defaults for the other options. [Platform: **Amazon Linux 2**, Timeout: **30 minutes**, Connection: **AWS Systems Manager (SSM)**]

AWS Cloud9 > Environments > Create environment

Create environment [Info](#)

Details

Name
FargateLab
Limit of 60 characters, alphanumeric, and unique per user.

Description - *optional*
Lab environment
Limit 200 characters.

Environment type [Info](#)
Determines what the Cloud9 IDE will run on.

☒ **New EC2 instance**
Cloud9 creates an EC2 instance in your account. The configuration of your EC2 instance cannot be changed by Cloud9 after creation.

☐ Existing compute
You have an existing instance or server that you'd like to use.

New EC2 instance

Instance type [Info](#)
The memory and CPU of the EC2 instance that will be created for Cloud9 to run on.

☐ t2.micro (1 GiB RAM + 1 vCPU)
Free-tier eligible. Ideal for educational users and exploration.

☐ t3.small (2 GiB RAM + 2 vCPU)
Recommended for small web projects.

☒ **m5.large (8 GiB RAM + 2 vCPU)**
Recommended for production and most general-purpose development.

- c. Choose **Create**.
- d. Wait for the environment to be created, which will take several minutes.

Activity: Building an Amazon ECS Service with an ASP.NET Website

Note: Even though the m5.large instance is not in the free tier, by using the 30-minute timeout before auto-hibernating, the cost will be minimal to finish this activity.

Note: If environment creation fails because the m5.large instance type is not available in the Region, repeat steps a–d and select a different small instance type. If the instance type is not in the free tier, be mindful of the rate you will be charged for the duration of the activity.

Check your work

In this section, you have done the following:

- ✓ Obtained an AWS account.
- ✓ Signed in to the AWS Management Console.
- ✓ Selected an AWS Region to work in.
- ✓ Accessed an available local environment or created an AWS Cloud9 environment.

Activity: Building an Amazon ECS Service with an ASP.NET Website

Section 2: Set up the development environment

In this section, you will install software to set up a development environment. Skip over items you already have installed.

1. Install the [AWS Command Line Interface](#) (AWS CLI).
2. [Configure the AWS CLI](#) so it is linked to a user in your AWS account.
3. From a command or terminal window, configure your credentials and default Region with the following command:

```
aws configure
```

4. Give your AWS user the permissions needed to deploy applications to Amazon ECS.
 - a. In the AWS Management Console, navigate to **IAM**.
 - b. Go to **Users** and choose your AWS user.
 - c. Choose **Add permissions**, then choose **Attach existing policies directly**.
 - d. Search for and select the check box for each of the following policies:
 - PowerUserAccess
 - AWSCloudFormationFullAccess
 - AmazonECS_FullAccessAmazonEC2ContainerRegistryFullAccess
 - AmazonSSMFullAccess
 - IAMFullAccess
5. Install the .NET 6 SDK.
 - **Local machine:** If you are using your local machine, [download and install the .NET 6 SDK](#) for your operating system.
 - **AWS Cloud9:** If you are using AWS Cloud9, install .NET using these commands:

```
sudo rpm -Uvh https://packages.microsoft.com/config/centos/7/packages-microsoft-prod.rpm
sudo yum -y update
sudo yum install dotnet-sdk-6.0
```

6. Install the [AWS Deploy tool for .NET CLI](#) with the following command:

```
dotnet tool install -g aws.deploy.tools
```

- If you are using AWS Cloud9, skip ahead to **Section 3: Create an Amazon DynamoDB table**
 - If you are using your local machine, continue as follows.
7. Install an integrated development environment (IDE), such as [Microsoft Visual Studio 2022](#) (Windows), or [Visual Studio Code](#) or [JetBrains Rider](#) (Linux, macOS, Windows). Ensure that you have installed the options or extensions for .NET web development in C#.
 8. If you are using Visual Studio, install and configure the AWS Toolkit. With the toolkit, you can examine your AWS deployments within the IDE.
 - a. [Download and Install](#)

Activity: Building an Amazon ECS Service with an ASP.NET Website

- b. [Configure](#)
9. Install [Docker Desktop](#), which must be running on your machine. If you already have Docker, be aware that you need version 17.05 or later of the Docker Engine.

Check your work

In this section, you have done the following:

- ✓ Installed all prerequisite software.
- ✓ Configured the AWS CLI for your AWS user and AWS Region.
- ✓ Configured the AWS Toolkit for your IDE.

Activity: Building an Amazon ECS Service with an ASP.NET Website

Section 3: Create an Amazon DynamoDB table

In this section, you will create a DynamoDB table named **Weather** and create some data records.

1. In a browser, sign in to the AWS Management Console.
2. At top right, set the Region you want to work in.
3. Navigate to Amazon DynamoDB and choose **Create table**.
 - a. For Table name, enter **Weather**.
 - b. For Partition key, enter **Location**.
 - c. For Sort key, enter **Timestamp**.
 - d. Choose **Create table**.
 - e. Wait for the table status to become **Active**.

The screenshot shows the 'Create table' page in the AWS Management Console. The breadcrumb trail at the top is 'DynamoDB > Tables > Create table'. The page title is 'Create table'. Below this, there is a 'Table details' section with an 'Info' link. It explains that DynamoDB is a schemaless database requiring only a table name and a primary key. The 'Table name' field is set to 'Weather'. The 'Partition key' is set to 'Location' with a 'String' data type. The 'Sort key - optional' is set to 'Timestamp' with a 'String' data type. Below these fields, there is a 'Table settings' section with two options: 'Default settings' (selected) and 'Customize settings'. Under 'Default settings', it says 'The fastest way to create your table. You can modify these settings now or after your table has been created.' Under 'Customize settings', it says 'Use these advanced features to make DynamoDB work better for your needs.' Below this, there is a link to 'View default table settings' with a description. At the bottom, there is a 'Tags' section with a description and an 'Add new tag' button. At the very bottom, there are 'Cancel' and 'Create table' buttons.

4. Choose the **Weather** table name to get to its detail page, then choose **Explore table items**.

Activity: Building an Amazon ECS Service with an ASP.NET Website

5. Add an item using the Form. You can also add items using the JSON view. See the [Appendix](#) for instructions.
 - a. Choose **Create item** and add an item with the following attributes: Location: **Dallas**, Timestamp: **2022-07-23T06:00:00**.
 - b. Choose **Add new attribute** to add attributes **TempC** (Number) **33**, **TempF** (Number) **92**, and **Summary** (String) **Hot**.

The screenshot shows the 'Edit item' form in the Amazon DynamoDB console for the 'Weather' table. The form has two tabs: 'Form' (selected) and 'JSON view'. Below the tabs, there's a message: 'You can add, remove, or edit the attributes of an item. You can nest attributes inside other attributes up to 32 levels deep. [Learn more](#)'. The 'Attributes' section contains a table with the following data:

Attribute name	Value	Type	
Location - Partition key	Dallas	String	New
Timestamp - Sort key	2022-07-23T06:00:00	String	New
TempC	33	Number	Remove
TempF	92	Number	Remove
Summary	Hot	String	Remove

At the bottom right, there are 'Cancel' and 'Save changes' buttons.

- c. Choose **Create item** to create the item.

The screenshot shows the Amazon DynamoDB console after successfully saving an item. A green banner at the top says 'The item has been saved successfully.' The main area shows the 'Weather' table. On the left, there's a 'Tables (1)' sidebar with 'Weather' selected. The main content area has a 'Scan/Query items' section with 'Scan' and 'Query' buttons, and a 'Filters' section. Below this, there's a 'Run' button and a 'Reset' button. The 'Items returned (1)' section shows a table with the following data:

Location	Timestamp	Summary	TempC	TempF
Dallas	2022-07-23T06:00:00	Hot	33	92

At the bottom right of the 'Items returned (1)' section, there is a 'Create item' button highlighted with a red box.

6. Add more items using the JSON view. See the [Appendix](#) for instructions.

Activity: Building an Amazon ECS Service with an ASP.NET Website

The completed table should have six items as listed in the following screenshot.

The screenshot shows the AWS DynamoDB console interface for a table named 'Weather'. On the left, a sidebar lists 'Tables (1)' with 'Weather' selected. The main panel is titled 'Weather' and includes an 'Autopreview' toggle and a 'View table details' link. Below this, the 'Scan/Query items' section shows a dropdown menu set to 'Weather' with 'Scan' and 'Query' buttons. A 'Run' button is visible. The 'Items returned (6)' section shows a table with 6 items. The table has columns: Location, Timestamp, Summary, TempC, and TempF. The items are as follows:

Location	Timestamp	Summary	TempC	TempF
Dallas	2022-07-23T06:00:00	Hot	33	92
Dallas	2022-07-23T12:00:00	Scorching	43	109
Dallas	2022-07-23T18:00:00	Hot	36	97
Minneapolis	2022-07-23T18:00:00	Cool	19	67
Minneapolis	2022-07-23T12:00:00	Balmy	22	72
Minneapolis	2022-07-23T06:00:00	Cool	13	56

Check your work

You should now have the following:

- ✓ A DynamoDB table named Weather
- ✓ Table rows populated with data

Activity: Building an Amazon ECS Service with an ASP.NET Website

2. Open the WeatherAPI project in your IDE.
3. Select Region. If you have an AWS Toolkit installed for your IDE, select the same AWS Region you chose in **Section 1: Set up an AWS environment.**
4. Disable HTTPS Redirection.
 - a. Open Program.cs in the code editor, and remove or comment out the following statement:

```
// app.UseHttpsRedirection();
```

5. Debug the project.

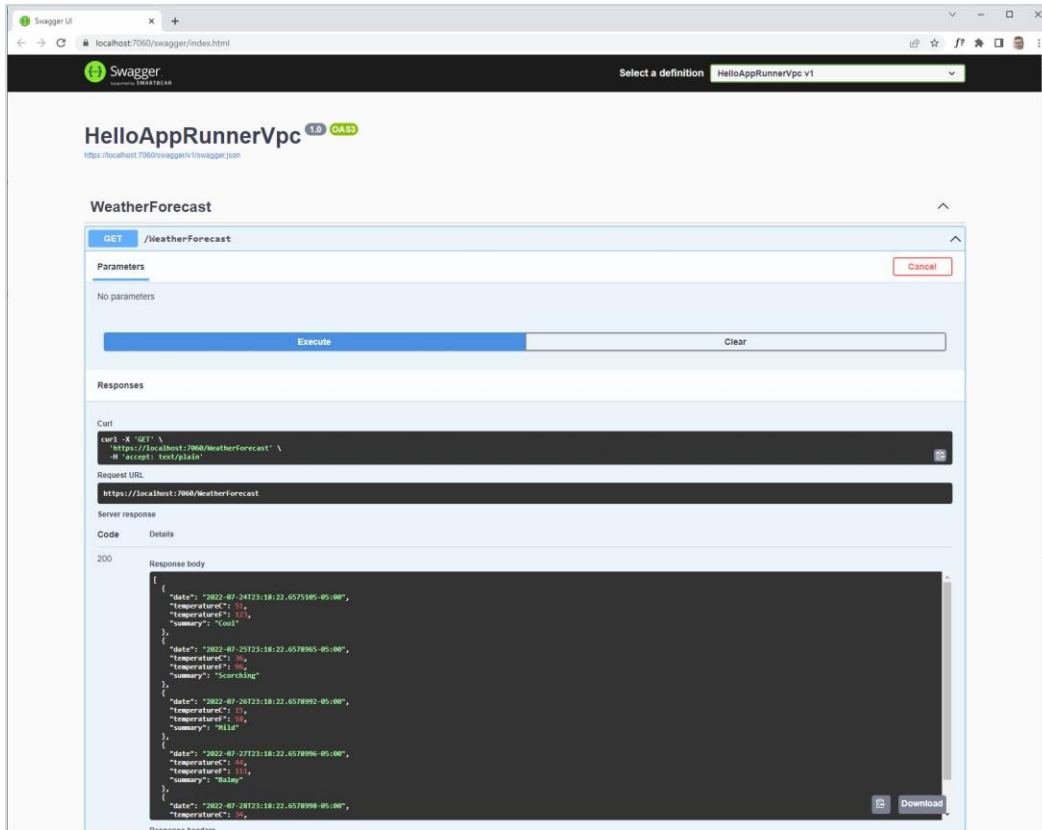
The generated project is a WeatherForecast API, commonly used in .NET samples.

 - a. To try it, do either of the following:
 - i. Press **F5** in your IDE.
 - ii. Run **dotnet run** from the command line and browse to the endpoint address, with **/swagger/index.html** at the end of the path.
 - b. In the browser tab that opens, you see a Swagger interface. Notice the service has a /WeatherForecast action that returns mock weather data JSON. Test it by choosing **GET**, then **Try it out**, and then **Execute**. Note the JSON response for each action.

You might or might not see a “Not secure” indicator about HTTPS in the browser, depending on whether or not you accepted a local SSL certificate.

- c. Stop the program from running.

Activity: Building an Amazon ECS Service with an ASP.NET Website



6. Configure ports.

The Web API will run at port 8080 (HTTP) and 8443 (SSL).

- Open Properties/LaunchSettings.json in the code editor.
- On line 17, change the **applicationUrl** values to the following:

```
"applicationUrl": "https://localhost:8443;http://localhost:8080",
```

7. Code the data structure.

Open WeatherForecast.cs and replace it with the following code. This record structure matches the DynamoDB items you created in **Section 3: Create an Amazon DynamoDB table**.

- Copy and paste the following code into WeatherForecast.cs:

Activity: Building an Amazon ECS Service with an ASP.NET Website

```
namespace WeatherAPI;

public class WeatherForecast
{
    public string? Location { get; set; }

    public DateTime Date { get; set; }

    public int TemperatureC { get; set; }

    public int TemperatureF { get; set; }

    public string? Summary { get; set; }
}
```

8. Code the Weather Forecast Controller.

- a. Open WeatherForecastController.cs in the Controllers folder, and replace (copy and paste) with the following code.
- b. On line 10, set **RegionEndpoint** to the AWS Region you are working in. For example, RegionEndpoint.USEast1. This code implements a health check method at the root path of the service, and a WeatherForecast method at /WeatherForecast.

The /WeatherForecast method takes a location parameter and retrieves data for it from the DynamoDB Weather table. It performs a table scan to find records whose partition key matches the location. Matching DynamoDB items are stored in a List of WeatherForecast objects. The results are returned as an array of JSON records.

