AWS Auto Scaling Uygulaması Dokümantasyonu

İçindekiler

1. Proje Genel Bakış
2. Teknik Mimari
3. Kod Analizi
4. Performans Testleri
5. Maliyet Analizi
6. Güvenlik
7. Kurulum ve Yapılandırma
8. Kodun Çalışma Sırası ve Detaylı Açıklamaları
9. Sonuç

Projede yer alanlar:

Zülal Dündar – Muhammed Kerem Üstün

---

1. Proje Genel Bakış

Bu proje, hesaplama görevlerinin paralel işlenmesi için AWS EC2 örneklerini kullanan bir dağıtık hesaplama sistemi uygulamaktadır. Sistem, iş yükünü verimli bir şekilde dağıtmak için birden fazla EC2 örneğinin oluşturulmasını, yönetimini ve temizlenmesini otomatikleştirir.

2. Teknik Mimari

2.1 Kullanılan Teknolojiler

- **Arka Uç**: C# (.NET)
- **Bulut Platformu**: AWS
- **AWS Servisleri**: EC2, Systems Manager (SSM)
- **İletişim**: HTTP/REST
- **Örnek Tipi**: t2.micro
- **Bölge**: EU-Central-1 (Frankfurt)

3. Kod Analizi

3.1 Ana Bileşenler

Sınıf Yapısı:

```
using Amazon;
using Amazon.EC2;
using Amazon.EC2.Model;
using Amazon.Runtime;
using Amazon.SimpleSystemsManagement;
using Amazon.SimpleSystemsManagement.Model;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Net.Http;
using System.Text;
using System.Text.Json;
using System.Threading.Tasks;

class Program
```

```
{
    private static AmazonEC2Client ec2Client = null!;
    private static AmazonSimpleSystemsManagementClient
ssmClient = null!;
    private const int NumberOfWork = 5000000;
    private static int TaskCount = 8;
    private static readonly Random random = new Random();
    private static List<string> instancePublicIPs = new
List<string>();
    private static readonly HttpClient httpClient = new
HttpClient() { Timeout = TimeSpan.FromSeconds(30) };

    static public int[] a = null!;
    static public int[] b = null!;
    public static bool allTasksCompleted;
    public static int batchSize = 1000;

    private static Dictionary<string, long> timings = new
Dictionary<string, long>();

    static async Task Main(string[] args)
    {
        // Kodun ana işlevi burada başlar
    }

    // Diğer metodlar burada yer alır
}
```

Türkçe Açıklamalar:

- **ec2Client**: AWS EC2 örnek işlemlerini yönetir. EC2 örneklerini başlatma, silme ve izleme işlemleri bu istemci aracılığıyla gerçekleştirilir.
- **ssmClient**: AWS Systems Manager işlemlerini yönetir. EC2 örneklerine komut göndermek için kullanılır.

- **NumberOfWork**: Toplam hesaplama görevi sayısını belirtir (5.000.000).
- **TaskCount**: Paralel olarak çalıştırılacak EC2 örnek sayısını belirler (8 örnek).
- **random**: Rastgele sayı üretmek için kullanılır. Görevlerin dağıtımında kullanılabilir.
- **instancePublicIPs**: Başlatılan EC2 örneklerinin genel IP adreslerini saklar.
- **httpClient**: EC2 örneklerine HTTP istekleri göndermek için kullanılır. Zaman aşımı süresi 30 saniye olarak ayarlanmıştır.
- **a ve b Dizileri**: Hesaplamada kullanılacak iki adet dizi. Her biri rastgele sayılarla doldurulur.
- **allTasksCompleted**: Tüm görevlerin tamamlanıp tamamlanmadığını kontrol eden bir bayrak.
- **batchSize**: Her bir EC2 örneğine gönderilecek görev sayısını belirtir (1000).
- **timings**: Farklı işlemlerin ne kadar sürdüğünü ölçmek için kullanılan bir sözlük.

3.2 Ana Metodlar

Kodun çalışması sırasında çağrılan ve belirli işlevleri yerine getiren ana metodlar aşağıda detaylıca açıklanmıştır.

3.2.1 Main Metodu

```
static async Task Main(string[] args)
{
    var totalStopwatch = Stopwatch.StartNew();
    var currentStopwatch = new Stopwatch();
    try
    {
        // AWS Credentials
        currentStopwatch.Start();
        string awsAccessKey =
Environment.GetEnvironmentVariable("AWS_ACCESS_KEY") ??
"YOUR_AWS_ACCESS_KEY";
        string awsSecretKey =
Environment.GetEnvironmentVariable("AWS_SECRET_KEY") ??
"YOUR_AWS_SECRET_KEY";

        var credentials = new
BasicAWSCredentials(awsAccessKey, awsSecretKey);
        ec2Client = new AmazonEC2Client(credentials,
RegionEndpoint.EUCentral1);
        ssmClient = new
AmazonSimpleSystemsManagementClient(credentials,
RegionEndpoint.EUCentral1);
        currentStopwatch.Stop();
        timings["AWS Configuration"] =
currentStopwatch.ElapsedMilliseconds;
        // EC2 Instances Start
        Console.WriteLine("\nStarting EC2 instances...");
        currentStopwatch.Restart();
        var instanceIds = await StartInstances(TaskCount);
        currentStopwatch.Stop();
        timings["Starting EC2 Instances"] =
currentStopwatch.ElapsedMilliseconds;

        // Instance Wait
```

```
        Console.WriteLine("\nWaiting for instances to be
ready...");
        currentStopwatch.Restart();
        await WaitForInstancesRunning(instanceIds);
        currentStopwatch.Stop();
        timings["Waiting for Instances"] =
currentStopwatch.ElapsedMilliseconds;

        // Get IP Addresses
        Console.WriteLine("\nGetting IP addresses...");
        currentStopwatch.Restart();
        instancePublicIPs = await
GetPublicIPAddresses(instanceIds);
        currentStopwatch.Stop();
        timings["Getting IP Addresses"] =
currentStopwatch.ElapsedMilliseconds;

        if (instancePublicIPs.Count < TaskCount)
        {
            throw new Exception("Not all instances have
public IPs assigned.");
        }

        // Prepare Arrays
        Console.WriteLine("\nPreparing work arrays...");
        currentStopwatch.Restart();
        PrepareWorkArrays();
        currentStopwatch.Stop();
        timings["Preparing Arrays"] =
currentStopwatch.ElapsedMilliseconds;

        // Process Tasks
        Console.WriteLine("\nProcessing tasks...");
        currentStopwatch.Restart();
        try
        {
            var tasks = new List<Task>();
```

```
            for (int i = 0; i < TaskCount; i++)
            {
                int taskIndex = i;
                tasks.Add(ProcessTaskWithRetry(taskIndex));
            }

            await Task.WhenAll(tasks);
        }
        finally
        {
            currentStopwatch.Stop();
            timings["Processing Tasks"] =
currentStopwatch.ElapsedMilliseconds;
        }

        // Cleanup
        Console.WriteLine("\nCleaning up...");
        currentStopwatch.Restart();
        await CleanupInstances(instanceIds);
        currentStopwatch.Stop();
        timings["Cleanup"] =
currentStopwatch.ElapsedMilliseconds;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"\nFatal error: {ex.Message}");
        Console.WriteLine(ex.StackTrace);
    }
    finally
    {
        totalStopwatch.Stop();
        timings["Total Execution"] =
totalStopwatch.ElapsedMilliseconds;

        // Zamanlamaları göster
        Console.WriteLine("\n=== Execution Times ===");
        Console.WriteLine("-----------------------");
```

```
        foreach (var timing in timings.OrderBy(t => t.Key))
        {
            Console.WriteLine($"{timing.Key,-25}:
{timing.Value,8:N0} ms
({TimeSpan.FromMilliseconds(timing.Value).TotalSeconds,6:N2}
seconds)");
        }
        Console.WriteLine("----------------------");

        Console.WriteLine("\nPress any key to exit...");
        Console.ReadKey();
    }
}
```

