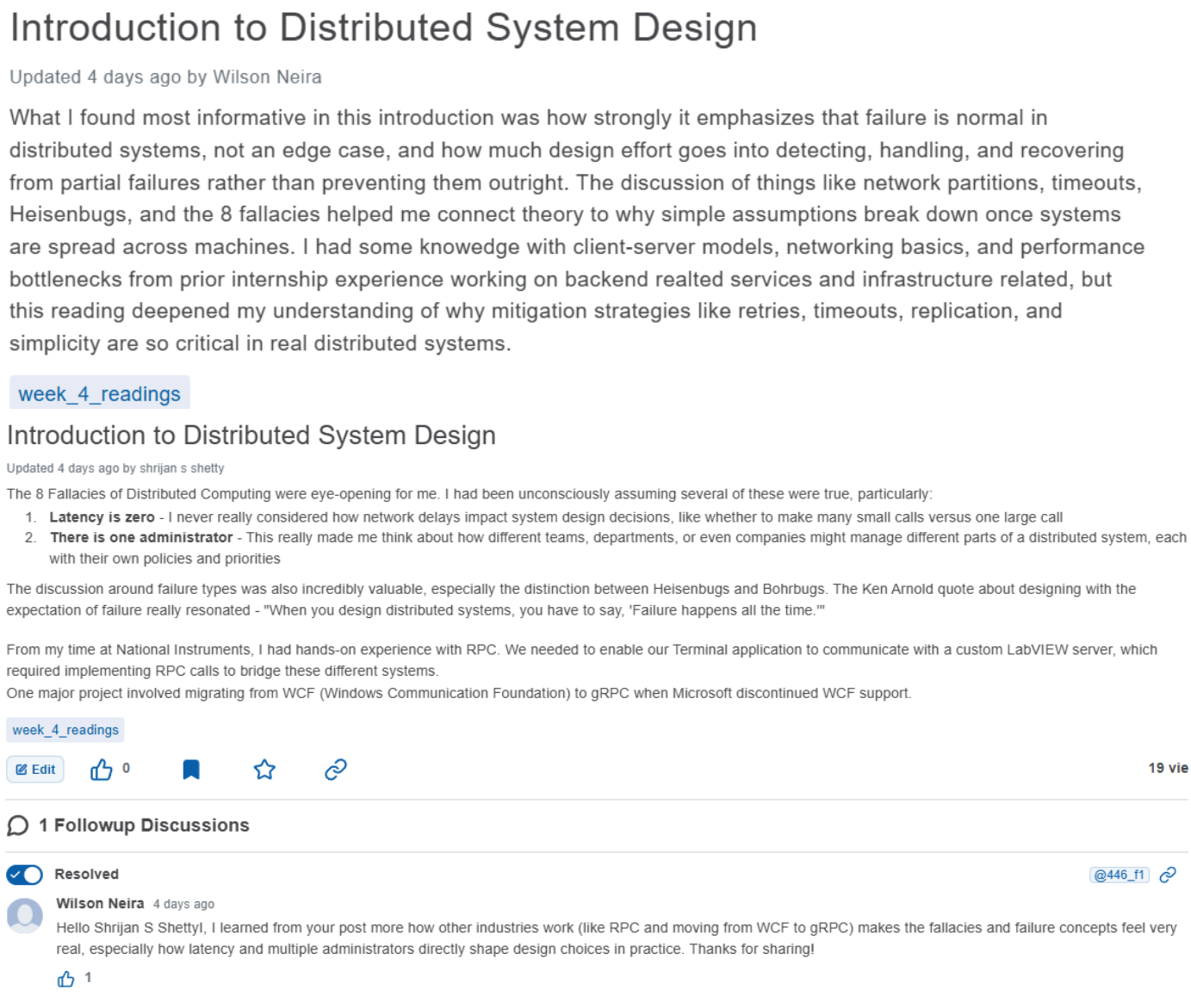
Homework 4 Report

Wilson Neira

Part I: Reading and Comments!



Introduction to Distributed System Design

Updated 4 days ago by Wilson Neira

What I found most informative in this introduction was how strongly it emphasizes that failure is normal in distributed systems, not an edge case, and how much design effort goes into detecting, handling, and recovering from partial failures rather than preventing them outright. The discussion of things like network partitions, timeouts, Heisenbugs, and the 8 fallacies helped me connect theory to why simple assumptions break down once systems are spread across machines. I had some knowedge with client-server models, networking basics, and performance bottlenecks from prior internship experience working on backend realted services and infrastructure related, but this reading deepened my understanding of why mitigation strategies like retries, timeouts, replication, and simplicity are so critical in real distributed systems.

week_4_readings

Introduction to Distributed System Design

Updated 4 days ago by shrijan s shetty

The 8 Fallacies of Distributed Computing were eye-opening for me. I had been unconsciously assuming several of these were true, particularly:

1. **Latency is zero** - I never really considered how network delays impact system design decisions, like whether to make many small calls versus one large call
2. **There is one administrator** - This really made me think about how different teams, departments, or even companies might manage different parts of a distributed system, each with their own policies and priorities

The discussion around failure types was also incredibly valuable, especially the distinction between Heisenbugs and Bohrbugs. The Ken Arnold quote about designing with the expectation of failure really resonated - "When you design distributed systems, you have to say, 'Failure happens all the time.'"

From my time at National Instruments, I had hands-on experience with RPC. We needed to enable our Terminal application to communicate with a custom LabVIEW server, which required implementing RPC calls to bridge these different systems.
One major project involved migrating from WCF (Windows Communication Foundation) to gRPC when Microsoft discontinued WCF support.

week_4_readings

Edit 0 19 vie

1 Followup Discussions

Resolved @446_f1

Wilson Neira 4 days ago

Hello Shrijan S Shetty!, I learned from your post more how other industries work (like RPC and moving from WCF to gRPC) makes the fallacies and failure concepts feel very real, especially how latency and multiple administrators directly shape design choices in practice. Thanks for sharing!

1

Part II: Infrastructure set up!

