Team Project Phase 2 Report

Summary

Team name:
Members:

Name	Andrew ID
Zewei Yu	zeweiy
Carmel Prosper Sagbo	csagbo
Jeremy Lou	jeremylo

Please color your responses as red and upload the report in PDF format.

Live Test Performance & Configurations

Number and types of instances:
Cost per hour of the entire system:
(Cost includes on-demand EC2, EBS, and ELB costs. Ignore S3, Network, and Disk I/O costs)

Your team's overall rank on the live test scoreboard:
Live test submission details:

	Blockchain	QR Code	Twitter	Mixed Test (Blockchain)	Mixed Test (QR Code)	Mixed Test (Twitter)
score	20	10.475389	0	10	5.3	0
submission id	1581524	1581537	1581550	1581564	1581564	1581564
throughput	61213.664022	48012.881092	254779.098683	17761.096	9942.12719	109984.31
latency	24.602	27.799	2.4	18.443	73.95	0.473
correctness	91.50188846922906	100	0	91.499	100.0	0
error	0.0007795716082488767	0.0014197	100	9.4365	0.00204	100

Rubric

- Each unanswered bullet point = -4%
- Each unsatisfactory answer = -2%
- Optional Questions = +4%

Guidelines

- Use the report as a record of your progress, and then condense it before submitting it.
- When documenting an observation, we expect you to also provide an explanation for it.
- Questions ending with "Why?" require evidence, not just reasoning.
- It is essential to express ideas in your own words, through paraphrasing. Please note that we will be checking for instances of plagiarism.

Task 1: Improvements of Microservices

Question 1: Cluster architecture

You are allowed to use several types of EC2 instances, and you are free to distribute the workload across your cluster in any way you'd like (for example, hosting MySQL only on some worker nodes, or having worker nodes of different sizes). When you try to improve the performance of your microservices, it might be helpful to think of how you make use of the available resources within the provided budget.

- List at least two reasonable varieties of cluster architectures. (e.g., draw schematics to show which pieces are involved in serving a request).

  - **Option 1: Dedicated Database Node with Shared Microservice Nodes**

    This first option is to have a dedicated node for mysql to host the twitter database backed with the EBS volume snapshot and all the other services sharing the other nodes.

    1. **Master Node**:
       - Controls all worker nodes in the cluster.
    2. **Dedicated Database Node**:
       - A single worker node that hosts only the MySQL database (Twitter database), backed by an EBS volume for persistent storage.
    3. **Microservice Nodes**:
       - Multiple worker nodes that run the three microservices:
         - **Twitter Service** (connects to MySQL database)
         - **QRCode Service** (with custom pods)

        - **Blockchain Service**
    - Each microservice pod is connected to an **Ingress Controller** backed by an **Application Load Balancer (ALB)** to manage incoming requests and direct traffic to the appropriate service.

- **Option 2: Distributed Database and Service Nodes**

  Second Option is to have some number of instances dedicated to host both the twitter Database and the twitter service while the other nodes host the other microservices.

