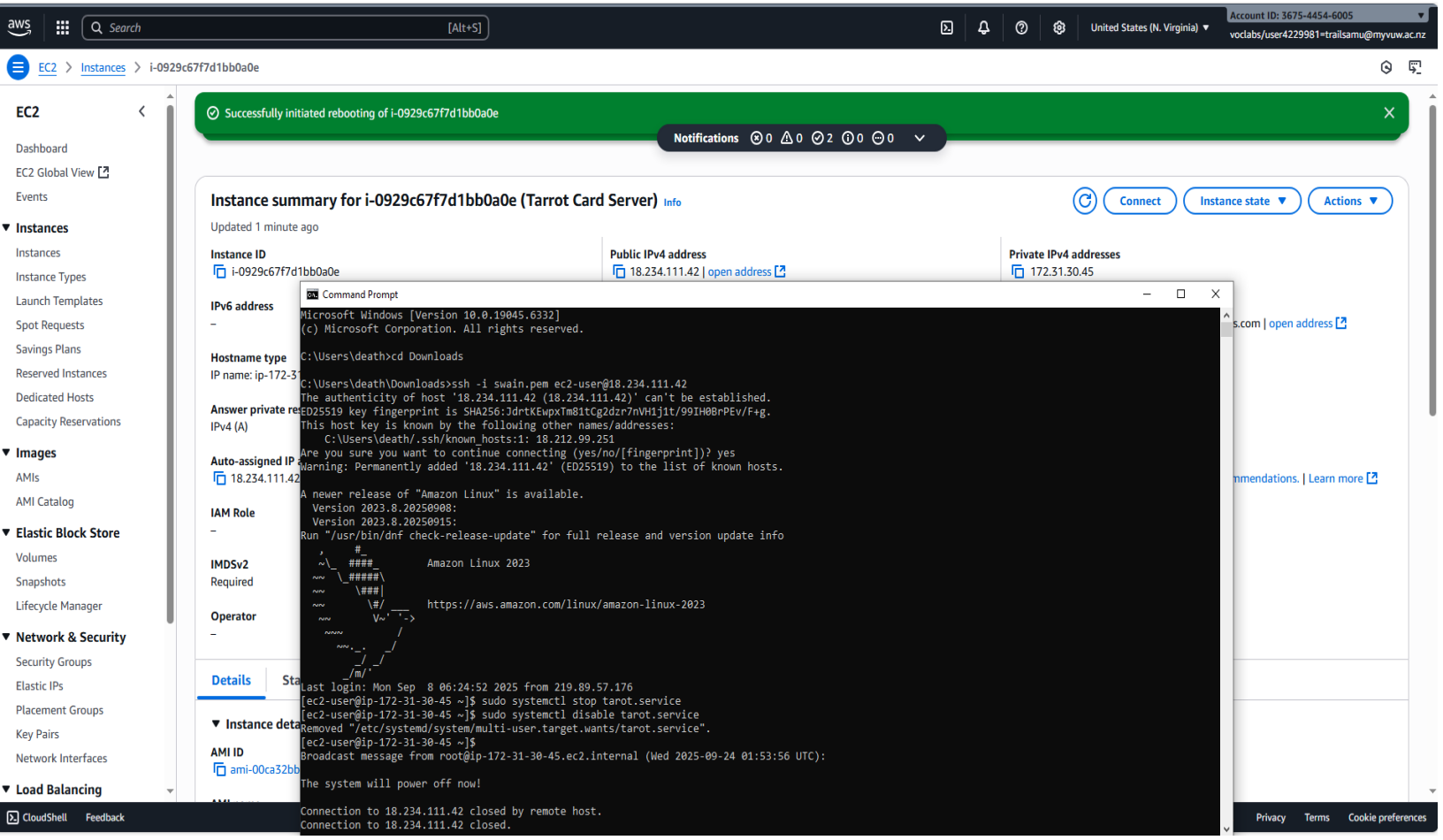
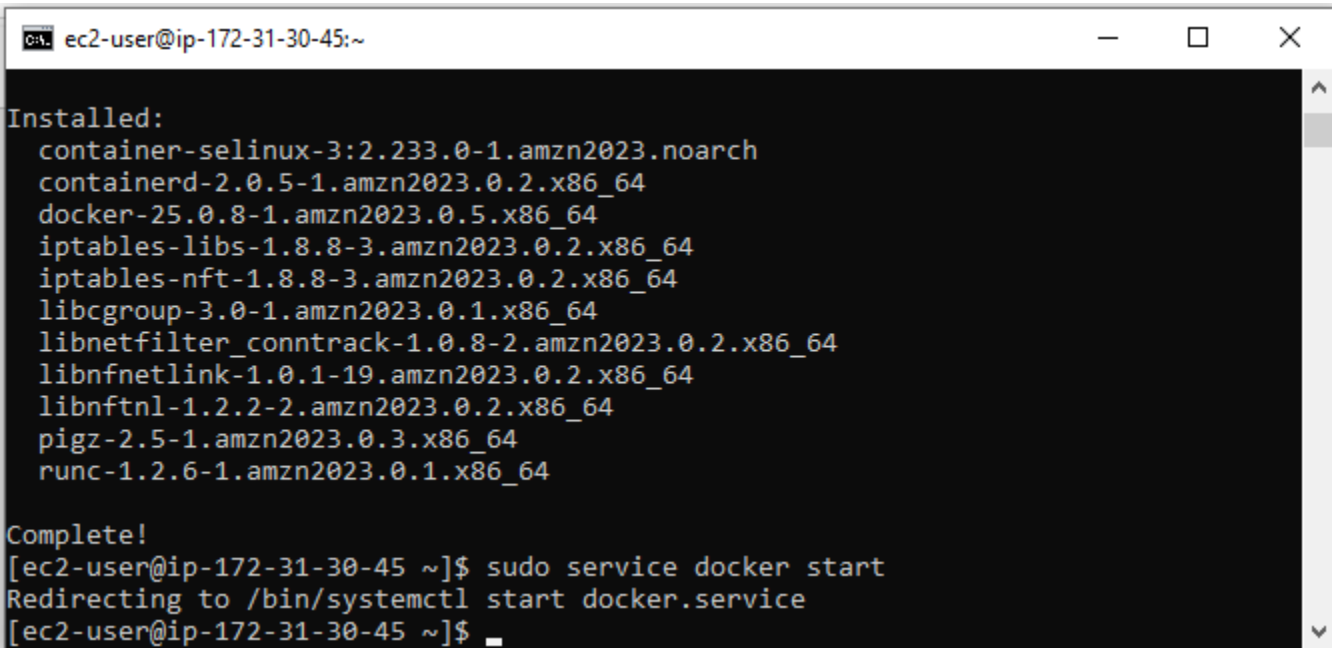
Docker installation and setup



AWS EC2 instance dashboard showing the Tarot Card Server instance after stopping existing services to prepare for Docker installation. The green notification indicates successful instance rebooting to ensure a clean state before containerizing the TarotCard server application.



Installation of Docker. The terminal shows the successful installation of container runtime dependencies. Demonstrating that the EC2 instance is being prepared as a Docker development environment after cleaning up previous services.

```
[ec2-user@ip-172-31-30-45 ~]$ docker info
Client:
 Version:    25.0.8
 Context:    default
 Debug Mode: false
 Plugins:
  buildx: Docker Buildx (Docker Inc.)
    Version:  0.12.1
    Path:     /usr/libexec/docker/cli-plugins/docker-buildx

Server:
 Containers: 0
  Running: 0
  Paused: 0
  Stopped: 0
 Images: 0
 Server Version: 25.0.8
 Storage Driver: overlay2
  Backing Filesystem: xfs
  Supports d_type: true
  Using metacopy: false
  Native Overlay Diff: true
  userxattr: false
 Logging Driver: json-file
 Cgroup Driver: systemd
 Cgroup Version: 2
 Plugins:
  Volume: local
  Network: bridge host ipvlan macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-file local splunk syslog
 Swarm: inactive
 Runtimes: io.containerd.runc.v2 runc
 Default Runtime: runc
 Init Binary: docker-init
 containerd version: fb4c30d4ede3531652d86197bf3fc9515e5276d9
 runc version: 6c52b3fc541fb26fe8c374d5f58112a0a5dbda66
 init version: de40ad0
 Security Options:
  seccomp
   Profile: builtin
  cgroupns
 Kernel Version: 6.1.147-172.266.amzn2023.x86_64
 Operating System: Amazon Linux 2023.8.20250818
 OSType: linux
 Architecture: x86_64
 CPUs: 1
 Total Memory: 949.4MiB
 Name: ip-172-31-30-45.ec2.internal
 ID: f3dcc424-4083-4037-9660-1e63113aa382
 Docker Root Dir: /var/lib/docker
 Debug Mode: false
 Experimental: false
 Insecure Registries:
  127.0.0.0/8
 Live Restore Enabled: false

[ec2-user@ip-172-31-30-45 ~]$
```

Output of docker info command showing Docker daemon is running successfully. This confirms the Docker installation was completed correctly and the service is operational on the cleaned instance.

