HW4 Report

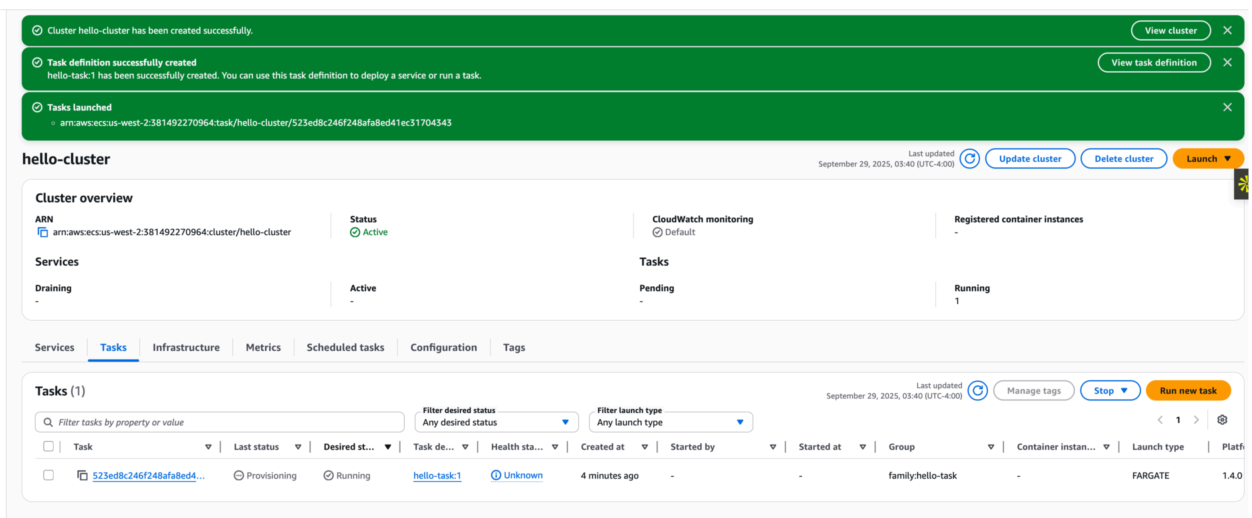
Rong Huang

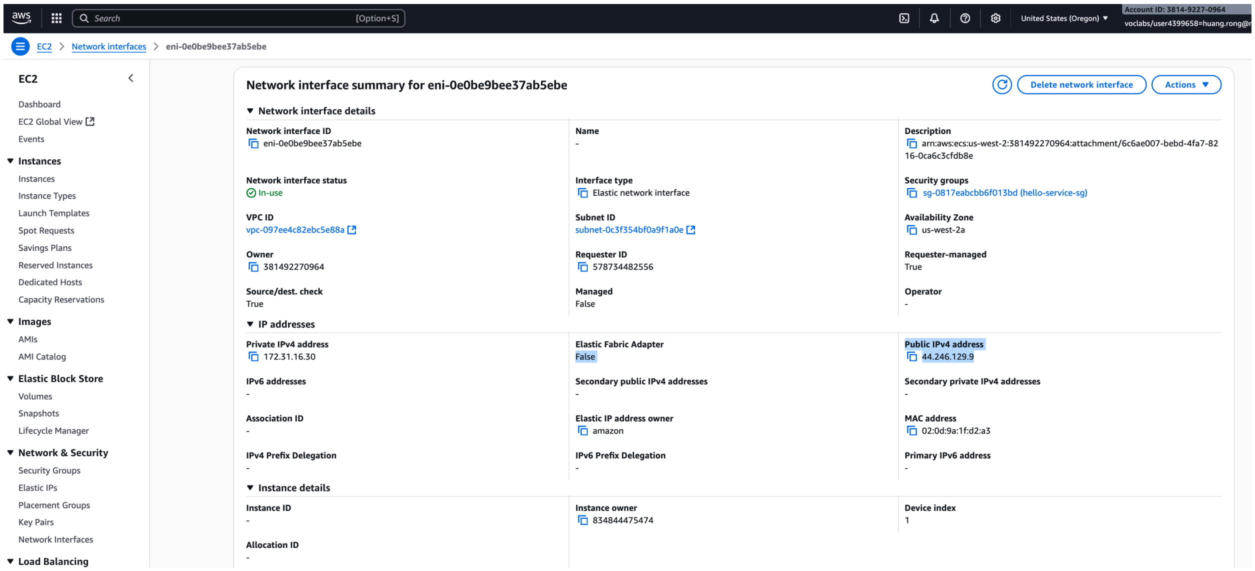
CS6650 25Fall

This week I finally got a full feel for the ECR/ECS workflow. It's a lot of clicking the first time, but once the pieces connect (push image to ECR, define a task, run it on Fargate), it starts to make sense. I can see why people automate this with tools like Terraform—we'll probably do that next.

Part II: Infrastructure set up!

Result






```
(base) ronghuang@Reginas-macbook hello-service % curl http://44.246.129.9:8080/albums
[
    {
        "id": "1",
        "title": "Blue Train",
        "artist": "John Coltrane",
        "price": 56.99
    },
    {
        "id": "2",
        "title": "Jeru",
        "artist": "Gerry Mulligan",
        "price": 17.99
    },
    {
        "id": "3",
        "title": "Sarah Vaughan and Clifford Brown",
        "artist": "Sarah Vaughan",
        "price": 39.99
    }
]%
(base) ronghuang@Reginas-macbook hello-service % curl http://44.246.129.9:8080/albums/1
{
    "id": "1",
    "title": "Blue Train",
    "artist": "John Coltrane",
    "price": 56.99