Açıklama:

Main metodu, uygulamanın başlangıç noktasıdır ve tüm ana işlemleri sırayla tetikler:

1. **AWS Kimlik Doğrulaması:** AWS erişim anahtarı ve gizli anahtar alınarak EC2 ve SSM istemcileri oluşturulur.
2. **EC2 Örneklerinin Başlatılması:** Belirlenen sayıda EC2 örneği başlatılır.
3. **Örneklerin Hazırlanması:** Başlatılan örneklerin çalışır duruma gelmesi beklenir.
4. **IP Adreslerinin Alınması:** Başlatılan EC2 örneklerinin genel IP adresleri alınır.
5. **Veri Dizilerinin Hazırlanması:** İşlenecek veriler hazırlanır.
6. **Görevlerin İşlenmesi:** Görevler EC2 örneklerine dağıtılır ve işlenir.
7. **Temizlik:** İşlem tamamlandıktan sonra EC2 örnekleri temizlenir.
8. **Zamanlamaların Gösterilmesi:** Her adımın ne kadar sürdüğü konsola yazdırılır.

3.2.2 StartInstances Metodu

```
private static async Task<List<string>> StartInstances(int
instanceCount)
{
    var runInstancesRequest = new RunInstancesRequest
    {
        ImageId = "ami-0c1d9dfbfa2c2ced5",
        InstanceType = InstanceType.T2Micro,
        MinCount = instanceCount,
        MaxCount = instanceCount,
        KeyName = "Yourkey",
        SecurityGroupIds = new List<string> {
"sg-071a84c40b521775b" },
        UserData =
Convert.ToBase64String(Encoding.UTF8.GetBytes(@"#!/bin/bash
            # API başlatma komutları buraya eklenebilir"))
    };

    var runResponse = await
ec2Client.RunInstancesAsync(runInstancesRequest);
    var instanceIds = new List<string>();

    foreach (var instance in
runResponse.Reservation.Instances)
    {
        instanceIds.Add(instance.InstanceId);
        Console.WriteLine($"Instance {instance.InstanceId}
started.");
    }

    return instanceIds;
}
```

Açıklama:

Bu metod, belirtilen sayıda EC2 örneğini başlatır ve başlatılan örneklerin Instance ID'lerini döner.

- **RunInstancesRequest**: EC2 örneklerinin nasıl başlatılacağını belirten parametreleri içerir. Burada AMI ID'si, örnek tipi, SSH anahtarı, güvenlik grubu ve kullanıcı verisi (örneğin başlatma komutları) tanımlanmıştır.
- **RunInstancesAsync**: EC2 örneklerini başlatır ve başlatılan örneklerin bilgilerini döner.
- **Instance ID'lerin Toplanması**: Başlatılan her bir örneğin ID'si toplanır ve listede saklanır.

3.2.3 WaitForInstancesRunning Metodu

```
private static async Task
WaitForInstancesRunning(List<string> instanceIds)
{
    Console.WriteLine("Waiting for instances to be
ready...");
    bool allRunning = false;
    int maxAttempts = 30;
    int attempt = 0;
    var waitStopwatch = Stopwatch.StartNew();

    while (!allRunning && attempt < maxAttempts)
    {
        var status = await GetInstanceStatus(instanceIds);
        allRunning = status.All(s => s.Value ==
InstanceStateName.Running);

        if (!allRunning)
        {
```

```
            Console.WriteLine($"Waiting for instances to be
ready... Attempt {attempt + 1}/{maxAttempts} " +
                              $"(Elapsed:
{waitStopwatch.ElapsedMilliseconds / 1000.0:F1} seconds)");
            await Task.Delay(10000);
            attempt++;
        }
    }

    waitStopwatch.Stop();
    if (!allRunning)
    {
        throw new TimeoutException($"Instances failed to
reach running state in {waitStopwatch.ElapsedMilliseconds /
1000.0:F1} seconds");
    }

    Console.WriteLine($"All instances are running after
{waitStopwatch.ElapsedMilliseconds / 1000.0:F1} seconds.
Waiting for API initialization...");
    await Task.Delay(60000);
}
```

Açıklama:

Bu metod, başlatılan tüm EC2 örneklerinin çalışır duruma gelmesini bekler.

- **Durum Kontrolü:** Örneklerin durumları periyodik olarak kontrol edilir. Tüm örnekler "Running" durumuna ulaştığında işlem devam eder.
- **Maksimum Deneme Sayısı:** Örneklerin belirli bir süre içinde çalışır duruma gelmemesi durumunda zaman aşımı hatası fırlatılır.
- **API Başlatma Beklemesi:** Örneklerin çalışır duruma gelmesinden sonra, API'nin başlatılması için ek süre tanınır (60 saniye).

