CS 551: Assignment 3

1) In the lecture, we discussed the benefits of virtual machines. What is a missing element that managed container orchestration services use, that virtual machines do not have out of the box?

   In the lecture, we learned about virtual machines. However, some benefits managed container orchestration services use that VMs don't have out of the box. The biggest elements are:

   Rapid Creation: Virtual machines require more overhead and take longer to start up than lightweight containers
   Short Lifetime: VMs often have a lengthy lifespan and stay in one location for extended periods, whereas containers are frequently transitory
   Replication: While expanding virtual machines requires additional resources, scaling containers makes it simple to replicate in response to demand.

   Apart from these 3 main things, there are some other added benefits of using container orchestration services:

   Auto-scaling: While virtual machines usually require additional tools or user intervention for scaling, orchestration systems can automatically grow the number of containers based on traffic loads or other indicators.
   Load Balancing: While virtual machines may need you to set up and maintain your load balancer, managed container orchestration platforms provide built-in ways to distribute traffic among instances of an application.

2) Chroot is a Linux command to isolate users to certain directories. How is this similar to what a Docker container does? Why would we not be able to use chroot instead of docker images to isolate programs?

   Chroot is a Linux command used to isolate users to certain directories. Docker also provides isolation like chroot, which modifies a process's root directory to allow it to access a restricted subset of the filesystem. We can restrict a process to a certain section of the filesystem using Docker's usage of the **mount** namespace. Hence it is similar in what it does when compared to chroot.

We can't use chroot instead of docker images to isolate programs since docker uses Linux namespaces to offer isolation for many components, including users (USER namespace), processes (PID namespace), networks (NET namespace), and the filesystem. On the other hand, chroot merely modifies the filesystem view and does not offer the same level of complete isolation. Also, all dependencies—binaries, libraries, and configurations—are encapsulated in Docker containers. As a result, Docker images are more adaptable when compared to chroot. Lastly, a process can be on a different virtual network than the host and have a distinct Internet address thanks to the network namespace which is used by Docker. Chroot doesn't provide inherent network isolation like this. All these reasons are why we can't substitute docker containers with chroot.

3) In assignment 1, you thought about load balancing and code deployments from using a simple EC2 instance. After completing this assignment, answer questions 4 and 5 from assignment 1 question 4 again with what you've learned so far about ECS.

   Question 4: In Assignment 1, we used tactics like load balancing, auto-scaling, blue-green deployments, and rolling updates for EC2 instances. In terms of ECS, when we want to scale up, we will use similar strategies but with a focus on containers, so we tag an image of a container and push it like we did in this assignment. We use strategies like:
   i) Use ALBs with ECS: Application load balancers (ALB) and ECS services work together to guarantee that traffic is sent only to healthy container instances. As containers are updated, the ALB will gradually shift traffic away from outdated versions and toward updated ones.
   ii) Auto-Scaling with ECS: Similar to EC2, ECS also supports auto-scaling at the service level. This means that as the load increases, additional tasks or containers can be automatically launched to handle the demand due to incoming traffic or resource usage.
   iii) ECS with AWS CodeDeploy using Blue/Green Deployment: The previous version (blue) and the new version (green) are displayed side by side during this kind of deployment. The traffic is progressively switched to the new version following testing and approval. Traffic may be switched back to the original version in case of problems, reducing risk and disruption.
   iv) Rolling updates is one of the several deployment methods that ECS provides. With this kind of update, a portion of the service is always available since the old version of the container will be gradually replaced with the new one.

These 4 techniques are way more efficient in ECS compared to EC2 since:

Containers can start much faster than virtual machines. When auto-scaling triggers, these container instances become operational way faster compared to booting up an entire EC2 instance since containers are lightweight and share the same OS kernel.

