Final Project Group 2

Project Overview: Running Web application (Twitter application) in ECS and performing schedule based auto scaling of tasks and ECS instances.

Cloud provider : AWS

Aws Components:

1. Code Build
2. Load Balancer
3. Target Groups
4. Amazon Elastic Contaner Service(ECS)

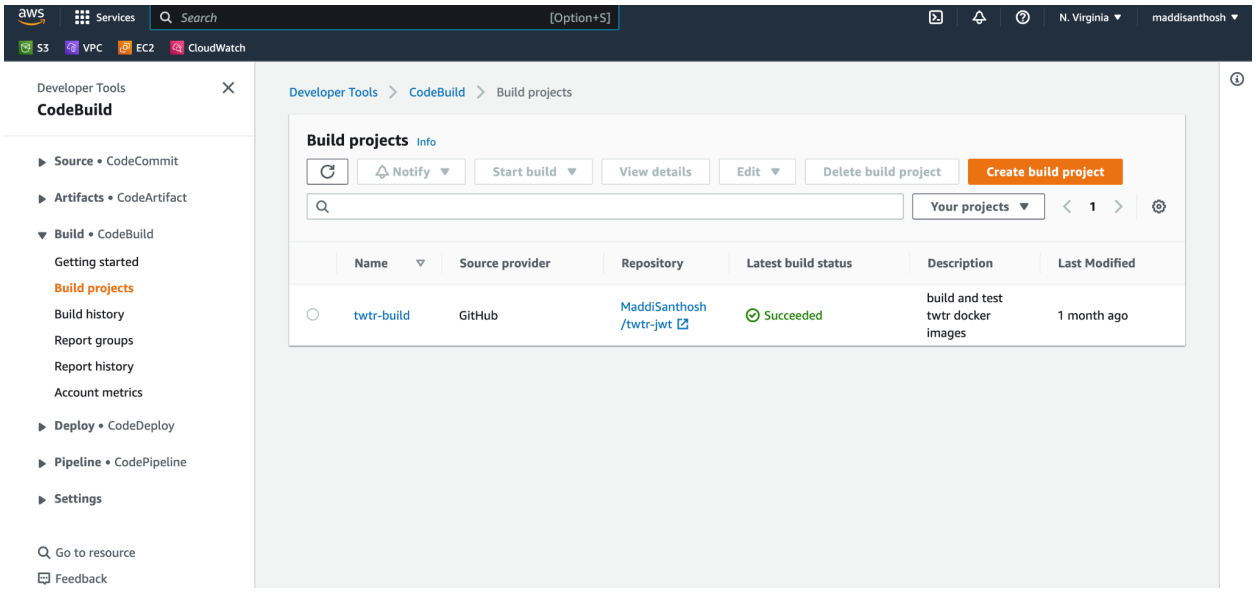
Deployment and application workflow

Code build and Github

1. Push twitter backend and frontend code along with buildspec and and docker files to a repository. (Reference link : https://github.com/dinorows/twtr-jwt)
2. Create a code build in AWS and use GitHub for the Source provider. Select Connect using OAuth, and click Connect to Github and allow access to your GitHub repo for twtr-jwt.
3. After authenticating, under Repository, select Repository in my GitHub account. Then, add the GitHub repository you created for this project.
4. Environment settings