3.2.4 GetInstanceStatus Metodu

```
private static async Task<Dictionary<string,
InstanceStateName>> GetInstanceStatus(List<string>
instanceIds)
{
    var request = new DescribeInstancesRequest { InstanceIds
= instanceIds };
    var response = await
ec2Client.DescribeInstancesAsync(request);

    return response.Reservations
        .SelectMany(r => r.Instances)
        .ToDictionary(i => i.InstanceId, i => i.State.Name);
}
```

Açıklama:

Bu metod, verilen EC2 örneklerinin durumlarını alır ve her bir örneğin durumunu bir sözlükte saklar.

- **DescribeInstancesRequest**: Belirtilen örneklerin bilgilerini almak için kullanılır.
- **Durum Sözlüğü:** Her bir örneğin ID'si ile durumunun adı arasında bir sözlük oluşturulur.

3.2.5 GetPublicIPAddresses Metodu

```
private static async Task<List<string>>
GetPublicIPAddresses(List<string> instanceIds)
{
    var describeInstancesRequest = new
DescribeInstancesRequest { InstanceIds = instanceIds };
    var describeInstancesResponse = await
ec2Client.DescribeInstancesAsync(describeInstancesRequest);
```

```
    var ipAddresses = describeInstancesResponse.Reservations
        .SelectMany(r => r.Instances)
        .Select(i => i.PublicIpAddress)
        .Where(ip => !string.IsNullOrEmpty(ip))
        .ToList();

    Console.WriteLine("\nRetrieved IP addresses:");
    foreach (var ip in ipAddresses)
    {
        Console.WriteLine($"- {ip}");
    }

    return ipAddresses;
}
```

Açıklama:

Bu metod, başlatılan EC2 örneklerinin genel IP adreslerini alır.

- **IP Adreslerinin Toplanması:** Her bir örneğin Public IP adresi toplanır ve listeye eklenir.
- **Görünürlük:** Alınan IP adresleri konsola yazdırılır.

3.2.6 PrepareWorkArrays Metodu

```
private static void PrepareWorkArrays()
{
    a = new int[NumberOfWork];
    b = new int[NumberOfWork];
    for (int i = 0; i < NumberOfWork; i++)
    {
        a[i] = random.Next(1, 11);
        b[i] = random.Next(1000, 2001);
    }
}
```

Açıklama:

Bu metod, işlenecek olan veri setlerini hazırlar.

- **Dizilerin Oluşturulması:** a ve b dizileri, belirlenen iş miktarı kadar (5.000.000) elemanla doldurulur.
- **Rastgele Sayı Ataması:** a dizisi 1 ile 10 arasında, b dizisi ise 1000 ile 2000 arasında rastgele sayılarla doldurulur.

3.2.7 ProcessTaskWithRetry Metodu

```
private static async Task ProcessTaskWithRetry(int
taskIndex)
{
    var taskStopwatch = Stopwatch.StartNew();
    int totalJobs = NumberOfWork / TaskCount;
    int startIndex = taskIndex * totalJobs;
    int endIndex = (taskIndex + 1) * totalJobs;
    int completedBatches = 0;
    int totalBatches = (endIndex - startIndex) / batchSize +
((endIndex - startIndex) % batchSize > 0 ? 1 : 0);

    try
    {
        for (int i = startIndex; i < endIndex; i +=
batchSize)
        {
            int batchEnd = Math.Min(i + batchSize,
endIndex);
            await SendBatchWithRetry(taskIndex, i,
batchEnd);
            completedBatches++;

            if (completedBatches % 5 == 0)  // Her 5
batch'te bir ilerleme raporu
            {
```

```
                Console.WriteLine($"Task {taskIndex}:
Completed {completedBatches}/{totalBatches} batches " +
                                  $"({(completedBatches *
100.0 / totalBatches):F1}%) in
{taskStopwatch.ElapsedMilliseconds / 1000.0:F1} seconds");
            }
        }
    }
    finally
    {
        taskStopwatch.Stop();
        Console.WriteLine($"\nTask {taskIndex} completed in
{taskStopwatch.ElapsedMilliseconds:N0} ms " +

$"({taskStopwatch.ElapsedMilliseconds / 1000.0:F1}
seconds)");
    }
}
```

Açıklama:

Bu metod, belirli bir görev indeksine sahip iş parçalarını EC2 örneklerine gönderir ve gerektiğinde yeniden deneme mekanizması uygular.

- **İş Bölümü:** Toplam iş miktarı, görev sayısına bölünerek her bir görev için işin hangi kısmının gönderileceği belirlenir.
- **Toplu İşleme:** İş parçaları batchSize kadar olacak şekilde bölünür ve her bir parça SendBatchWithRetry metoduyla gönderilir.
- **İlerleme Raporu:** Her 5 batch'ten sonra ilerleme durumu konsola yazdırılır.
- **Zaman Takibi:** Görevin ne kadar sürdüğü ölçülür ve tamamlandığında konsola yazdırılır.

3.2.8 SendBatchWithRetry Metodu

```
private static async Task SendBatchWithRetry(int taskIndex,
int start, int end)
{
    const int maxRetries = 3;
    var batchStopwatch = Stopwatch.StartNew();

    for (int retry = 0; retry < maxRetries; retry++)
    {
        try
        {
            var data = new
            {
                TaskIndex = taskIndex,
                ArrayA = a.Skip(start).Take(end -
start).ToArray(),
                ArrayB = b.Skip(start).Take(end -
start).ToArray()
            };

            var jsonContent = new StringContent(
                JsonSerializer.Serialize(data),
                Encoding.UTF8,
                "application/json");

            var response = await httpClient.PostAsync(

$"http://{instancePublicIPs[taskIndex]}:5039/api/Worker/comp
uteTask",
                jsonContent);
            if (response.IsSuccessStatusCode)
            {
                batchStopwatch.Stop();
                Console.WriteLine($"Task {taskIndex}: Batch
{start}-{end} completed in
{batchStopwatch.ElapsedMilliseconds} ms");
```

```
                return;
            }

            string errorResponse = await
response.Content.ReadAsStringAsync();
            Console.WriteLine($"Task {taskIndex}: Attempt
{retry + 1} failed with status {response.StatusCode}. Error:
{errorResponse}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Task {taskIndex}: Attempt
{retry + 1} failed with error: {ex.Message}");
            if (retry == maxRetries - 1) throw;
        }

        await Task.Delay(2000 * (retry + 1));
    }
}
```

Açıklama:

Bu metod, belirli bir iş parçasını EC2 örneğine HTTP üzerinden gönderir ve gerektiğinde yeniden deneme yapar.