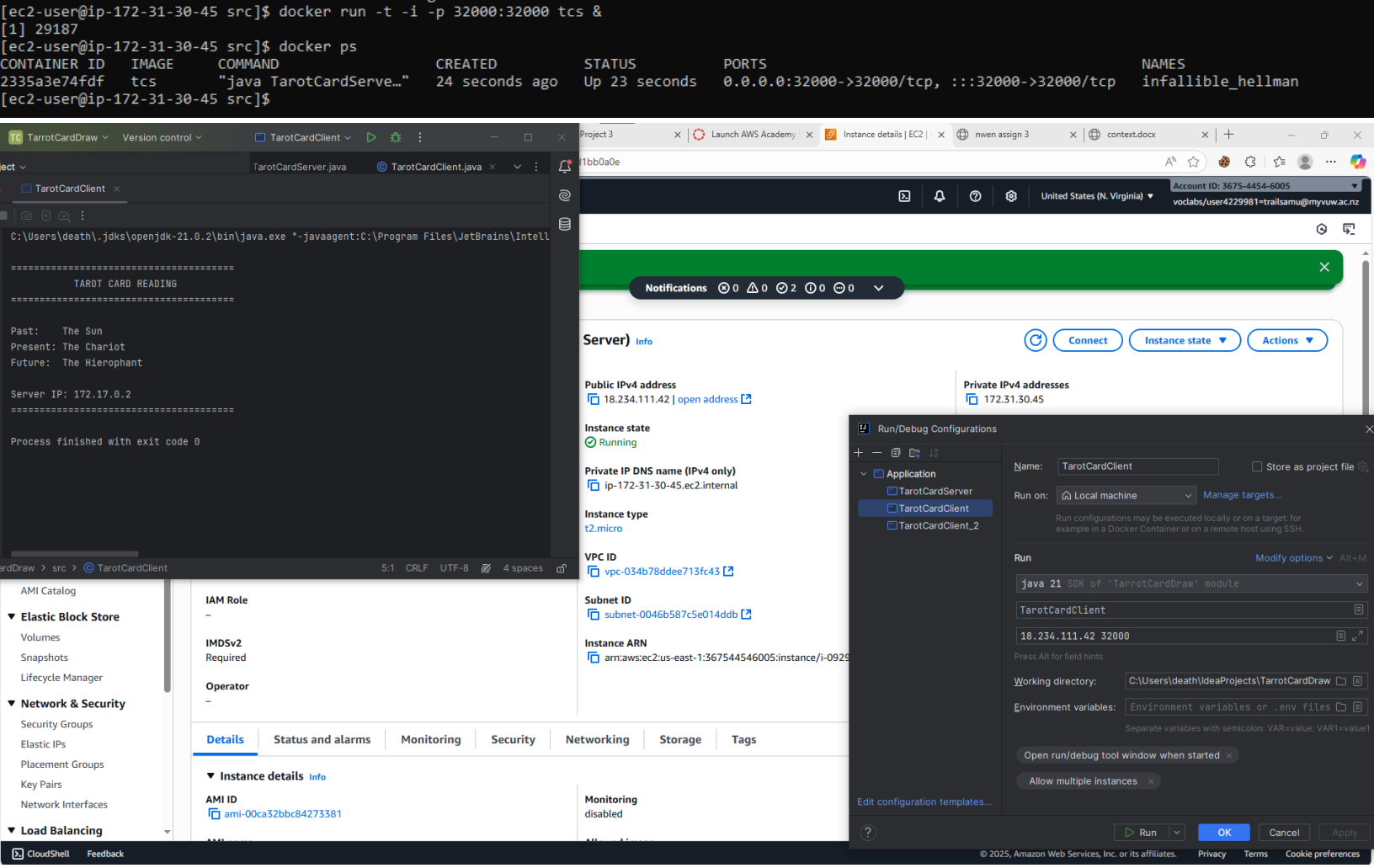
Creating docker image

```
[ec2-user@ip-172-31-30-45 ~]$ ls
TarrotCardDraw.iml  out  run.sh  src
[ec2-user@ip-172-31-30-45 ~]$ cd src
[ec2-user@ip-172-31-30-45 src]$ ls
ClientHandler.class    TarotCardClient.java   TarotCardServer.java
TarotCardClient.class  TarotCardServer.class
[ec2-user@ip-172-31-30-45 src]$ cat > Dockerfile << 'EOF'
> FROM openjdk:8
> COPY *.java /usr/src/TCS/
> WORKDIR /usr/src/TCS
> RUN javac TarotCardServer.java
> EXPOSE 32000
> CMD ["java", "TarotCardServer", "32000"]
> EOF
[ec2-user@ip-172-31-30-45 src]$ cat Dockerfile
FROM openjdk:8
COPY *.java /usr/src/TCS/
WORKDIR /usr/src/TCS
RUN javac TarotCardServer.java
EXPOSE 32000
CMD ["java", "TarotCardServer", "32000"]
[ec2-user@ip-172-31-30-45 src]$
```

Creation of the Dockerfile using the cat command. The Dockerfile contains instructions to build a container image from OpenJDK 8 base image, copy Java source files, compile the TarotCardServer.java, expose port 32000, and execute the server on startup.

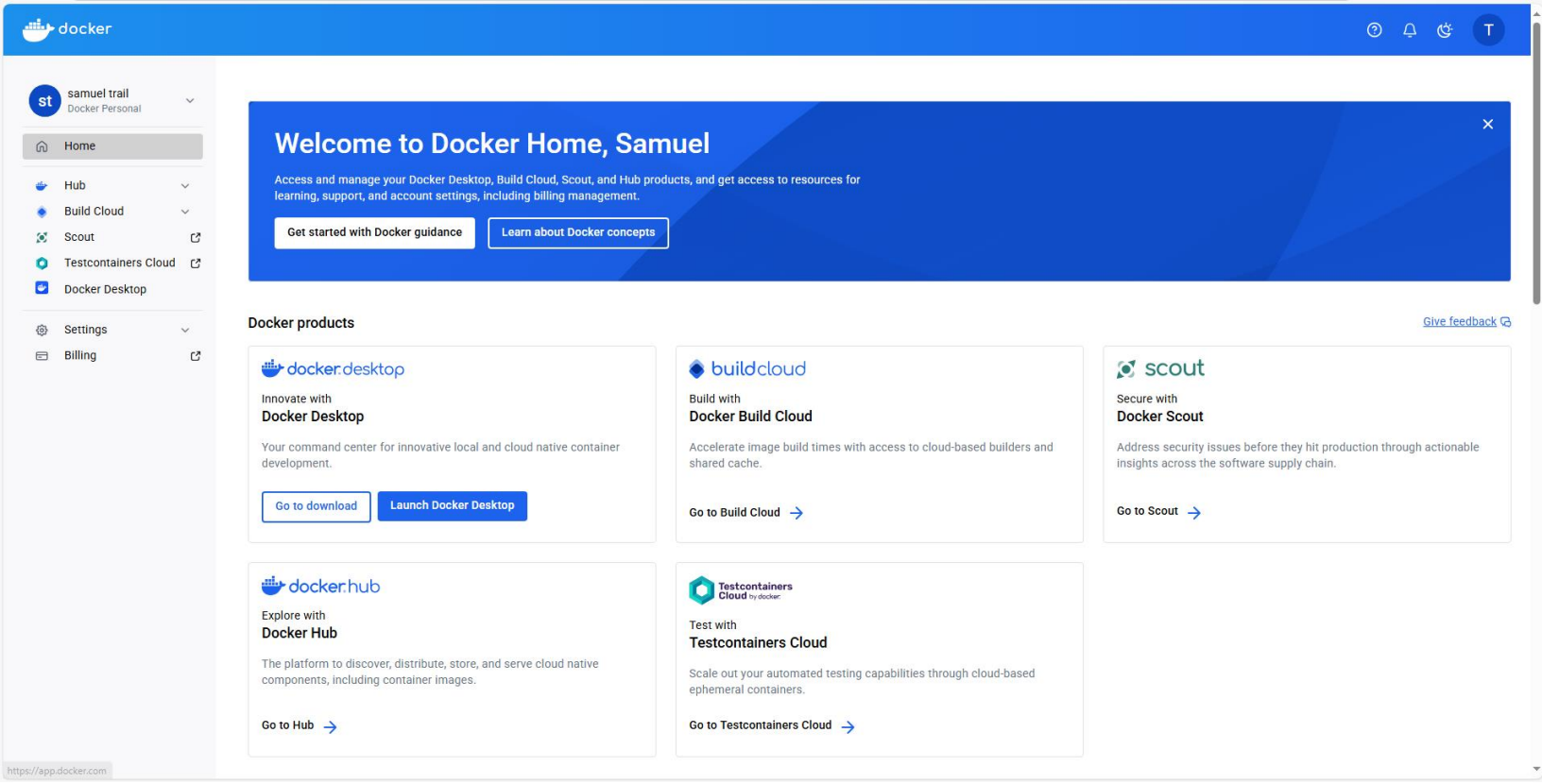
```
[ec2-user@ip-172-31-30-45 src]$ docker images --filter reference=tcs
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
tcs          latest    51fa9aa8b170   8 seconds ago   526MB
[ec2-user@ip-172-31-30-45 src]$
```

Successful execution of docker build -t tcs . command and docker images --filter reference=tcs showing the newly created container image.



Testing the containerized server locally using docker run -t -i -p 32000:32000 tcs & and verifying it's running with docker ps. The container shows proper port mapping (0.0.0.0:32000->32000/tcp) and successful client connection from local machine, demonstrating the containerization works correctly.