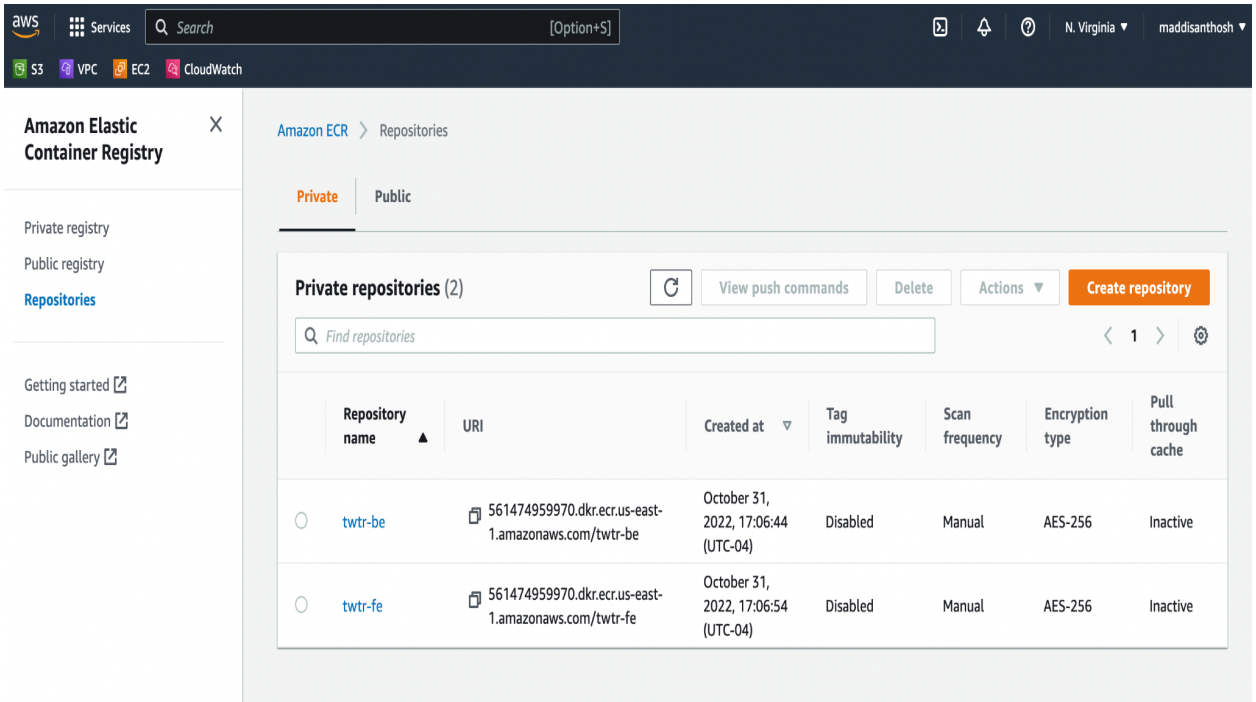
   - Environment image - use the Managed image
   - Operating system - Ubuntu
   - Runtime - Standard
   - Image - aws/codebuild/standard:4.0
   - Image version - Always use the latest image for this runtime version
   - Privileged - check the flag
   - Service role - New service role
   - Role name - flask-react-build-role
5. Add your AWS account ID and AWS region as environment variables called AWS_ACCOUNT_ID and AWS_REGION in plaintext
6. Under Build specifications, select Use a buildspec file.

Screenshot of Code Build in AWS



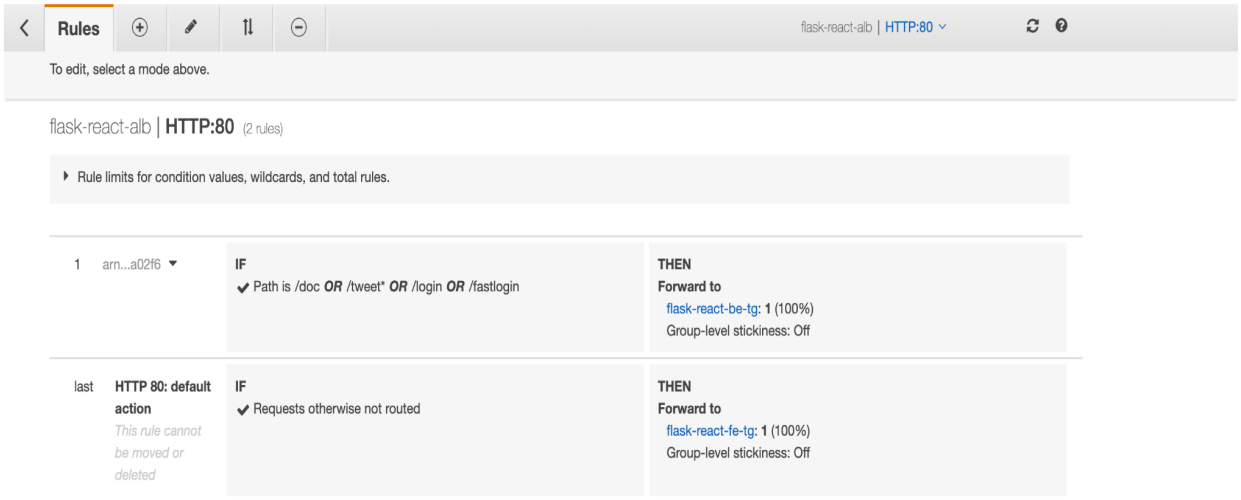
Note: If code build fails because of docker throttling add docker login command to the pre build section in build spec file.