- **Veri Hazırlama:** Görev indeksi, ArrayA ve ArrayB dizilerinden ilgili iş parçaları alınır ve JSON formatında paketlenir.
- **HTTP İsteği:** EC2 örneğine POST isteği gönderilir. Başarılı olursa işlem tamamlanır.
- **Hata Yönetimi:** İstek başarısız olursa, belirlenen sayıda yeniden deneme yapılır. Her denemede belirli bir gecikme uygulanır.
- **Zaman Takibi:** Görevin ne kadar sürdüğü ölçülür ve başarılı olduğunda konsola yazdırılır.

3.2.9 CleanupInstances Metodu

```
private static async Task CleanupInstances(List<string>
instanceIds)
{
      Console.WriteLine("Terminating instances...");
        var cleanupStopwatch = Stopwatch.StartNew();
        try
        {
            var runningInstances = await
GetRunningInstances();
            if (runningInstances.Any())
            {
                Console.WriteLine("Terminating running
instances:");
                foreach (var instanceId in runningInstances)
                {
                    Console.WriteLine($"- {instanceId}");
                }
                var terminateRequest = new
TerminateInstancesRequest { InstanceIds = runningInstances
};
                var terminateResponse = await
ec2Client.TerminateInstancesAsync(terminateRequest);
                foreach (var instance in
terminateResponse.TerminatingInstances)
                {
                    Console.WriteLine($"Instance
{instance.InstanceId} is terminating. Current state:
{instance.CurrentState.Name}");
                }
            }
            else
```

```
            {
                Console.WriteLine("No running instances to
terminate.");
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error during cleanup:
{ex.Message}");
        }
        finally
        { cleanupStopwatch.Stop();
            Console.WriteLine($"Cleanup completed in
{cleanupStopwatch.ElapsedMilliseconds:N0} ms");
        }
}
```

Açıklama:

Bu metod, tüm işlerin tamamlanmasının ardından EC2 örneklerini siler ve temizler.

- **Çalışan Örneklerin Alınması:** GetRunningInstances metodu kullanılarak çalışan tüm örnekler belirlenir.
- **Silme İşlemi:** Çalışan örnekler TerminateInstancesAsync metodu ile siler.
- **Durum Takibi:** Silme işlemi sırasında her bir örneğin mevcut durumu konsola yazdırılır.
- **Hata Yönetimi:** Temizlik sırasında oluşabilecek hatalar yakalanır ve konsola yazdırılır.
- **Zaman Takibi:** Temizlik işleminin ne kadar sürdüğü ölçülür ve konsola yazdırılır.

3.2.10 GetRunningInstances Metodu

```
private static async Task<List<string>>
GetRunningInstances()
{
    var describeInstancesRequest = new
DescribeInstancesRequest();
    var describeInstancesResponse = await
ec2Client.DescribeInstancesAsync(describeInstancesRequest);

    return describeInstancesResponse.Reservations
        .SelectMany(r => r.Instances)
        .Where(i => i.State.Name ==
InstanceStateName.Running)
        .Select(i => i.InstanceId)
        .ToList();
}
```

Açıklama:

Bu metod, AWS hesabınızda çalışan tüm EC2 örneklerinin ID'lerini döner.

- **DescribeInstancesRequest**: Tüm örneklerin bilgilerini almak için kullanılır.
- **Filtreleme:** Sadece "Running" durumunda olan örnekler seçilir.
- **Sonuç:** Çalışan örneklerin ID'leri bir liste olarak döner.

3.3 Hata Yönetimi

Sistem, birden fazla hata yönetim mekanizması içerir:

- **Yeniden Deneme Mantığı:** Başarısız API çağrıları için yeniden deneme yapılır.
- **Örnek Durumu İzleme:** EC2 örneklerinin durumları düzenli olarak izlenir.
- **Temizleme Prosedürleri:** Başarısız işlemler için örneklerin temizlenmesi sağlanır.
- **Zaman Aşımı Yönetimi:** Uzun süren işlemler için zaman aşımı uygulanır.

4. Performans Testleri

4.1 Test Senaryoları

t2.micro İşlemcisi

Test No	İş Öğeleri	TASK	Batch Boyutu	Toplam Süre (s)
1	5,000,000	1	1,000	613.61
2	500,000	8	1,000	116.38
3	5,000,000	8	100	527.63
4	5,000,000	8	10,000	437.20
5	5,000,000	8	1,000	247.799
6	5,000,000	12	1,000	162.418
7	5,000,000	16	1,000	147.428
8	5,000,000	24	1,000	137.155

8 TASK - 1,000 BATCH - 5000000 İŞ ÖĞESİ

=== Execution Times ===

AWS Configuration	236 ms (0,24 seconds)
Cleanup	1.261 ms (1,26 seconds)
Getting IP Addresses	495 ms (0,49 seconds)
Preparing Arrays	54 ms (0,05 seconds)
Processing Tasks	140.470 ms (140,47 seconds)
Starting EC2 Instances	3.737 ms (3,74 seconds)
Total Execution	247.799 ms (247,80 seconds)
Waiting for Instances	101.534 ms (101,53 seconds)

12 TASK - 1,000 BATCH - 5000000 İŞ ÖĞESİ

=== Execution Times ===

AWS Configuration	190 ms (0,19 seconds)
Cleanup	1.244 ms (1,24 seconds)
Getting IP Addresses	332 ms (0,33 seconds)
Preparing Arrays	60 ms (0,06 seconds)
Processing Tasks	64.980 ms (64,98 seconds)
Starting EC2 Instances	3.237 ms (3,24 seconds)
Total Execution	162.418 ms (162,42 seconds)
Waiting for Instances	92.364 ms (92,36 seconds)