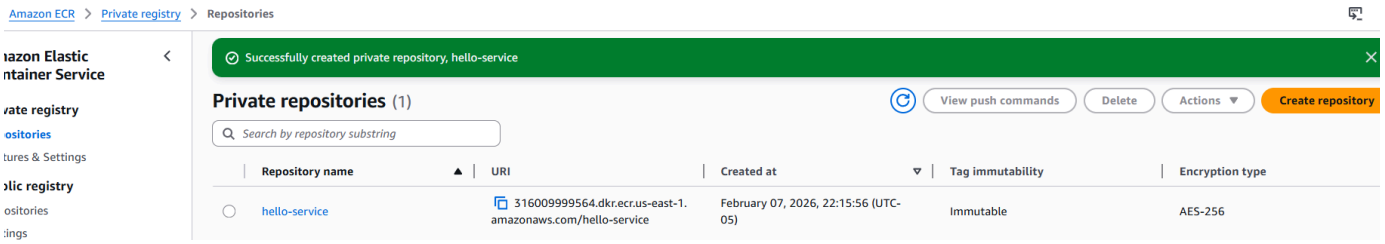
```
Owner@DESKTOP-5FCT03U MINGW64 ~/Documents/Audacity/Building-Scalable-Distributed-Systems (main)
$ ssh -i "/c/Users/Owner/Documents/Audacity/Northeastern University/CS 6650 Building Scalable Distributed Systems/Week 1 Whh
hhat is all of this!/web-service-gin.pem" ec2-user@ec2-54-234-94-111.compute-1.amazonaws.com

A newer release of "Amazon Linux" is available.
  Version 2023.10.20260120:
  Version 2023.10.20260202:
Run "/usr/bin/dnf check-release-update" for full release and version update info
   ,     #_
   ~\_  ####_
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   Amazon Linux 2023 (ECS Optimized)
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'

For documentation, visit http://aws.amazon.com/documentation/ecs
Last login: Mon Feb  2 12:43:35 2026 from 98.15.96.233
[ec2-user@ip-172-31-19-156 ~]$ docker images
REPOSITORY                 TAG            IMAGE ID       CREATED        SIZE
docker-gs-ping             latest         643981103e08   12 days ago    1.27GB
amazon/amazon-ecs-agent    latest         192298286f1f   3 weeks ago    120MB
ebs-csi-driver             latest         702835b18362   3 weeks ago    58.8MB
ecs-service-connect-agent  interface-v1   369a1f77771a   6 weeks ago    204MB
amazon/amazon-ecs-pause    0.1.0          9dd4685d3644   11 years ago   702kB
[ec2-user@ip-172-31-19-156 ~]$
```

Step 1: Configure

```
AwksAD5hC/83/r+iQsVVyjd4oxExLMUOOPE3VDE6dLCCqyAghsEAEaDDMxNjAwOTk5OTU2NCIMztkGvaCnoTbhlsrtKo8C5c+5aoODdIcLdl+tAhkRnM6YhXqQcs
iCF1Ev9iolU04JOjfzx8Px49KQ5+cLVeuMq/wE06lx7L6F9siR7WkhHXKI0cOAaUFIfAYpIKTOY8oMacEjJB11XBJY7RQc3iy2HAqjHclmE9ZJYpB6kTDEoPNPLg
c84JJaYBkbi/vz4k+fuFvCbooj5GuCaZjvwSXPTyNkLD09rzldSnoYyc7hkX/oS+lD564Pp8zArLqrZG814PPY+aUzAMa9bWzfftYpgNGkEVEVAJ403cn31EoUMw
h3BFFemwsKcjeS/4Xgbj4QAwFa81Ojbq7tpRl+e73TNSwRf1EnNCFKXTEmVO8y4cEsHOHS0YSik1zKQV1RpDD77J/MBjqeAUAJoGBuBXJMwADRIh+1CiNiSYr8pr
NHoNDHFSn21Yhba4/USoptTqg3Wl0MGZZNWxUgzBxxsHzkBDkfdUeUgUFgasqtb7Zxh5wJTMRDTlarV8TH2Uxsf3hsIzBmg2BOxFEV3UFxRfmb407Qk4bFncW5jy
2r1Ry3Bz7+pEG8tR2tzG7JNJeySnEXtqCe/5b76L/Eat6Dpk1V541J/WUu
Default region name [us-east-1]:
Default output format [json]:
PS C:\Users\Owner\Documents\Audacity\Building-Scalable-Distributed-Systems> ssh -i "C:/Users/Owner/Documents/Audacity/Northe
```

Step 2: Setup ECR

1. Navigate to Amazon ECR
2. Create Repository



3. Copy the Repository URI

316009999564.dkr.ecr.us-east-1.amazonaws.com/hello-service

Step 3: Push Docker Image to ECR

1. Get your ECR repository URL

```
Owner@DESKTOP-5FCT03U MINGW64 ~/Documents/Audacity/Building-Scalable-Distributed-Systems (main)
$ ECR_URL=$(aws ecr describe-repositories \
  --repository-names hello-service \
  --region us-east-1 \
  --query 'repositories[0].repositoryUri' \
  --output text)

Owner@DESKTOP-5FCT03U MINGW64 ~/Documents/Audacity/Building-Scalable-Distributed-Systems (main)
$ echo "ECR URL: $ECR_URL"
ECR URL: 316009999564.dkr.ecr.us-east-1.amazonaws.com/hello-service
```

2. Authenticate Docker to ECR

```
Owner@DESKTOP-5FCT03U MINGW64 ~/Documents/Audacity/Building-Scalable-Distributed-Systems (main)
$ ECR_BASE=$(echo $ECR_URL | cut -d'/' -f1)

Owner@DESKTOP-5FCT03U MINGW64 ~/Documents/Audacity/Building-Scalable-Distributed-Systems (main)
$ aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin $ECR_BASE
Login Succeeded

Owner@DESKTOP-5FCT03U MINGW64 ~/Documents/Audacity/Building-Scalable-Distributed-Systems (main)
$ password=$(aws ecr get-login-password --region us-east-1)

Owner@DESKTOP-5FCT03U MINGW64 ~/Documents/Audacity/Building-Scalable-Distributed-Systems (main)
$ echo "$password" | docker login --username AWS --password-stdin "$ECR_BASE"
Login Succeeded
```

3. Tag your image for ECR

```
[ec2-user@ip-172-31-19-156 ~]$ docker tag hello-service:latest "$ECR_URL:latest"
```

4. Build and push the image

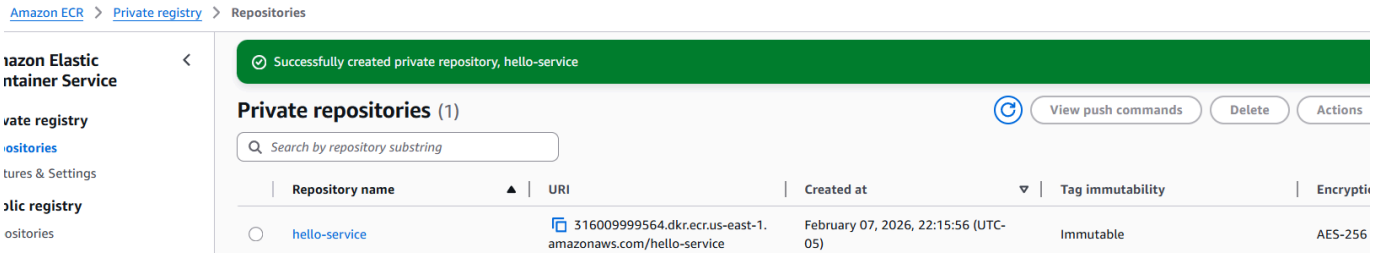
```
[ec2-user@ip-172-31-19-156 ~]$ docker push "$ECR_URL:latest"
The push refers to repository [316009999564.dkr.ecr.us-east-1.amazonaws.com/hello-service]
44d3a0ac86f9: Preparing
a1153bed92ba: Preparing
8fe368f16a1e: Preparing
93bf13679d88: Preparing
bca63f61669a: Preparing
5f70bf18a086: Preparing
53bf8f783f7e: Preparing
5a797796f075: Preparing
5296ce17e466: Preparing
d7d9aa728fbb: Preparing
1a0f650a8421: Preparing
da7213941eca: Preparing
no basic auth credentials
```

5. Verify upload

```
[ec2-user@ip-172-31-19-156 ~]$ aws ecr list-images \
  --repository-name hello-service \
  --region us-east-1 \
  --query 'imageIds[*].imageTag' \
  --output table
[ec2-user@ip-172-31-19-156 ~]$
```