Docker images in ECR (Built and pushed by code build)

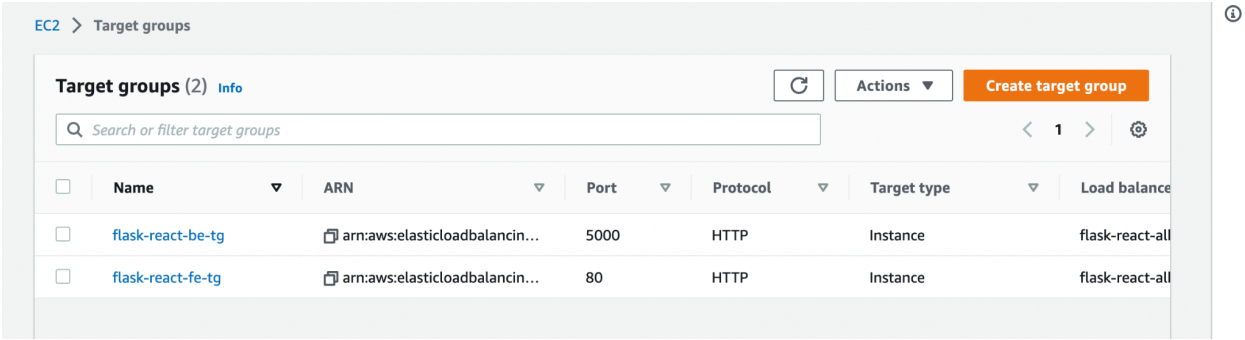


Application Load Balancer

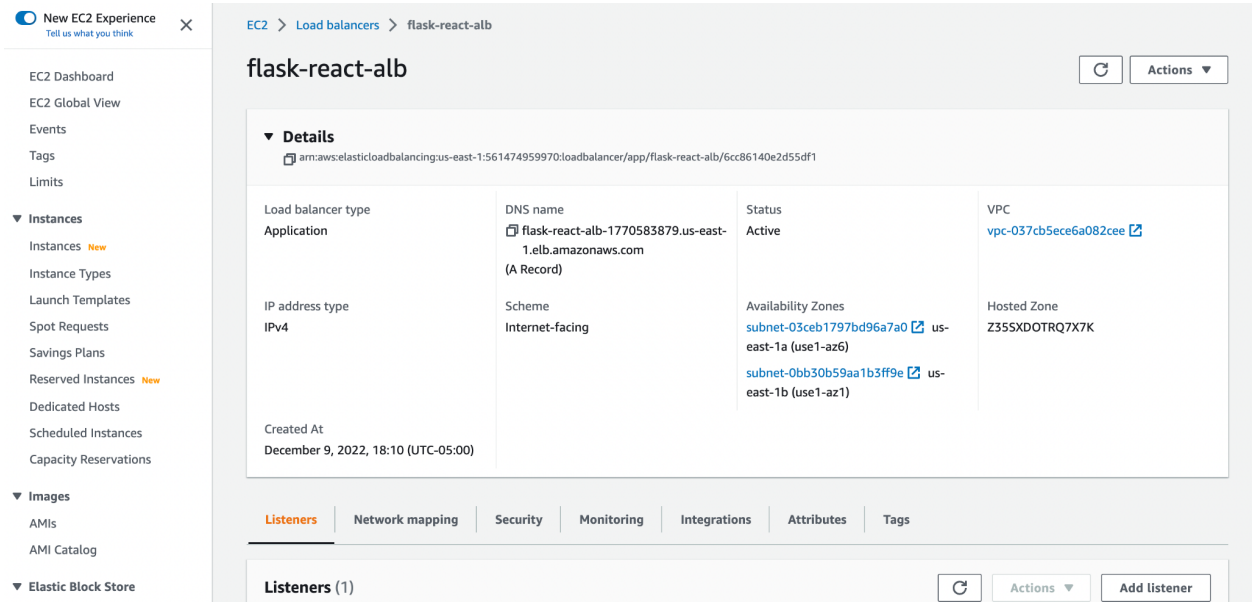
1. Configure Load Balancer – Name:flask-react-alb
   – Scheme:internet-facing
   – IP address type: ipv4
   – Listeners: HTTP / Port 80
   – VPC: Select the default VPC to keep things simple
   – Availability Zones: Select at least two available subnets
2. Skip configure security settings
3. Configure security groups : select default security group.
4. Create a target group called “flask-react-fe-tg”, type : instance, Port: 80, Path: “/’
5. Create a target group called “flask-react-be-tg”, type : instance, Port: 5000, Path: “/doc”
6. Register both target groups with load balancer
7. Edit listener rules of the load balancer as shown in below screenshot



Target Groups :