16 TASK - 1,000 BATCH - 5000000 İŞ ÖĞESİ

=== Execution Times ===

AWS Configuration	174 ms (0,17 seconds)
Cleanup	1.421 ms (1,42 seconds)
Getting IP Addresses	281 ms (0,28 seconds)
Preparing Arrays	48 ms (0,05 seconds)
Processing Tasks	49.165 ms (49,16 seconds)
Starting EC2 Instances	3.811 ms (3,81 seconds)
Total Execution	147.428 ms (147,43 seconds)
Waiting for Instances	92.475 ms (92,47 seconds)

24 TASK - 1,000 BATCH - 5000000 İŞ ÖĞESİ

=== Execution Times ===

AWS Configuration	227 ms (0,23 seconds)
Cleanup	2.295 ms (2,29 seconds)
Getting IP Addresses	501 ms (0,50 seconds)
Preparing Arrays	48 ms (0,05 seconds)
Processing Tasks	34.474 ms (34,47 seconds)
Starting EC2 Instances	5.163 ms (5,16 seconds)
Total Execution	137.155 ms (137,16 seconds)
Waiting for Instances	94.435 ms (94,44 seconds)

t.2 Performans Analizi

- **Optimal Konfigürasyon:** 24 task ve 1,000 batch boyut
- **İşlem Verimliliği:** Batch boyutu 1,000'den büyük olduğunda düşüyor
- **Task Sayısına Göre Verim Artışı:** Task sayısı arttıkça verim artışı da artıyor

i7-11370H CPU @ 3.30GHz İşlemcisi

Test No	İş Öğeleri	TASK	Batch Boyutu	Toplam Süre (s)
1	500,000	8	1,000	7.445
2	5,000,000	8	1,000	50.411
3	5,000,000	8	100	56.761
4	5,000,000	1	1,000	164.059
5	5,000,000	8	100	527.63
6	50,000,000	8	10,000	465.120
7	5,000,000	16	1,000	49.649
8	5,000,000	12	1,000	49.153

i7-11370H Performans Analizi

- **Optimal Konfigürasyon:** 12 task ve 1,000 batch boyutu
- **Task Sayısına Göre Verim Artışı:** 8 task in üzerine çıkıldıkça verim artışı azalıyor

i7-10750H CPU @ 2.60GHz İşlemcisi

Test No	İş Öğeleri	TASK	Batch Boyutu	Toplam Süre (s)
1	5,000,000	8	1,000	80,04
2	5,000,000	8	10,000	73,29
3	5,000,000	8	100	88,91
4	5,000,000	1	1,000	415,44
5	5,000,000	16	1,000	65,43
6	5,000,000	12	100	81,47
7	5,000,000	12	1,000	61,50
8	5,000,000	12	10,000	64,43

i7-10750H Performans Analizi

- **Optimal Konfigürasyon:** 12 task ve 1,000 batch boyutu
- **Task Sayısına Göre Verim Artışı:** 12 task dışında verim artışı azalıyor

5. Maliyet Analizi

5.1 Altyapı Maliyetleri

- **t2.micro Maliyeti:** Saat başı $0.0116
- **Ortalama Çalışma Süresi:** 210.08 saniye
- **Çalıştırma Başına Maliyet (8 örnek):** $0.0054
- **Aylık Maliyet (100 çalıştırma):** $0.54

6. Güvenlik

6.1 Mevcut Güvenlik Önlemleri

- AWS kimlik bilgileri yönetimi
- Güvenlik grubu kısıtlamaları
- Anahtar çiftleri ile örnek erişim kontrolü
- HTTP istemci zaman aşımı ayarları

6.2 Güvenlik Önerileri

1. AWS IAM rollerini uygulama
2. AWS Secrets Manager kullanımı
3. VPC güvenliğini etkinleştirme
4. İstek doğrulama uygulaması
5. SSL/TLS şifreleme ekleme

7. Kurulum ve Yapılandırma

7.1 Worker API Kurulumu

Worker API'nin EC2 instance'larına kurulumu için aşağıdaki adımlar izlenmelidir:

1. **Dosyaların Kopyalanması**

```
sudo mkdir -p /var/www/Worker_api/
sudo cp -r~/publish/Worker_api/Worker_api/bin/Debug/net8.0/*
/var/www/Worker_api/
```

2. **Systemd Servis Yapılandırması** a. Servis dosyasının oluşturulması:

```
sudo nano /etc/systemd/system/worker_api.service
```

b. Servis dosyası içeriği:

```
[Unit]
Description=Worker API Service
After=network.target

[Service]
Type=simple
WorkingDirectory=/var/www/Worker_api
ExecStart=/usr/bin/dotnet /var/www/Worker_api/Worker_api.dll
Restart=always
RestartSec=10
SyslogIdentifier=worker_api

[Install]
WantedBy=multi-user.target
```

3. **Servisin Etkinleştirilmesi**

```
sudo systemctl daemon-reload
sudo systemctl start worker_api
sudo systemctl enable worker_api
```

4. **Servis Durumu Kontrolü**

```
sudo systemctl status worker_api
```

7.2 AMI Oluşturma

Worker API kurulumu tamamlandıktan sonra, bu yapılandırmayı içeren bir AMI (Amazon Machine Image) oluşturulmalıdır. Bu AMI, dağıtık hesaplama sisteminde kullanılacak diğer EC2 örnekleri için temel teşkil edecektir.

AMI oluşturma adımları:

1. AWS Console üzerinden ilgili EC2 instance'ı seçin
2. Actions > Image and templates > Create image
3. AMI için uygun bir isim ve açıklama girin
4. Create image'a tıklayarak AMI oluşturma işlemini başlatın

Bu AMI, **Bölüm 3.2.1**'de belirtilen StartInstances metodunda kullanılacak olan ImageId değeri için kullanılmalıdır.

7.3 Güvenlik Notları

- Worker API servis dosyasındaki izinler ve çalıştırma hakları doğru yapılandırılmalıdır
- API'nin çalıştığı port (5039) güvenlik gruplarında açık olmalıdır
- Systemd servis yapılandırması en az yetki prensibi gözetilerek yapılmalıdır

8. Kodun Çalışma Sırası ve Detaylı Açıklamaları

8.1 Genel Kod Yapısı ve Akışı

Kod, AWS SDK kullanarak EC2 örneklerini başlatmak, bu örneklere iş dağıtmak ve iş tamamlandıktan sonra örnekleri temizlemek amacıyla yazılmıştır. İşlem sırasında SSM (AWS Systems Manager) kullanılarak EC2 örneklerine komutlar gönderilir ve örneklerin IP adresleri üzerinden HTTP istekleri yapılır.