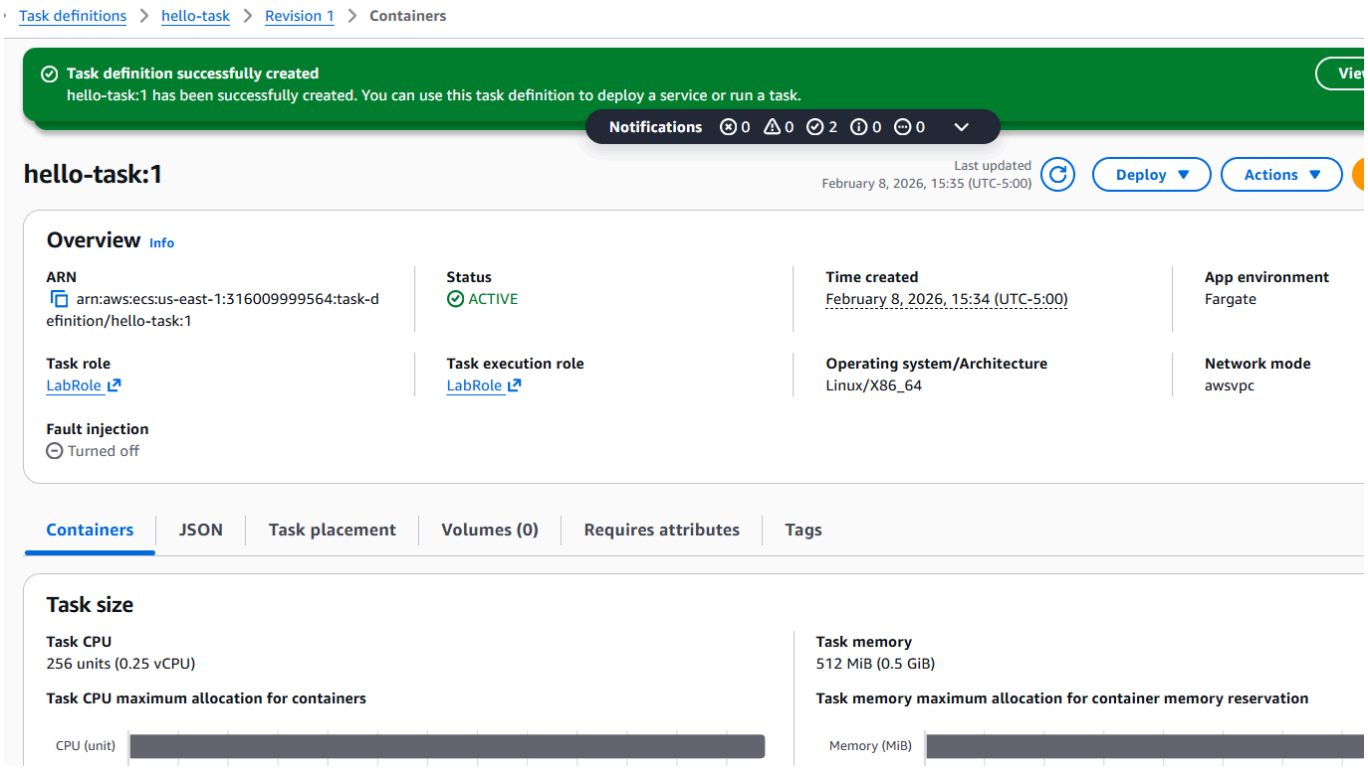
Step 4: Create ECS Cluster

1. Navigate to Amazon ECS
2. Create Cluster



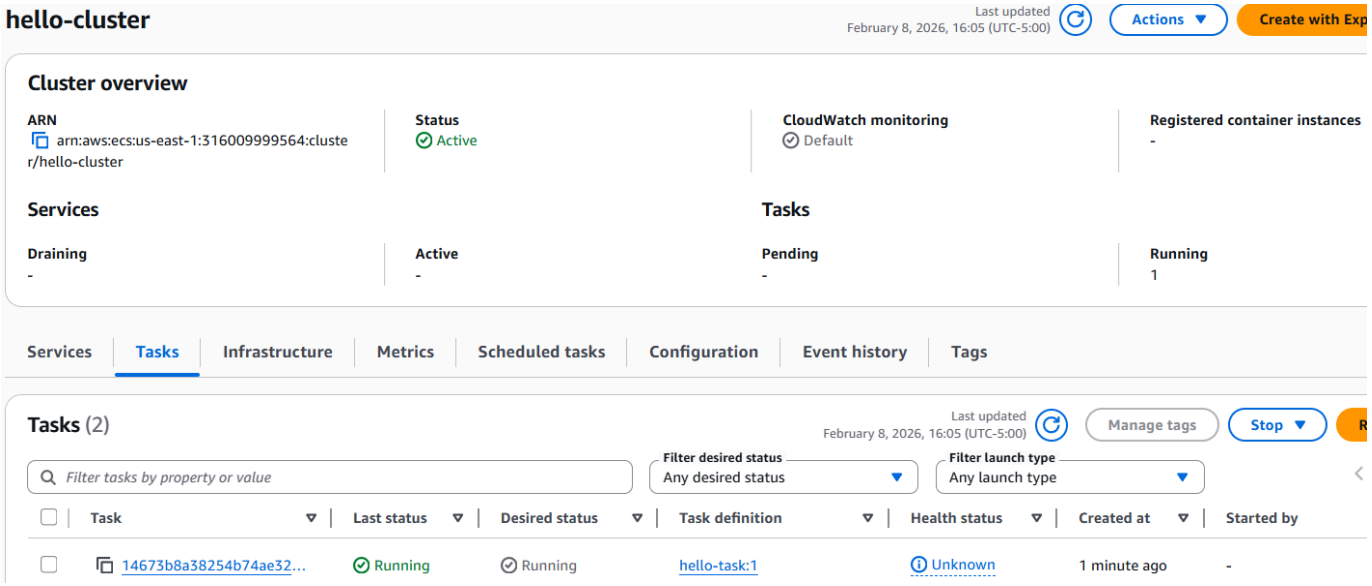
Step 5: Create Task Definition

1. Navigate to Task Definitions
2. Configure Task Definition
3. Add Container
4. Create the Task Definition



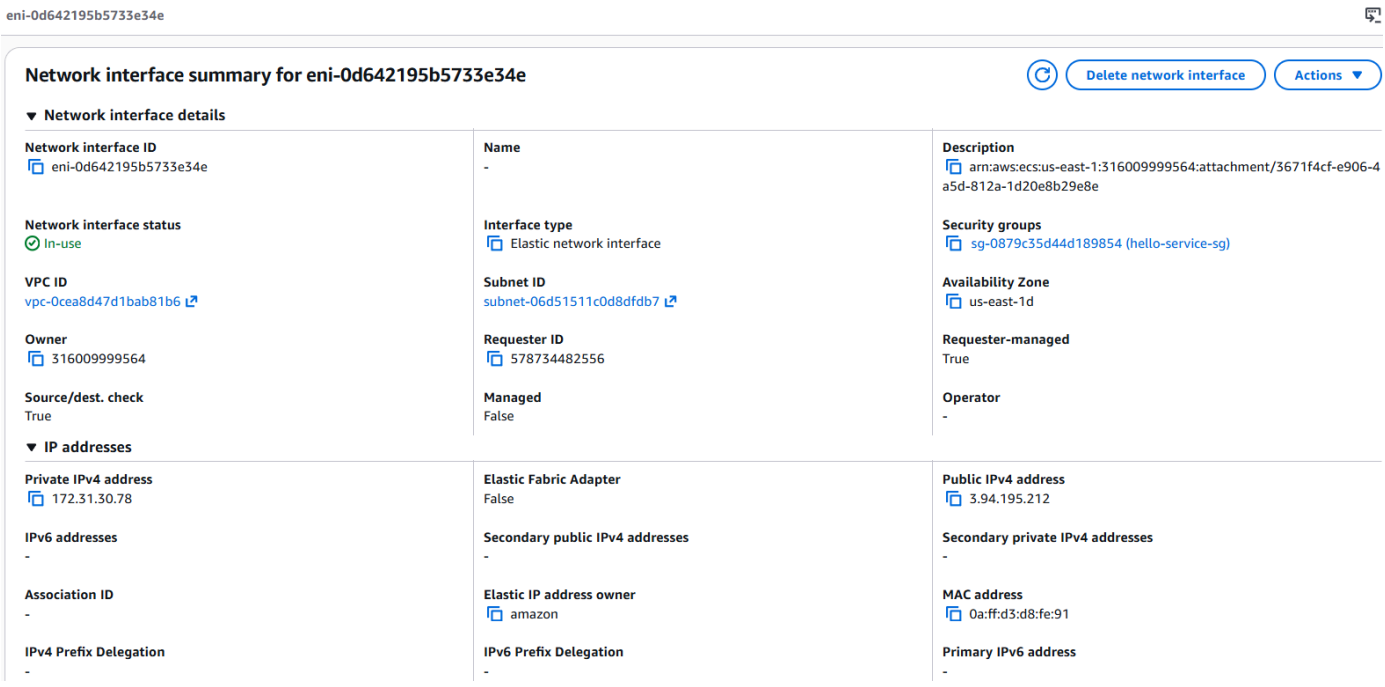
Step 6: Run a Task

1. Navigate to your Cluster
2. Run New Task
3. Configure Networking
4. Run the Task



Step 7: Getting the Public IP and Testing!

1. View Task Details



2. Find Public IP

3.94.195.212

3. Test Your Service

```
Owner@DESKTOP-5FCT03U MINGW64 ~/Documents/Audacity/Building-Scalable-Distributed-Systems (main)
$ # Get albums
curl http://3.94.195.212:8080/albums

# Get album with ID 1
curl http://3.94.195.212:8080/albums/1
[
    {
        "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
    }
]{
    "id": "1",
    "title": "Blue Train",
    "artist": "John Coltrane",
    "price": 56.99
}
Owner@DESKTOP-5FCT03U MINGW64 ~/Documents/Audacity/Building-Scalable-Distributed-Systems (main)
$
```

Thinking about your result…