}%
(base) ronghuang@Reginas-macbook hello-service %
```

Things I explored and what I learned

1. Difference between EC2 and ECS

- EC2: I manage the virtual machines myself.
  I have to think about installing stuff, scaling up, patching, etc.
  It’s flexible but more work.

- ECS (with Fargate): I just tell AWS
  what container to run and how much CPU/memory it needs. No servers to manage. This was easier for my small tasks.

2. What is a VPC and a subnet? How did I access the default VPC?

- VPC is like my private network in AWS. A subnet is
  a slice of that network in a specific availability zone.
- I used the default VPC and a public subnet. I enabled a public IP on my task and opened port 8080 in the security group, so I could curl the service from my laptop.

3. What is TCP and how is it different from UDP?

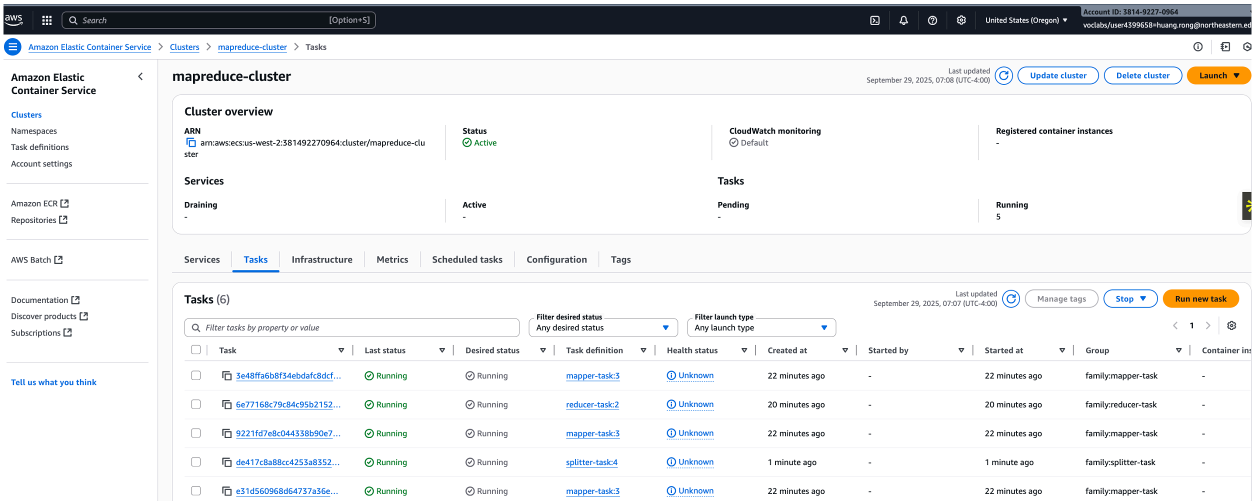
- TCP is a reliable connection (it makes sure all data arrives and in order).
  It’s good for HTTP and my JSON requests.
- UDP is faster but doesn’t guarantee delivery or order.
  It’s more for streaming or custom cases where I handle reliability myself.