Kodun genel akışı şu şekildedir:

1. **AWS İstemcilerinin Oluşturulması:** EC2 ve SSM istemcileri oluşturulur.
2. **EC2 Örneklerinin Başlatılması:** Belirlenen sayıda EC2 örneği başlatılır.
3. **Veri Dizilerinin Hazırlanması:** İşlenecek veriler hazırlanır.
4. **Görevlerin Dağıtılması ve İşlenmesi:** Görevler EC2 örneklerine dağıtılır ve işlenir.
5. **Örneklerin Temizlenmesi:** Tüm işler tamamlandığında EC2 örnekleri kapatılır.
6. **Zamanlamaların Gösterilmesi:** Her adımın süresi ölçülür ve konsola yazdırılır.

8.2 Adım Adım Metodlar ve İşlevleri

Adım 1: AWS İstemcilerinin Oluşturulması

```
// AWS Credentials
currentStopwatch.Start();
string awsAccessKey =
Environment.GetEnvironmentVariable("AWS_ACCESS_KEY") ?? "
YOUR_AWS_ACCESS_KEY ";
string awsSecretKey =
Environment.GetEnvironmentVariable("AWS_SECRET_KEY") ?? "
YOUR_AWS_SECRET_KEY ";

var credentials = new BasicAWSCredentials(awsAccessKey,
awsSecretKey);
ec2Client = new AmazonEC2Client(credentials,
RegionEndpoint.EUCentral1);
ssmClient = new
AmazonSimpleSystemsManagementClient(credentials,
RegionEndpoint.EUCentral1);
currentStopwatch.Stop();
timings["AWS Configuration"] =
currentStopwatch.ElapsedMilliseconds;
```

Açıklama:

- **AWS Kimlik Bilgileri:** Ortam değişkenlerinden AWS erişim anahtarı ve gizli anahtar alınır. Bu kimlik bilgileri, EC2 ve SSM istemcilerini oluşturmak için kullanılır.
- **İstemcilerin Oluşturulması:** AmazonEC2Client ve AmazonSimpleSystemsManagementClient nesneleri oluşturulur. Bu istemciler, AWS hizmetlerine erişim sağlar.
- **Zaman Takibi:** AWS konfigürasyonunun ne kadar sürdüğü ölçülür ve timings sözlüğüne eklenir.

Adım 2: EC2 Örneklerinin Başlatılması

```
// EC2 Instances Start
Console.WriteLine("\nStarting EC2 instances...");
currentStopwatch.Restart();
var instanceIds = await StartInstances(TaskCount);
currentStopwatch.Stop();
timings["Starting EC2 Instances"] =
currentStopwatch.ElapsedMilliseconds;
```

Açıklama:

- **Görev Sayısı:** TaskCount değişkeni (8) kadar EC2 örneği başlatılır.
- **StartInstances Metodu:** Bu metod çağrılarak EC2 örnekleri başlatılır ve başlatılan örneklerin ID'leri alınır.
- **Zaman Takibi:** EC2 örneklerinin başlatılmasının ne kadar sürdüğü ölçülür ve timings sözlüğüne eklenir.

Adım 3: Örneklerin Hazırlanması

```
// Instance Wait
Console.WriteLine("\nWaiting for instances to be ready...");
currentStopwatch.Restart();
await WaitForInstancesRunning(instanceIds);
currentStopwatch.Stop();
timings["Waiting for Instances"] =
currentStopwatch.ElapsedMilliseconds;
```

Açıklama:

- **WaitForInstancesRunning Metodu:** Başlatılan tüm EC2 örneklerinin çalışır duruma gelmesi beklenir.
- **Zaman Takibi:** Örneklerin hazır hale gelmesinin ne kadar sürdüğü ölçülür ve timings sözlüğüne eklenir.

Adım 4: IP Adreslerinin Alınması

```
// Get IP Addresses
Console.WriteLine("\nGetting IP addresses...");
currentStopwatch.Restart();
instancePublicIPs = await GetPublicIPAddresses(instanceIds);
currentStopwatch.Stop();
timings["Getting IP Addresses"] =
currentStopwatch.ElapsedMilliseconds;

if (instancePublicIPs.Count < TaskCount)
{
    throw new Exception("Not all instances have public IPs
assigned.");
}
```

Açıklama:

- **GetPublicIPAddresses Metodu:** Başlatılan EC2 örneklerinin genel IP adresleri alınır.
- **Kontrol:** Tüm örneklerin IP adreslerine sahip olup olmadığı kontrol edilir. Eksik varsa hata fırlatılır.
- **Zaman Takibi:** IP adreslerinin alınmasının ne kadar sürdüğü ölçülür ve timings sözlüğüne eklenir.

Adım 5: Veri Dizilerinin Hazırlanması

```
// Prepare Arrays
Console.WriteLine("\nPreparing work arrays...");
currentStopwatch.Restart();
PrepareWorkArrays();
currentStopwatch.Stop();
timings["Preparing Arrays"] =
currentStopwatch.ElapsedMilliseconds;
```

Açıklama:

- **PrepareWorkArrays Metodu:** İşlenecek olan veri setleri (diziler a ve b) hazırlanır.
- **Zaman Takibi:** Veri dizilerinin hazırlanmasının ne kadar sürdüğü ölçülür ve timings sözlüğüne eklenir.

Adım 6: Görevlerin İşlenmesi

```
// Process Tasks
Console.WriteLine("\nProcessing tasks...");
currentStopwatch.Restart();
try
{
    var tasks = new List<Task>();
    for (int i = 0; i < TaskCount; i++)
    {
        int taskIndex = i;
        tasks.Add(ProcessTaskWithRetry(taskIndex));
    }

    await Task.WhenAll(tasks);
}
finally
```

```
{
    currentStopwatch.Stop();
    timings["Processing Tasks"] =
currentStopwatch.ElapsedMilliseconds;
}
```

Açıklama:

- **ProcessTaskWithRetry Metodu:** Her bir EC2 örneği için görev işlenir ve gerektiğinde yeniden deneme yapılır.
- **Görevlerin Paralel İşlenmesi:** Task.WhenAll ile tüm görevlerin tamamlanması beklenir.
- **Zaman Takibi:** Görevlerin işlenmesinin ne kadar sürdüğü ölçülür ve timings sözlüğüne eklenir.