Activity: Building an Amazon ECS Service with an ASP.NET Website

```
using Amazon;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DocumentModel;
using Microsoft.AspNetCore.Mvc;

namespace WeatherAPI.Controllers;

[ApiController]
[Route("")]
public class WeatherForecastController : ControllerBase
{
    static readonly RegionEndpoint region = RegionEndpoint.USWest2; //← Add your Region here

    private readonly ILogger<WeatherForecastController> _logger;

    public WeatherForecastController(ILogger<WeatherForecastController> logger)
    {
        _logger = logger;
    }

    [HttpGet("")]
    public string GetHealthcheck()
    {
        return "Healthcheck: Healthy";
    }

    [HttpGet("WeatherForecast")]
    public async Task<IEnumerable<WeatherForecast>> GetWeatherForecast(string location = "Dallas")
    {
        List<WeatherForecast> forecasts = new List<WeatherForecast>();

        try
        {
            _logger.LogInformation($"00 enter GET, location = {location}");

            var client = new AmazonDynamoDBClient(region);
            Table table = Table.LoadTable(client, "Weather");

            var filter = new ScanFilter();
            filter.AddCondition("Location", ScanOperator.Equal, location);

            var scanConfig = new ScanOperationConfig()
            {
                Filter = filter,
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string> { "Location", "Timestamp", "TempC", "TempF", "Summary" }
            };
        }
    }
}
```

Activity: Building an Amazon ECS Service with an ASP.NET Website

```
_logger.LogInformation($"10 table.Scan");  
  
Search search = table.Scan(scanConfig);  
  
    List<Document> matches;  
    do  
    {  
        _logger.LogInformation($"20 table.GetNextSetAsync");  
        matches = await search.GetNextSetAsync();  
        foreach (var match in matches)  
        {  
            forecasts.Add(new WeatherForecast  
            {  
                Location = Convert.ToString(match["Location"]),  
                Date = Convert.ToDateTime(match["Timestamp"]),  
                TemperatureC = Convert.ToInt32(match["TempC"]),  
                TemperatureF = Convert.ToInt32(match["TempF"]),  
                Summary = Convert.ToString(match["Summary"])  
            });  
        }  
    } while (!search.IsDone);  
  
    _logger.LogInformation($"30 exited results loop");  
  
}  
catch (Exception ex)  
{  
    _logger.LogError(ex, "90 Exception");  
}  
  
_logger.LogInformation($"99 returning {forecasts.Count} results");  
return forecasts.ToArray();  
}  
}
```

Activity: Building an Amazon ECS Service with an ASP.NET Website

9. Save and Build.

- a. Save your changes and ensure the project builds successfully.

Check your work

In this section, you have done the following:

- ✓ Generated a WeatherAPI project.
- ✓ Installed the AWSDK.DynamoDBv2 NuGet package.
- ✓ Set WeatherForecastController.cs Region variable to your AWS Region.
- ✓ Disabled HTTPS Redirection.
- ✓ Updated WeatherForecast.cs with a structure that matches the DynamoDB items.
- ✓ Updated WeatherForecastController.cs with a root healthcheck method and WeatherForecast method that queries the DynamoDB weather table.

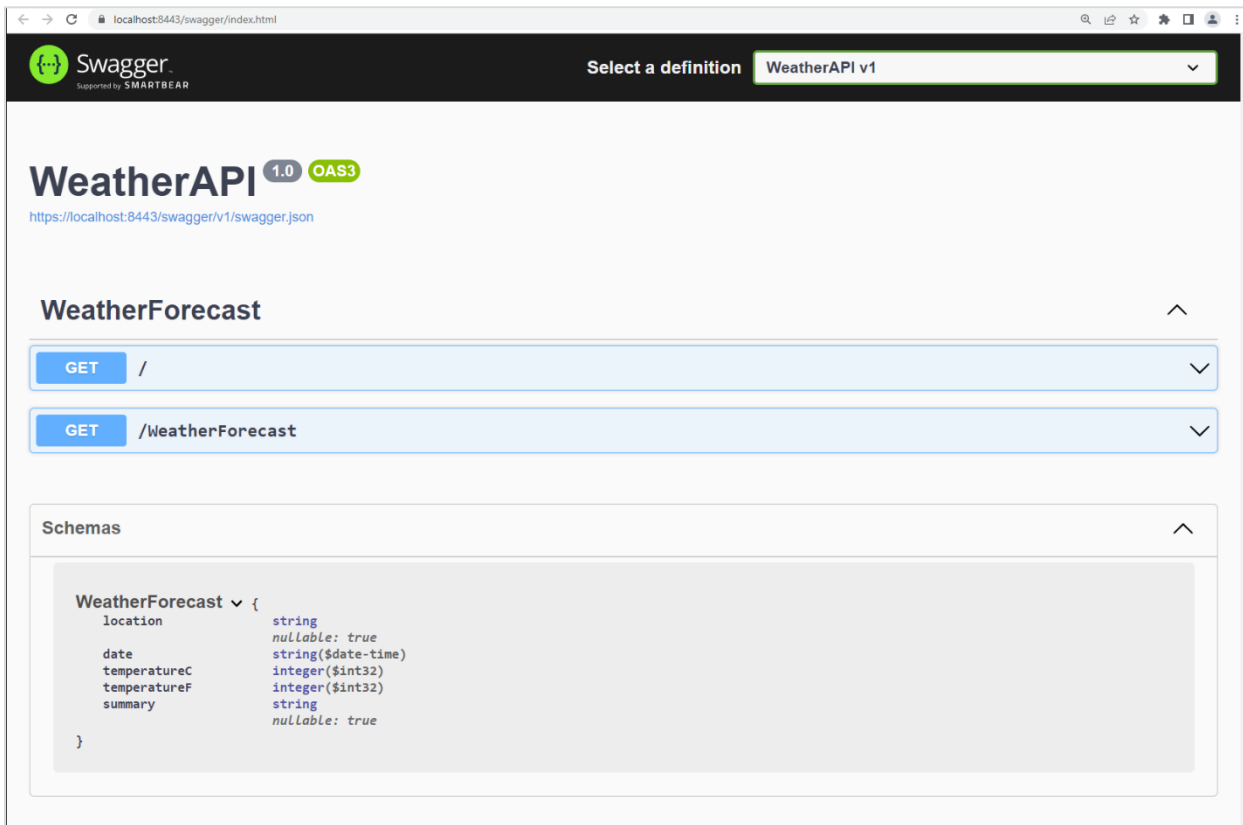
Activity: Building an Amazon ECS Service with an ASP.NET Website

Section 5: Test WeatherAPI locally

In this section, you will test the Web API locally and confirm data retrieval from DynamoDB.

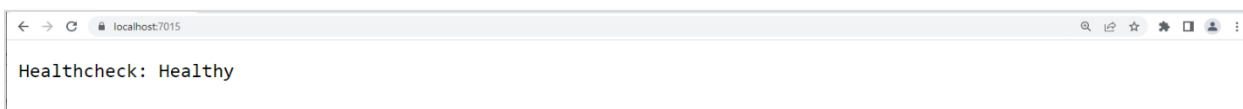
1. Verify access to the application.

- In your IDE, press **F5** and wait for the app to build and launch in a browser. Or run `dotnet run` from the command line and browse to `http://localhost:8080/swagger/index.html`.
- In the Swagger page, you should now find your two GET methods and the WeatherForecast schema.



2. Test the healthcheck action.

- In the browser, remove the Swagger path from the URL to hit the service root, and you should find a health check message. AWS App Runner will regularly ping the site to check health.



3. Test WeatherForecast action.

- Add `/WeatherForecast?location=Dallas` to the end of the URL path.
- You should find weather forecast data JSON, with values you created in the DynamoDB table in **Section 3: Create an Amazon DynamoDB table.**

Activity: Building an Amazon ECS Service with an ASP.NET Website

```
localhost:7015/WeatherForecast?location=Dallas
[{"location": "Dallas", "date": "2022-07-23T06:00:00", "temperatureC": 33, "temperatureF": 92, "summary": "Hot"}, {"location": "Dallas", "date": "2022-07-23T12:00:00", "temperatureC": 43, "temperatureF": 109, "summary": "Scorching"}, {"location": "Dallas", "date": "2022-07-23T18:00:00", "temperatureC": 36, "temperatureF": 97, "summary": "Hot"}]
```

4. Test another location.

- Change the URL path to end in **/WeatherForecast?location=Minneapolis**.
- Now you should find the data for that city.

```
localhost:7015/WeatherForecast?location=Minneapolis
[{"location": "Minneapolis", "date": "2022-07-23T06:00:00", "temperatureC": 13, "temperatureF": 56, "summary": "Cool"}, {"location": "Minneapolis", "date": "2022-07-23T12:00:00", "temperatureC": 22, "temperatureF": 72, "summary": "Balmy"}, {"location": "Minneapolis", "date": "2022-07-23T18:00:00", "temperatureC": 19, "temperatureF": 67, "summary": "Balmy"}]
```

5. Test an invalid location.

- Try a different location name, and you should find an empty response because there is no data for it in the table.

```
localhost:7015/WeatherForecast?location=Utopia
[]
```

6. Stop testing.

- Stop the program from running.

Check your work

In this section, you have done the following:

- ✓ Tested the WeatherAPI locally.
- ✓ Confirmed the WeatherAPI can retrieve data from the DynamoDB table.

Activity: Building an Amazon ECS Service with an ASP.NET Website

Section 6: Deploy WeatherAPI to Amazon ECS

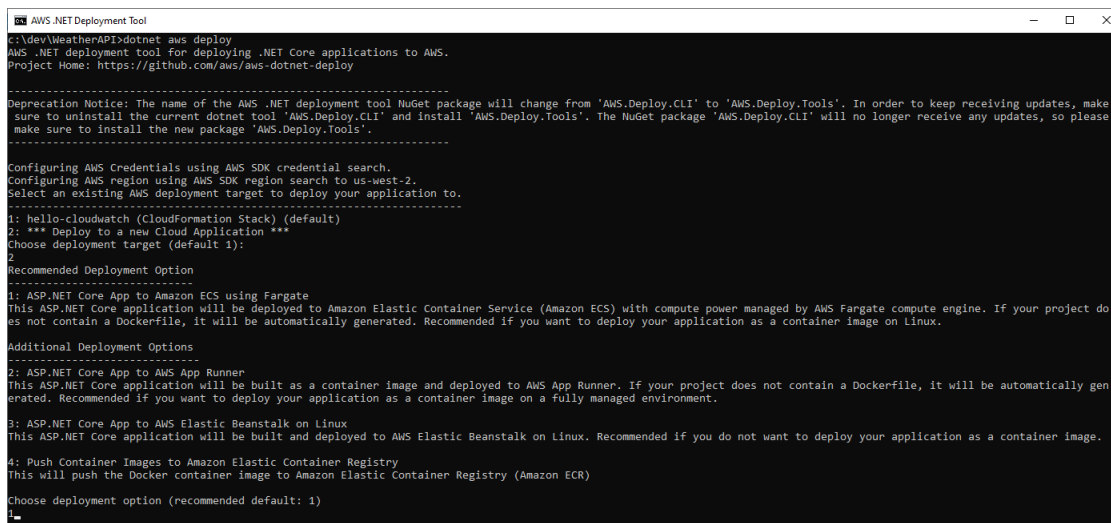
In this section, you will containerize and deploy the WeatherAPI project to Amazon ECS using the AWS deployment tool for .NET CLI.

1. Run the AWS deployment tool.
 - a. In a command prompt or terminal window, navigate to the WeatherAPI folder.
 - b. Run the following command:

```
dotnet aws deploy
```

Note: The following deployment will fail if you do not have the Node.js and Docker dependencies installed on your local machine. In that case, follow the instructions for installing them: [Node.js](#) and [Docker](#).

- c. If you are prompted to **Select an existing AWS deployment target**, select the choice for **Deploy to a new Cloud Application**.
- d. Select the choice for **ASP.NET Core App to Amazon ECS using Fargate**.



```

AWS .NET Deployment Tool
C:\dev\WeatherAPI>dotnet aws deploy
AWS .NET deployment tool for deploying .NET Core applications to AWS.
Project Home: https://github.com/aws/aws-dotnet-deploy

-----
Deprecation Notice: The name of the AWS .NET deployment tool NuGet package will change from 'AWS.Deploy.CLI' to 'AWS.Deploy.Tools'. In order to keep receiving updates, make
sure to uninstall the current dotnet tool 'AWS.Deploy.CLI' and install 'AWS.Deploy.Tools'. The NuGet package 'AWS.Deploy.CLI' will no longer receive any updates, so please
make sure to install the new package 'AWS.Deploy.Tools'.
-----

Configuring AWS Credentials using AWS SDK credential search.
Configuring AWS region using AWS SDK region search to us-west-2.
Select an existing AWS deployment target to deploy your application to.
-----
1: hello-cloudwatch (CloudFormation Stack) (default)
2: *** Deploy to a new Cloud Application ***
Choose deployment target (default 1):
2

Recommended Deployment Option
-----
1: ASP.NET Core App to Amazon ECS using Fargate
This ASP.NET Core application will be deployed to Amazon Elastic Container Service (Amazon ECS) with compute power managed by AWS Fargate compute engine. If your project do
es not contain a Dockerfile, it will be automatically generated. Recommended if you want to deploy your application as a container image on Linux.

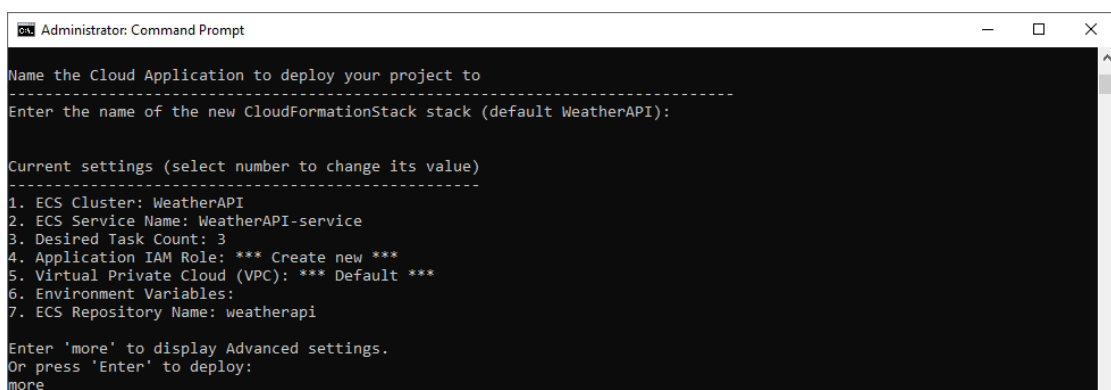
Additional Deployment Options
-----
2: ASP.NET Core App to AWS App Runner
This ASP.NET Core application will be built as a container image and deployed to AWS App Runner. If your project does not contain a Dockerfile, it will be automatically gen
erated. Recommended if you want to deploy your application as a container image on a fully managed environment.

3: ASP.NET Core App to AWS Elastic Beanstalk on Linux
This ASP.NET Core application will be built and deployed to AWS Elastic Beanstalk on Linux. Recommended if you do not want to deploy your application as a container image.

4: Push Container Images to Amazon Elastic Container Registry
This will push the Docker container image to Amazon Elastic Container Registry (Amazon ECR)

Choose deployment option (recommended default: 1)
1
```

- e. For name of new CloudFormation stack, press **Enter** to accept the default name of **WeatherAPI**.



```

Administrator: Command Prompt

Name the Cloud Application to deploy your project to
-----
Enter the name of the new CloudFormationStack stack (default WeatherAPI):

Current settings (select number to change its value)
-----
1. ECS Cluster: WeatherAPI
2. ECS Service Name: WeatherAPI-service
3. Desired Task Count: 3
4. Application IAM Role: *** Create new ***
5. Virtual Private Cloud (VPC): *** Default ***
6. Environment Variables:
7. ECS Repository Name: weatherapi

Enter 'more' to display Advanced settings.
Or press 'Enter' to deploy:
more
```