Now if the deployment fails in ECS, the rollback strategies would include the following:

i) **Task definition versions:**
Versions of task definitions are maintained by ECS. You can immediately roll back modifications if a new deployment fails by going back to an earlier version of the task specification.

ii) **AWS CodeDeploy Alarms:**
It is possible to establish CloudWatch Alarms when deploying ECS applications with AWS CodeDeploy. A deployment can be automatically turned back if something goes wrong. This ensures that healthy containers are active, and admins are notified about unhealthy ones.

iii) **Service Auto Scaling for Rollbacks:**
If scaling controls are in place and a deployment produces problems, ECS may automatically switch from the number of jobs executing the unstable, newer version to the more stable, older version.

Question 5: For scaling one to many containers in ECS, we can use:

i) **ECS Service Auto Scaling with Load Balancing:** Used in conjunction with an Application Load Balancer to effectively divide incoming traffic among many container activities.

ii) **Fargate:** Scaling becomes easier and more responsive when you use AWS Fargate to deploy containers without having to worry about maintaining the underlying infrastructure. Instead, you can concentrate just on the quantity of jobs or containers needed.

Trade-Offs:

i) **Cost Repercussions:**
Although Fargate and ECS in particular make scaling simpler, they can also result in different cost structures. A thorough grasp of the pricing model and effective monitoring are necessary to prevent unforeseen costs.

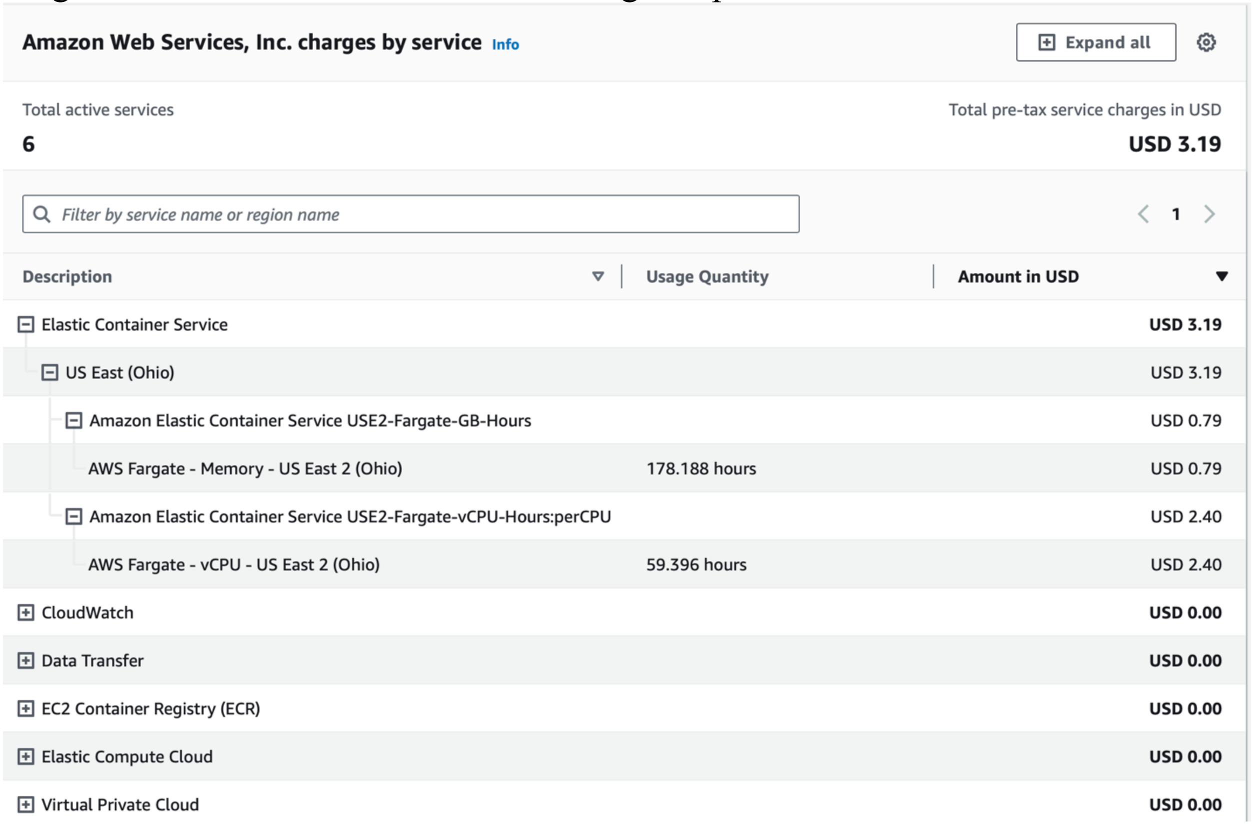
ii) **Added complexity:** Although containers offer deployment simplicity and versatility, they can complicate orchestration, service-to-service communication, and general administration, particularly when expanding out quickly. Effective handling of this requires the use of the right instruments and tactics.