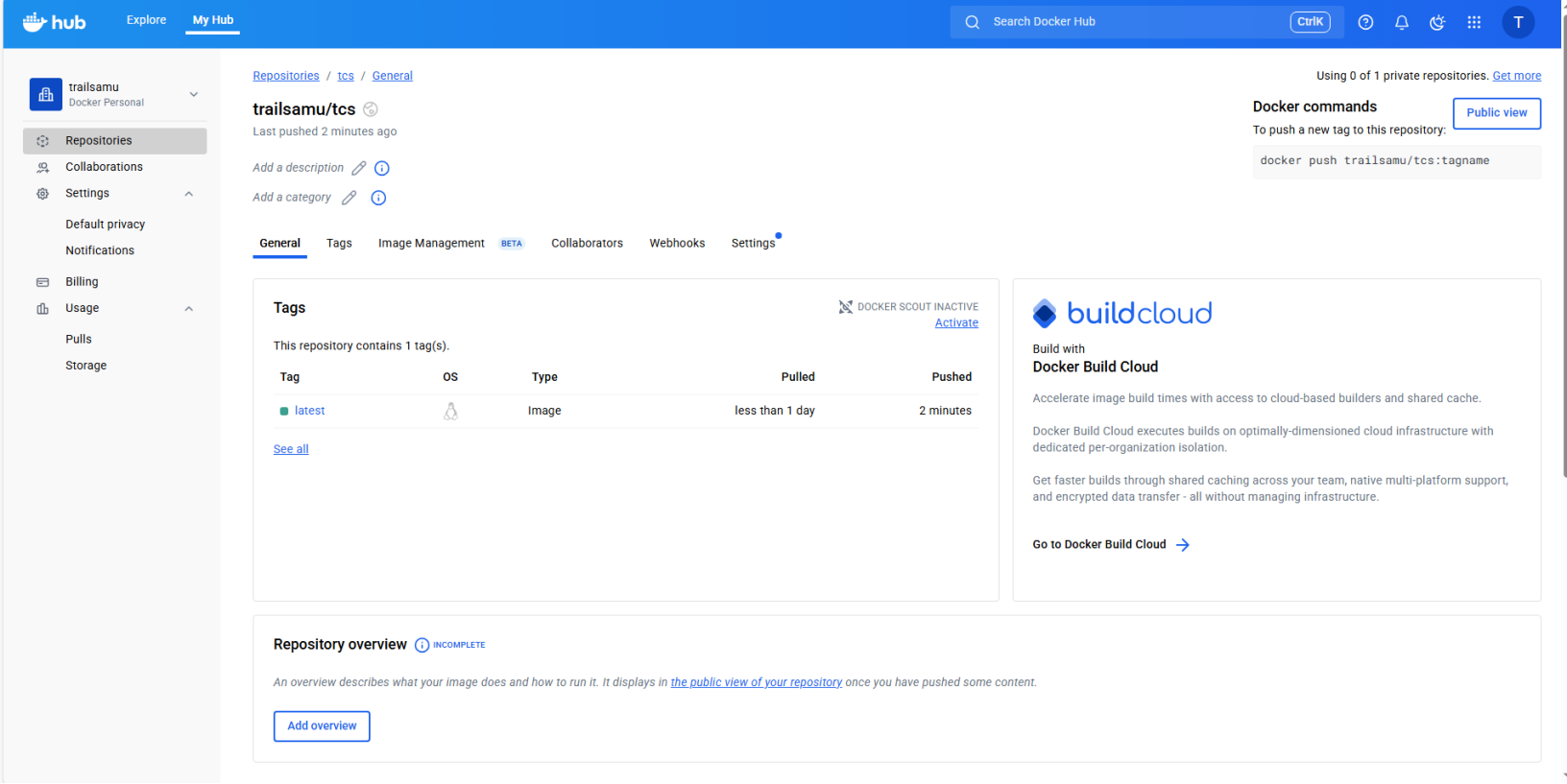
Docker hub repo



Docker Hub dashboard showing successful account creation and repo setup.

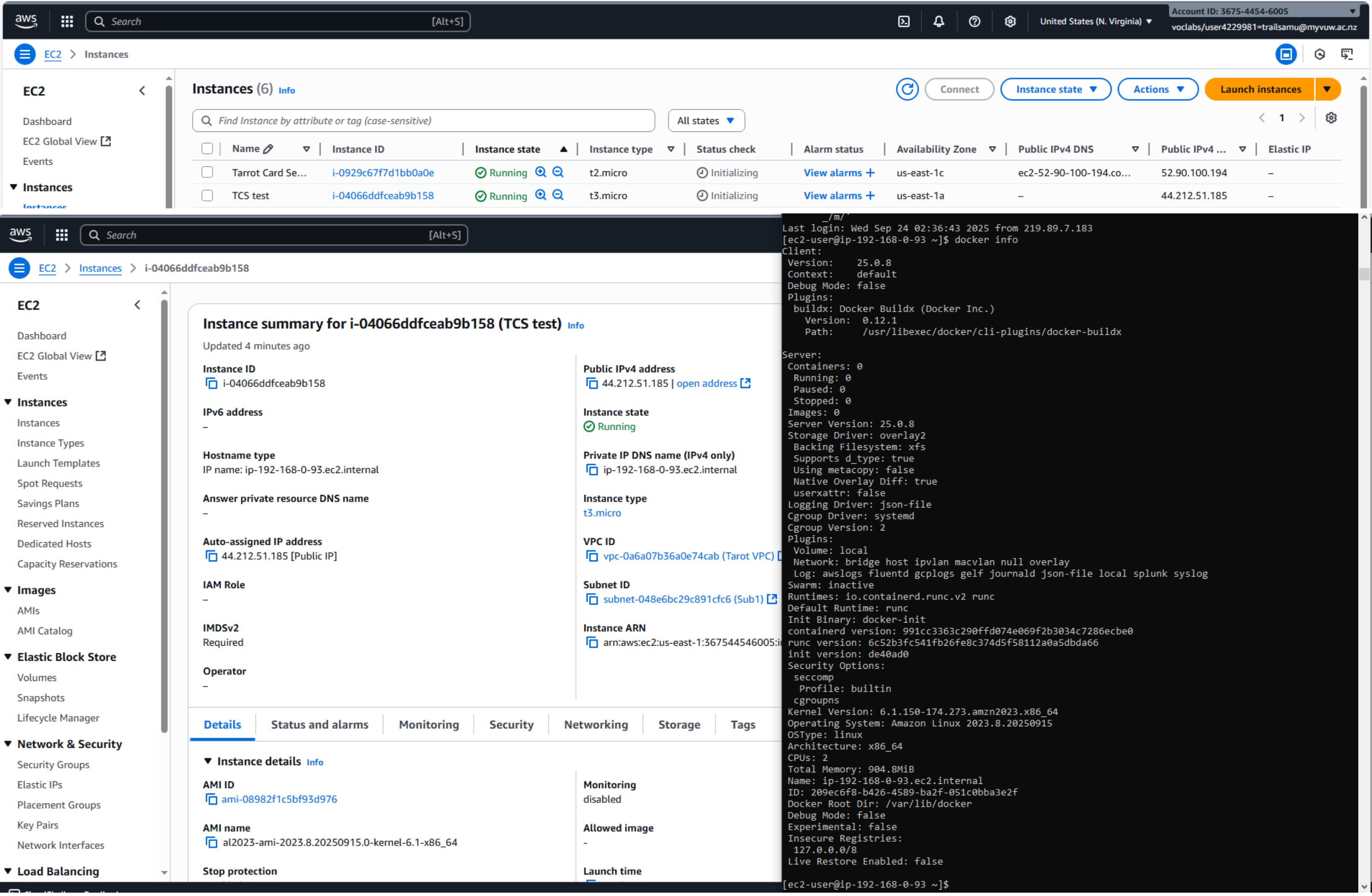
```
[ec2-user@ip-172-31-30-45 src]$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS
    PORTS                                         NAMES
2335a3e74fdf   tcs       "java TarotCardServe…"   12 minutes ago   Up 12 minute
s   0.0.0.0:32000->32000/tcp, :::32000->32000/tcp   infallible_hellman
[ec2-user@ip-172-31-30-45 src]$ docker tag tcs trailsamu/tcs
[ec2-user@ip-172-31-30-45 src]$ docker push trailsamu/tcs
Using default tag: latest
The push refers to repository [docker.io/trailsamu/tcs]
21b608edf65a: Pushed
5f70bf18a086: Pushed
8a6ebb8f03d8: Pushed
6b5aaff44254: Mounted from library/openjdk
53a0b163e995: Mounted from library/openjdk
b626401ef603: Mounted from library/openjdk
9b55156abf26: Mounted from library/openjdk
293d5db30c9f: Mounted from library/openjdk
03127cdb479b: Mounted from library/openjdk
9c742cd6c7a5: Mounted from library/openjdk
latest: digest: sha256:4a4d52a92b56971ba67de35460e1f8d5b6c95fda1cf83e562146ed11
9525b20c size: 2417
[ec2-user@ip-172-31-30-45 src]$
```

Pushing the container image to Docker Hub using docker tag and docker push commands. The push process shows multiple layers being uploaded, with the final confirmation that the image is now available in the trailsamu/tcs repo.