Adım 7: Örneklerin Temizlenmesi

```
// Cleanup
Console.WriteLine("\nCleaning up...");
currentStopwatch.Restart();
await CleanupInstances(instanceIds);
currentStopwatch.Stop();
timings["Cleanup"] = currentStopwatch.ElapsedMilliseconds;
```

Açıklama:

- **CleanupInstances Metodu:** Tüm işler tamamlandıktan sonra EC2 örnekleri siler ve temizlenir.
- **Zaman Takibi:** Temizlik işleminin ne kadar sürdüğü ölçülür ve timings sözlüğüne eklenir.

Adım 8: Zamanlamaların Gösterilmesi

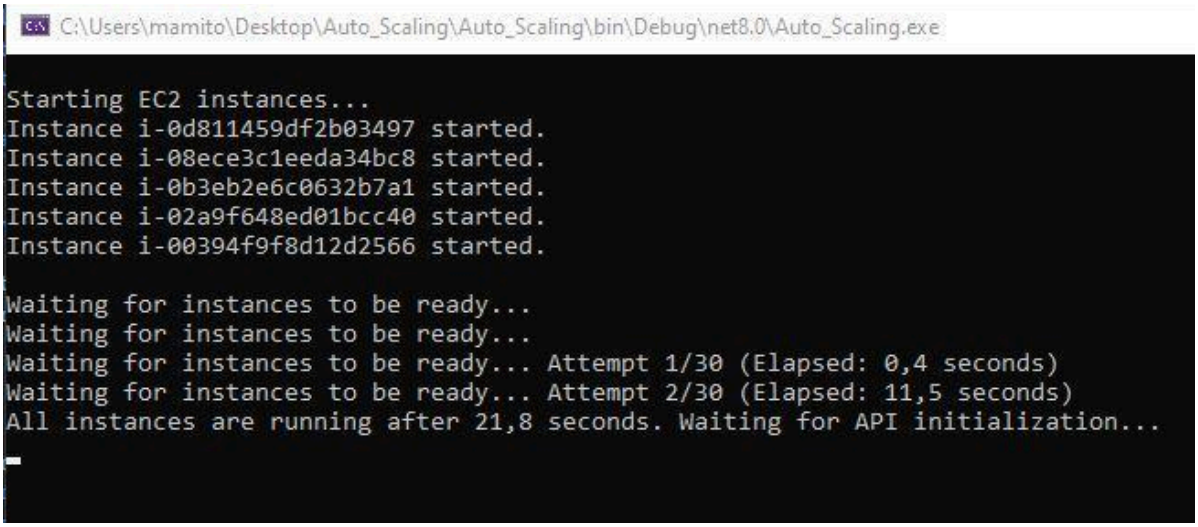
```
finally
{
    totalStopwatch.Stop();
    timings["Total Execution"] =
totalStopwatch.ElapsedMilliseconds;

    // Zamanlamaları göster
    Console.WriteLine("\n=== Execution Times ===");
    Console.WriteLine("----------------------");
    foreach (var timing in timings.OrderBy(t => t.Key))
    {
        Console.WriteLine($"{timing.Key,-25}:
{timing.Value,8:N0} ms
({TimeSpan.FromMilliseconds(timing.Value).TotalSeconds,6:N2}
seconds)");
    }
    Console.WriteLine("----------------------");

    Console.WriteLine("\nPress any key to exit...");
    Console.ReadKey();
}
```

Açıklama:

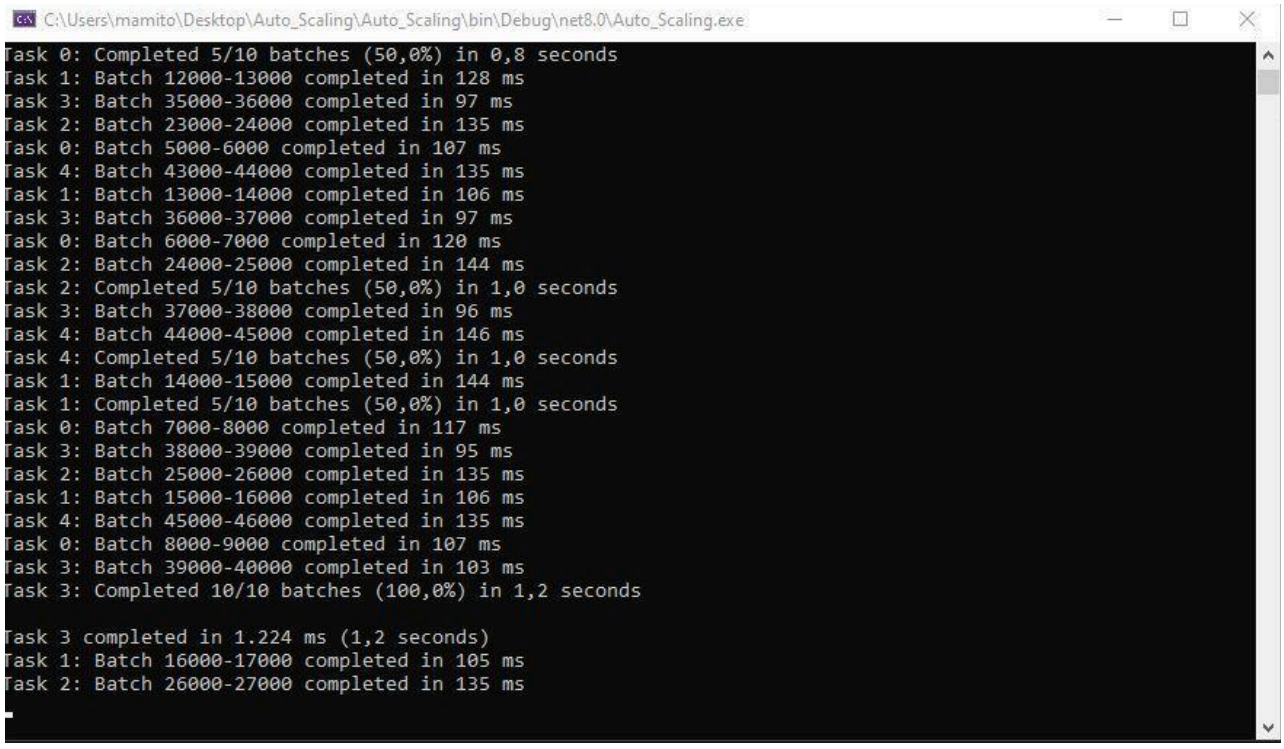
- **Toplam Süre:** Tüm işlemlerin toplam süresi ölçülür.
- **Detaylı Zamanlama:** Her bir adımın ne kadar sürdüğü konsola yazdırılır.
- **Çıkış:** Kullanıcı herhangi bir tuşa bastığında program sonlanır.