4) Note how much your cluster costs and compare that to the cost of deploying a single instance in assignment 1.

So, for this section I ran:

```
$ docker buildx build --platform=linux/amd64 -t <image-name> .
```

Since I had a M1 chip Macbook, I had to build in x86 architecture. Unfortunately, this wasn't working with the free tier of t2.micro that we get. So, I used Fargate instead. Fargate is a pay-as-you-go service, that charges $0.04048 per vCPU per hour. I could compile, built and run my docker file on the cloud now. As compared to the free tier of EC2 which supported t2.micro architecture, I paid a little for a cluster since I used Fargate. Attached is a screenshot of billing and price breakdown:



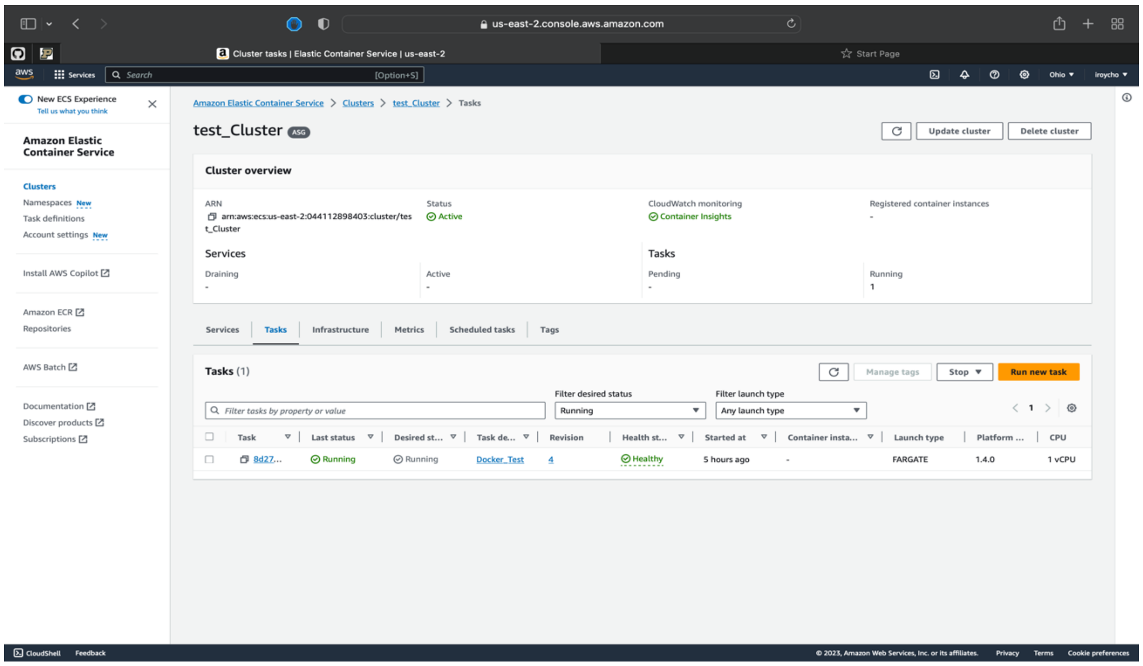
Amazon Web Services, Inc. charges by service Info