Activity: Building an Amazon ECS Service with an ASP.NET Website

2. Edit settings.

- a. When Current settings are displayed, enter **more** to see advanced settings. A numbered list of settings is displayed.
- b. Enter **9** to select Elastic Load Balancer and respond to prompts as follows:
 - Create New Load Balancer: **y**
 - Deregistration delay: **60**
 - Health Check path: **/**
 - Health Check Timeout: **5**
 - Health Check Interval: **30**
 - Healthy Threshold Count: **5**
 - Unhealthy Threshold Count: **2**
 - Internet-Facing: **y**

```
Current settings (select number to change its value)
-----
1. ECS Cluster: WeatherAPI
2. ECS Service Name: WeatherAPI-service
3. Desired Task Count: 3
4. Application IAM Role: *** Create new ***
5. Virtual Private Cloud (VPC): *** Default ***
6. Environment Variables:
7. ECR Repository Name: weatherapi

Enter 'more' to display Advanced settings.
Or press 'Enter' to deploy:
more

Current settings (select number to change its value)
-----
1 . ECS Cluster: WeatherAPI
2 . ECS Service Name: WeatherAPI-service
3 . Desired Task Count: 3
4 . Application IAM Role: *** Create new ***
5 . Virtual Private Cloud (VPC): *** Default ***
6 . ECS Service Security Groups:
7 . Task CPU: 256
8 . Task Memory: 512
9 . Elastic Load Balancer:
    Create New Load Balancer: True
    Deregistration delay (seconds): 60
    Health Check Path: /
    Health Check Timeout: 5
    Health Check Interval: 30
    Healthy Threshold Count: 5
    Unhealthy Threshold Count: 2
    Internet-Facing: True
10. AutoScaling:
    Enable: False
11. Environment Variables:
12. Docker Build Args:
13. Dockerfile Path: .\Dockerfile
14. Docker Execution Directory:
15. ECR Repository Name: weatherapi

Or press 'Enter' to deploy:
|
```

Activity: Building an Amazon ECS Service with an ASP.NET Website

3. Deploy the project.

- Press **Enter** to start deployment. The deployment will create a load balancer, three ECS tasks, and new roles.

```
AWS .NET Deployment Tool
-----
Configuring AWS Credentials using AWS SDK credential search.
Configuring AWS region using AWS SDK region search to us-west-2.
Select an existing AWS deployment target to deploy your application to.
-----
1: hello-cloudwatch (CloudFormation Stack) (default)
2: *** Deploy to a new Cloud Application ***
Choose deployment target (default 1):
2
Recommended Deployment Option
-----
1: ASP.NET Core App to Amazon ECS using Fargate
This ASP.NET Core application will be deployed to Amazon Elastic Container Service (Amazon ECS) with compute power managed by AWS Fargate compute engine. If your project does not contain a Dockerfile, it will be automatically generated. Recommended if you want to deploy your application as a container image on Linux.
Additional Deployment Options
-----
2: ASP.NET Core App to AWS App Runner
This ASP.NET Core application will be built as a container image and deployed to AWS App Runner. If your project does not contain a Dockerfile, it will be automatically generated. Recommended if you want to deploy your application as a container image on a fully managed environment.
3: ASP.NET Core App to AWS Elastic Beanstalk on Linux
This ASP.NET Core application will be built and deployed to AWS Elastic Beanstalk on Linux. Recommended if you do not want to deploy your application as a container image.
4: Push Container Images to Amazon Elastic Container Registry
This will push the Docker container image to Amazon Elastic Container Registry (Amazon ECR)
Choose deployment option (recommended default: 1)
1
Name the Cloud Application to deploy your project to
-----
Enter the name of the new CloudFormationStack stack (default WeatherAPI):
Current settings (select number to change its value)
-----
1. ECS Cluster: WeatherAPI
2. ECS Service Name: WeatherAPI-service
3. Desired Task Count: 3
4. Application IAM Role: *** Create new ***
5. Virtual Private Cloud (VPC): *** Default ***
6. Environment Variables:
7. ECS Repository Name: weatherapi
Enter 'more' to display Advanced settings.
On press 'Enter' to deploy:
```

- Wait while the service is containerized and deployed, which will take several minutes. When it completes, note the deployment details at the end of the output.
- Record the details under Application Endpoint. You will need the *endpoint* to test the service.

Endpoint URL: _____

```
WeatherAPI
Deployment time: 177.89s
arn:aws:cloudformation:us-west-2:██████████:stack/WeatherAPI/9373c170-2184-11ed-879f-06f12ef73b73
Stack ARN:
Total time: 195.27s
Resources
-----
Application Endpoint:
Id: arn:aws:elasticloadbalancing:us-west-2:██████████:loadbalancer/app/Weath-Recip-Z91T6NJ3ED7H/ee106e79fec51b
Type: AWS::ElasticLoadBalancingV2::LoadBalancer
Endpoint: http://Weath-Recip-Z91T6NJ3ED7H-1064726783.us-west-2.elb.amazonaws.com/
ECS Service:
Id: arn:aws:ecs:us-west-2:██████████:service/WeatherAPI/WeatherAPI-service
Type: AWS::ECS::Service
ECS Cluster:
Id: WeatherAPI
Type: AWS::ECS::Cluster
c:\dev\WeatherAPI>
```

Activity: Building an Amazon ECS Service with an ASP.NET Website

4. Review Dockerfile.

In your project folder, notice that a **Dockerfile** file has been added. The AWS deployment tool created this file and used it with Docker to containerize the project. The Dockerfile should look similar to the following. You might see 6.0 or 7.0 depending on which version you are using.

```
FROM mcr.microsoft.com/dotnet/aspnet:7.0 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

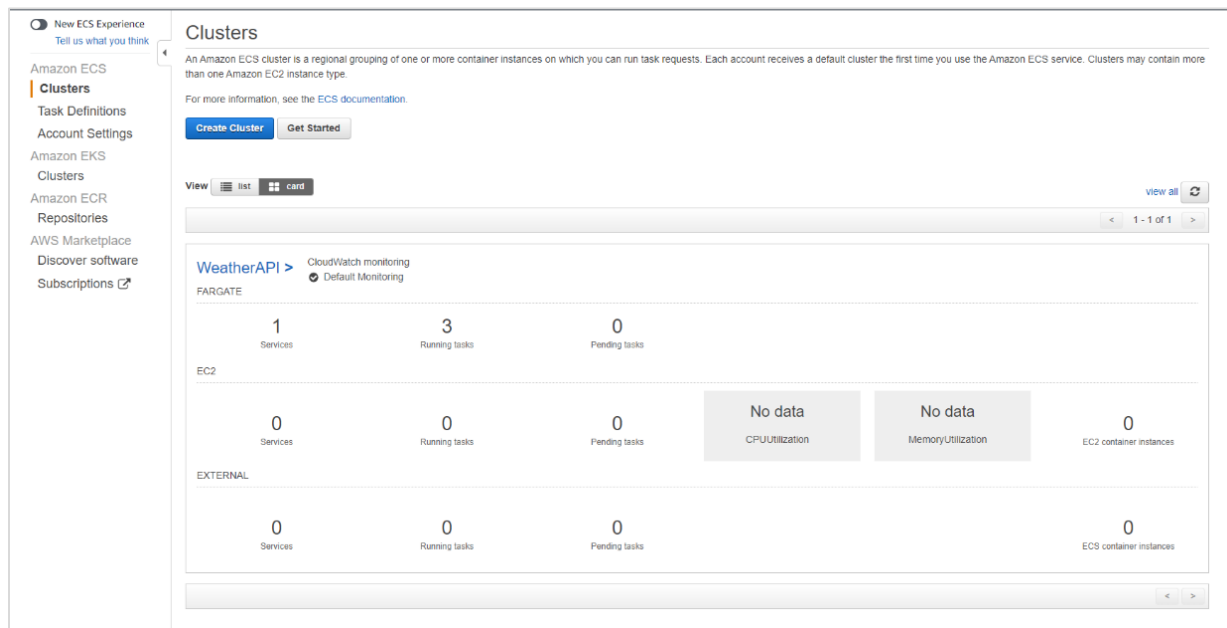
FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build
WORKDIR /src
COPY ["WeatherAPI.csproj", ""]
RUN dotnet restore "WeatherAPI.csproj"
COPY . .
WORKDIR "/src/"
RUN dotnet build "WeatherAPI.csproj" -c Release -o /app/build

FROM build AS publish
RUN apt-get update -yq \
    && apt-get install curl gnupg -yq \
    && curl -sL https://deb.nodesource.com/setup_14.x | bash \
    && apt-get install nodejs -yq
RUN dotnet publish "WeatherAPI.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "WeatherAPI.dll"]
```

Activity: Building an Amazon ECS Service with an ASP.NET Website

5. **Review in the Amazon ECS console.** Examine what's been deployed in the AWS console.
 - a. In the AWS Management Console, navigate to **Amazon ECS**.
 - b. Choose **Clusters** from the navigation pane, and the **WeatherAPI** cluster should be listed in the content pane. You should find one FARGATE service with three running tasks.
 - c. Choose the **WeatherAPI** cluster name to examine its detail.



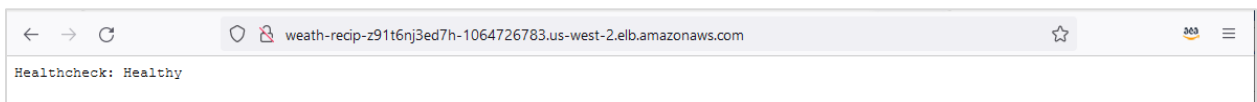
Activity: Building an Amazon ECS Service with an ASP.NET Website

- d. In the content pane for **WeatherAPI**, explore the tabs to find what was deployed. You should find three running tasks of launch type FARGATE.

The screenshot shows the Amazon ECS console for the 'WeatherAPI' cluster. The 'Cluster overview' section displays the cluster's ARN, status (Active), CloudWatch monitoring (Default), and registered container instances (-). Below this, the 'Services' tab is selected, showing a table of services. The table has columns for Service name, Status, ARN, Service type, Deployments and tasks, Last deployment, Task definition, Revision, and Launch type. One service, 'WeatherAPI-service', is listed with a status of 'Active', ARN 'arn:aws:ecs:us-west-2:123456789012:cluster/WeatherAPI/service/WeatherAPI-service', service type 'REPLICA', and 3/3 tasks running. The last deployment is 'Completed' with task definition 'WeatherAPIRecipeAppT' and revision '9'. The launch type is 'FARGATE'.

Service name	Status	ARN	Service type	Deployments and tasks	Last deployment	Task definition	Revision	Launch...
WeatherAPI-service	Active	arn:aws:ecs:us-west-2:123456789012:cluster/WeatherAPI/service/WeatherAPI-service	REPLICA	3/3 Tasks r...	Completed	WeatherAPIRecipeAppT	9	FARGATE

6. Test the ECS endpoint.
- a. Open another browser tab, and visit the endpoint URL you recorded earlier in Step 3c. You should see a **Healthy** health check response. This test shows that the ECS-hosted service is responding.



Check your work

In this section, you have done the following:

- ✓ Containerized and deployed the project to Amazon ECS with the AWS deployment tool for .NET CLI.
- ✓ Confirmed the deployment in the Amazon ECS console.
- ✓ Confirmed the base endpoint URL is returning a healthy health check.

Activity: Building an Amazon ECS Service with an ASP.NET Website

Section 7: Update ECS task role and test API

In this section, you will create an AWS Identity and Access Management (IAM) policy for DynamoDB access, update the ECS task role permissions, and test the Amazon ECS service.

1. Test API and note missing permission.
 - a. In a browser, visit the endpoint URL with **/WeatherForecast?location=Dallas** added to the end of the path.
 - b. You should notice that no data is returned. This is because the ECS task role that is running the WeatherAPI service does not have permissions to access the DynamoDB Weather table. Keep this browser open; you will return to it later in this section.
2. Create an IAM policy that allows access to the Weather DynamoDB table.
 - a. In the AWS Management Console, navigate to **IAM**.
 - b. Choose **Policies** from the navigation pane, and then choose **Create policy**.

Create policy

1 2 3

Review policy

Name* ddb-weather
Use alphanumeric and "+, @, _" characters. Maximum 128 characters.

Description
Maximum 1000 characters. Use alphanumeric and "+, @, _" characters.

Summary
This policy defines some actions, resources, or conditions that do not provide permissions. To grant access, policies must have an action that has an applicable resource or condition. For details, choose [Show remaining](#). [Learn more](#)

Q Filter

Service	Access level	Resource	Request condition
Allow (1 of 329 services) Show remaining 328			
DynamoDB	Full Tagging Limited	Read, Write	TableName string like WeatherForecast

Tags
Key Value
No tags associated with the resource.

* Required

Cancel Previous Create policy

- c. Create the policy and enter (copy and paste) the following JSON. Replace **[account]** with your 12-digit AWS account number, and **[region]** with your AWS Region (for example: us-west-2).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Action": "dynamodb:*",
      "Resource": "arn:aws:dynamodb:[region]:[account]:table/Weather"
    }
  ]
}
```

Activity: Building an Amazon ECS Service with an ASP.NET Website

The screenshot shows the 'Create policy' page in the AWS IAM console. The 'JSON' tab is selected, and the following policy is defined:

```
1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Sid": "1",
6       "Effect": "Allow",
7       "Action": "dynamodb:*",
8       "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/WeatherForecast"
9     }
10  ]
11 }
```

At the bottom, there are buttons for 'Cancel' and 'Next: Tags'. A status bar at the bottom indicates 'Security: 0', 'Errors: 0', 'Warnings: 0', and 'Suggestions: 1'. The character count is 163 of 6,144.

- e. Choose **Next: Tags** and then **Next: Review**.
- f. Name the policy **ddb-weather** and choose **Create policy**.
3. Modify the task role. The Amazon ECS service doesn't yet have permissions to access the Weather DynamoDB table. Update the role to add two policies.
 - a. In the Amazon ECS console, choose the **WeatherAPI** cluster to view its detail.
 - b. Choose **WeatherAPI-service** to examine the service configuration.

The screenshot shows the 'WeatherAPI' cluster details in the AWS ECS console. The cluster is in an 'ACTIVE' state. Below the cluster information, there are tabs for 'Services', 'Tasks', 'ECS Instances', 'Metrics', 'Scheduled Tasks', 'Tags', and 'Capacity Providers'. The 'Services' tab is selected, showing a table of services.

Service Name	Status	Service type	Task Definition	Desired tasks	Running tasks	Launch type	Platform version...
WeatherAPI-service	ACTIVE	REPLICA	WeatherAPIRecip...	3	3	FARGATE	LATEST(1.4.0)

Activity: Building an Amazon ECS Service with an ASP.NET Website

- c. Choose the **Configuration and tasks tab**, and then choose the **Task definition** link.

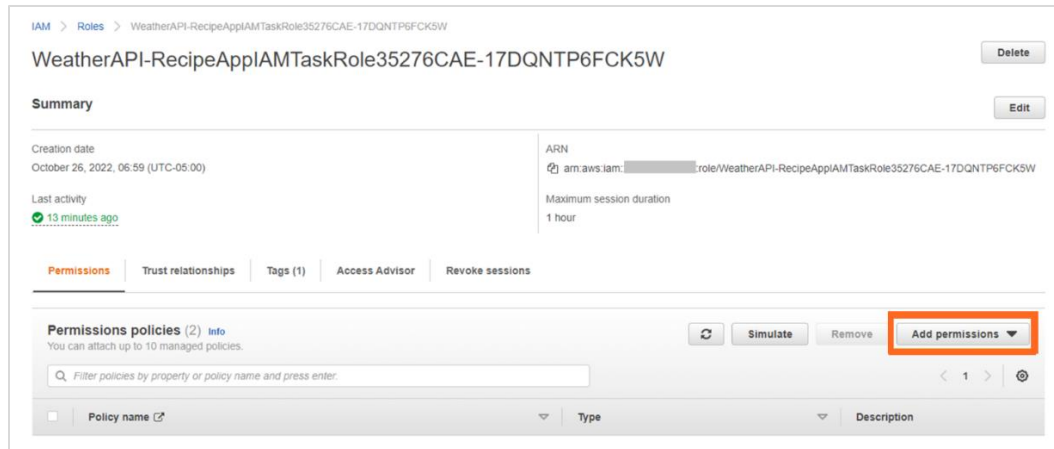
The screenshot shows the 'Configuration and tasks' tab for the 'WeatherAPI-service'. The 'Service configuration' section is expanded, showing details for the 'Task definition: revision' link, which is highlighted with a red box. The 'Launch type' is 'FARGATE', 'Service type' is 'REPLICA', and 'Created by' is 'aws:iam:aws:role/cdk-hnb659fds-cfn-exec-role-us-west-2'. The 'Load balancer' section shows 'Application Load Balancer' as the 'Load balancer type'. The 'Auto Scaling' section shows 'Desired tasks' as 3, 'Min tasks' as 1, and 'Max tasks' as 3. A message at the bottom states 'No Auto Scaling resources configured for this service.'

- d. Choose the **Task role** link. This takes you to the Role definition in the IAM console.

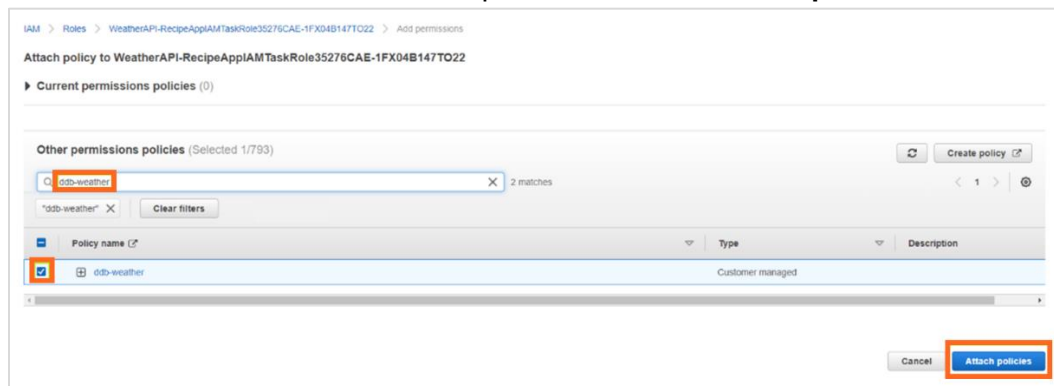
The screenshot shows the 'Task definition' page for 'WeatherAPIRecipeAppTaskDefinition1990ADDC:9'. The 'Overview' tab is selected, showing details for the 'Task role' link, which is highlighted with a red box. The 'Task role' is 'WeatherAPI-RecipeAppIAMTaskRole35276CAE-17DQNT6GCK5W'. The 'Task execution role' is 'WeatherAPI-RecipeAppTaskDefinitionExecutionRole3EC-DVHHEB2C6X'. The 'Time created' is '10/26/2022, 11:59:58 UTC', 'App environment' is 'FARGATE', and 'Network mode' is 'awsvpc'. The 'Task size' section shows 'Task CPU' as '.25 vCPU' and 'Task memory' as '.5 GB'. The 'Containers' section shows a single container named 'AppContainerDefinition' with image 'dkr.ecr.us-west-2.amazonaws.com/weatherapi:638023822291416951', 'Essential' set to 'true', 'CPU' as 0, and 'Memory' as -.

Activity: Building an Amazon ECS Service with an ASP.NET Website

- e. Choose **Add permissions**, and then choose **Attach policies**.



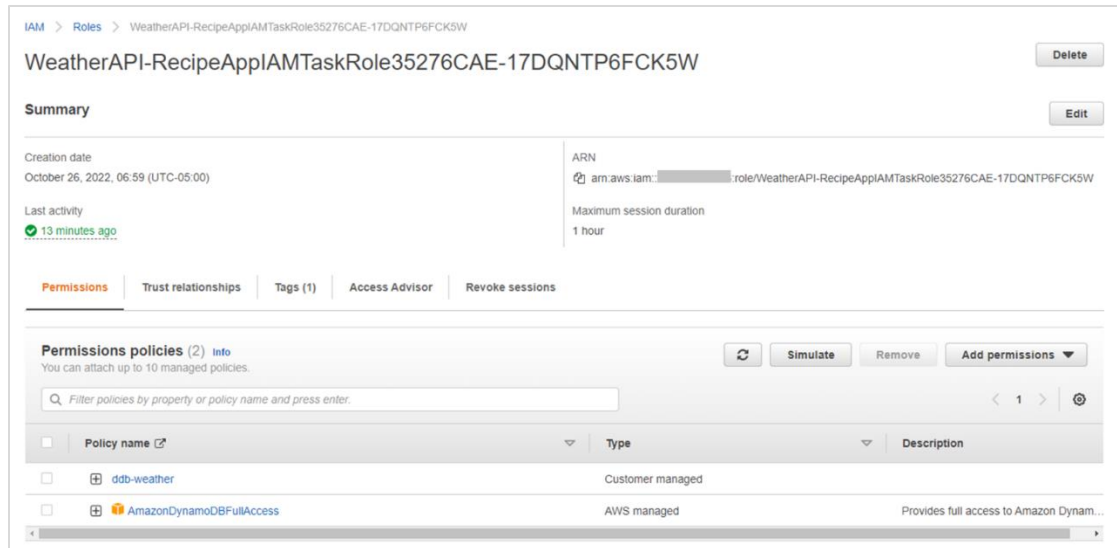
- f. Search for and select the **ddb-weather** permission. Choose **Attach policies**.



- g. Search for and select the **AmazonDynamoDBFullAccess** permission.

Activity: Building an Amazon ECS Service with an ASP.NET Website

- h. Choose **Attach policies**.
- i. You should now have both policies attached to the role as listed in the following screenshot.



- 4. Re-test the API.
 - a. Refresh the page that returned no results in Step 1b. This time, you should find the JSON data returned. The IAM role update provided access to the DynamoDB table.



Check your work

In this section, you have done the following:

- ✓ Tested the WeatherAPI ECS service.
- ✓ Created a policy named ddb-weather.
- ✓ Updated the Amazon ECS task role with new permissions for DynamoDB.
- ✓ Re-tested the WeatherAPI Amazon ECS service and had data returned successfully.

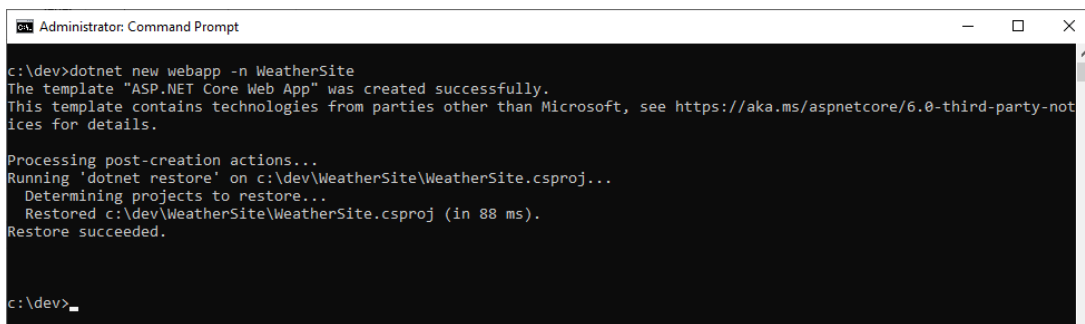
Activity: Building an Amazon ECS Service with an ASP.NET Website

Section 8: Create WeatherSite project

In this section, you will use the **dotnet new** command to create a .NET web project and update its code to retrieve data from the DynamoDB table.

1. Generate a web project.
 - a. Open a command prompt or terminal window and navigate to a development folder.
 - b. Run the following **dotnet new** command to create a new Web API project named WeatherSite.

```
dotnet new webapp -n WeatherSite
```



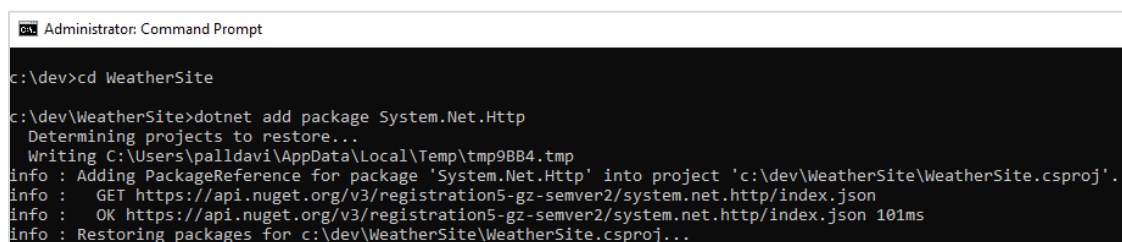
```
Administrator: Command Prompt
c:\dev>dotnet new webapp -n WeatherSite
The template "ASP.NET Core Web App" was created successfully.
This template contains technologies from parties other than Microsoft, see https://aka.ms/aspnetcore/6.0-third-party-notices for details.

Processing post-creation actions...
Running 'dotnet restore' on c:\dev\WeatherSite\WeatherSite.csproj...
  Determining projects to restore...
  Restored c:\dev\WeatherSite\WeatherSite.csproj (in 88 ms).
Restore succeeded.

c:\dev>
```

- c. Navigate to the WeatherSite project folder.
- d. Run the following **dotnet add package** command to add the **System.Net.Http** package to the project. You will need this library to communicate with the WeatherAPI service.

```
dotnet add package System.Net.Http
```



```
Administrator: Command Prompt
c:\dev>cd WeatherSite
c:\dev\WeatherSite>dotnet add package System.Net.Http
  Determining projects to restore...
  Writing C:\Users\palldavi\AppData\Local\Temp\tmp98B4.tmp
info : Adding PackageReference for package 'System.Net.Http' into project 'c:\dev\WeatherSite\WeatherSite.csproj'.
info :   GET https://api.nuget.org/v3/registration5-gz-semver2/system.net.http/index.json
info :   OK https://api.nuget.org/v3/registration5-gz-semver2/system.net.http/index.json 101ms
info : Restoring packages for c:\dev\WeatherSite\WeatherSite.csproj...
```

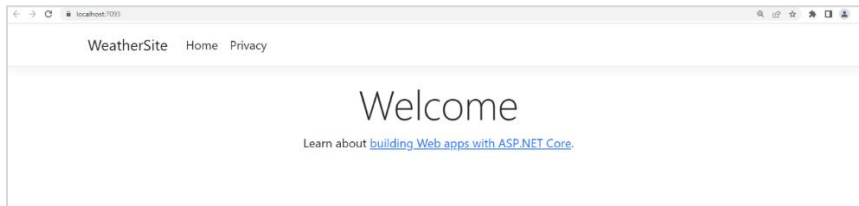
2. Open the project in the IDE.
 - a. Open the WeatherSite project in your IDE.
3. Disable HTTPS Redirection.
 - a. Open **Program.cs** in the code editor and remove or comment out the following statement:

```
// app.UseHttpsRedirection();
```

4. Debug the project. The generated project is a simple web application, commonly used in .NET samples.
 - a. To try it, do either of the following:
 - i. Press **F5** in your IDE.
 - ii. Run **dotnet run** from the command line and browse to the endpoint address, with **http://localhost:<port>**.

Activity: Building an Amazon ECS Service with an ASP.NET Website

- b. In the browser tab that opens, there is a simple page with a welcome message.



- c. Stop the program from running.

5. Add the WeatherAPI configuration setting.

- a. Open appsettings.json in the code editor and replace (copy and paste) with the code below, replacing **[service-address]** with the endpoint address of the WeatherAPI ECS service. **Note:** This is the WeatherAPI service URL from Section 6. It should not have a trailing slash.

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "AppSettings": {
    "WeatherAPI": "http://[service-address]"
  }
}
```


Activity: Building an Amazon ECS Service with an ASP.NET Website

6. Configure ports. The Web site will run at port 80 (HTTP) and 443 (SSL), but you might not be able to use those ports locally. If your local development machine is running a local web server such as IIS Express, they might not be available. For now, use ports 8080 and 8443.
 - a. Open Properties/LaunchSettings.json in the code editor.
 - b. On line 17, change the **applicationUrl** values to the following:

```
"applicationUrl": "https://localhost:8443;http://localhost:8080",
```

7. Code the WeatherSite record structure. Add file WeatherForecast.cs to the project with the following code. This record structure matches the items returned by the Weather API.

```
namespace WeatherSite;

public class WeatherForecast
{
    public string? Location { get; set; }

    public DateTime Date { get; set; }

    public int TemperatureC { get; set; }

    public int TemperatureF { get; set; }

    public string? Summary { get; set; }
}
```

8. Code the service client.
 - a. Open WeatherSite/Pages/Index.cshtml.cs in the code editor and replace (copy and paste) with the following code.

This is the code-behind file for the Razor page Index.cshtml. The URL for the WeatherAPI service is read from a setting in appSettings.json. The /WeatherForecast route takes a location parameter and retrieves data for it from the DynamoDB Weather table. The results are returned as an array of JSON records.

Activity: Building an Amazon ECS Service with an ASP.NET Website

```
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace WeatherSite.Pages;

public class IndexModel : PageModel
{
    public string Location { get; set; }
    public WeatherForecast[] Items { get; set; }
    public string Message { get; set; }

    private readonly ILogger<IndexModel> _logger;

    public IndexModel(ILogger<IndexModel> logger)
    {
        _logger = logger;
    }

    public async Task OnGet()
    {
        try
        {
            var apiUrl = new
ConfigurationBuilder().AddJsonFile("appsettings.json").Build().GetSection("AppSettings")["WeatherAPI"];
            Location = Request.Query["location"].FirstOrDefault() ?? "Dallas";
            var client = new HttpClient() { BaseAddress = new Uri(apiUrl) };
            var response = await
client.GetAsync($"WeatherForecast?location={Location}");
            response.EnsureSuccessStatusCode();
            var items = await
response.Content.ReadFromJsonAsync<WeatherForecast[]>();
            Items = items;
            Message = $"{Items.Count()} forecast records found ";
        }
        catch (HttpRequestException ex)
        {
            Message = ex.Message;
        }
    }
}
```

9. Code the page.

- a. Open WeatherSite/Pages/Index.cshtml in the code editor, and replace (copy and paste) with the following code.