4. How do I control resources for a task?

- In the task definition, I set CPU and memory (for example, 0.25 vCPU and 0.5 GB). Fargate enforces that. If I need more power,
  I can bump these numbers or run more tasks at once.

Part III: Map Reduce

Result




```
(base) ronghuang@Reginas-macbook reducer % aws s3 ls s3://mapreduce-regina-1759134570
/input/ --region us-west-2 | cat                                     aws s3 ls s3://mapre
duce-regina-1759134570/input/ --region us-west-2 | cat
2025-09-29 04:32:11        123 test.txt
(base) ronghuang@Reginas-macbook reducer % aws ec2 describe-network-interfaces --netw
ork-interface-ids eni-064995fef87407b71 --region us-west-2 --query 'NetworkInterfaces
[0].Association.PublicIp' --output text | cat
  aws ec2 describe-network-interfaces --network-interface-ids eni-064995fef87407b71 -
-region us-west-2 --query 'NetworkInterfaces[0].Association.PublicIp' --output text |
 cat
35.90.79.175
<TTER $MAPPER1 $MAPPER2 $MAPPER3 $REDUCER $BUCKET_URL
http://35.88.206.27:8080 http://44.244.94.248:8080 http://35.86.204.227:8080 http://3
5.86.204.227:8080 http://35.90.79.175:8080 s3://mapreduce-regina-1759134570/input/tes
t.txt
(base) ronghuang@Reginas-macbook reducer % curl -s -X POST "$SPLITTER/split-s3" -H "C
ontent-Type: application/json" -d "{\"s3_url\":\"$BUCKET_URL\",\"curl -s -X POST "$SP
LITTER/split-s3" -H "Content-Type: application/json" -d "{\"s3_url\":\"$BUCKET_URL\",
\"chunks\":3}" | tee /tmp/split_resp.json | cat
{"chunk_urls":["s3://mapreduce-regina-1759134570/chunks/1759151384/chunk_0.txt","s3:/
/mapreduce-regina-1759134570/chunks/1759151384/chunk_1.txt","s3://mapreduce-regina-17
59134570/chunks/1759151384/chunk_2.txt"],"total_chunks":3}
(base) ronghuang@Reginas-macbook reducer %
```

```json
{
    "final_count": {
        "awesome": 2,
        "hello": 4,
        "is": 1,
        "mapreduce": 3,
        "test": 4,
        "world": 5
    },
    "total_words": 19,
```