Difference in choosing EC2 vs ECS.

EC2 is me managing an actual server/VM myself, while ECS (with Fargate) is me just deploying containers and AWS manages the servers for me.

What is a VPCLinks to an external site and subnetLinks to an external site.? How did you get access into the default VPC?

A VPC is like my own private network inside AWS and a subnet is a smaller IP range inside that network in one availability zone, and I got access because AWS automatically gives my account a default VPC in each region and I selected it when launching the task.

What is TCP? How is it different than UDP?

TCP is like a reliable confirmed delivery connection where packets arrive in order, while UDP is faster but doesn't guarantee delivery or order.

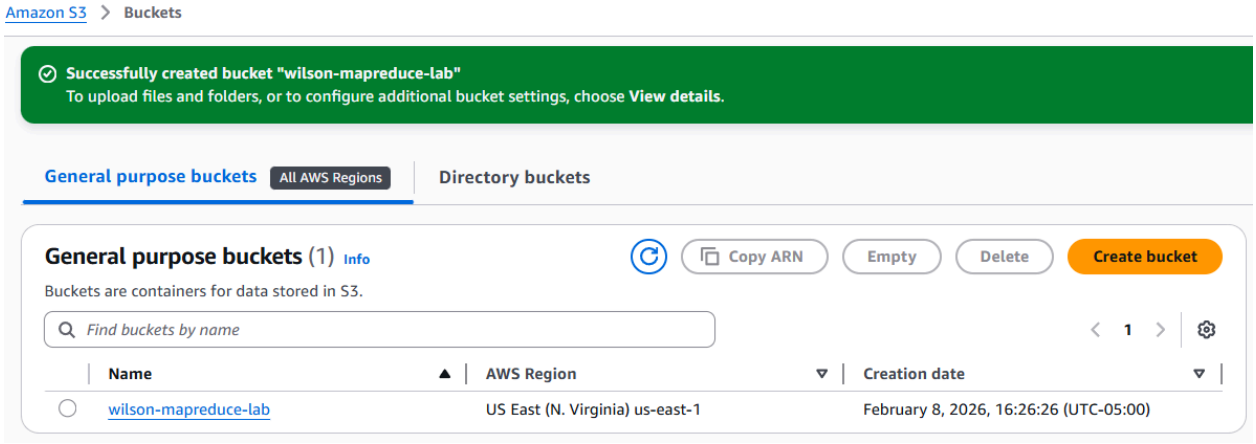
How do you control resources allocated to a task?

I control task resources by setting the CPU and memory values in the ECS Task Definition (and that limits what each running task/container can use).

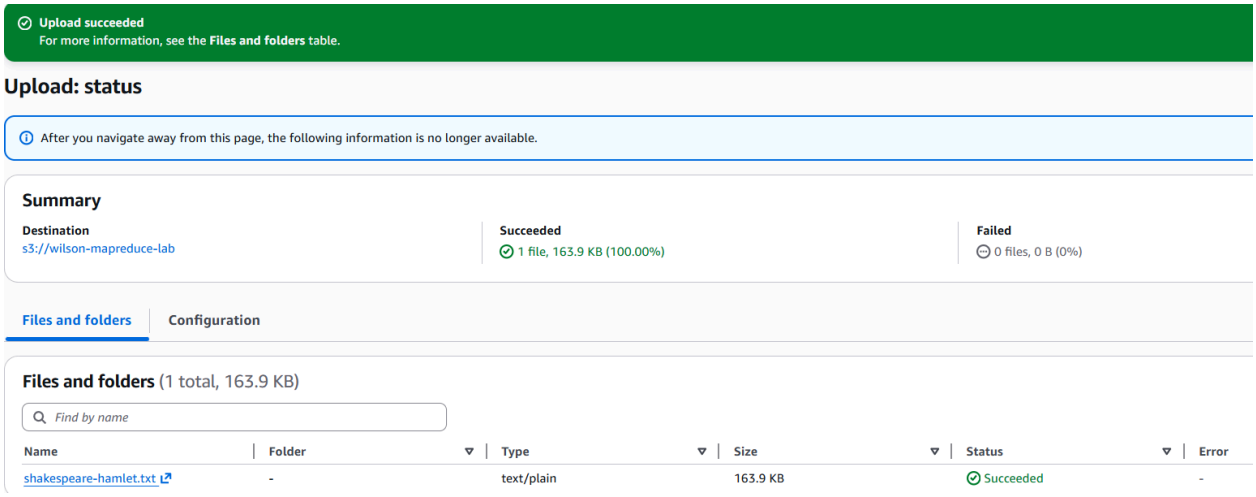
Part III: A Fun Experiment!