The Razor page displays some values from the code-behind model using @ notation: location, success/error message, and a table of weather forecast data.

Activity: Building an Amazon ECS Service with an ASP.NET Website

```
@page
@model IndexModel
@{
    ViewData["Title"] = "Home page";
}

<style>
    .styled-table {
        border-collapse: collapse;
        font-size: 0.9em;
        font-family: sans-serif;
        min-width: 400px;
        box-shadow: 0 0 20px rgba(0, 0, 0, 0.15);
        margin-left: auto;
        margin-right: auto;
    }

    .styled-table thead tr {
        background-color: cornflowerblue;
        color: #ffffff;
        text-align: left;
    }

    .styled-table th,
    .styled-table td {
        padding: 12px 15px;
    }

    .styled-table tbody tr {
        border-bottom: 1px solid #dddddd;
    }

    .styled-table tbody tr:nth-of-type(even) {
        background-color: #f3f3f3;
    }

    .styled-table tbody tr:last-of-type {
        border-bottom: 2px solid cornflowerblue;
    }

    .styled-table tbody tr.active-row {
        font-weight: bold;
        color: cornflowerblue;
    }
</style>
```

Activity: Building an Amazon ECS Service with an ASP.NET Website

```
<div class="text-center">
  <h1 class="display-4">@Model.Location Weather</h1>
  <p>Weather Forecast</p>
  <p>@Model.Message</p>

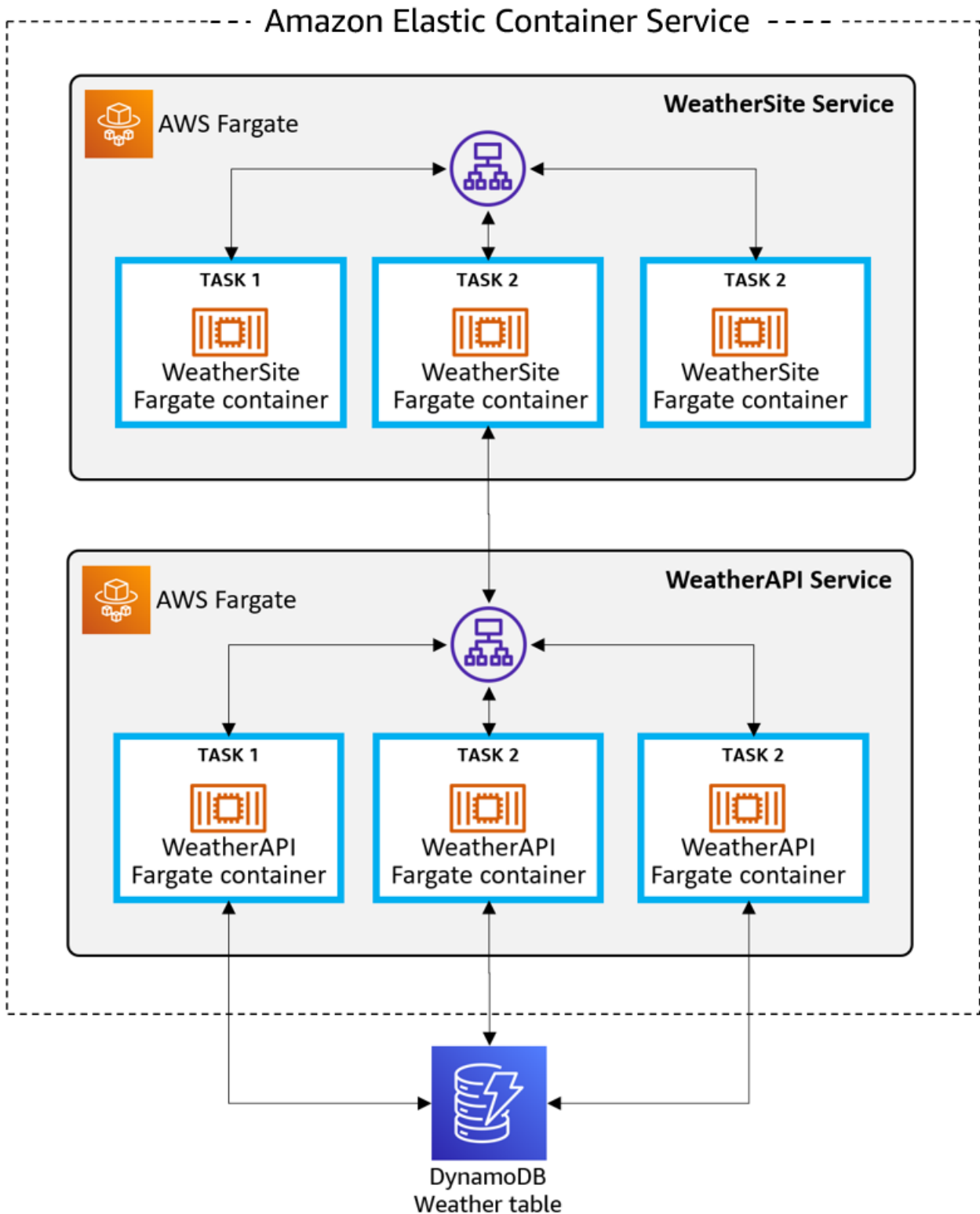
  <p><a href="/?location=Dallas">Dallas</a></p>
  <p><a href="/?location=Minneapolis">Minneapolis</a></p>

  @if (@Model.Items != null)
  {
    <table class="styled-table">
      <thead>
        <tr><th>Location</th><th>Time</th><th>Temp (°C)</th><th>Temp
(°F)</th><th>Summary</th>
        </thead>
      <tbody>
        @foreach (var forecast in Model.Items)
        {
          <tr><td>@forecast.Location</td><td>@forecast.Date.ToString("yyyy-MM-dd
HH:MM")</td><td>@forecast.TemperatureC</td><td>@forecast.TemperatureF</td><td>@for
ecast.Summary</td></tr>
        }
      </tbody>
    </table>
  }
</div>
```

10. Save and build.
 - a. Save your changes and ensure the project builds successfully.

At this point, you have created all the components for the architecture: database table, API, and website.

Activity: Building an Amazon ECS Service with an ASP.NET Website



Activity: Building an Amazon ECS Service with an ASP.NET Website

Check your work

After completing this section, you have done the following:

- ✓ Generated the WeatherSite project.
- ✓ Disabled HTTPS Redirection.
- ✓ Debugged the project.
- ✓ Added the WeatherAPI to the configuration settings.
- ✓ Configured HTTP ports.
- ✓ Updated WeatherForecast.cs with a record structure that matches the DynamoDB items.
- ✓ Coded the service client.
- ✓ Coded the web page.

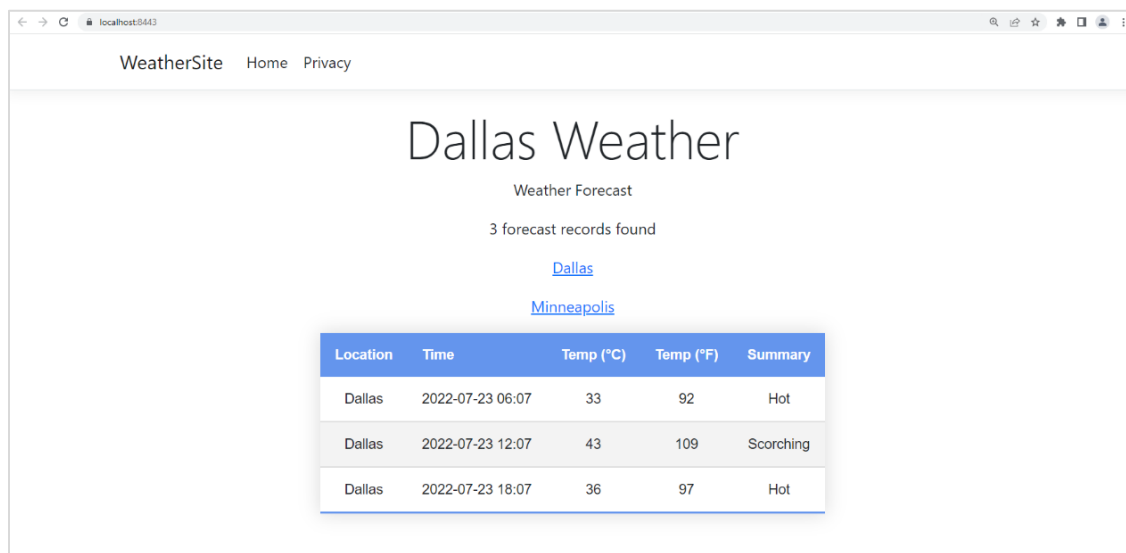
Activity: Building an Amazon ECS Service with an ASP.NET Website

Section 9: Test WeatherSite locally

In this section, you will test the web site locally and confirm that it can communicate with the Weather table.

1. Debug the project.
 - a. In your IDE, press F5 and wait for the web app to build and launch in a browser. The website should load and display weather data for Dallas.

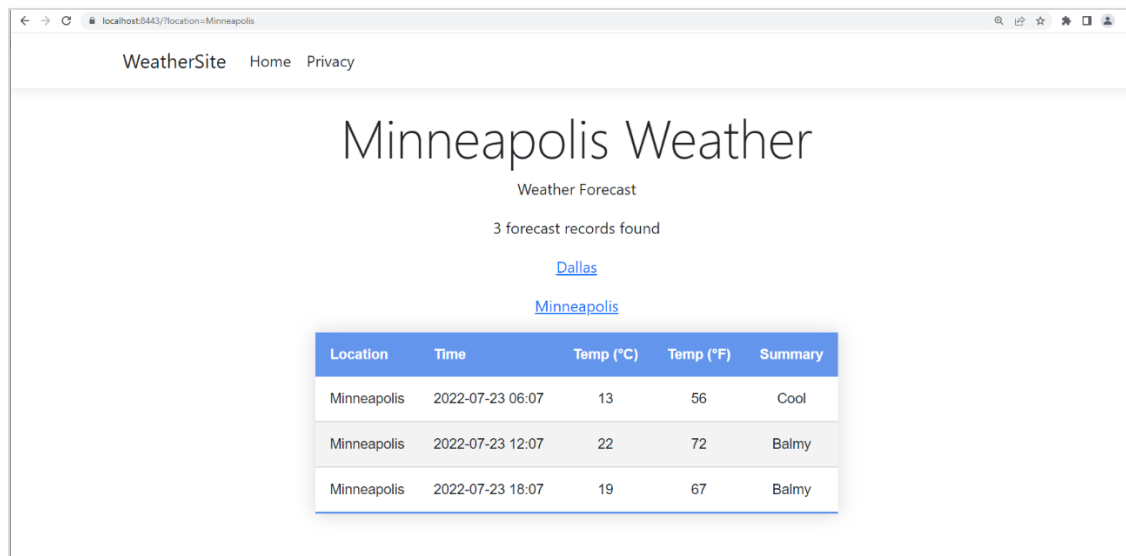
Note: If you are using Visual Studio, set WeatherSite as the startup project before starting the debugger.



The screenshot shows a web browser at localhost:5443 displaying the WeatherSite. The page title is "Dallas Weather" with a subtitle "Weather Forecast". It indicates "3 forecast records found". Below this are two links: "Dallas" (highlighted) and "Minneapolis". A table displays the weather data for Dallas.

Location	Time	Temp (°C)	Temp (°F)	Summary
Dallas	2022-07-23 06:07	33	92	Hot
Dallas	2022-07-23 12:07	43	109	Scorching
Dallas	2022-07-23 18:07	36	97	Hot

2. Choose the **Minneapolis** link. The page should refresh with data for that city.



The screenshot shows the same web browser at localhost:5443, but the URL now includes "/location=Minneapolis". The page title is "Minneapolis Weather" with a subtitle "Weather Forecast". It indicates "3 forecast records found". Below this are two links: "Dallas" and "Minneapolis" (highlighted). A table displays the weather data for Minneapolis.

Location	Time	Temp (°C)	Temp (°F)	Summary
Minneapolis	2022-07-23 06:07	13	56	Cool
Minneapolis	2022-07-23 12:07	22	72	Balmy
Minneapolis	2022-07-23 18:07	19	67	Balmy

3. Stop debugging. Stop the WeatherSite debugger in your IDE.

Activity: Building an Amazon ECS Service with an ASP.NET Website

4. **Optional:** Configure production ports. Perform this step only if you have access to 80 and 443 to use as production ports for the web site.
 - a. In the IDE with the WeatherSite project open, open Properties/LaunchSettings.json in the code editor.
 - b. On line 17, change the **applicationUrl** values as follows:

```
"applicationUrl": "https://localhost:443;http://localhost:80",
```

- c. Save your changes and confirm the project builds successfully.

Check your work

After completing this section, you have done the following:

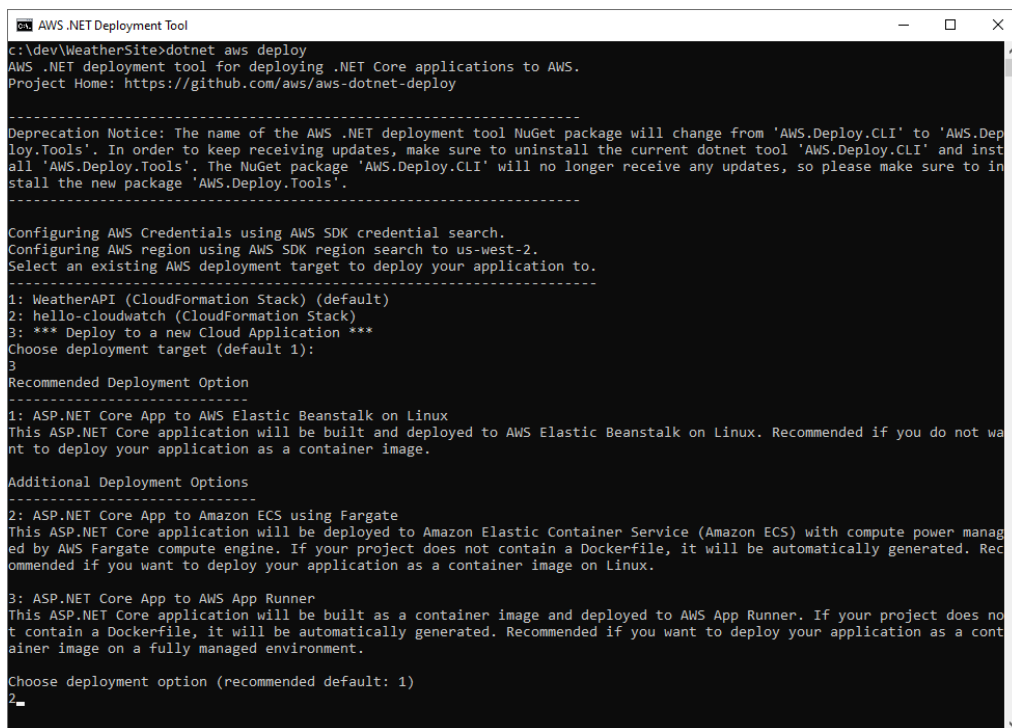
- ✓ Tested the WeatherSite project locally.
- ✓ Confirmed data is retrieved from the WeatherAPI backend Amazon ECS service.