TAB to Dockerfile →

```
SPLITTER=http://35.88.206.27:8080; MAPPER1=http://44.244.94.248:8080; MAPPER2=http://35.86.204.227:8080; MAPPER3=http://35.8
OK
http://35.86.204.227:8080/health
OK
http://35.86.204.227:8080/health
OK
http://35.90.79.175:8080/health
OK
--- SPLIT ---
{"chunk_urls":["s3://mapreduce-regina-1759134570/chunks/1759151520/chunk_0.txt","s3://mapreduce-regina-175913
4570/chunks/1759151520/chunk_1.txt","s3://mapreduce-regina-1759134570/chunks/1759151520/chunk_2.txt"],"total_
chunks":3}
Chunks:
s3://mapreduce-regina-1759134570/chunks/1759151520/chunk_0.txt
s3://mapreduce-regina-1759134570/chunks/1759151520/chunk_1.txt
s3://mapreduce-regina-1759134570/chunks/1759151520/chunk_2.txt
--- MAP ---
{"result_url":"s3://mapreduce-regina-1759134570/map-results/1759151521/chunk_0_result.json"}
{"result_url":"s3://mapreduce-regina-1759134570/map-results/1759151521/chunk_1_result.json"}
{"result_url":"s3://mapreduce-regina-1759134570/map-results/1759151521/chunk_2_result.json"}
Results:
s3://mapreduce-regina-1759134570/map-results/1759151521/chunk_0_result.json
s3://mapreduce-regina-1759134570/map-results/1759151521/chunk_1_result.json
s3://mapreduce-regina-1759134570/map-results/1759151521/chunk_2_result.json
--- REDUCE ---
{"final_result_url":"s3://mapreduce-regina-1759134570/final-results/1759151522/final_word_count.json"}
Final: s3://mapreduce-regina-1759134570/final-results/1759151522/final_word_count.json
download: s3://mapreduce-regina-1759134570/final-results/1759151522/final_word_count.json to ./final_word_cou
nt.json
{"final_count":{"awesome":2,"hello":4,"is":1,"mapreduce":3,"test":4,"world":5},"total_words":19,"unique_words
":6,"top_10_words":[{"word":"world","count":5},{"word":"hello","count":4},{"word":"test","count":4},{"word":"
mapreduce","count":3},{"word":"awesome","count":2},{"word":"is","count":1}]}%
(base) ronghuang@Reginas-macbook cs6650 %
```

Mini MapReduce (word count) — what I built

- One Splitter task: reads a text file from S3, splits it into 3 chunks, and writes the chunks back to S3.
- Three Mapper tasks: each reads one chunk, counts words, and writes a small JSON to S3.
- One Reducer task: combines the three mapper JSON files, sums the counts, and writes the final result to S3.

I used simple HTTP endpoints on port 8080 for each. All tasks ran on ECS Fargate. IAM role had S3 read/write and ECR pull.

My results (end-to-end time)

- Data: Shakespeare’s Hamlet plain text, about 159 KB, stored in S3.
  Source: shakespeare-hamlet.txt.
- Because the file is small, total time is dominated by startup and network/S3 overhead. The actual counting is essentially instant.
- With such a small input, using 1, 2, or 3 mappers showed no meaningful speedup; overhead hides the benefit.

Questions

- The first run was slower due to image pull (“cold start”). Warm runs were quicker but still dominated by overhead at this input size.
- To see clear parallel gains, I would use a larger input (e.g., 50–100 MB) made by concatenating the same text multiple times before uploading to S3.
- S3 I/O and network can become the bottleneck when inputs are small.

1. What if a mapper fails? How would I recover?

- Retry the failed chunk. Outputs use clear S3 keys, so re-running is safe (overwrite with the same name). The reducer can wait/retry until all expected outputs exist.

2. How to scale to 10 or 100 mappers?

- Split into more chunks (e.g., 10 or 100).
- Launch more mappers in parallel on ECS.
- Keep S3 keys organized (job ID + chunk number) for easy listing.
- For many chunks, consider a tree-style reduce to avoid a single reducer bottleneck.

3. What was the challenging part of coordinating tasks manually?

- Setting public IPs and opening the right ports initially.
- Waiting for tasks to be RUNNING and ready to accept requests.
- Cold starts added variance to timing.

- Manually tracking chunk/result URLs is clumsy; a small script or workflow tool would help.