Total active services: 6

Total pre-tax service charges in USD: **USD 3.19**

Description	Usage Quantity	Amount in USD
Elastic Container Service		**USD 3.19**
US East (Ohio)		USD 3.19
Amazon Elastic Container Service USE2-Fargate-GB-Hours		USD 0.79
AWS Fargate - Memory - US East 2 (Ohio)	178.188 hours	USD 0.79
Amazon Elastic Container Service USE2-Fargate-vCPU-Hours:perCPU		USD 2.40
AWS Fargate - vCPU - US East 2 (Ohio)	59.396 hours	USD 2.40
CloudWatch		**USD 0.00**
Data Transfer		**USD 0.00**
EC2 Container Registry (ECR)		**USD 0.00**
Elastic Compute Cloud		**USD 0.00**
Virtual Private Cloud		**USD 0.00**

Screenshots of Cluster Deployed:



Amazon Elastic Container Service > Clusters > test_Cluster > Tasks

test_Cluster ASG

Cluster overview

ARN	Status	CloudWatch monitoring	Registered container instances
arn:aws:ecs:us-east-2:044112898403:cluster/test_Cluster	Active	Container Insights	-

Services

Draining	Active
-	-

Tasks

Pending	Running
-	1

Tasks (1)

Filter desired status: Running

Filter launch type: Any launch type

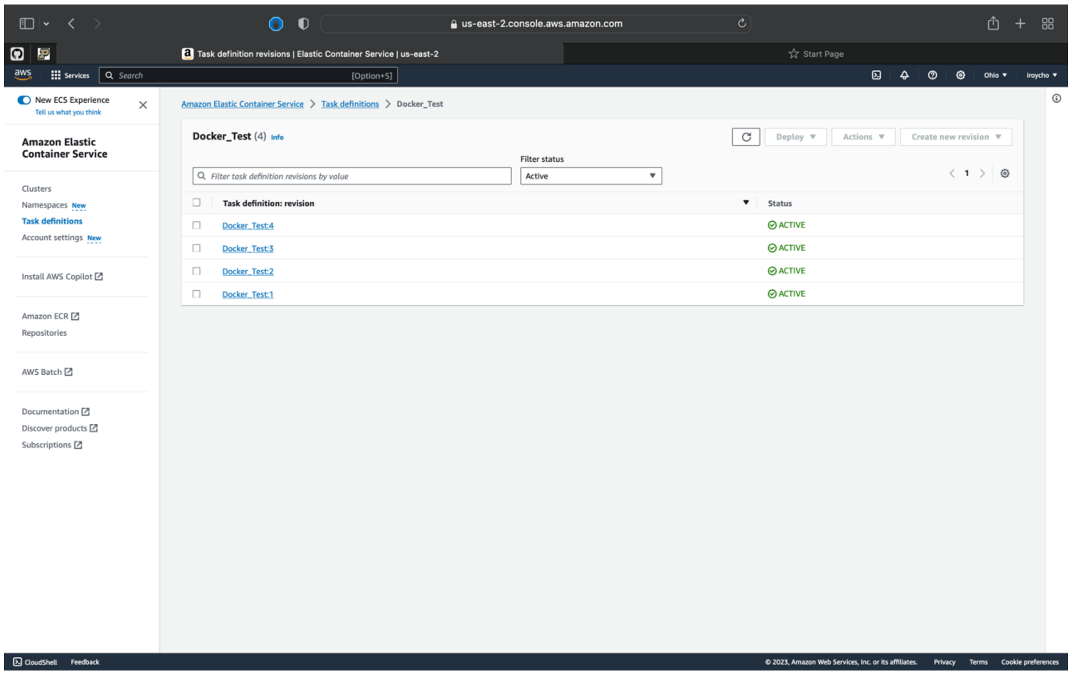
Task	Last status	Desired st...	Task de...	Revision	Health st...	Started at	Container insta...	Launch type	Platform ...	CPU
8d27...	Running	Running	Docker_Test	4	Healthy	5 hours ago	-	FARGATE	1.4.0	1 vCPU

© 2023, Amazon Web Services, Inc. or its affiliates.