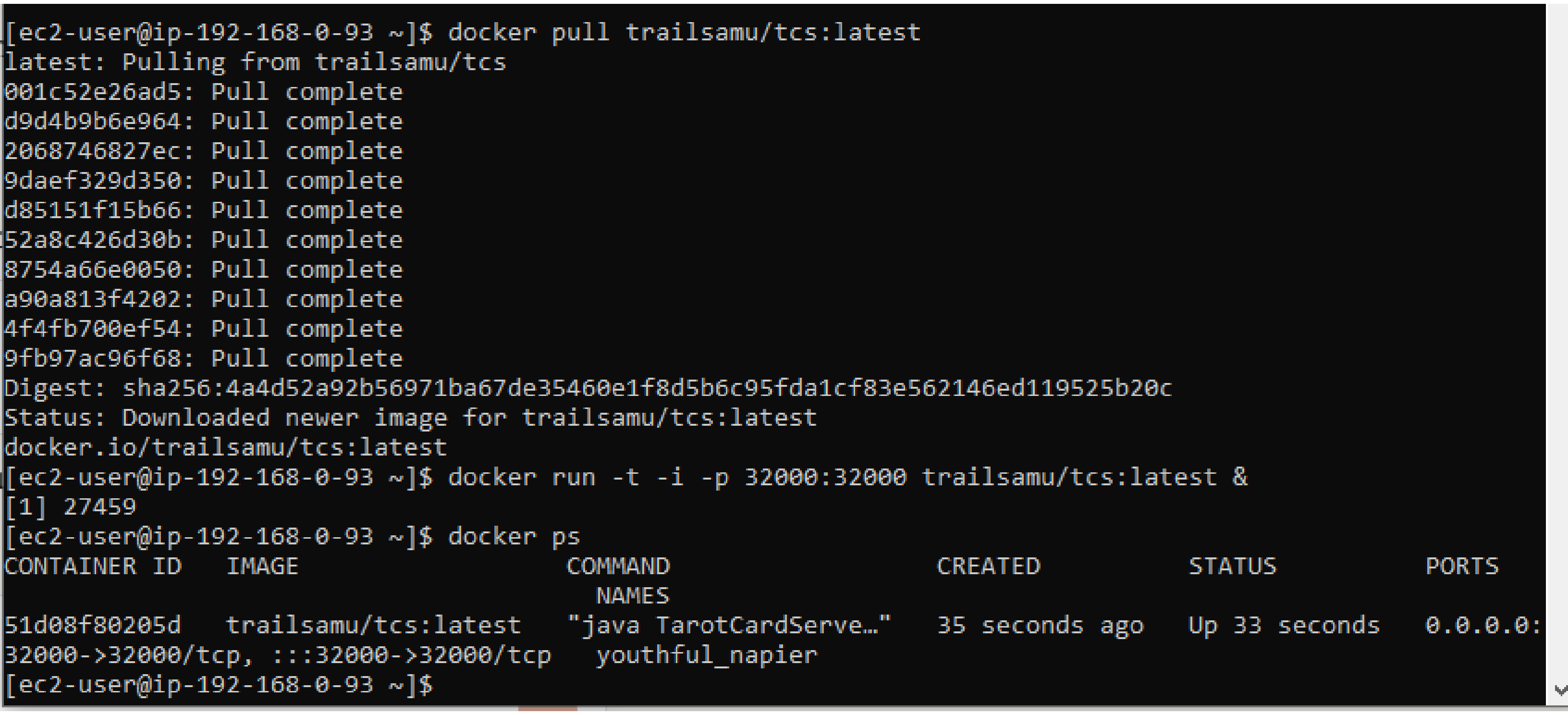


Docker Hub repo page showing the successfully uploaded TCS container image with "latest" tag, confirming the image is publicly accessible for deployment on other systems.

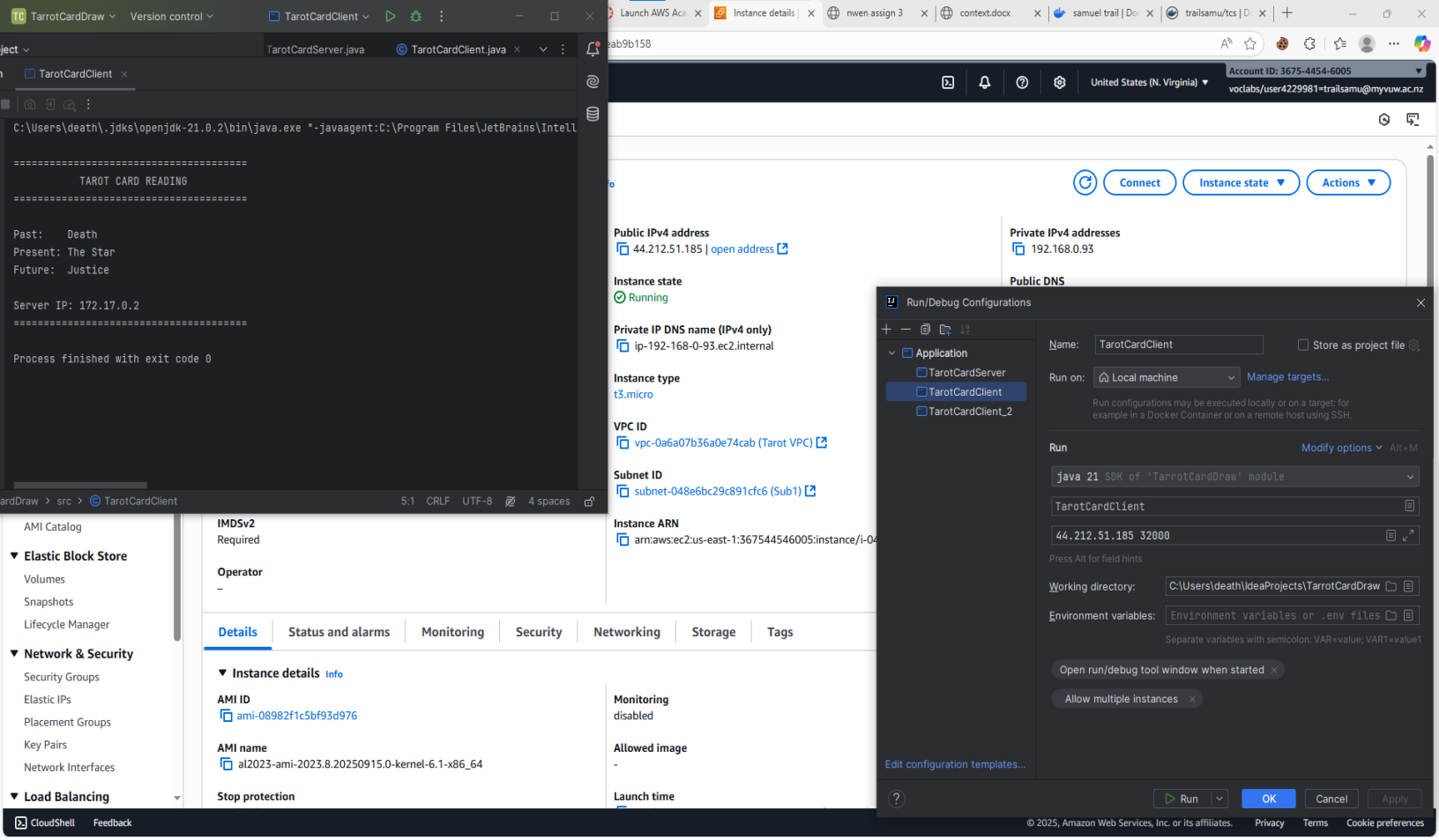
Instance testing



Creation of a new TCS test EC2 instance and installation of Docker on the clean system

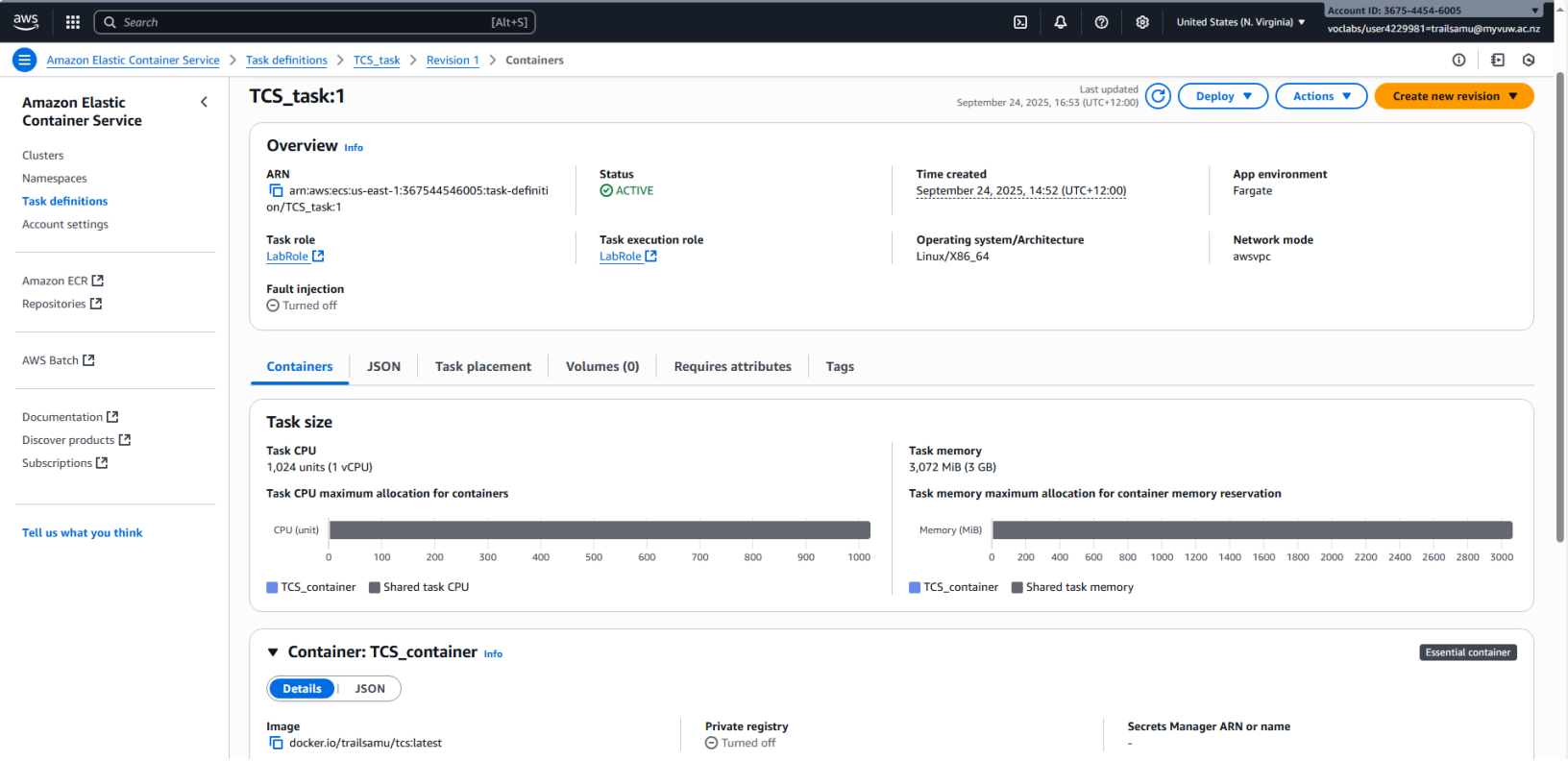


Pulling the container image from Docker Hub using docker pull trailsamu/tcs:latest and running it on the test instance. This proves the containerized application works.