Load Balancer



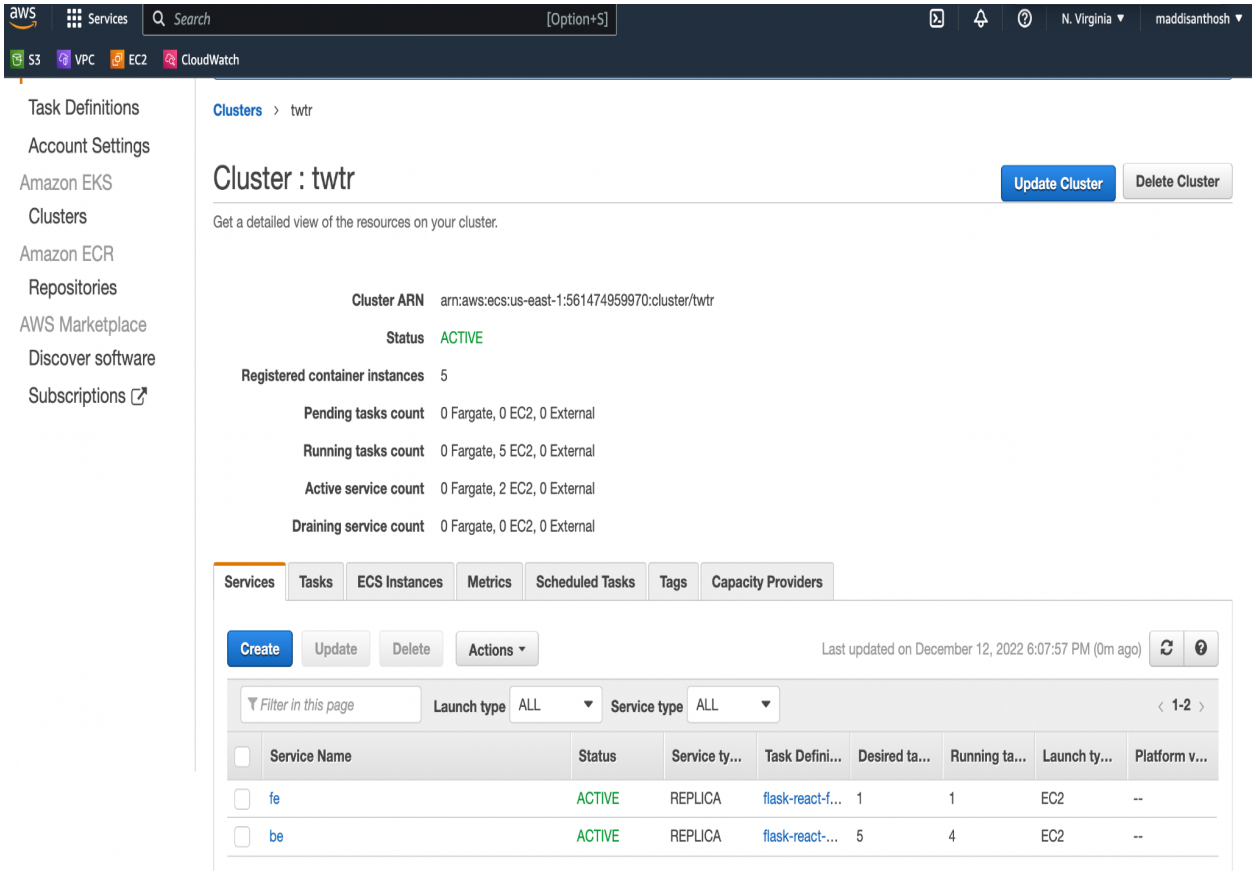
Note： Get the DNS of the load balancer from Load balancer details page. Replace the url (REACT_APP_API_SERVICE_URL) in build spec file with load balancer DNS. Make sure that code build triggers after updating build spec file.

AWS ECS

Create Ecs cluster

1. Select EC2 Linux + Networking
2. Cluster name: flask-react-cluster
3. EC2 instance type: t2.micro
4. Number of instances: 2
5. Select default VPC(same as load balancer vpc)
6. Select existing key pair or create new key pair