Implementation

1. Create S3



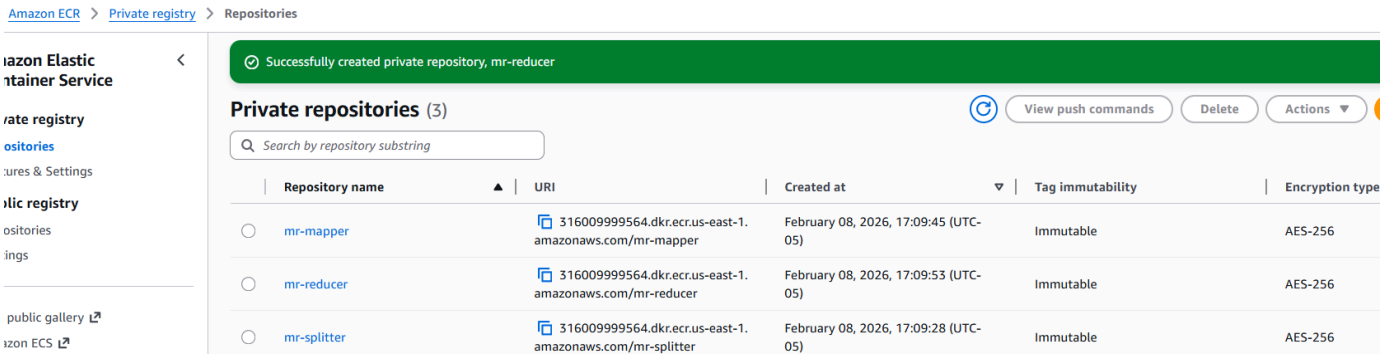
2. Upload file



3. Build image for each code mapper, reducer, and splitter

```
docker build -t reducer-service  --build-arg SERVICE=reducer .
View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/xo40frxvtlbr9l54ggultw8af
[+] Building 38.9s (15/15) FINISHED                                                                    docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                  0.0s
 => => transferring dockerfile: 469B                                                                                  0.0s
 => [internal] load metadata for docker.io/library/alpine:3.20                                                        0.4s
 => [internal] load metadata for docker.io/library/golang:1.22-alpine                                                 0.4s
 => [internal] load .dockerignore                                                                                     0.0s
 => => transferring context: 2B                                                                                       0.0s
 => [build 1/6] FROM docker.io/library/golang:1.22-alpine@sha256:1699c10032ca2582ec89a24a1312d986a3f094aed3d5c1147b19  0.1s
 => => resolve docker.io/library/golang:1.22-alpine@sha256:1699c10032ca2582ec89a24a1312d986a3f094aed3d5c1147b19880afe  0.1s
 => [stage-1 1/3] FROM docker.io/library/alpine:3.20@sha256:a4f4213abb84c497377b8544c81b3564f313746700372ec4fe84653e4  0.1s
 => => resolve docker.io/library/alpine:3.20@sha256:a4f4213abb84c497377b8544c81b3564f313746700372ec4fe84653e4fb03805   0.1s
 => [internal] load build context                                                                                     0.0s
 => => transferring context: 310B                                                                                     0.0s
 => CACHED [build 2/6] WORKDIR /app                                                                                   0.0s
 => CACHED [build 3/6] COPY go.mod go.sum ./                                                                          0.0s
 => CACHED [build 4/6] RUN go mod download                                                                            0.0s
 => CACHED [build 5/6] COPY . .                                                                                       0.0s
 => [build 6/6] RUN CGO_ENABLED=0 GOOS=linux GOARCH=amd64 go build -o /out/app ./reducer                             35.8s
 => CACHED [stage-1 2/3] WORKDIR /app                                                                                 0.0s
 => [stage-1 3/3] COPY --from=build /out/app /app/app                                                                 0.1s
 => exporting to image                                                                                                1.9s
 => => exporting layers                                                                                               1.4s
 => => exporting manifest sha256:3bba8412d9013aba3ac08deaa9c0b93aa704c1da052cd05819aa39011d12d386                     0.0s
 => => exporting config sha256:b950cef85314551a8595ea43ad3bb946dcd317a3712ca67352ded90434ac05be                       0.0s
 => => exporting attestation manifest sha256:3e2f27bde32e326159a1fc58080d3dad793c9a7347216c0f665d7a42d6f4fc44         0.1s
 => => exporting manifest list sha256:dae09b1629711781744ac8c9ceda2791f756da09c4a5f521b27756f867205bdd                0.0s
 => => naming to docker.io/library/reducer-service:latest                                                             0.0s
 => => unpacking to docker.io/library/reducer-service:latest                                                          0.3s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/kc2koj9vseo40lv9yh1lo74cq

Owner@DESKTOP-5FCT03U MINGW64 ~/Documents/Audacity/Building-Scalable-Distributed-Systems/HW4/mapreduce-lab (main)
$ ls
Dockerfile  go.mod  go.sum  mapper/  reducer/  shakespeare-hamlet.txt  splitter/
```