Activity: Building an Amazon ECS Service with an ASP.NET Website

Section 10: Deploy WeatherSite to Amazon ECS

In this section, you will containerize and deploy the WeatherSite project to Amazon ECS using the AWS deploy tool for .NET CLI. This is similar to **Section 6: Deploy WeatherAPI to Amazon ECS**, but this time you deploy the frontend website.

1. Run the AWS deployment tool.
 - a. In a command prompt or terminal window, navigate to the WeatherSite folder.
 - b. Run the **dotnet aws deploy** command:
`dotnet aws deploy`
 - c. If prompted to select a new or existing deployment, choose **Deploy to a new Cloud Application**.
 - d. Select the choice for **ASP.NET Core App to Amazon ECS using Fargate**.



```

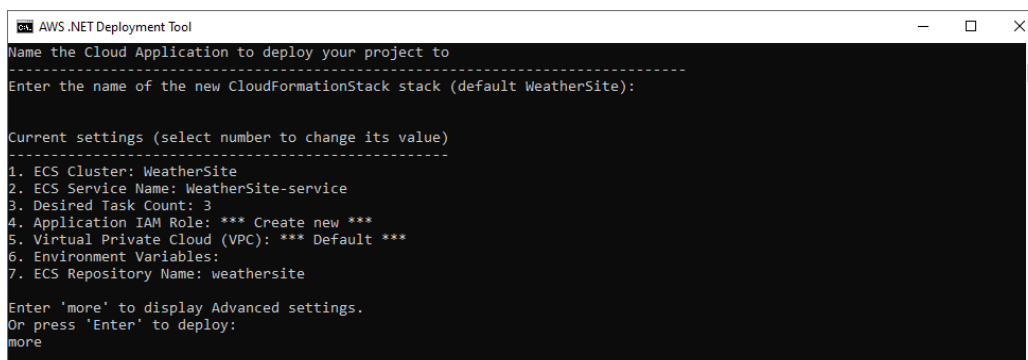
AWS .NET Deployment Tool
c:\dev\WeatherSite>dotnet aws deploy
AWS .NET deployment tool for deploying .NET Core applications to AWS.
Project Home: https://github.com/aws/aws-dotnet-deploy

-----
Deprecation Notice: The name of the AWS .NET deployment tool NuGet package will change from 'AWS.Deploy.CLI' to 'AWS.Deploy.Tools'. In order to keep receiving updates, make sure to uninstall the current dotnet tool 'AWS.Deploy.CLI' and install 'AWS.Deploy.Tools'. The NuGet package 'AWS.Deploy.CLI' will no longer receive any updates, so please make sure to install the new package 'AWS.Deploy.Tools'.
-----

Configuring AWS Credentials using AWS SDK credential search.
Configuring AWS region using AWS SDK region search to us-west-2.
Select an existing AWS deployment target to deploy your application to.
-----
1: WeatherAPI (CloudFormation Stack) (default)
2: hello-cloudwatch (CloudFormation Stack)
3: *** Deploy to a new Cloud Application ***
Choose deployment target (default 1):
3
Recommended Deployment Option
-----
1: ASP.NET Core App to AWS Elastic Beanstalk on Linux
This ASP.NET Core application will be built and deployed to AWS Elastic Beanstalk on Linux. Recommended if you do not want to deploy your application as a container image.
-----
Additional Deployment Options
-----
2: ASP.NET Core App to Amazon ECS using Fargate
This ASP.NET Core application will be deployed to Amazon Elastic Container Service (Amazon ECS) with compute power managed by AWS Fargate compute engine. If your project does not contain a Dockerfile, it will be automatically generated. Recommended if you want to deploy your application as a container image on Linux.
-----
3: ASP.NET Core App to AWS App Runner
This ASP.NET Core application will be built as a container image and deployed to AWS App Runner. If your project does not contain a Dockerfile, it will be automatically generated. Recommended if you want to deploy your application as a container image on a fully managed environment.
-----
Choose deployment option (recommended default: 1)
2

```

- e. For name of new CloudFormation stack, press **Enter** to accept the default name of **WeatherSite**.



```

AWS .NET Deployment Tool
Name the Cloud Application to deploy your project to
Enter the name of the new CloudFormationStack stack (default WeatherSite):

Current settings (select number to change its value)
-----
1. ECS Cluster: WeatherSite
2. ECS Service Name: WeatherSite-service
3. Desired Task Count: 3
4. Application IAM Role: *** Create new ***
5. Virtual Private Cloud (VPC): *** Default ***
6. Environment Variables:
7. ECS Repository Name: weathersite

Enter 'more' to display Advanced settings.
Or press 'Enter' to deploy:
more

```


Activity: Building an Amazon ECS Service with an ASP.NET Website

2. Edit settings.

- a. When Current settings are displayed, enter **more** to list the advanced settings. A numbered list of settings is displayed. Enter **9** to select Elastic Load Balancer and respond to prompts as follows:
 - Create New Load Balancer: **y**
 - Deregistration delay: **60**
 - Health Check path: **/**
 - Health Check Timeout: **5**
 - Health Check Interval: **30**
 - Healthy Threshold Count: **5**
 - Unhealthy Threshold Count: **2**
 - Internet-Facing: **y**

```
Current settings (select number to change its value)
-----
1 . ECS Cluster: WeatherSite
2 . ECS Service Name: WeatherSite-service
3 . Desired Task Count: 3
4 . Application IAM Role: *** Create new ***
5 . Virtual Private Cloud (VPC): *** Default ***
6 . ECS Service Security Groups:
7 . Task CPU: 256
8 . Task Memory: 512
9 . Elastic Load Balancer:
    Create New Load Balancer: True
    Deregistration delay (seconds): 60
    Health Check Path: /
    Health Check Timeout: 5
    Health Check Interval: 30
    Healthy Threshold Count: 5
    Unhealthy Threshold Count: 2
    Internet-Facing: True
10. AutoScaling:
    Enable: False
11. Environment Variables:
12. Docker Build Args:
13. Dockerfile Path:
14. Docker Execution Directory:
15. ECR Repository Name: weathersite

Or press 'Enter' to deploy:
```

Activity: Building an Amazon ECS Service with an ASP.NET Website

3. Deploy the WeatherSite project.

- Press **Enter** to start deployment. The deployment will create a load balancer, three ECS tasks, and new roles.

```
AWS .NET Deployment Tool
#16 5.038 Unpacking libgpm2:amd64 (1.20.7-8) ...
#16 5.081 Setting up libksba8:amd64 (1.5.0-3) ...
#16 5.098 Setting up libgpm2:amd64 (1.20.7-8) ...
#16 5.111 Setting up libsqlite3-0:amd64 (3.34.1-3) ...
#16 5.125 Setting up libnpt0:amd64 (1.6-3) ...
#16 5.138 Setting up libassuan0:amd64 (2.5.3-7.1) ...
#16 5.152 Setting up gnupg-l10n (2.2.27-2+deb11u2) ...
#16 5.166 Setting up libncursesw6:amd64 (6.2+20201114-2) ...
#16 5.181 Setting up readline-common (8.1-1) ...
#16 5.196 Setting up pinentry-curses (1.1.0-4) ...
#16 5.218 Setting up libreadline8:amd64 (8.1-1) ...
#16 5.231 Setting up gpgconf (2.2.27-2+deb11u2) ...
#16 5.245 Setting up gpg (2.2.27-2+deb11u2) ...
#16 5.259 Setting up gnupg-utils (2.2.27-2+deb11u2) ...
#16 5.272 Setting up gpg-agent (2.2.27-2+deb11u2) ...
#16 5.622 Setting up gpgsm (2.2.27-2+deb11u2) ...
#16 5.641 Setting up dirmngr (2.2.27-2+deb11u2) ...
#16 5.744 Setting up gpg-wks-server (2.2.27-2+deb11u2) ...
#16 5.760 Setting up gpg-wks-client (2.2.27-2+deb11u2) ...
#16 5.773 Setting up gnupg (2.2.27-2+deb11u2) ...
#16 5.786 Processing triggers for libc-bin (2.31-13+deb11u3) ...
#16 6.065
#16 6.065 ## Installing the NodeSource Node.js 14.x repo...
#16 6.065
#16 6.070
#16 6.070 ## Populating apt-get cache...
#16 6.070
#16 6.070 + apt-get update
#16 6.126 Hit:1 http://deb.debian.org/debian bullseye InRelease
#16 6.136 Hit:2 http://deb.debian.org/debian-security bullseye-security InRelease
#16 6.148 Hit:3 http://deb.debian.org/debian bullseye-updates InRelease
#16 6.266 Reading package lists...
#16 6.745
#16 6.745 ## Installing packages required for setup: lsb-release...
#16 6.745
#16 6.745 + apt-get install -y lsb-release > /dev/null 2>&1
#16 10.13
#16 10.13 ## Confirming "bullseye" is supported...
#16 10.13
#16 10.13 + curl -sLf -o /dev/null 'https://deb.nodesource.com/node_14.x/dists/bullseye/Release'
```

- Wait while the website is containerized and deployed, which will take several minutes. When it completes, note the deployment details at the end of the output.

Record the details under **Application Endpoint**. You will need *endpoint* to test the website.
Endpoint URL: _____

```
arn:aws:cloudformation:us-west-2:██████████:stack/WeatherSite/cbe9e650-2199-11ed-b8b6-069cd7692927
WeatherSite

Deployment time: 161.82s

Stack ARN:
Total time: 179.85s

Resources
-----
Application Endpoint:
  Id: arn:aws:elasticloadbalancing:us-west-2:██████████:loadbalancer/app/Weath-Recip-10JHBC4QKHQVT/ada167c2da6f6
  Type: AWS::ElasticLoadBalancingV2::LoadBalancer
  Endpoint: http://Weath-Recip-10JHBC4QKHQVT-622638868.us-west-2.elb.amazonaws.com/
ECS Service:
  Id: arn:aws:ecs:us-west-2:██████████:service/WeatherSite/WeatherSite-service
  Type: AWS::ECS::Service
ECS Cluster:
  Id: WeatherSite
  Type: AWS::ECS::Cluster

c:\dev\WeatherSite>
```

Activity: Building an Amazon ECS Service with an ASP.NET Website

4. Review Dockerfile. In your project folder, notice that a **Dockerfile** file has been added. The AWS deployment tool created this and used it with Docker to containerize the project. The Dockerfile should look similar to the following. You might see 6.0 or 7.0 depending on which version you are using.

```
FROM mcr.microsoft.com/dotnet/aspnet:7.0 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

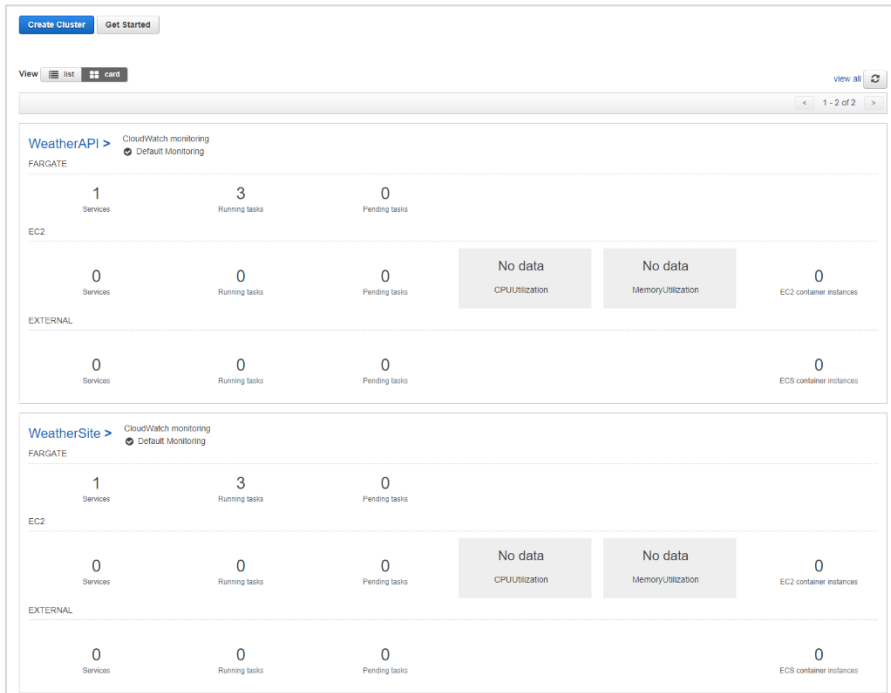
FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build
WORKDIR /src
COPY ["WeatherSite.csproj", ""]
RUN dotnet restore "WeatherSite.csproj"
COPY . .
WORKDIR "/src/"
RUN dotnet build "WeatherSite.csproj" -c Release -o /app/build

FROM build AS publish
RUN apt-get update -yq \
    && apt-get install curl gnupg -yq \
    && curl -sL https://deb.nodesource.com/setup_14.x | bash \
    && apt-get install nodejs -yq
RUN dotnet publish "WeatherSite.csproj" -c Release -o /app/publish

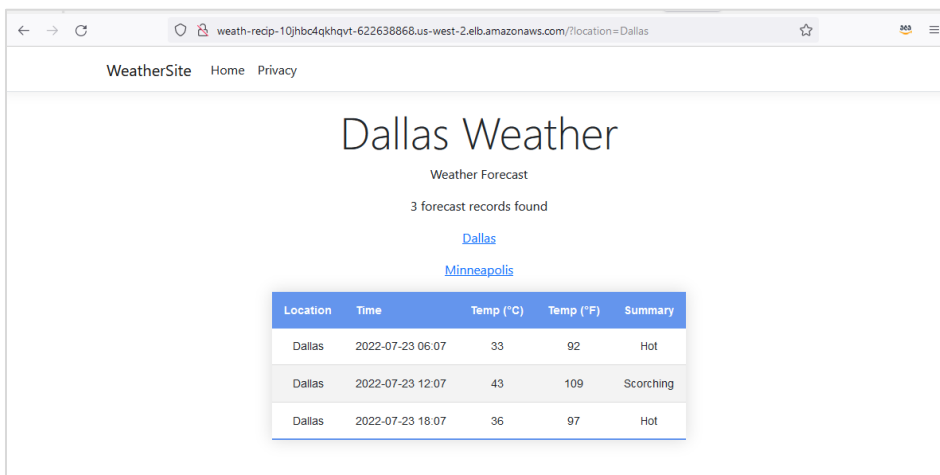
FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "WeatherSite.dll"]
```

Activity: Building an Amazon ECS Service with an ASP.NET Website

5. Review in the Amazon ECS console. Examine what's been deployed in the AWS console:
 - a. In the Amazon ECS console, navigate to **ECS**.
 - b. Select **Clusters** from the navigation pane. The **WeatherSite** cluster should be listed. There should be one FARGATE service with three running tasks.