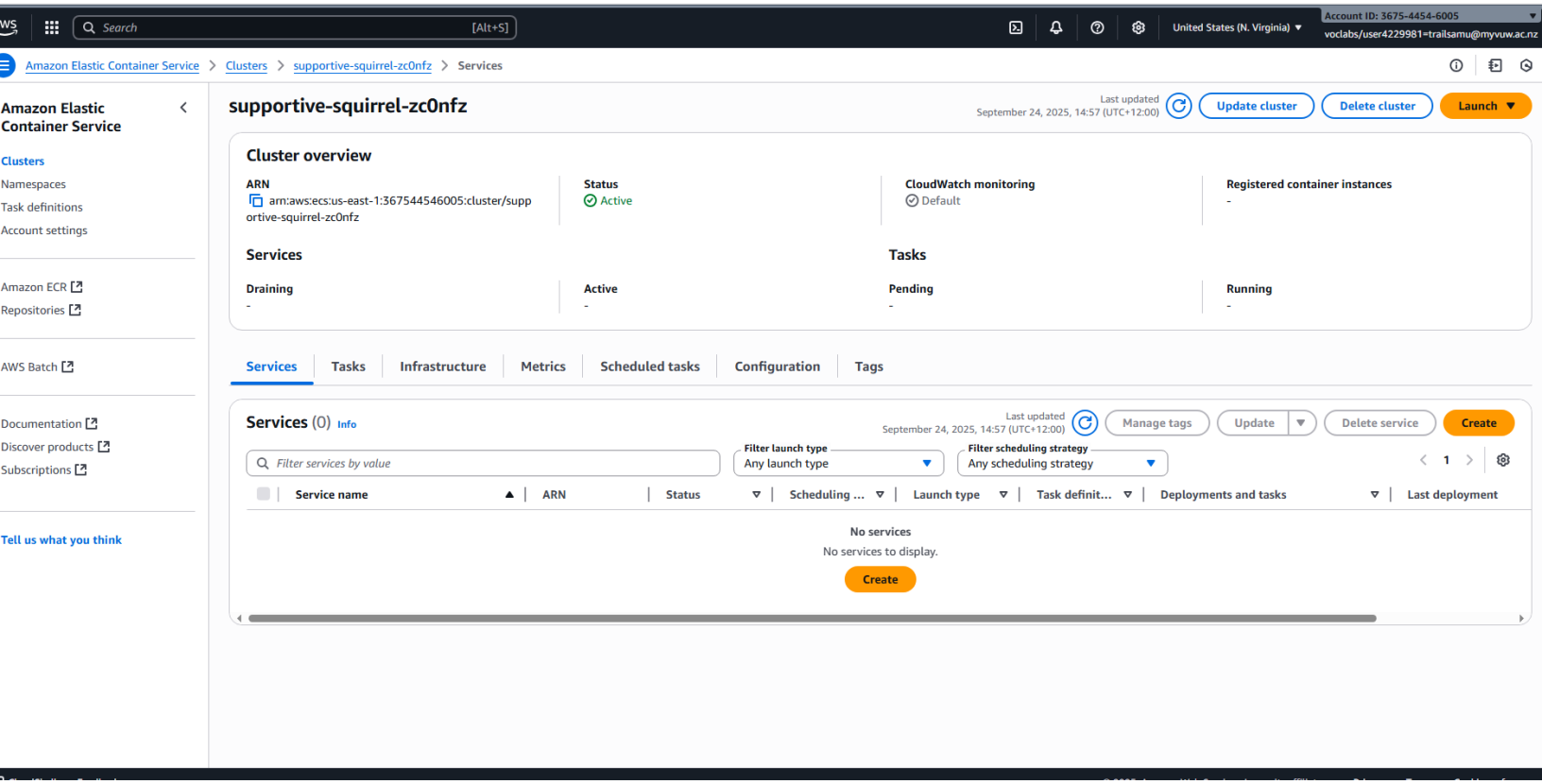


Client successfully connecting to the containerized server on the test instance, receiving a different tarot card reading