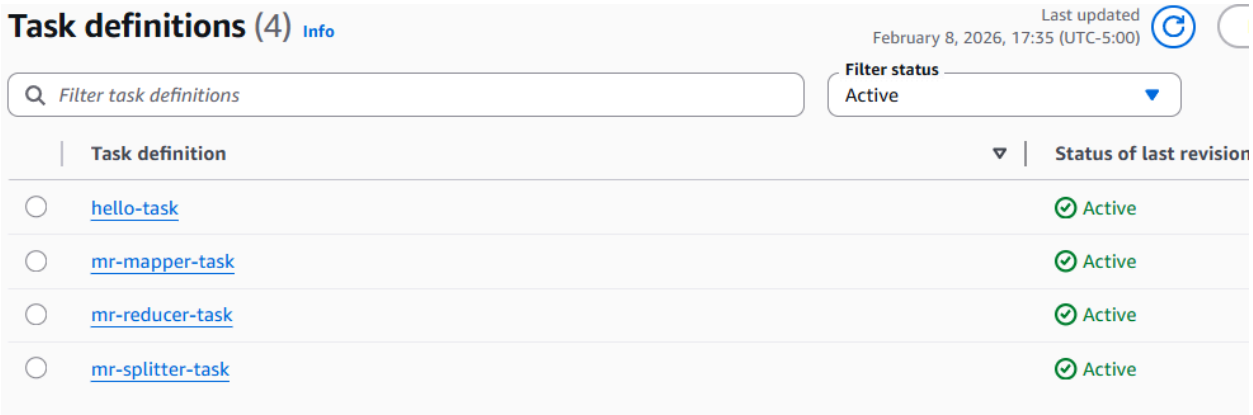
4. Create 3 ECR



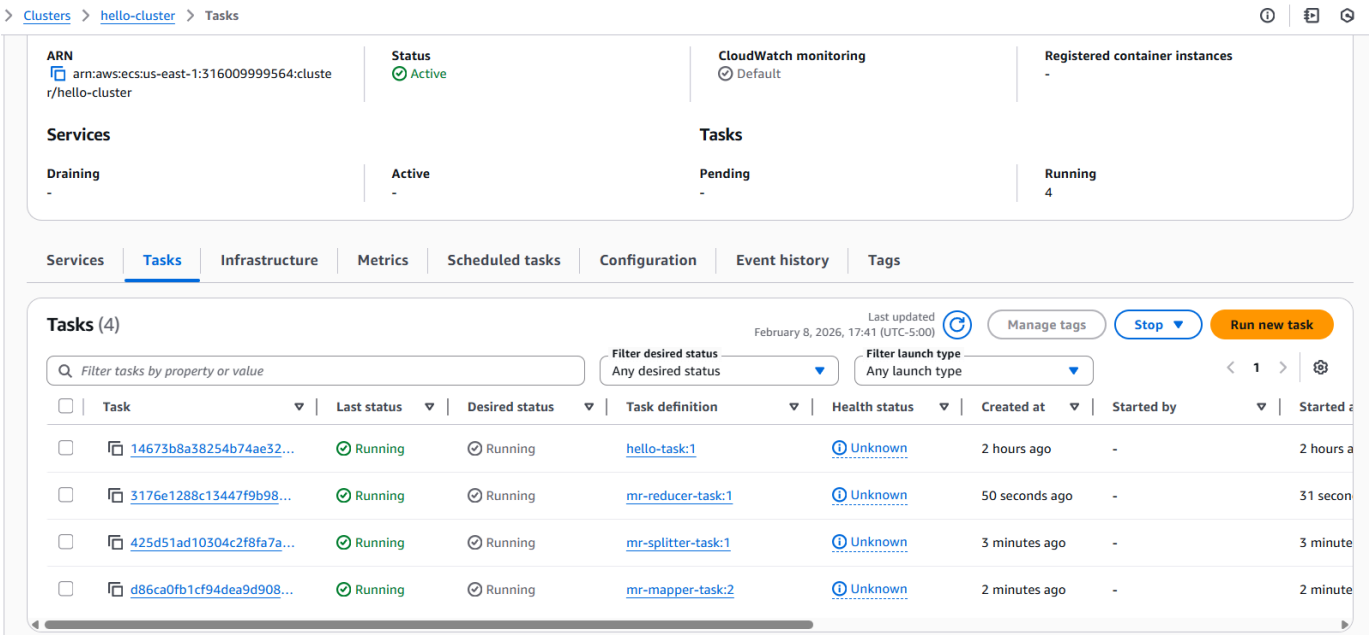
5. Tag + push each image

```
Owner@DESKTOP-5FCT03U MINGW64 ~/Documents/Audacity/Building-Scalable-Distributed-Systems/HW4/mapreduce-lab (main)
$ ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
BASE="316009999564.dkr.ecr.us-east-1.amazonaws.com"

docker tag splitter-service:latest $BASE/mr-splitter:latest
docker tag mapper-service:latest   $BASE/mr-mapper:latest
docker tag reducer-service:latest  $BASE/mr-reducer:latest

docker push $BASE/mr-splitter:latest
docker push $BASE/mr-mapper:latest
docker push $BASE/mr-reducer:latest
The push refers to repository [316009999564.dkr.ecr.us-east-1.amazonaws.com/mr-splitter]
1f1feb02db15: Pushed
811d8fb87323: Pushed
fc332be8cea2: Pushed
76eb174b37c3: Pushed
latest: digest: sha256:9bba91c6d79cf92555dfcba09a757f57b2bb2e216fd66c895bb2f464d1da5069 size: 855
The push refers to repository [316009999564.dkr.ecr.us-east-1.amazonaws.com/mr-mapper]
1f1feb02db15: Pushed
66239dc487ad: Pushed
52e27d47c2ba: Pushed
76eb174b37c3: Pushed
latest: digest: sha256:282ce5f9f5d45aa0316ef5c90f3f5a9bba502726bb12761f6cb7e9d10ed8aca3 size: 855
The push refers to repository [316009999564.dkr.ecr.us-east-1.amazonaws.com/mr-reducer]
df6c8daab628: Pushed
e9c7f95ffbb7: Pushed
1f1feb02db15: Pushed
76eb174b37c3: Pushed
latest: digest: sha256:dae09b1629711781744ac8c9ceda2791f756da09c4a5f521b27756f867205bdd size: 855

Owner@DESKTOP-5FCT03U MINGW64 ~/Documents/Audacity/Building-Scalable-Distributed-Systems/HW4/mapreduce-lab (main)
```