Cluster



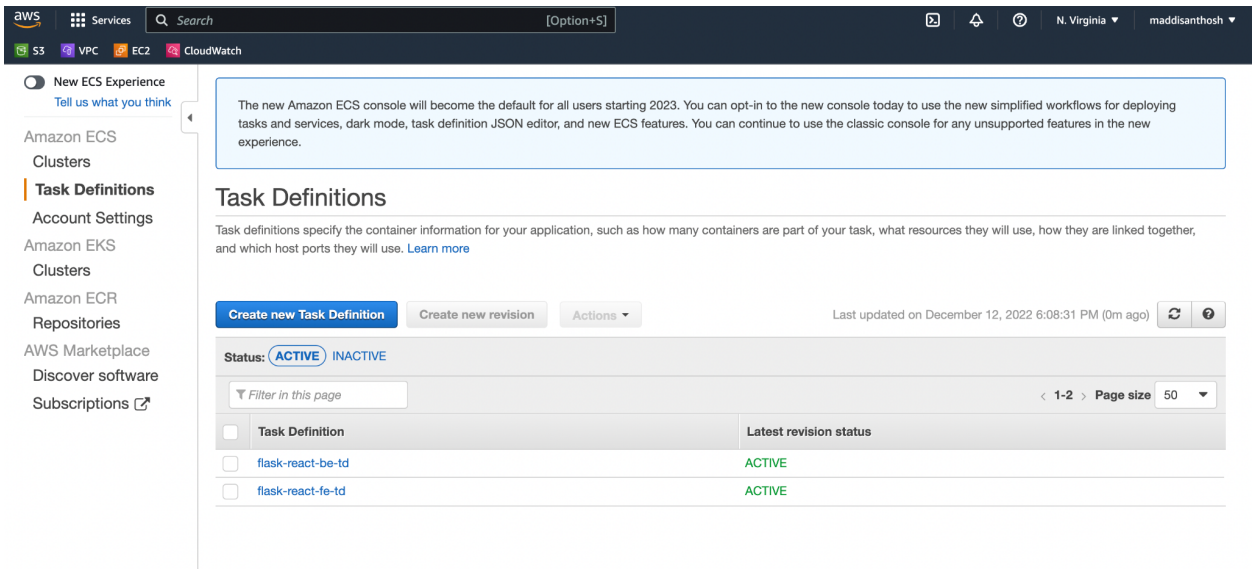
Create frontend Task Defination:

1. 2select EC2 in the Select launch type compatibility screen
2. Set Task Definition Name to flask-react-fe-td Task Role: do not select
3. Network Mode: do not select
4. Task execution IAM role: ecsTaskExecutionRole
5. Add container: Container name: fe
6. Image: <YOUR_AWS_ACCOUNT_ID>.dkr.ecr.us-east- 1.amazonaws.com/twtr-fe:prod
7. Memory Limits (MB): 500 soft limit
8. Port mappings: 0 or empty host, 80 container

Create backend Task Defination:

1. select EC2 in the Select launch type compatibility screen
2. Set Task Definition Name to flask-react-be-td Task Role: do not select
3. Network Mode: do not select
4. Task execution IAM role: ecsTaskExecutionRole
5. Add container: Container name: be
6. Image: <YOUR_AWS_ACCOUNT_ID>.dkr.ecr.us-east- 1.amazonaws.com/twtr-be:prod
7. Memory Limits (MB): 500 soft limit
8. Port mappings: 0 or empty host, 5000 container

Task Defination:



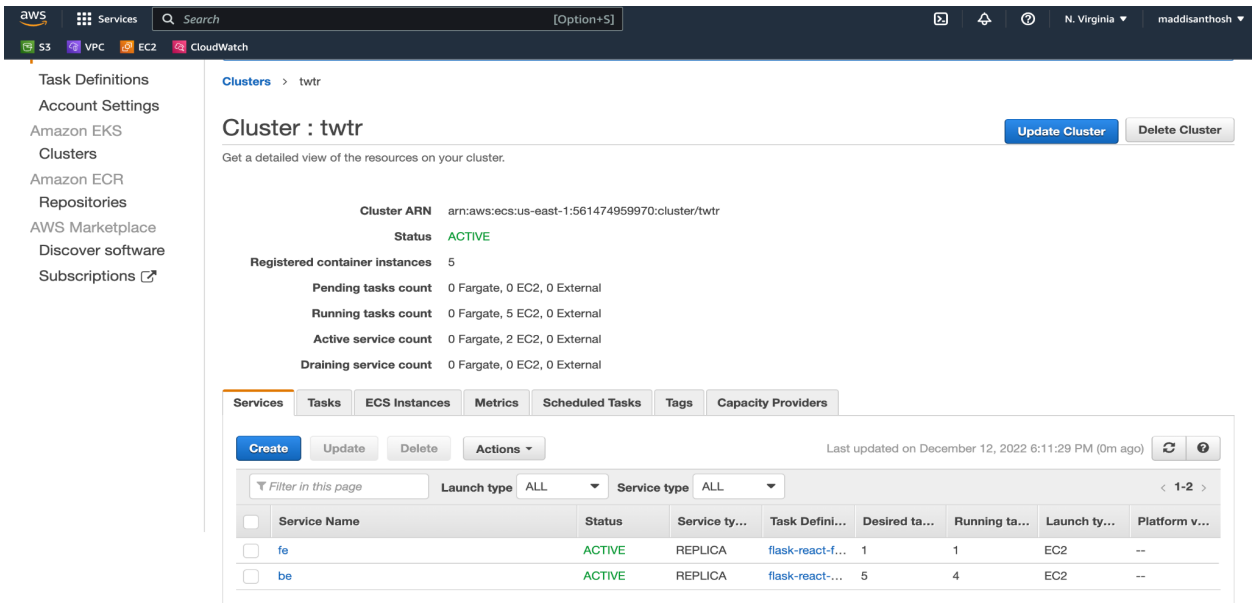
Create backend Service:

1. Launch type: EC2
2. Task Definition:
    a. – Family:flask-react-be-td
    b. – Revision:LATEST_REVISION_NUMBER
3. Service name: flask-react-be-service Number of tasks: 1
4. Select the Application Load Balancer under Load balancer type
5. Load balancer name: flask-react-alb Container name: port: be:0:5000
6. Click Add to load balancer
7. Production listener port: 80:HTTP
8. Target group name: flask-react-be-tg

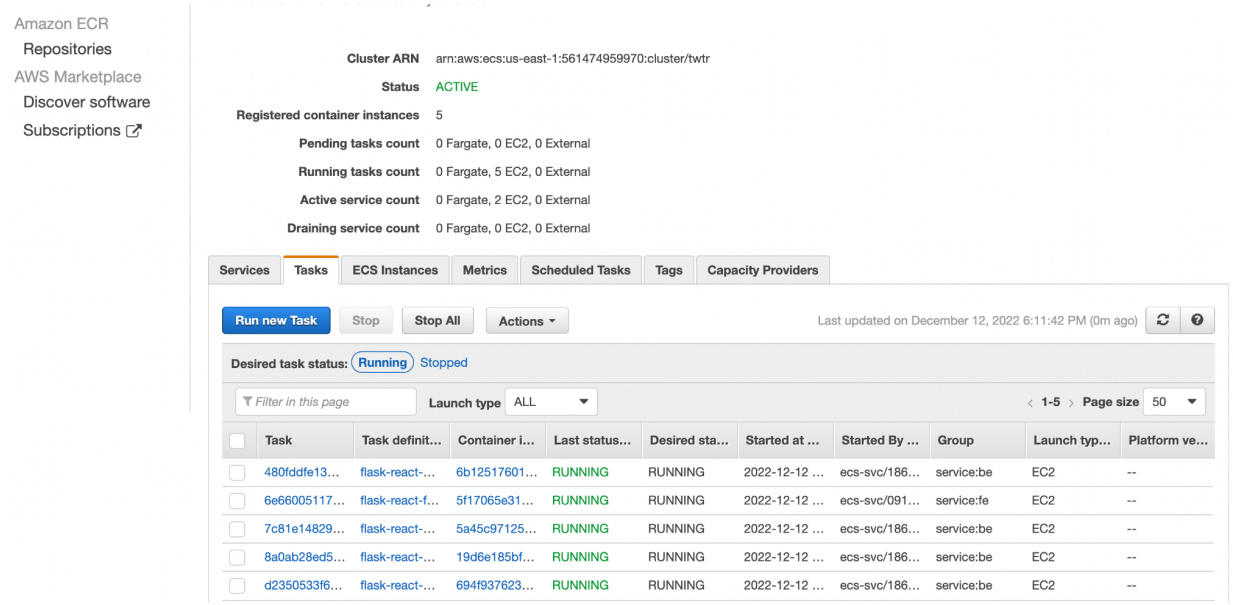
Create fropmtend Service:

1. Launch type: EC2
2. Task Definition:
    a. – Family:flask-react-fe-td
    b. – Revision:LATEST_REVISION_NUMBER
3. Service name: flask-react-fe-service, Number of tasks: 1
4. Select the Application Load Balancer under Load balancer type
5. Load balancer name: flask-react-alb Container name: port: be:0:80
6. Click Add to load balancer
7. Production listener port: 80:HTTP
8. Target group name: flask-react-fe-tg