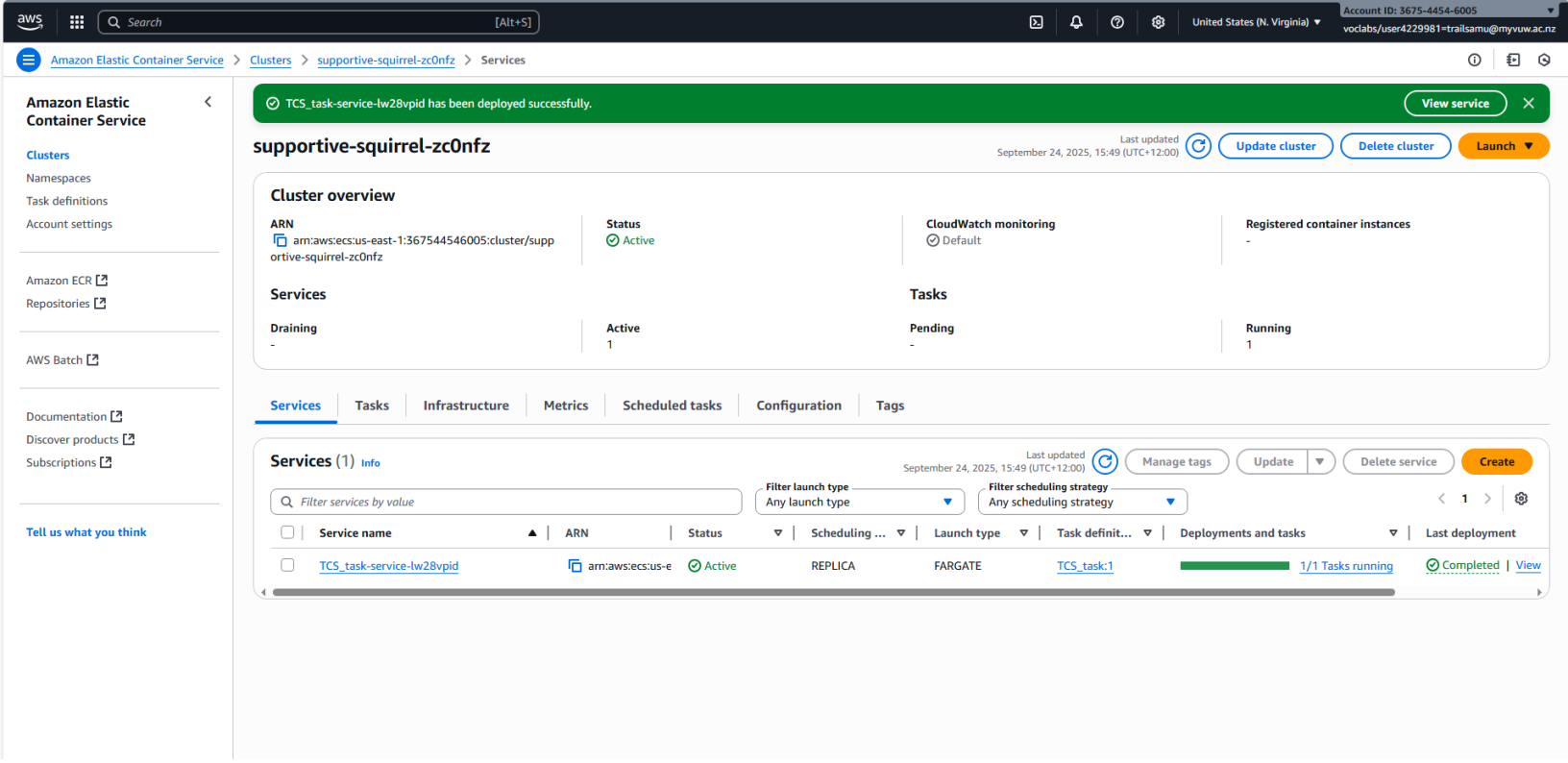
AWS/Fargate stuff



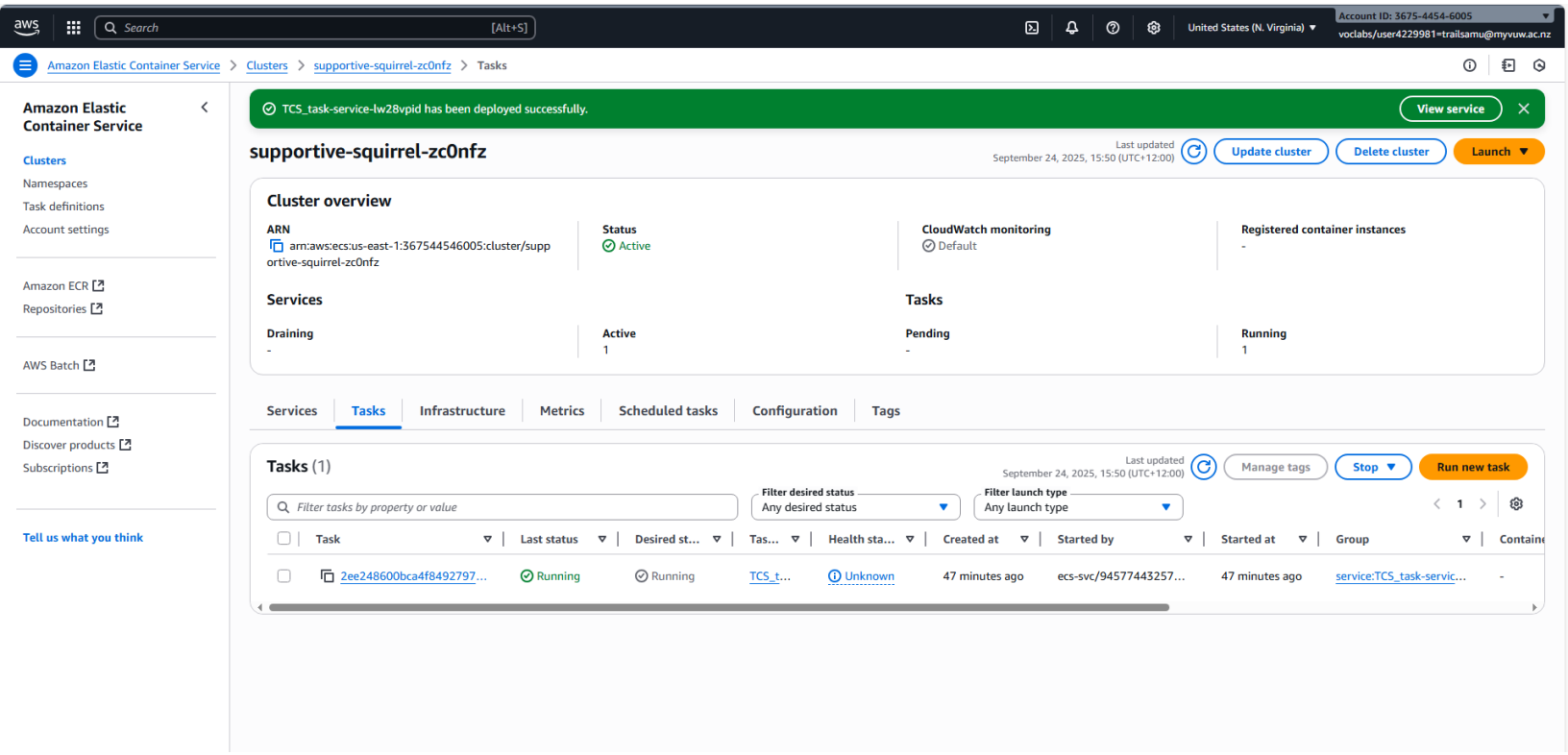
ECS Task Definition creation showing the configuration for containerized deployment using AWS Fargate. The task definition specifies the Docker image location (trailsamu/tcs:latest)



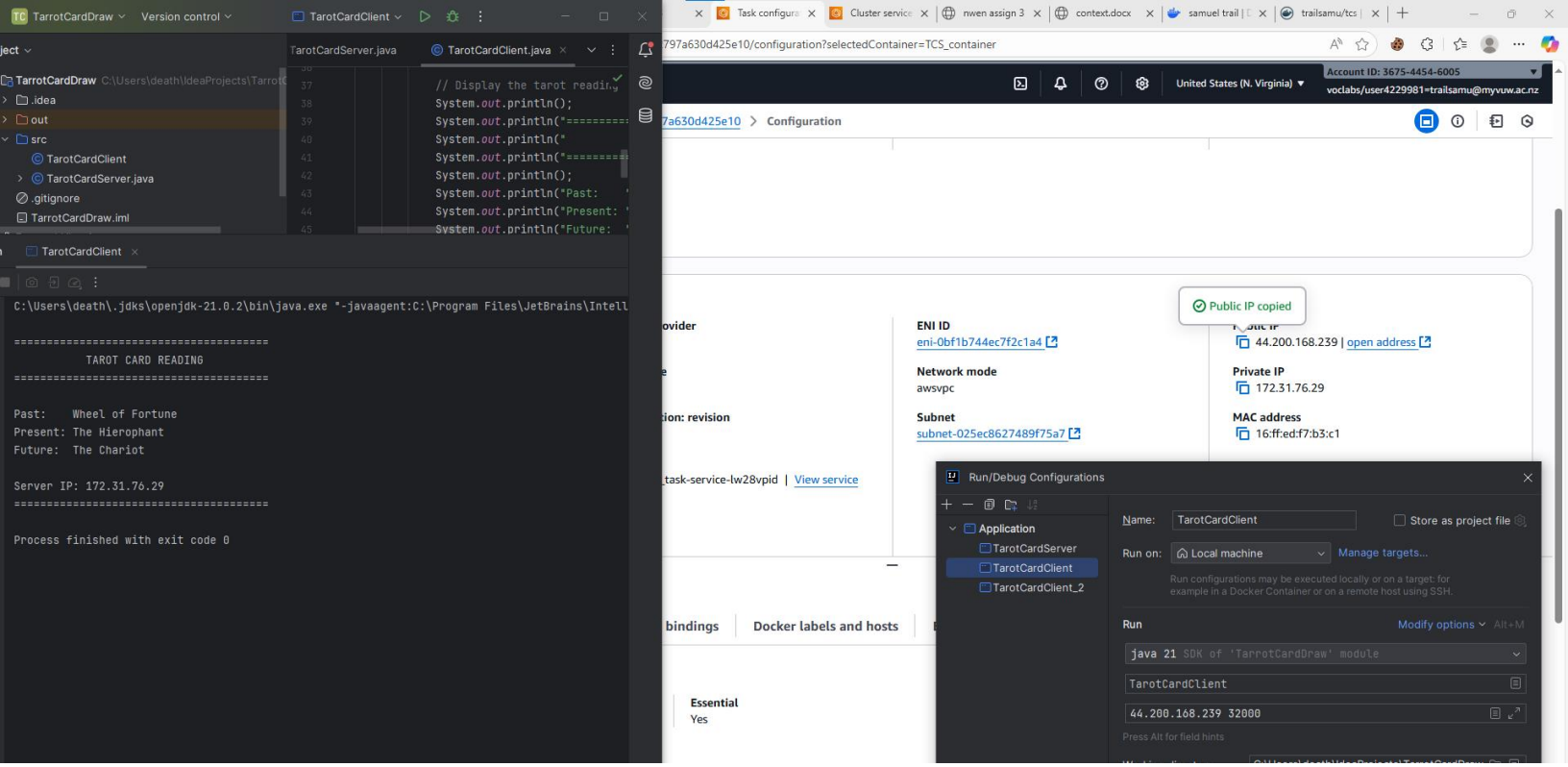
ECS Cluster creation with AWS Fargate serverless infrastructure.

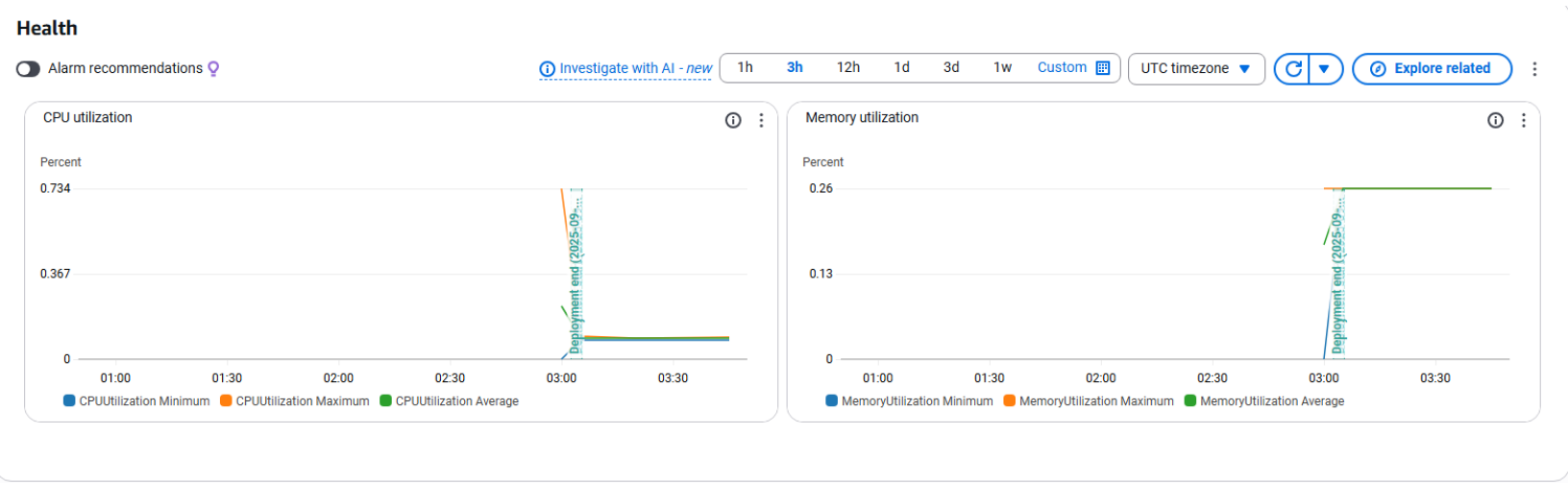


ECS Service deployment showing successful creation of the containerized service. The service is configured to maintain one running task using the Fargate launch type for serverless execution