6. Create task definitions (3 total)

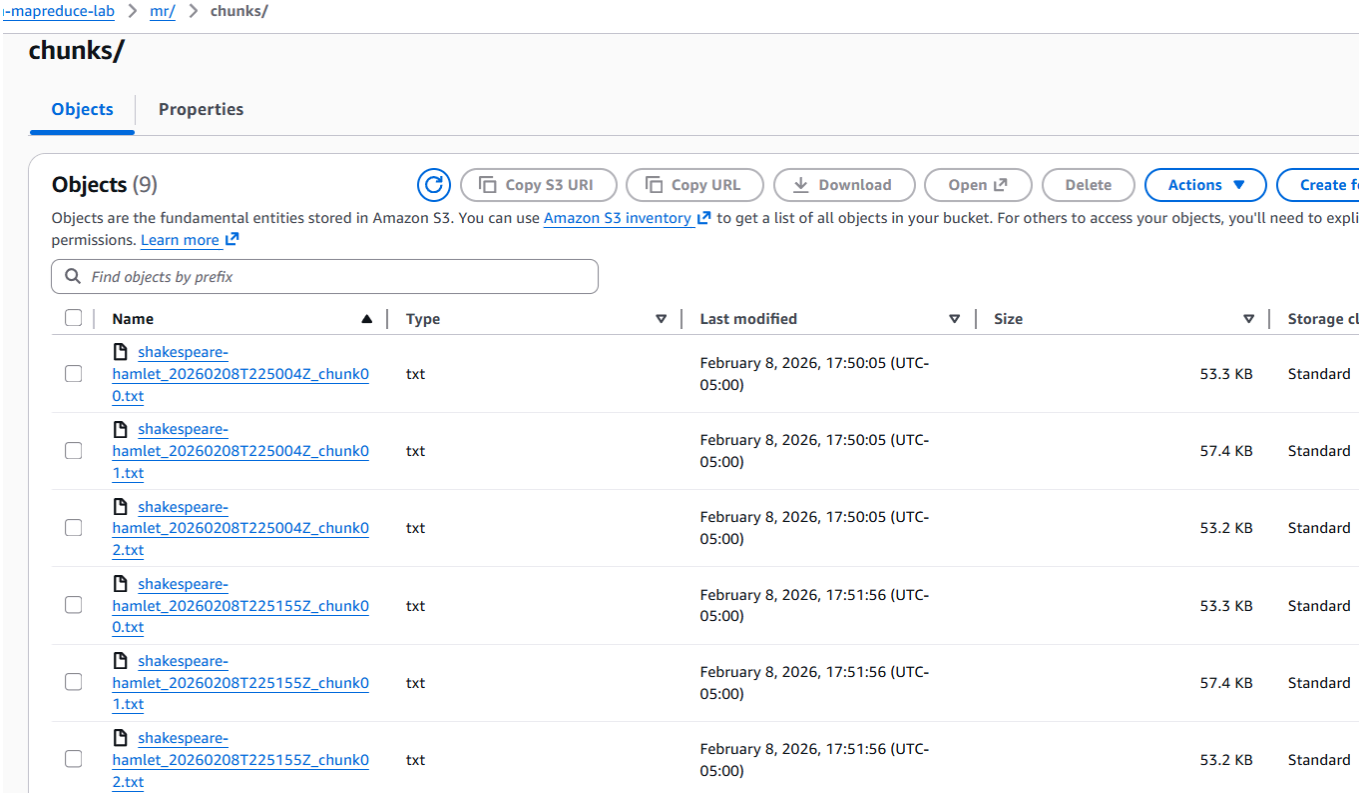


7. Run Tasks



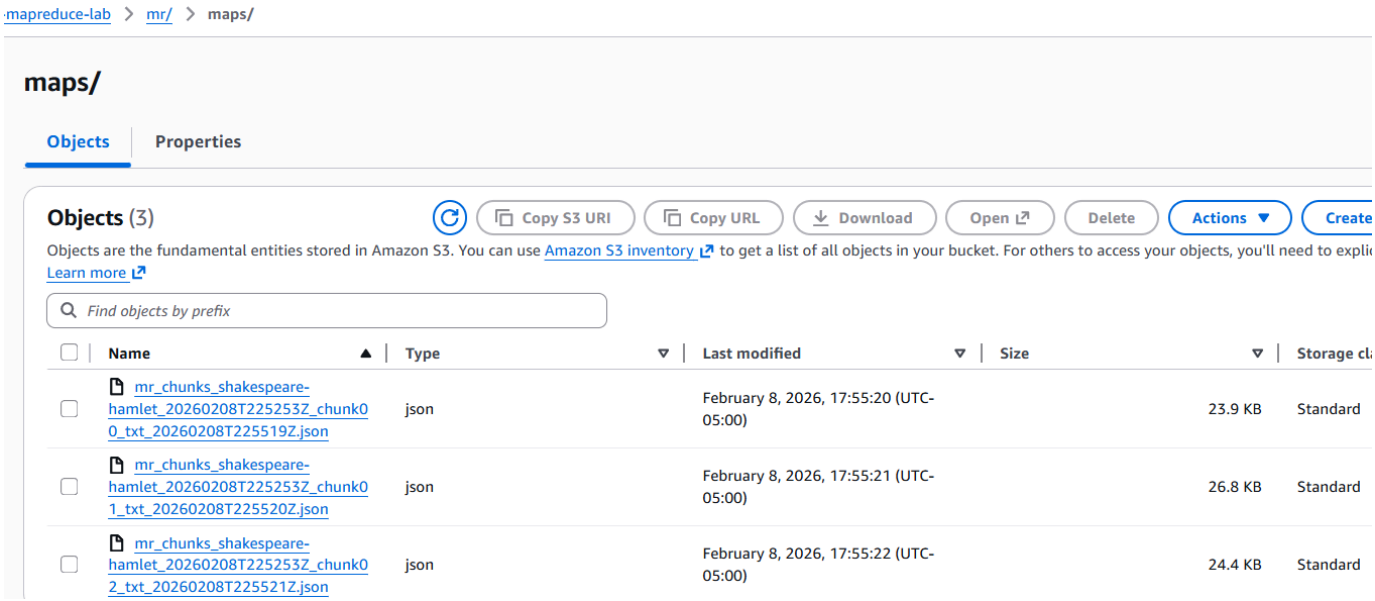
8. Split

```
Owner@DESKTOP-5FCT03U MINGW64 ~/Documents/Audacity/Building-Scalable-Distributed-Systems/HW4/mapreduce-lab (main)
$ curl "http://3.238.198.242:8080/split?s3=s3://wilson-mapreduce-lab/shakespeare-hamlet.txt"
{"chunks":["s3://wilson-mapreduce-lab/mr/chunks/shakespeare-hamlet_20260208T225253Z_chunk00.txt","s3://wilson-mapreduce-lab/mr/chunks/shakespeare-hamlet_20260208T225253Z_chunk01.txt","s3://wilson-mapreduce-lab/mr/chunks/shakespeare-hamlet_20260208T225253Z_chunk02.txt"]}
```



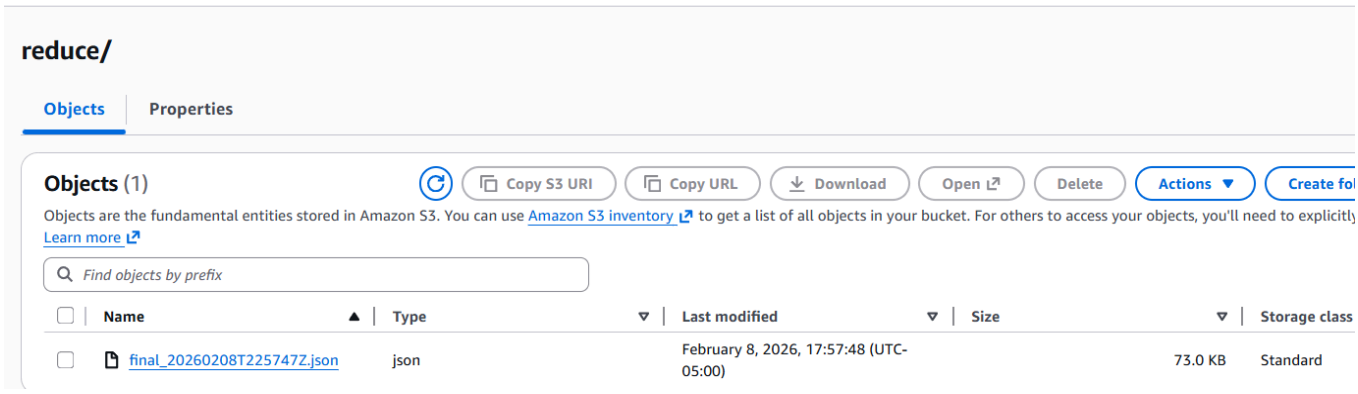
9. Map

```
Owner@DESKTOP-5FCT03U MINGW64 ~/Documents/Audacity/Building-Scalable-Distributed-Systems/HW4/mapreduce-lab (main)
$ curl "http://54.237.79.199:8080/map?s3=s3://wilson-mapreduce-lab/mr/chunks/shakespeare-hamlet_20260208T225253Z_chunk00.txt
"
curl "http://54.237.79.199:8080/map?s3=s3://wilson-mapreduce-lab/mr/chunks/shakespeare-hamlet_20260208T225253Z_chunk01.txt"
curl "http://54.237.79.199:8080/map?s3=s3://wilson-mapreduce-lab/mr/chunks/shakespeare-hamlet_20260208T225253Z_chunk02.txt"
{"out":"s3://wilson-mapreduce-lab/mr/maps/mr_chunks_shakespeare-hamlet_20260208T225253Z_chunk00_txt_20260208T225519Z.json"}
{"out":"s3://wilson-mapreduce-lab/mr/maps/mr_chunks_shakespeare-hamlet_20260208T225253Z_chunk01_txt_20260208T225520Z.json"}
{"out":"s3://wilson-mapreduce-lab/mr/maps/mr_chunks_shakespeare-hamlet_20260208T225253Z_chunk02_txt_20260208T225521Z.json"}
```