9.Sonuç

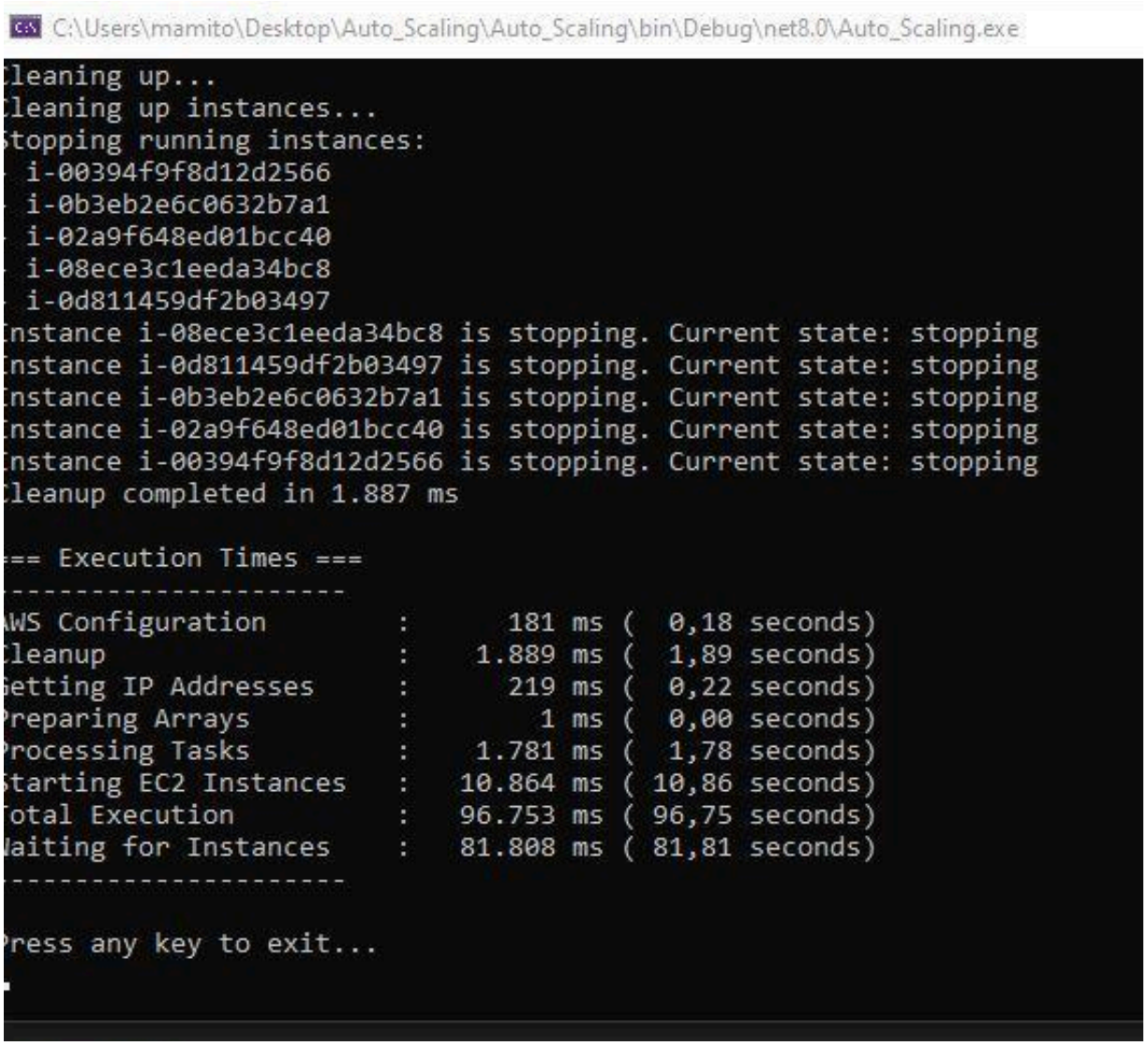
9.1 Açılış



9.2 Worker işlemleri



9.3 Sonuç


C:\Users\mamito\Desktop\Auto_Scaling\Auto_Scaling\bin\Debug\net8.0\Auto_Scaling.exe
leaning up...
leaning up instances...
topping running instances:
i-00394f9f8d12d2566
i-0b3eb2e6c0632b7a1
i-02a9f648ed01bcc40
i-08ece3c1eeda34bc8
i-0d811459df2b03497
nstance i-08ece3c1eeda34bc8 is stopping. Current state: stopping
nstance i-0d811459df2b03497 is stopping. Current state: stopping
nstance i-0b3eb2e6c0632b7a1 is stopping. Current state: stopping
nstance i-02a9f648ed01bcc40 is stopping. Current state: stopping
nstance i-00394f9f8d12d2566 is stopping. Current state: stopping
leanup completed in 1.887 ms
== Execution Times ===
WS Configuration : 181 ms (0,18 seconds)
leanup : 1.889 ms (1,89 seconds)
etting IP Addresses : 219 ms (0,22 seconds)
reparing Arrays : 1 ms (0,00 seconds)
rocessing Tasks : 1.781 ms (1,78 seconds)
tarting EC2 Instances : 10.864 ms (10,86 seconds)
otal Execution : 96.753 ms (96,75 seconds)
aiting for Instances : 81.808 ms (81,81 seconds)
ress any key to exit...