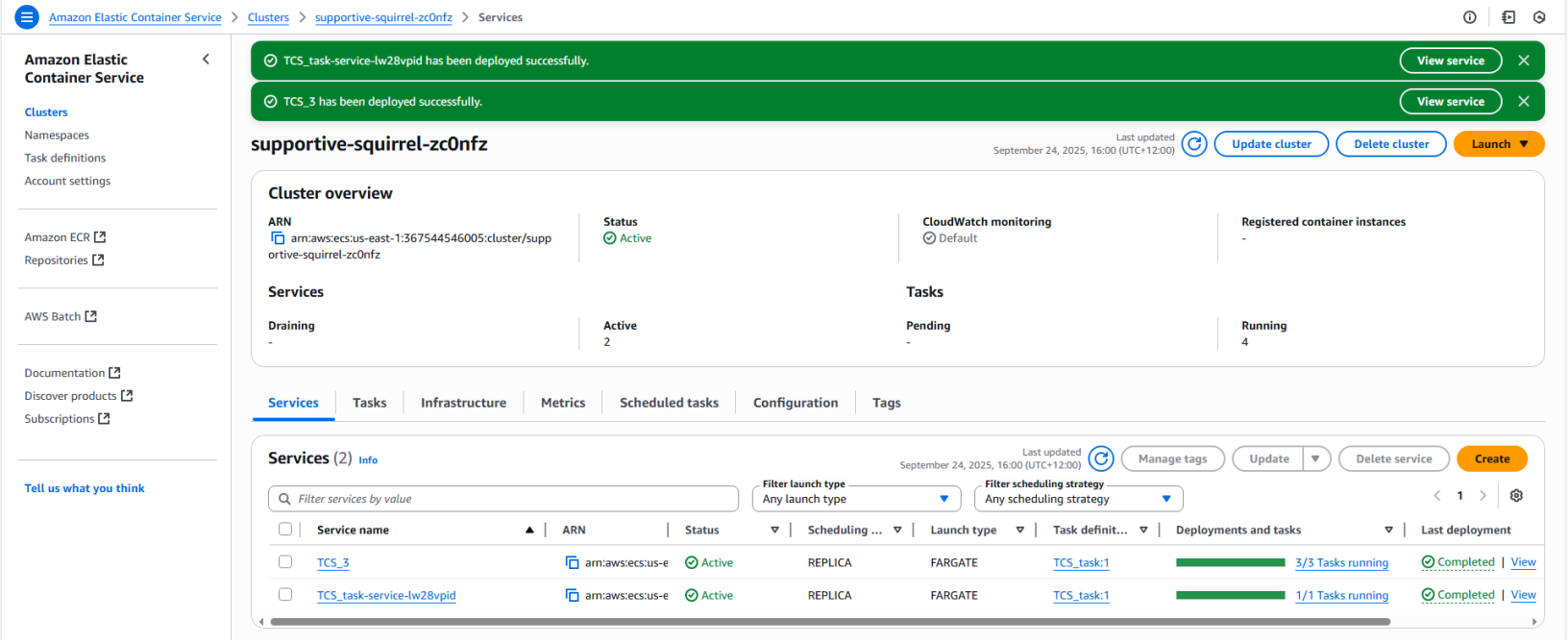
ECS Tasks overview showing one task successfully running in the cluster.