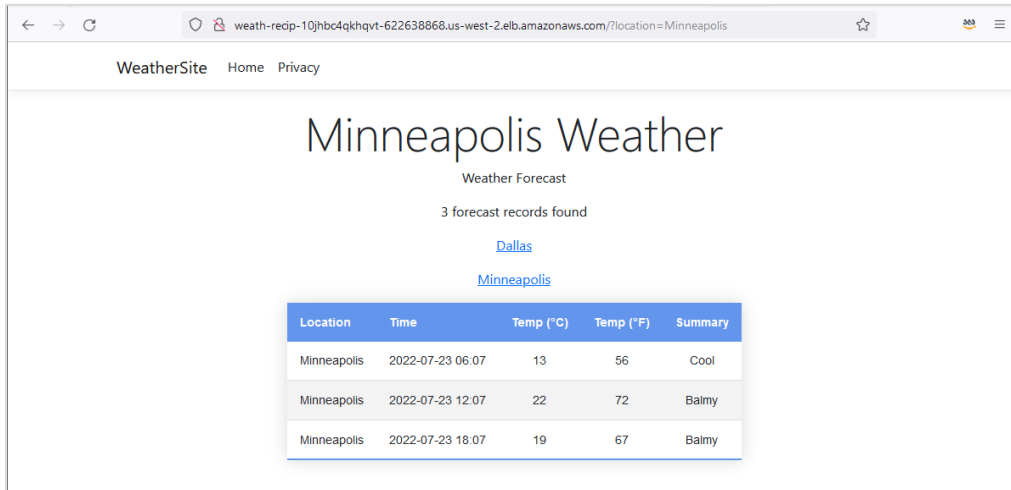


- c. Choose the **WeatherSite** cluster name to view its detail. Explore the tabs to see what was deployed.
6. Test the ECS endpoint.
 - a. In another browser tab, visit the endpoint URL you recorded earlier in Step 3b. The website should display, defaulting to location Dallas.



- b. Choose the link for **Minneapolis**, to display the data for that locale.

Activity: Building an Amazon ECS Service with an ASP.NET Website



WeatherSite Home Privacy

Minneapolis Weather

Weather Forecast

3 forecast records found

[Dallas](#)

[Minneapolis](#)

Location	Time	Temp (°C)	Temp (°F)	Summary
Minneapolis	2022-07-23 06:07	13	56	Cool
Minneapolis	2022-07-23 12:07	22	72	Balmy
Minneapolis	2022-07-23 18:07	19	67	Balmy

Congratulations, you have created a DynamoDB table and a website and API hosted in Amazon ECS on Fargate.

Check your work

You should now have done the following:

- ✓ Containerized and deployed the website to Amazon ECS with the AWS deployment tool for .NET CLI.
- ✓ Confirmed deployment in the Amazon ECS console.
- ✓ Confirmed the website is live and able to access the Weather API service.
- ✓ Verified that the website, API, and DynamoDB table work together on AWS.

Activity: Building an Amazon ECS Service with an ASP.NET Website

Section 11: Deploy an Update

In this section, you will add more data to your DynamoDB table, update the website with a third location, and deploy the updated website. You will also change the task count from 3 to 2.

1. Add data for another location to the DynamoDB weather table.
 - a. In the AWS Management Console, navigate to **DynamoDB**, then **Tables**, and choose the **Weather** table.
 - b. Choose **Explore table items**. Add three more items for a new location such as **Seattle** or a location of your choosing, just as you did earlier in **Section 3: Create an Amazon DynamoDB table**.

Completed Read capacity units consumed: 0.5

Items returned (9)

Actions Create item

<input type="checkbox"/>	Location	Timestamp	Summary	TempC	TempF
<input type="checkbox"/>	Minneapolis	2022-07-23T06:00:00	Cool	13	56
<input type="checkbox"/>	Minneapolis	2022-07-23T12:00:00	Balmy	22	72
<input type="checkbox"/>	Minneapolis	2022-07-23T18:00:00	Balmy	19	67
<input type="checkbox"/>	Seattle	2022-07-23T06:00:00	Cool	12	54
<input type="checkbox"/>	Seattle	2022-07-23T12:00:00	Comfortable	18	66
<input type="checkbox"/>	Seattle	2022-07-23T18:00:00	Cool	14	58
<input type="checkbox"/>	Dallas	2022-07-23T06:00:00	Hot	33	92
<input type="checkbox"/>	Dallas	2022-07-23T12:00:00	Scorching	43	109
<input type="checkbox"/>	Dallas	2022-07-23T18:00:00	Hot	36	97

2. Update the website for the new location.
 - a. Open the WeatherSite project in your IDE.
 - b. Open Pages/Index.cshtml in the code editor, and insert the following code after the link for Minneapolis:

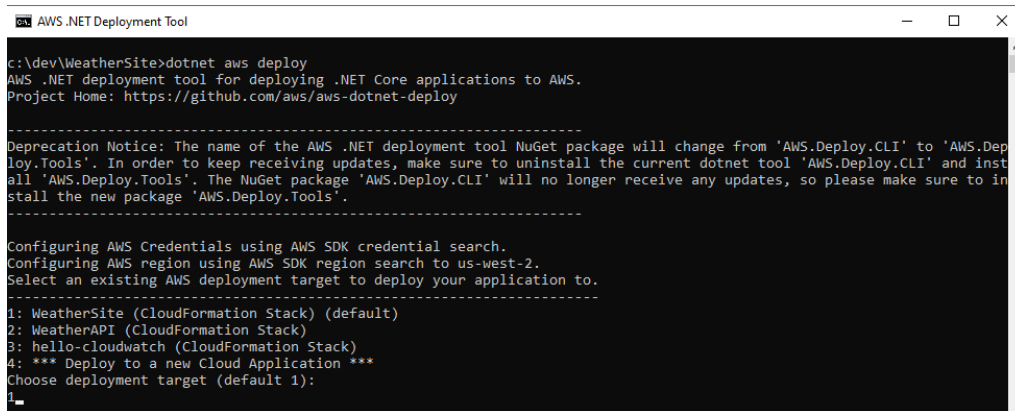
```
<p><a href="/?location=Seattle">Seattle</a></p>
```

- c. Save your changes and build the project.
3. Deploy an update.
 - a. Open a command prompt or terminal window and navigate to the WeatherSite project folder.
 - b. Run the command **dotnet aws deploy** to deploy an update:

```
dotnet aws deploy
```

Activity: Building an Amazon ECS Service with an ASP.NET Website

- c. Select the choice for **WeatherSite (CloudFormation Stack) (default)**.



```

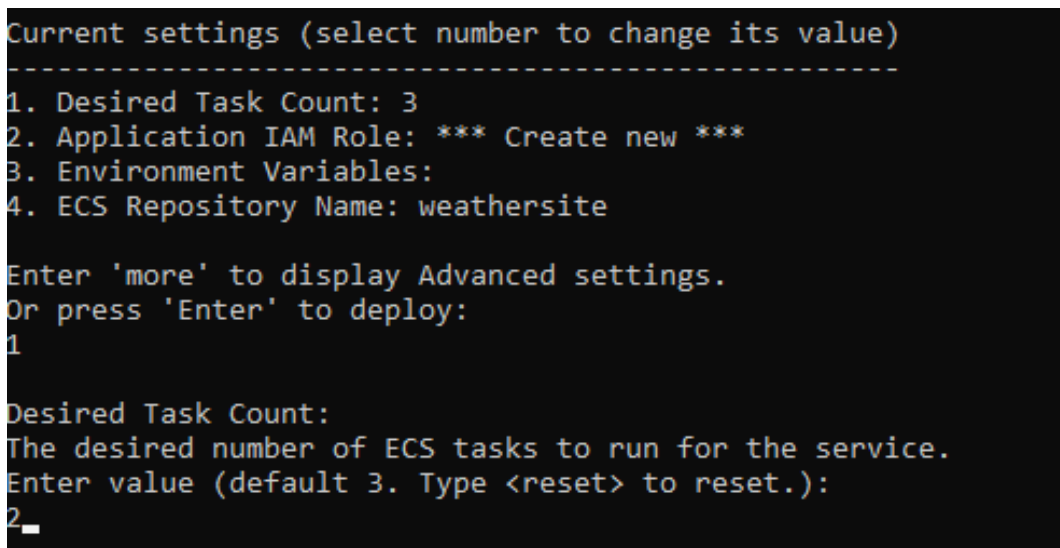
AWS .NET Deployment Tool
c:\dev\WeatherSite>dotnet aws deploy
AWS .NET deployment tool for deploying .NET Core applications to AWS.
Project Home: https://github.com/aws/aws-dotnet-deploy

-----
Deprecation Notice: The name of the AWS .NET deployment tool NuGet package will change from 'AWS.Deploy.CLI' to 'AWS.Deploy.Tools'. In order to keep receiving updates, make sure to uninstall the current dotnet tool 'AWS.Deploy.CLI' and install 'AWS.Deploy.Tools'. The NuGet package 'AWS.Deploy.CLI' will no longer receive any updates, so please make sure to install the new package 'AWS.Deploy.Tools'.
-----

Configuring AWS Credentials using AWS SDK credential search.
Configuring AWS region using AWS SDK region search to us-west-2.
Select an existing AWS deployment target to deploy your application to.
-----
1: WeatherSite (CloudFormation Stack) (default)
2: WeatherAPI (CloudFormation Stack)
3: hello-cloudwatch (CloudFormation Stack)
4: *** Deploy to a new Cloud Application ***
Choose deployment target (default 1):
1_

```

- d. When Current settings are displayed, select **1. Desired Task Count** and change it from 3 to 2.



```

Current settings (select number to change its value)
-----
1. Desired Task Count: 3
2. Application IAM Role: *** Create new ***
3. Environment Variables:
4. ECS Repository Name: weathersite

Enter 'more' to display Advanced settings.
Or press 'Enter' to deploy:
1

Desired Task Count:
The desired number of ECS tasks to run for the service.
Enter value (default 3. Type <reset> to reset.):
2_

```

Activity: Building an Amazon ECS Service with an ASP.NET Website

- e. Press **Enter** to begin the deployment, which will take a few minutes.

```
AWS .NET Deployment Tool
92a4e8a3140f: Waiting
a3616206dbdf: Layer already exists
a916c95ae016: Layer already exists
91961d2d529c: Layer already exists
5f70bf18a086: Layer already exists
d0084e19df22: Layer already exists
66ec52466d81: Layer already exists
e985411fba9e: Layer already exists
92a4e8a3140f: Layer already exists
637967170661528946: digest: sha256:e589d6462d4622dac833cda0ac291361003b3bd37821700b22abaf593766d9a size: 1995

Initiating deployment: ASP.NET Core App to Amazon ECS using Fargate
Configuring AWS Cloud Development Kit (CDK)...
Generating AWS Cloud Development Kit (AWS CDK) deployment project
Deploying AWS CDK project
npm WARN config global '--global', '--local' are deprecated. Use '--location=global' instead.
Using bootstrapping template from C:\Users\palldavi\.aws-dotnet-deploy\CDKBootstrapTemplate.yaml
  Bootstrapping environment aws://393840302653/us-west-2...
Trusted accounts for deployment: (none)
Trusted accounts for lookup: (none)
Using default execution policy of 'arn:aws:iam::aws:policy/AdministratorAccess'. Pass '--cloudformation-execution-policy' to customize.
Bootstrap stack already at version '13'. Not downgrading it to version '12' (use --force if you intend to downgrade)
  Environment aws://393840302653/us-west-2 bootstrapped (no changes).

npm WARN config global '--global', '--local' are deprecated. Use '--location=global' instead.
  Synthesis time: 19.18s

WeatherSite: deploying...
[0%] start: Publishing 2e707c23ff53f285d29bce022d1aa92a01d3caf285d5f613dd8a8b4a334c0446:393840302653-us-west-2
[100%] success: Published 2e707c23ff53f285d29bce022d1aa92a01d3caf285d5f613dd8a8b4a334c0446:393840302653-us-west-2
WeatherSite: creating CloudFormation changeset...
```

- f. Wait for the update to complete. Notice the endpoint for the Amazon ECS service has not changed, because this is an update.

4. Confirm task count change in the Amazon ECS console. Navigate to **Amazon ECS** and choose **Clusters**. Your **WeatherSite** task count, previously 3, should now be 2.

Amazon Elastic Container Service > Clusters

All Clusters [Info](#)

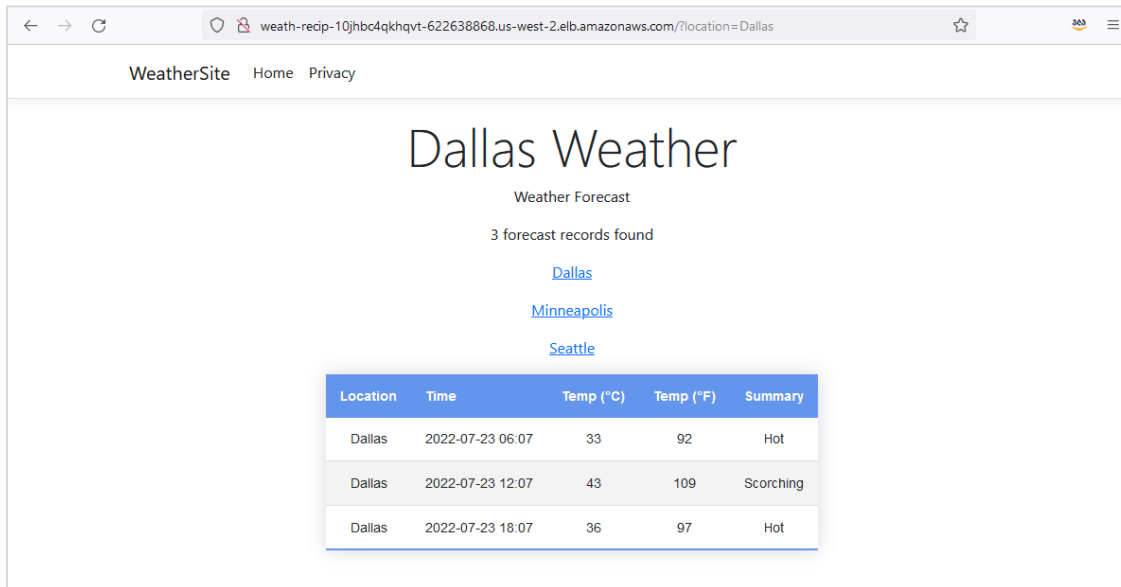
Clusters (2) [Create cluster](#)

Cluster	Services	Tasks	CloudWatch monitoring	Capacity provider strategy
WeatherAPI	1	0 Pending 3 Running	Default	No default found
WeatherSite	1	0 Pending 2 Running	Default	No default found

Activity: Building an Amazon ECS Service with an ASP.NET Website

5. Test the updated website.

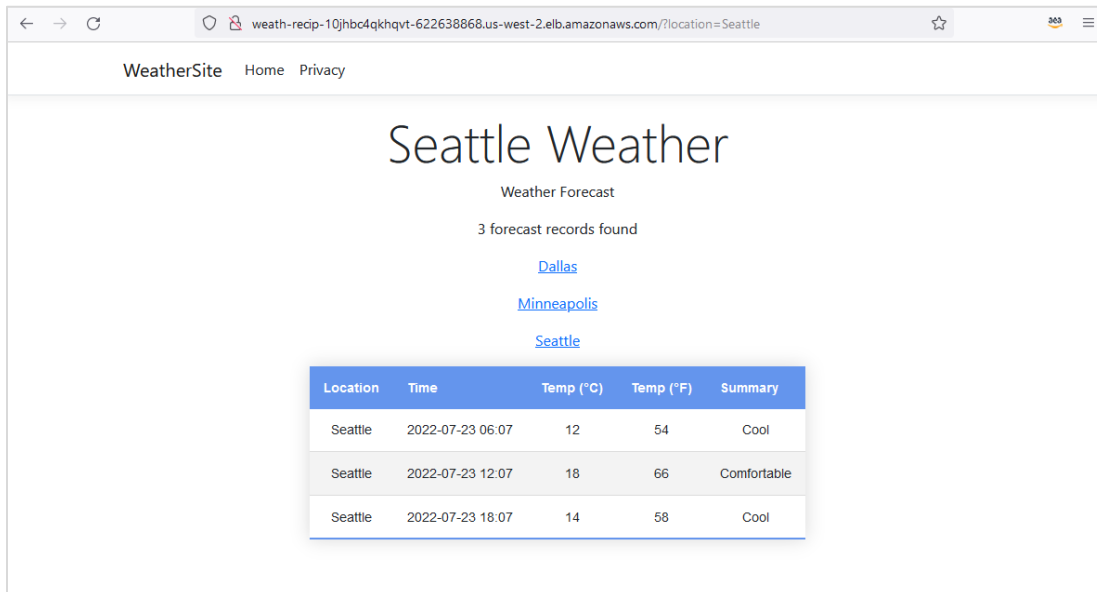
- In a browser, visit your website at the same endpoint you used previously in **Section 10, Step 3b**. Confirm the web page now shows the third location for Seattle.