Here I ran a health check with the command, to make sure the right container was tagged and pushed:

CMD-SHELL, curl -d "username=ProfComer&email=comer@cs.purdue.edu&address=West Lafayette, IN" -H "Content-Type: application/x-www-form-urlencoded" -X POST || exit 1

Screenshots of Task Definition:



Amazon Elastic Container Service > Task definitions > Docker_Test

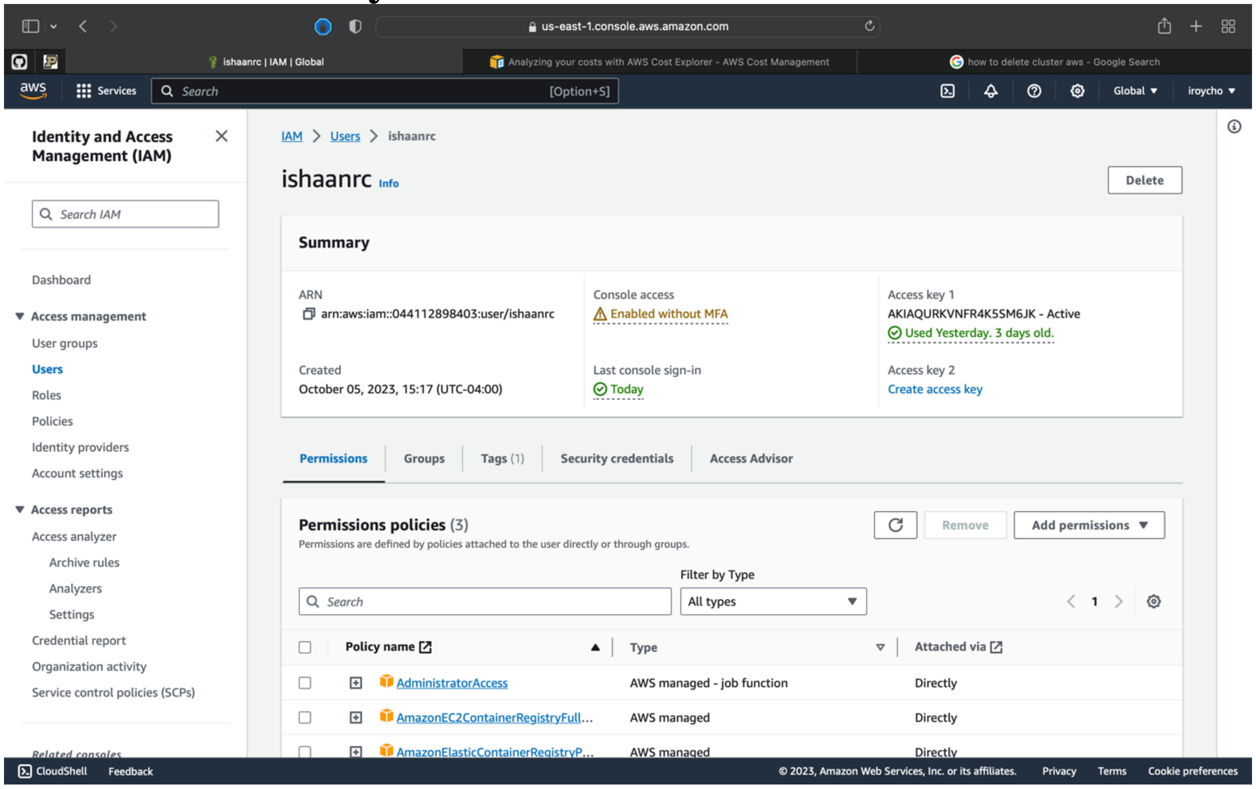
Docker_Test (4)

Filter status: Active

Task definition: revision	Status
Docker_Test:4	ACTIVE
Docker_Test:3	ACTIVE
Docker_Test:2	ACTIVE
Docker_Test:1	ACTIVE

© 2023, Amazon Web Services, Inc. or its affiliates.