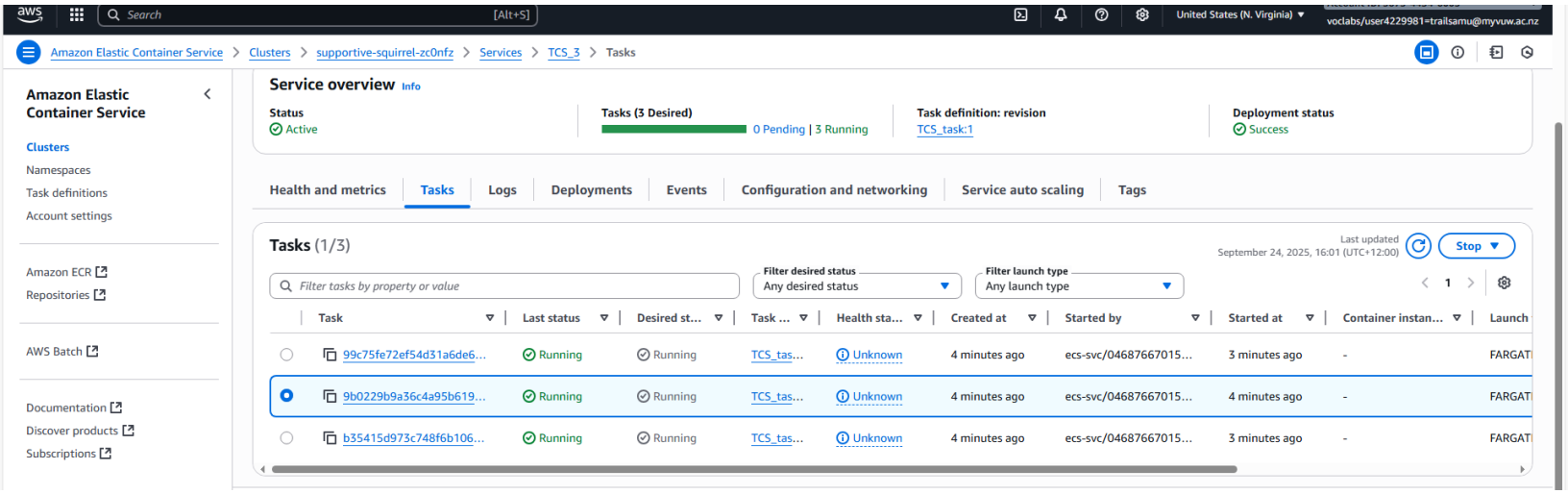


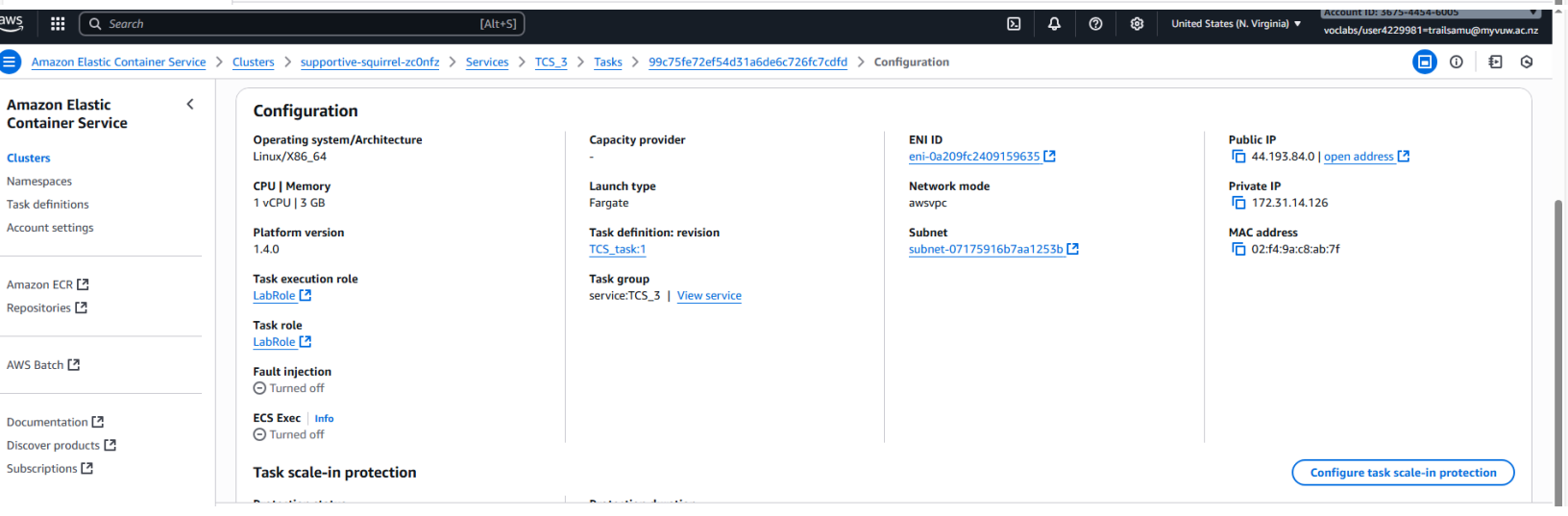


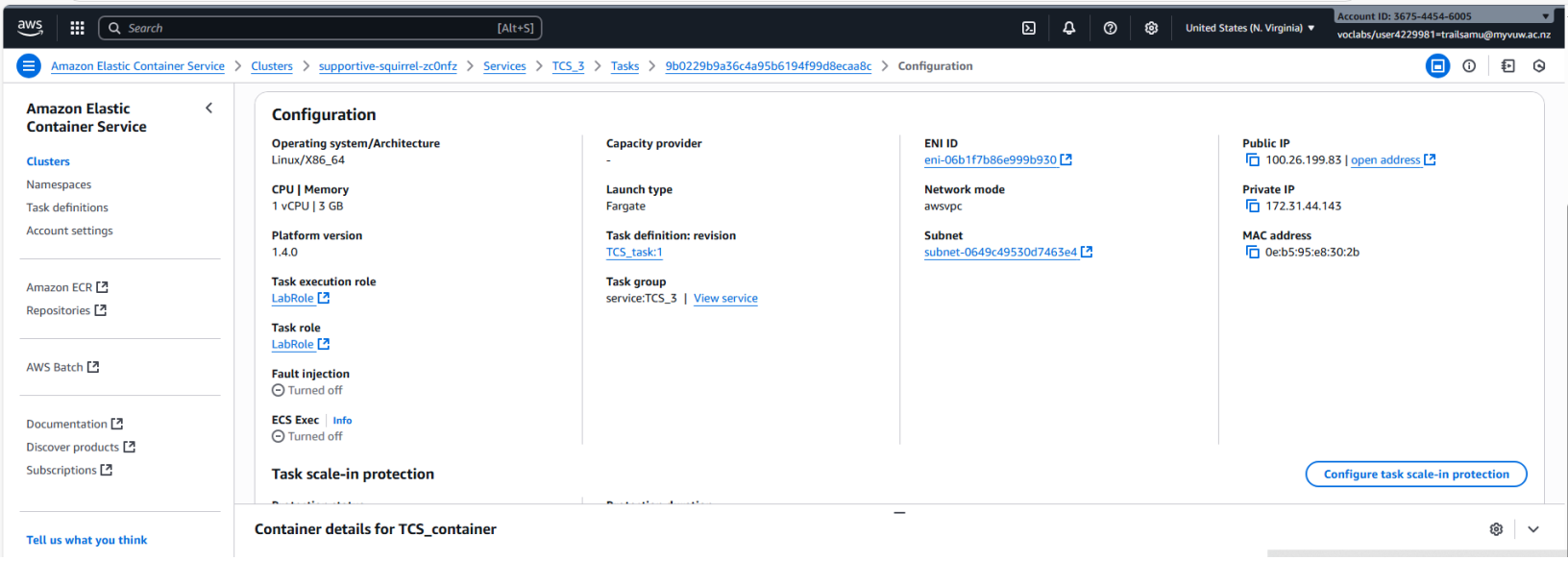
Replicas and Notes

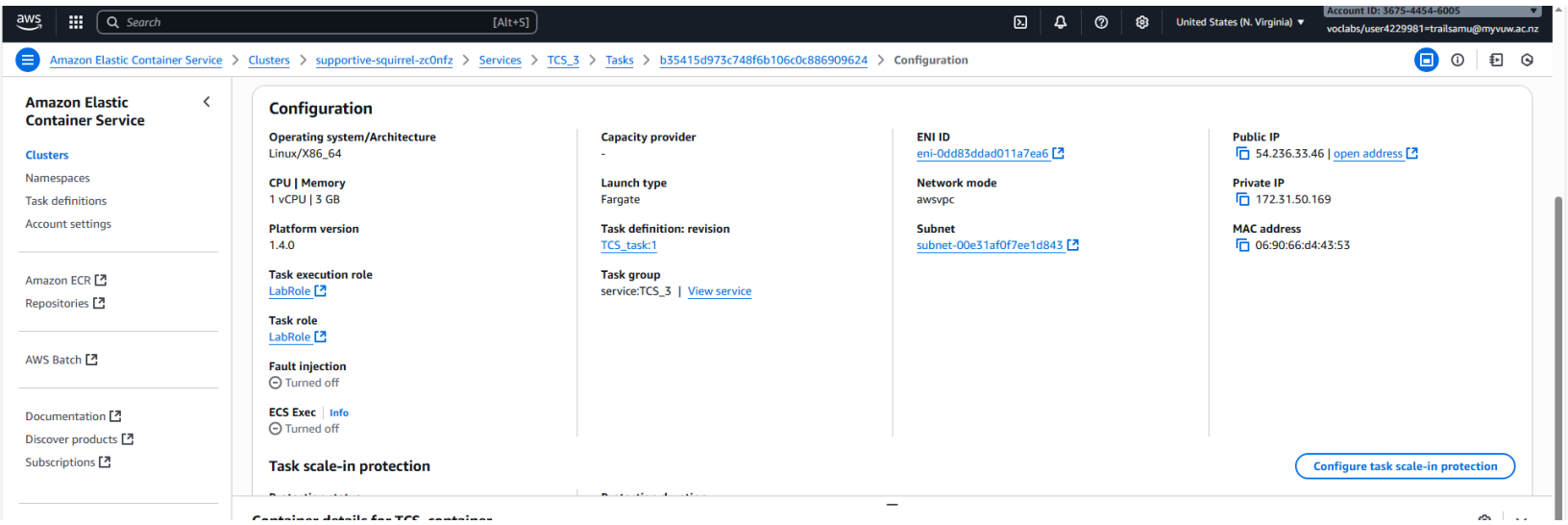
Each replica gets its own unique IP address (both public and private). Even though they're running identical container images with the same code, AWS assigns each task instance its own network interface and IP addresses.


Amazon Elastic Container Service > Clusters > supportive-squirrel-zc0nfz > Services
Amazon Elastic Container Service
Clusters
Namespaces
Task definitions
Account settings
Amazon ECR
Repositories
AWS Batch
Documentation
Discover products
Subscriptions
Tell us what you think
TCS_task-service-lw28vpid has been deployed successfully.
View service
TCS_3 has been deployed successfully.
View service
supportive-squirrel-zc0nfz
Last updated September 24, 2025, 16:00 (UTC+12:00)
Update cluster
Delete cluster
Launch
Cluster overview
ARN
arn:aws:ecs:us-east-1:367544546005:cluster/supportive-squirrel-zc0nfz
Status
Active
CloudWatch monitoring
Default
Registered container instances
-
Services
Draining
-
Active
2
Tasks
Pending
-
Running
4
Services
Tasks
Infrastructure
Metrics
Scheduled tasks
Configuration
Tags
Services (2) Info
Last updated September 24, 2025, 16:00 (UTC+12:00)
Manage tags
Update
Delete service
Create
Filter services by value
Filter launch type
Any launch type
Filter scheduling strategy
Any scheduling strategy
Service name
ARN
Status
Scheduling ...
Launch type
Task definit...
Deployments and tasks
Last deployment
TCS_3
arn:aws:ecs:us-e
Active
REPLICA
FARGATE
TCS_task:1
3/3 Tasks running
Completed | View
TCS_task-service-lw28vpid
arn:aws:ecs:us-e
Active
REPLICA
FARGATE
TCS_task:1
1/1 Tasks running
Completed | View


Search
[Alt+S]
United States (N. Virginia)
voclabs/user4229981=trailsamu@myvuw.ac.nz
Amazon Elastic Container Service > Clusters > supportive-squirrel-zc0nfz > Services > TCS_3 > Tasks
Amazon Elastic Container Service
Clusters
Namespaces
Task definitions
Account settings
Amazon ECR
Repositories
AWS Batch
Documentation
Discover products
Subscriptions
Service overview Info
Status
Active
Tasks (3 Desired)
0 Pending | 3 Running
Task definition: revision
TCS_task:1
Deployment status
Success
Health and metrics
Tasks
Logs
Deployments
Events
Configuration and networking
Service auto scaling
Tags
Tasks (1/3)
Last updated September 24, 2025, 16:01 (UTC+12:00)
Stop
Filter tasks by property or value
Filter desired status
Any desired status
Filter launch type
Any launch type
Task
Last status
Desired st...
Task ...
Health sta...
Created at
Started by
Started at
Container instan...
Launch
99c75fe72ef54d31a6de6...
Running
Running
TCS_tas...
Unknown
4 minutes ago
ecs-svc/04687667015...
3 minutes ago
-
FARGAT
9b0229b9a36c4a95b619...
Running
Running
TCS_tas...
Unknown
4 minutes ago
ecs-svc/04687667015...
4 minutes ago
-
FARGAT
b35415d973c748f6b106...
Running
Running
TCS_tas...
Unknown
4 minutes ago
ecs-svc/04687667015...
3 minutes ago
-
FARGAT


Search
[Alt+S]
United States (N. Virginia)
voclabs/user4229981=trailsamu@myvuw.ac.nz
Amazon Elastic Container Service > Clusters > supportive-squirrel-zc0nfz > Services > TCS_3 > Tasks > 99c75fe72ef54d31a6de6c726fc7cdfd > Configuration
Amazon Elastic Container Service
Clusters
Namespaces
Task definitions
Account settings
Amazon ECR
Repositories
AWS Batch
Documentation
Discover products
Subscriptions
Configuration
Operating system/Architecture
Linux/X86_64
CPU | Memory
1 vCPU | 3 GB
Platform version
1.4.0
Task execution role
LabRole
Task role
LabRole
Fault injection
Turned off
ECS Exec Info
Turned off
Capacity provider
-
Launch type
Fargate
Task definition: revision
TCS_task:1
Task group
service:TCS_3 | View service
ENI ID
eni-0a209fc2409159635
Network mode
awsvpc
Subnet
subnet-07175916b7aa1253b
Public IP
44.193.84.0 | open address
Private IP
172.31.14.126
MAC address
02:f4:9a:c8:ab:7f
Task scale-in protection
Configure task scale-in protection





Load Balancer Solution: To provide a single endpoint for clients, a load balancer is implemented. Instead of clients connecting directly to individual replica IPs, they connect to a single load balancer endpoint that distributes requests across all healthy replicas automatically. This provides better availability, automatic failover, and distributes load evenly