The screenshot shows a web browser displaying the WeatherSite. The URL is `weath-recip-10jhbc4qkhqvt-622638868.us-west-2.elb.amazonaws.com/?location=Dallas`. The page title is "WeatherSite" with links for "Home" and "Privacy". The main heading is "Dallas Weather". Below it, it says "Weather Forecast" and "3 forecast records found". There are three links: "Dallas", "Minneapolis", and "Seattle". A table displays the forecast for Dallas:

Location	Time	Temp (°C)	Temp (°F)	Summary
Dallas	2022-07-23 06:07	33	92	Hot
Dallas	2022-07-23 12:07	43	109	Scorching
Dallas	2022-07-23 18:07	36	97	Hot

- Click the link for **Seattle** and confirm that the data changes.



The screenshot shows the same web browser displaying the WeatherSite, but the URL is now `weath-recip-10jhbc4qkhqvt-622638868.us-west-2.elb.amazonaws.com/?location=Seattle`. The page title is "WeatherSite" with links for "Home" and "Privacy". The main heading is "Seattle Weather". Below it, it says "Weather Forecast" and "3 forecast records found". There are three links: "Dallas", "Minneapolis", and "Seattle". A table displays the forecast for Seattle:

Location	Time	Temp (°C)	Temp (°F)	Summary
Seattle	2022-07-23 06:07	12	54	Cool
Seattle	2022-07-23 12:07	18	66	Comfortable
Seattle	2022-07-23 18:07	14	58	Cool

Activity: Building an Amazon ECS Service with an ASP.NET Website

Check your work

In this section, you have done the following:

- ✓ Added Seattle weather data to the DynamoDB table.
- ✓ Updated the website project main page with a Seattle location link.
- ✓ Deployed the updated website to Amazon ECS with a reduced task count.
- ✓ Tested the updated website hosted in Amazon ECS.

Activity: Building an Amazon ECS Service with an ASP.NET Website

Section 12: Shut it down

When you're done with this exercise, follow these steps to deallocate the resources. You don't want to accrue AWS charges for something you're no longer using.

1. Delete the deployment for WeatherSite.
 - a. In a command or terminal window, navigate to the **WeatherSite** project folder.
 - b. Run the command **dotnet aws delete-deployment WeatherSite** and confirm the deletion:

```
dotnet aws delete-deployment WeatherSite
```

- c. Wait for the delete process to complete.
 - d. In the AWS Management Console, navigate to Amazon ECS and confirm that the **WeatherSite** cluster is no longer listed.
2. Delete the deployment for WeatherAPI.
 - a. In a command or terminal window, navigate to the **WeatherAPI** project folder.
 - b. Run the command **dotnet aws delete-deployment WeatherAPI** and confirm the deletion.

```
dotnet aws delete-deployment WeatherAPI
```

- c. In the AWS Management Console, confirm that the **WeatherAPI** cluster is no longer listed.
3. Delete the DynamoDB Weather table.
 - A. In the AWS Management Console, navigate to **Amazon DynamoDB**.
 - B. Choose the **Weather** table, and then choose **Delete** from the Actions menu.
 - C. Confirm the delete and wait for it to complete.
4. If you are using an **AWS Cloud9** environment,
 - a. In the AWS Management Console, navigate to **Cloud9**.
 - b. Delete the **FargateLab** environment.

Check your work

You should now have done the following:

- ✓ Deleted your two Amazon ECS deployments.
- ✓ Deleted all other resources you manually allocated from the AWS Management Console.

Activity: Building an Amazon ECS Service with an ASP.NET Website

Summary

In this lab activity, you achieved the following:

1. You created a DynamoDB table and populated it with weather data.
2. You developed a .NET Web API project that retrieves weather data from the DynamoDB table.
3. You deployed the WeatherAPI project to Amazon ECS on Fargate, using the AWS deployment tool for .NET CLI.
4. You developed a website that retrieves data from WeatherAPI.
5. You deployed the WeatherSite project to Amazon ECS on Fargate, using the AWS deployment tool for .NET CLI.
6. You updated weather data and the website and deployed the updated website to Amazon ECS and modified its task count.
7. You deleted the deployments for WeatherSite and WeatherAPI, and deleted the Weather DynamoDB table in the AWS console.

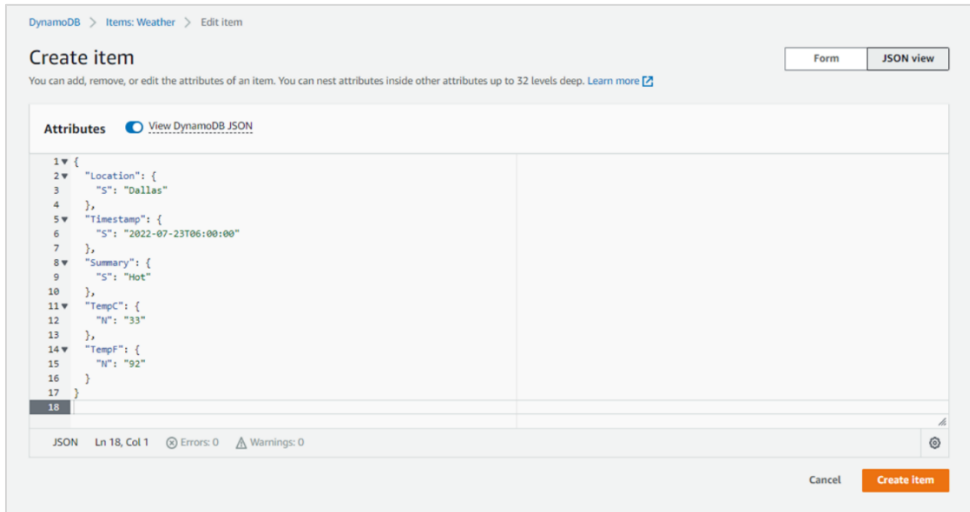
Activity: Building an Amazon ECS Service with an ASP.NET Website

Appendix: Populate the Weather table using JSON

1. In the AWS Management Console, navigate to Amazon DynamoDB.
2. Choose the Weather table name to get to its detail page. Then choose **Explore table items**.
3. Add the table items.
 - a. Choose **Create item** and then select the **JSON view**.
 - b. Enter the following JSON code (Dallas, morning) and choose **Create Item**.

```
{
  "Location": {
    "S": "Dallas"
  },
  "Timestamp": {
    "S": "2022-07-23T06:00:00"
  },
  "Summary": {
    "S": "Hot"
  },
  "TempC": {
    "N": "33"
  },
  "TempF": {
    "N": "92"
  }
}
```

Activity: Building an Amazon ECS Service with an ASP.NET Website



The screenshot shows the AWS DynamoDB 'Create item' console. The breadcrumb navigation is 'DynamoDB > Items: Weather > Edit item'. The title is 'Create item'. Below the title is a note: 'You can add, remove, or edit the attributes of an item. You can nest attributes inside other attributes up to 32 levels deep. [Learn more](#)'. There are two tabs: 'Form' and 'JSON view', with 'JSON view' selected. The 'Attributes' section has a toggle for 'View DynamoDB JSON'. The JSON document is displayed in a text area with line numbers 1 through 18. The document is a JSON object with the following structure: { "Location": { "S": "Dallas" }, "Timestamp": { "S": "2022-07-23T06:00:00" }, "Summary": { "S": "Hot" }, "TempC": { "N": "33" }, "TempF": { "N": "92" } }. At the bottom, there is a status bar showing 'JSON Ln 18, Col 1', 'Errors: 0', and 'Warnings: 0'. There are 'Cancel' and 'Create item' buttons at the bottom right.

```
1 {  
2   "Location": {  
3     "S": "Dallas"  
4   },  
5   "Timestamp": {  
6     "S": "2022-07-23T06:00:00"  
7   },  
8   "Summary": {  
9     "S": "Hot"  
10  },  
11  "TempC": {  
12    "N": "33"  
13  },  
14  "TempF": {  
15    "N": "92"  
16  }  
17 }  
18
```

- c. In the same way, add the second item (Dallas, mid-day) with the following JSON.

```
{  
  "Location": {  
    "S": "Dallas"  
  },  
  "Timestamp": {  
    "S": "2022-07-23T12:00:00"  
  },  
  "Summary": {  
    "S": "Scorching"  
  },  
  "TempC": {  
    "N": "43"  
  },  
  "TempF": {  
    "N": "109"  
  }  
}
```

Activity: Building an Amazon ECS Service with an ASP.NET Website

- d. Add the third item (Dallas, evening) with the following JSON.

```
{
  "Location": {
    "S": "Dallas"
  },
  "Timestamp": {
    "S": "2022-07-23T18:00:00"
  },
  "Summary": {
    "S": "Hot"
  },
  "TempC": {
    "N": "36"
  },
  "TempF": {
    "N": "97"
  }
}
```

- e. Add the fourth item (Minnesota, morning) with the following JSON.

```
{
  "Location": {
    "S": "Minneapolis"
  },
  "Timestamp": {
    "S": "2022-07-23T06:00:00"
  },
  "Summary": {
    "S": "Cool"
  },
  "TempC": {
    "N": "13"
  },
  "TempF": {
    "N": "56"
  }
}
```

Activity: Building an Amazon ECS Service with an ASP.NET Website

- f. Add the fifth item (Minnesota, mid-day) with the following JSON.

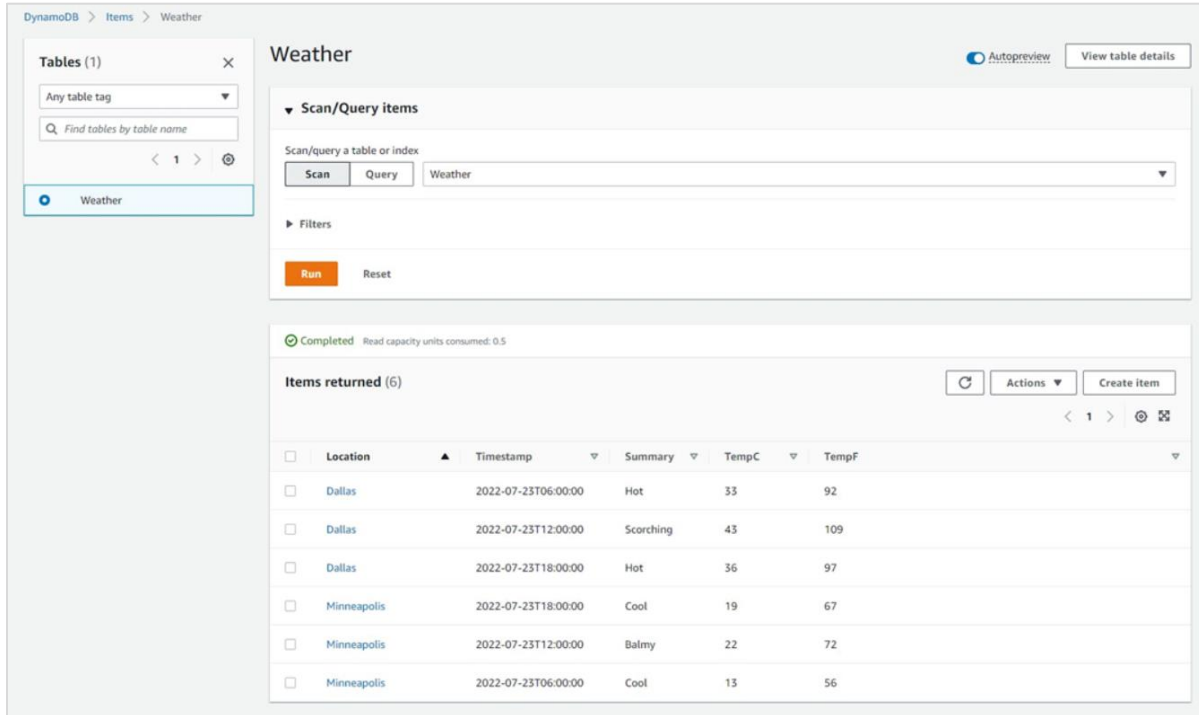
```
{
  "Location": {
    "S": "Minneapolis"
  },
  "Timestamp": {
    "S": "2022-07-23T12:00:00"
  },
  "Summary": {
    "S": "Balmy"
  },
  "TempC": {
    "N": "22"
  },
  "TempF": {
    "N": "72"
  }
}
```

- g. Add the sixth item (Minnesota, evening) with the following JSON.

```
{
  "Location": {
    "S": "Minneapolis"
  },
  "Timestamp": {
    "S": "2022-07-23T18:00:00"
  },
  "Summary": {
    "S": "Cool"
  },
  "TempC": {
    "N": "19"
  },
  "TempF": {
    "N": "67"
  }
}
```


Activity: Building an Amazon ECS Service with an ASP.NET Website

4. Verify your Weather table items from the DynamoDB console.



The screenshot shows the AWS DynamoDB console interface for the 'Weather' table. On the left, a sidebar lists the tables, with 'Weather' selected. The main area displays the 'Scan/Query items' section. A 'Scan' button is active, and the table 'Weather' is selected. Below this, a 'Run' button is visible. The results section shows 'Items returned (6)' with a table of weather data. The table has columns for Location, Timestamp, Summary, TempC, and TempF. The data includes entries for Dallas and Minneapolis with various timestamps and weather conditions.

Location	Timestamp	Summary	TempC	TempF
Dallas	2022-07-23T06:00:00	Hot	33	92
Dallas	2022-07-23T12:00:00	Scorching	43	109
Dallas	2022-07-23T18:00:00	Hot	36	97
Minneapolis	2022-07-23T18:00:00	Cool	19	67
Minneapolis	2022-07-23T12:00:00	Balmy	22	72
Minneapolis	2022-07-23T06:00:00	Cool	13	56

5. Optional: If you want, add more table items for your favorite locations.