Screenshot of my IAM role:



ishaanrc Info

Summary

ARN	Console access	Access key 1
arn:aws:iam::044112898403:user/ishaanrc	Enabled without MFA	AKIAQURKVNFR4K5SM6JK - Active Used Yesterday. 3 days old.
Created	**Last console sign-in**	**Access key 2**
October 05, 2023, 15:17 (UTC-04:00)	Today	Create access key

Permissions | Groups | Tags (1) | Security credentials | Access Advisor

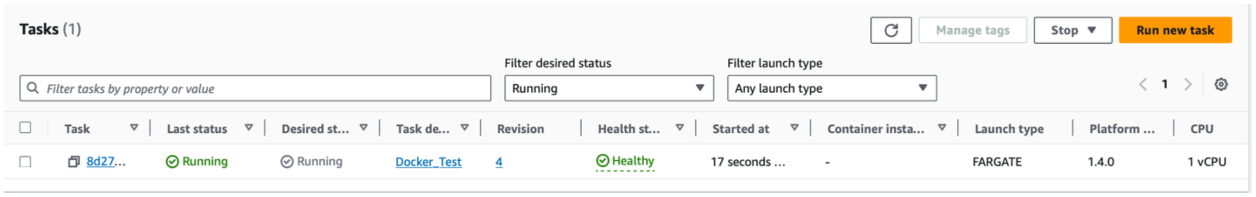
Permissions policies (3)

Permissions are defined by policies attached to the user directly or through groups.

Filter by Type: All types

Policy name	Type	Attached via
AdministratorAccess	AWS managed - job function	Directly
AmazonEC2ContainerRegistryFull...	AWS managed	Directly
AmazonElasticContainerRegistryP...	AWS managed	Directly

Screenshot of my cluster working:

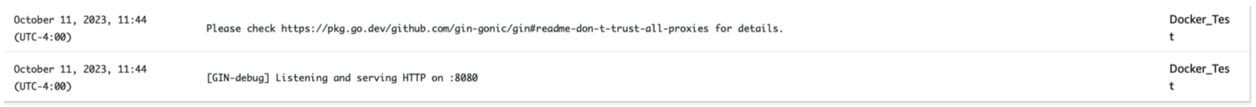


Tasks (1)

Filter desired status: Running
Filter launch type: Any launch type

Task	Last status	Desired st...	Task de...	Revision	Health st...	Started at	Container insta...	Launch type	Platform ...	CPU
8d27...	Running	Running	Docker_Test	4	Healthy	17 seconds ...	-	FARGATE	1.4.0	1 vCPU

Screenshot of my logs:

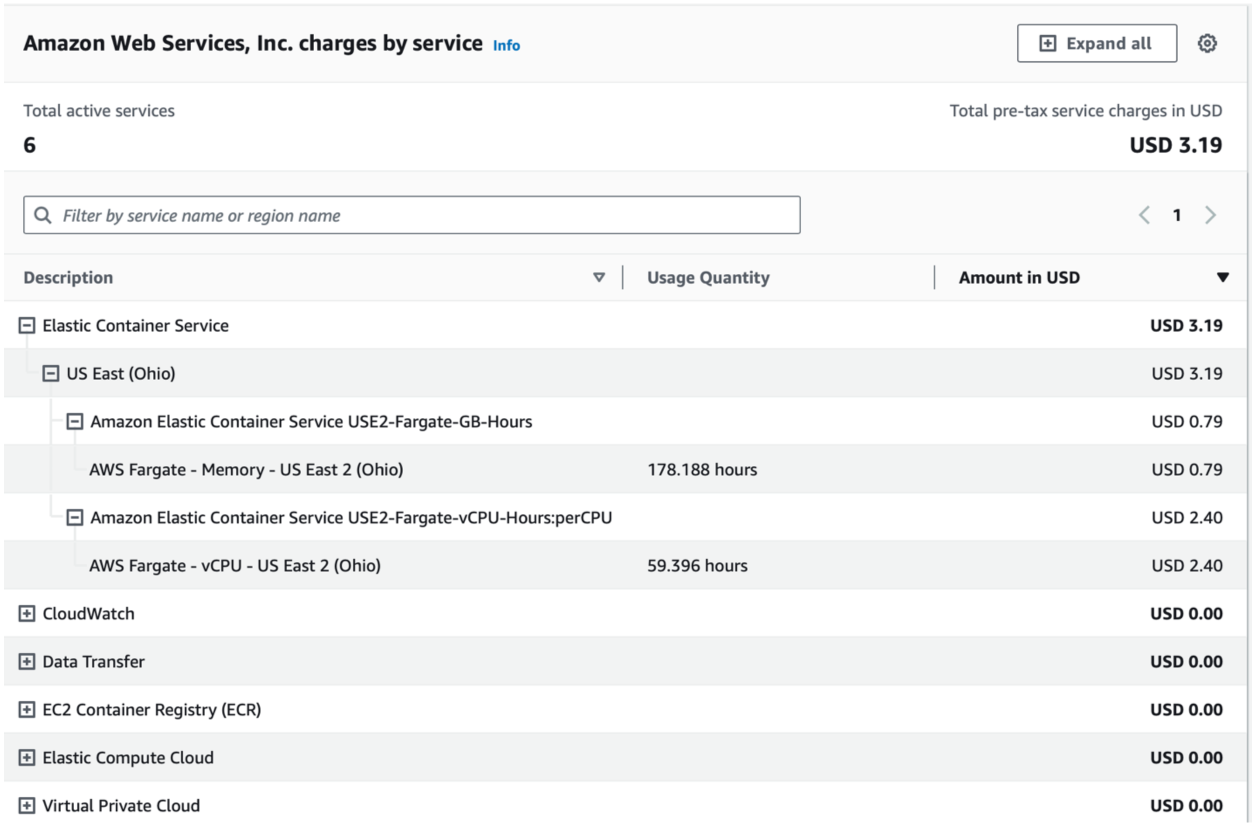


```
October 11, 2023, 11:44   Please check https://pkg.go.dev/github.com/gin-gonic/gin#readme-don-t-trust-all-proxies for details.   Docker_Tes
(UTC-4:00)                                                                                                                      t
October 11, 2023, 11:44   [GIN-debug] Listening and serving HTTP on :8080                                                       Docker_Tes
(UTC-4:00)                                                                                                                      t
```

As mentioned above the cost of EC2 has 750 hours of free tier usage and 5GB of AWS S3 storage. My cost was a little higher than free since I used Fargate because

I had a M1 mac and the free tier wasn't compatible. Fargate is a pay-as-you-go service, that charges $0.04048 per vCPU per hour. I could compile, built and run my docker file on the cloud now. As compared to the free tier of EC2 which supported t2.micro architecture, I paid a little for a cluster since I used Fargate.

Screenshot of Costs:



Amazon Web Services, Inc. charges by service Info

Total active services
6

Total pre-tax service charges in USD
USD 3.19

Description	Usage Quantity	Amount in USD
Elastic Container Service		**USD 3.19**
US East (Ohio)		USD 3.19
Amazon Elastic Container Service USE2-Fargate-GB-Hours		USD 0.79
AWS Fargate - Memory - US East 2 (Ohio)	178.188 hours	USD 0.79
Amazon Elastic Container Service USE2-Fargate-vCPU-Hours:perCPU		USD 2.40
AWS Fargate - vCPU - US East 2 (Ohio)	59.396 hours	USD 2.40
CloudWatch		**USD 0.00**
Data Transfer		**USD 0.00**
EC2 Container Registry (ECR)		**USD 0.00**
Elastic Compute Cloud		**USD 0.00**
Virtual Private Cloud		**USD 0.00**