10. Reduce

```
Owner@DESKTOP-5FCT03U MINGW64 ~/Documents/Audacity/Building-Scalable-Distributed-Systems/HW4/mapreduce-lab (main)
$ curl "http://50.19.146.108:8080/reduce?in=s3://wilson-mapreduce-lab/mr/maps/mr_chunks_shakespeare-hamlet_20260208T225253Z_
chunk00_txt_20260208T225519Z.json&in=s3://wilson-mapreduce-lab/mr/maps/mr_chunks_shakespeare-hamlet_20260208T225253Z_chunk01
_txt_20260208T225520Z.json&in=s3://wilson-mapreduce-lab/mr/maps/mr_chunks_shakespeare-hamlet_20260208T225253Z_chunk02_txt_20
260208T225521Z.json"
{"out":"s3://wilson-mapreduce-lab/mr/reduce/final_20260208T225747Z.json","files":3}
```

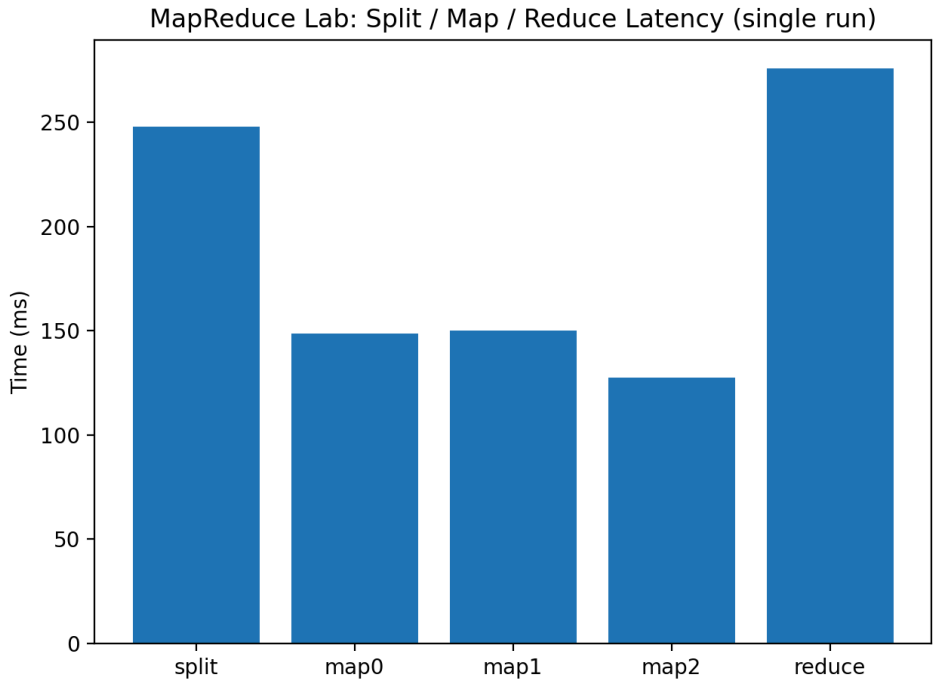


11. Verify the final JSON produced by reducer and it matches with the real results!

```
PS C:\Users\Owner\Documents\Audacity\Building-Scalable-Distributed-Systems\HW4\mapreduce-lab> & C:/Users/Owner/AppData/Local
/Programs/Python/Python310/python.exe c:/Users/Owner/Documents/Audacity/Building-Scalable-Distributed-Systems/HW4/mapreduce-
lab/verify_json.py
Total unique words truth: 4815
Total unique words reduced: 4817
Total mismatched words: 0
PASS
```

12. Performance Plots

```
Owner@DESKTOP-5FCT03U MINGW64 ~/Documents/Audacity/Building-Scalable-Distributed-Systems/HW4/mapreduce-lab (main)
$ curl -s -o /dev/null -w "split ms=%{time_total}\n" "http://3.238.198.242:8080/split?s3=s3://wilson-mapreduce-lab/shakespea
re-hamlet.txt"
split ms=0.247902

Owner@DESKTOP-5FCT03U MINGW64 ~/Documents/Audacity/Building-Scalable-Distributed-Systems/HW4/mapreduce-lab (main)
$ curl -s -o /dev/null -w "map0 ms=%{time_total}\n" "http://54.237.79.199:8080/map?s3=s3://wilson-mapreduce-lab/mr/chunks/sh
akespeare-hamlet_20260208T225253Z_chunk00.txt"
curl -s -o /dev/null -w "map1 ms=%{time_total}\n" "http://54.237.79.199:8080/map?s3=s3://wilson-mapreduce-lab/mr/chunks/shak
espeare-hamlet_20260208T225253Z_chunk01.txt"
curl -s -o /dev/null -w "map2 ms=%{time_total}\n" "http://54.237.79.199:8080/map?s3=s3://wilson-mapreduce-lab/mr/chunks/shak
espeare-hamlet_20260208T225253Z_chunk02.txt"
map0 ms=0.148496
map1 ms=0.150162
map2 ms=0.127319

Owner@DESKTOP-5FCT03U MINGW64 ~/Documents/Audacity/Building-Scalable-Distributed-Systems/HW4/mapreduce-lab (main)
$ curl -s -o /dev/null -w "reduce ms=%{time_total}\n" "http://50.19.146.108:8080/reduce?in=s3://wilson-mapreduce-lab/mr/maps
/mr_chunks_shakespeare-hamlet_20260208T225253Z_chunk00_txt_20260208T225519Z.json&in=s3://wilson-mapreduce-lab/mr/maps/mr_chu
nks_shakespeare-hamlet_20260208T225253Z_chunk01_txt_20260208T225520Z.json&in=s3://wilson-mapreduce-lab/mr/maps/mr_chunks_sha
kespeare-hamlet_20260208T225253Z_chunk02_txt_20260208T225521Z.json"
reduce ms=0.275870
```



This shows that splitting and reducing take the most time, while the three map tasks each run faster and at about the same speed, which proves the work is being divided and processed in parallel.

Results

What happen if one of the mapper failed? How would you recover?

The final reduce would be missing that mapper's output, so I'd re-run that mapper on the same chunk until it successfully writes its JSON to S3, then run reduce again with all the outputs.

How can you scale this system into 10 or 100 mappers?

I'd split the file into 10/100 chunks and run 10/100 mapper tasks in parallel, then have the reducer read all the mapper output URLs from S3 (or a list the splitter writes) instead of hardcoding just three.

What was the challenging part of coordinating tasks manually?

Keeping track of all the moving pieces (IPs, S3 paths, correct query params, and the right order of split -> map -> reduce) without making a tiny typo that breaks everything.

What to hand in and when!

Did you get a speed up? How much?!

Yes, I did get a speed up because the three mapper tasks ran in parallel instead of one after the other, so the total map phase time was about the time of the slowest mapper (~0.15s) instead of the sum of all three (~0.43s), which gives roughly a 3x speedup compared to running them sequentially.