Services



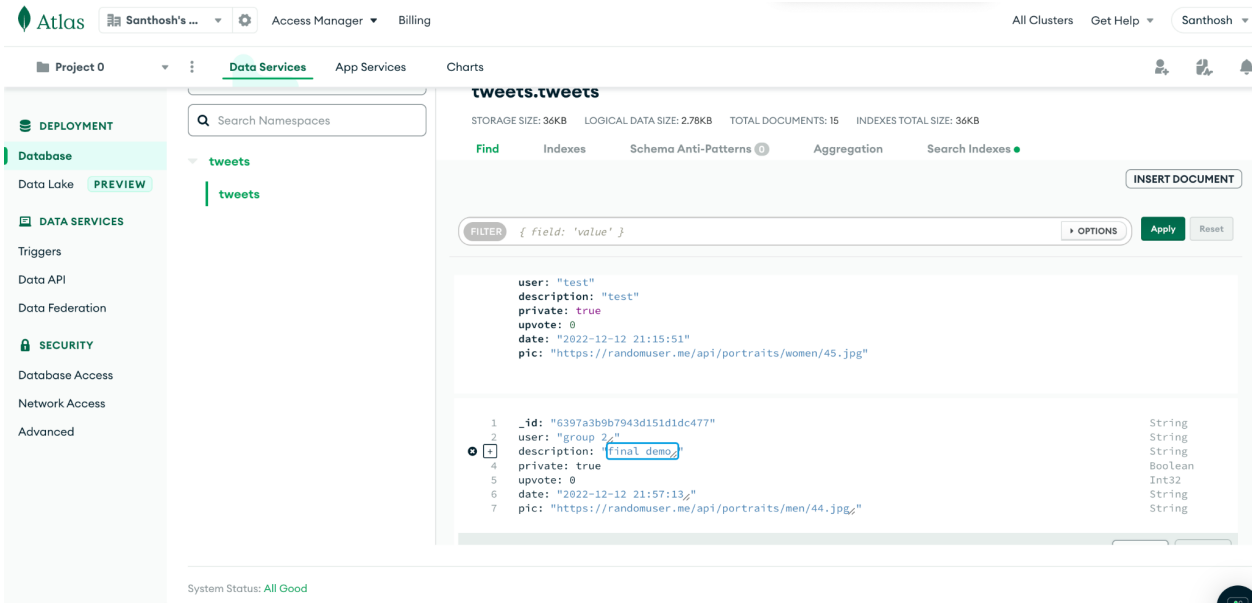
Note: Verify that new tasks are created and status is running as shown in below image.
TASKS:



Note: In Ec2's security groups page select the Security Group associated with the containers, and allow traffic with selecting source as ALB's security group.

FRONTEND:(Access using DNS of load balancer)

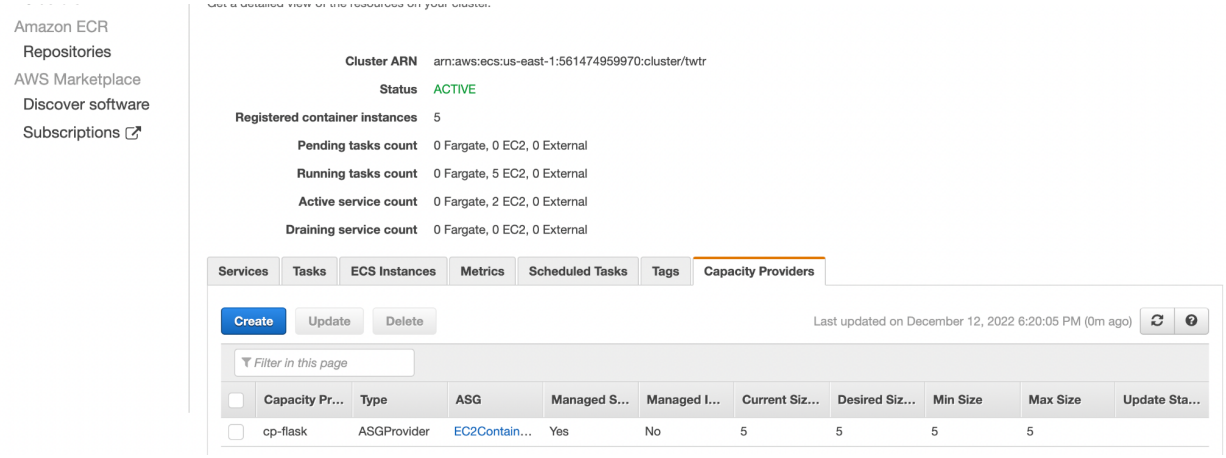
ATLAS:



AutoScaling

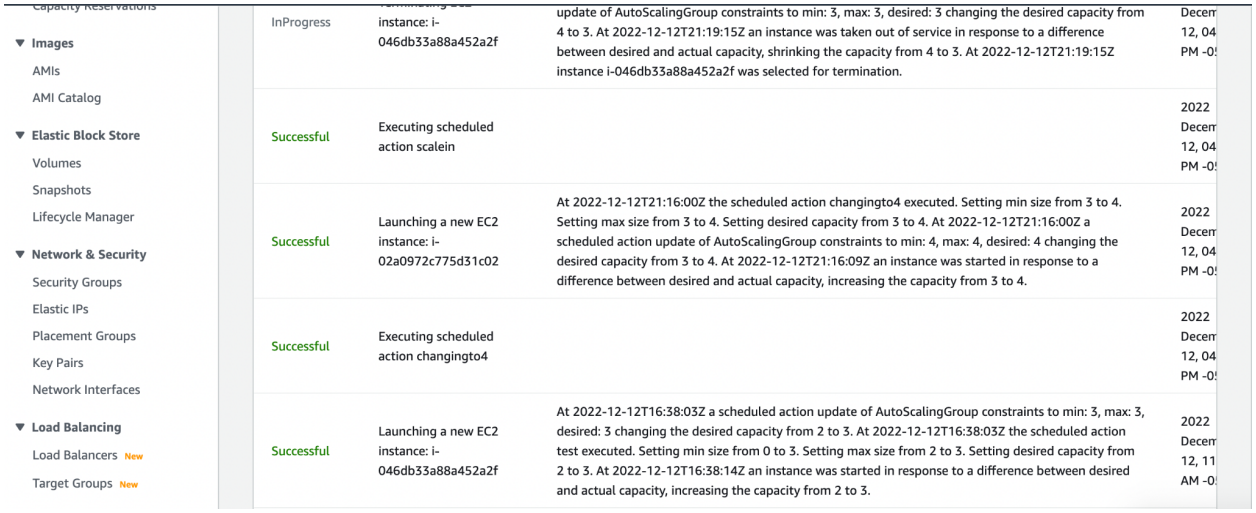
CAPACITY PROVIDERS:

1. Create a capacity provider in the twitter cluster and select the auto scaling group associated with cluster.

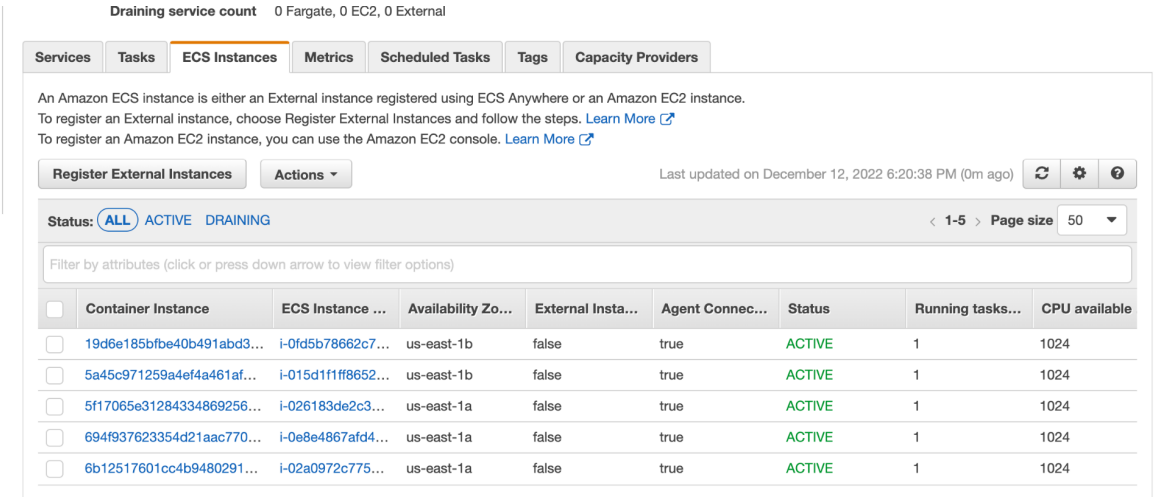


2. Go the auto scaling group and create a scheduled action and set desire capacity to 5 and time stamp to the current time stamp and submit
3. Go back to the cluster and verify that cluster has 5 ECS instances.

Verify the scheduled action status in activity of ASG as shown below



ECS INSTANCES:



Auto scaling Service Tasks:

Use below AWS CLI commands to scale n and scale out tasks of a service .

1. **Register target :**

   aws application-autoscaling register-scalable-target --service-namespace ecs --scalable-dimension ecs:service:DesiredCount --resource-id service/<cluset-name>/<service-name> --min-capacity 1 --max-capacity 10 --region us-east-1

2. **Put scheduled action**

   aws application-autoscaling put-scheduled-action --service-namespace ecs --scalable-dimension ecs:service:DesiredCount --resource-id service/<cluset-name>/<service-name> --scheduled-action-name single-scaleout-action --schedule "at(2022-12-12T21:17:00)" --scalable-target-action MinCapacity=1,MaxCapacity=1 --region us-east-1

After running the command by providing correct cluster name, service name and current time stamp go back to the service and verify that number of task are changed as expected.

References:

1. https://aws.amazon.com/blogs/containers/optimizing-amazon-elastic-container-service-for-cost-using-scheduled-scaling/
2.