AWS ECS CheatSheet

1. Cloned my existing GitHub public repo for the project with the cmd **git clone https://github.com/sriharshaperi/twtr-jwt.git**
2. Added global variable in terminal **export REACT_APP_API_SERVICE_URL=http://localhost:5004**
3. Removed jet auth from twtr backend and fetch calls in React app
4. Replaced fetch call hosts routing to localhost with **REACT_APP_API_SERVICE_URL**
5. Added Dockerfiles in both fe and be folders of the project repo
6. **docker build -f be/Dockerfile -t \ 101871923216.dkr.ecr.us-east- 1.amazonaws.com/twtr-be:dev ./be**
7. **docker build -f fe/Dockerfile -t \ 101871923216.dkr.ecr.us-east- 1.amazonaws.com/twtr-fe:dev ./fe**
8. Authorized AWS CLI with docker credentials **aws ecr get-login-password --region us-east-1 \ | docker login --username AWS --password-stdin \ 101871923216.dkr.ecr.us-east-1.amazonaws.com**
9. **docker push 101871923216.dkr.ecr.us-east-1.amazonaws.com/twtr-be:dev**
10. **docker push 101871923216.dkr.ecr.us-east-1.amazonaws.com/twtr-fe:dev**
11. Created build repository on AWS CodeBuild with settings enabled to trigger build with respect to commits and pushes to GitHub repository
12. Attached **AmazonEC2ContainerRegistryReadOnly, AmazonEC2ContainerRegistryFullAccess and EC2InstanceProfileForImageBuilderECRContainerBuilds** policies to IAM Role created in AWS CodeBuild repo
13. Created a new Application Load Balancer and attached a frontend target group to it
14. Added a new listener rule to ALB to route /doc OR /tweet* OR /login OR /fastlogin routes to the backend target group
15. Replaced the REACT_APP_API_SERVICE_URL with the DNS Name of the ALB in the buildspec.yaml file and pushed into GitHub repo and that triggered auto build on AWS CodeBuild
16. Created AWS ECS Cluster and TaskDefinitions under the cluster for frontend and backend
17. Created services for frontend and backend and assigned the initial number of tasks to be run on services.
18. **Concept :** *AWS ECS Cluster has Services and Services have associations with the respective TaskDefinitions. Service has tasks associated and EC2 instances created by AWS ECS cluster run the tasks within these.*
19. After the two services have reached the steady state, the application can be accessed through the DNS name of the load balancer
20. Auto Scaling group gets created. Assign the AutoScaling group to the Capacity Provider and assign base and weights. Capacity Provider with base 1 and weight 4 indicates minimum number of tasks to be run is 1 and all 4 tasks must be run under a single EC2 instance
21. Scheduled a scheduler for upscaling and downscaling EC2 instances using a cron scheduler in AutoScaling Group
22. Scheduled a scheduler for upscaling and downscaling tasks using a task scheduler in each service
23. Estimated and compared Cost Analysis for Spot Instance vs OnDemand Instance and used OnDemand Instance in the project to ensure more convenience in upscaling of EC2 instances