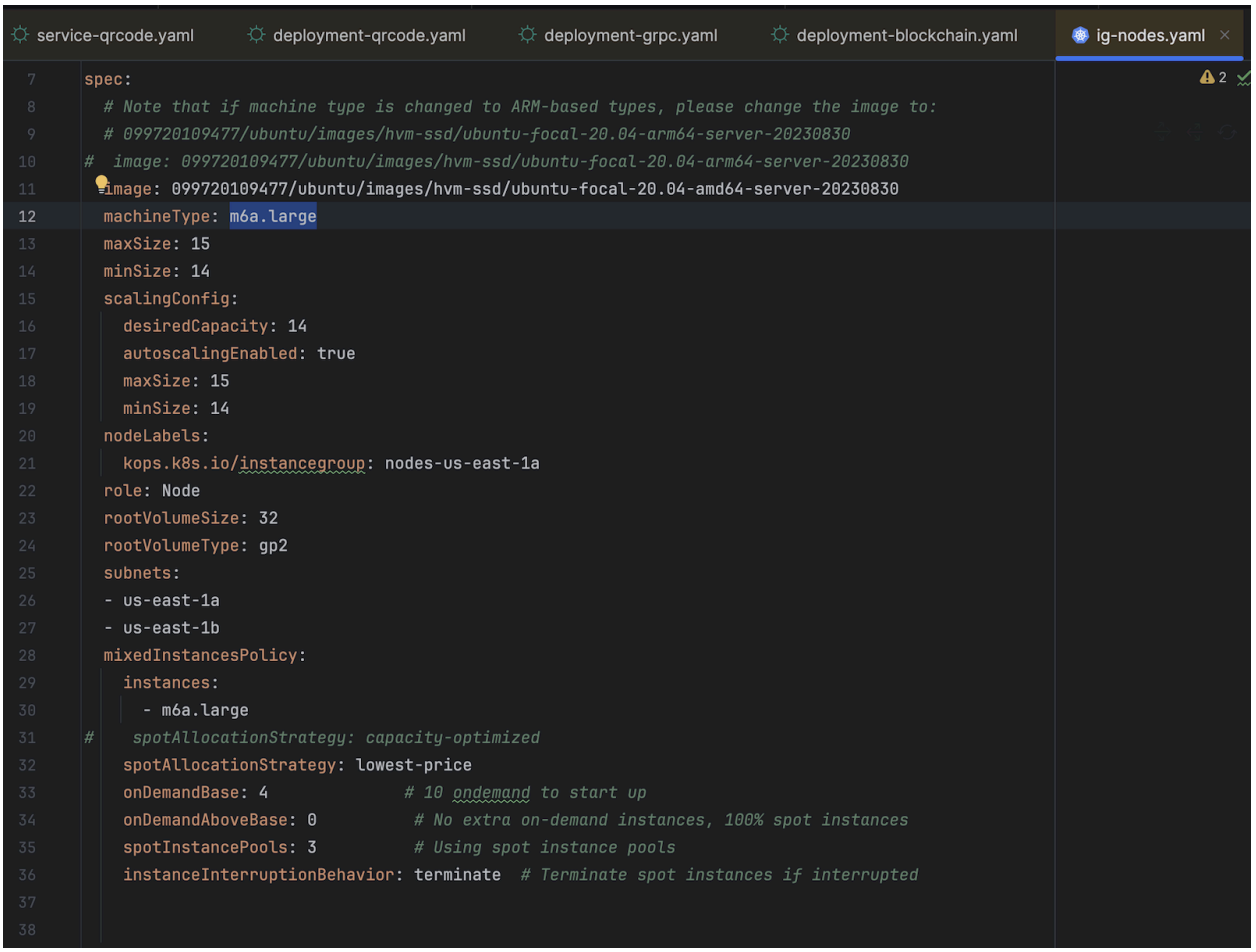
  1. **Master Node**:
     - Controls all worker nodes in the cluster.
  2. **Combined Database and Twitter Service Nodes**:
     - Several worker nodes each hosting:
       - **MySQL Database (Twitter Database)** with an EBS volume
       - **Twitter Service**
     - This co-location reduces latency for database queries as both the service and database are on the same node.
  3. **Microservice Nodes**:
     - Separate nodes run the other services:
       - **QRCode Service** (with custom pods)
       - **Blockchain Service**
     - These nodes are also connected to the **Ingress Controller** backed by an **Application Load Balancer (ALB)**

- Discuss how your design choices mentioned above affect performance, and why.
  - The first gives a dedicated resource for performance isolation to the mysql database allowing for efficient resource allocation on the other microservices. In that particular case the failure on the database side can be quickly fixed if there are at least two nodes hosting the database as we are only performing read operations on it. This allows for the twitter service to continue serving users while getting the failed node back. However there might be some latency in the communication between the database and the twitter microservice.
  - The second option will resolve the latency issue , but a failure to this service node will also create failure to respond to some query on the twitter microservice, leading to low throughput or poor performance.

- Describe your final choice of instances and architecture for your microservices. If it is different from what you had in Phase 1, describe whether the performance improved and why.

  Our final choice of the instances is:

  m6a.large



Question 2: Database and schema

If you want to make Twitter Service faster, the first thing to look at is the database because the amount of computation at the web tier is not as much as in Blockchain Service and QR Code Service. As you optimize your Twitter Service, you will find out that various schemas can result in a very large difference in performance! Also, you might want to look at different metrics to understand the bottleneck and think of what to optimize.

- What schema did you use for Twitter in Phase 1?

  Three tables are used, which is User, Interactions, Tweets.
  The user stores user's name and description, indexed by user id.
  The interactions stores interactions between users.
  The tweets stores tweet's information, indexed by id.

Did you change it in Phase 2?
Didn't change it due to the time limit.

If so, please discuss how and why you changed it, and how did the new schema affect performance?
In phase 3, the user information and the interactions can be combined into a single table to reduce the query number for each request.

- Compare any two schemas for Twitter. Try to explain why one schema is more performant than another.
  If combining the User information into the interaction table, the twitter service then can directly get the required information without several additional requests to user table.

- Explain briefly the theory behind at least 3 performance optimization techniques for databases in general. How are each of these optimizations implemented in your storage tier?
  Indexing: index each table by id, which can make retrieval speed faster.
  Reduce table: try to combine information as much as possible into a single table so that it can reduce the query number.
  Caching: for some row in the table, they are been searched a lot, so cache them can make the retrieval much faster.

Question 3: Your ETL process

It's highly likely that you need to re-run your ETL process in Phase 2 each time you implement a new database schema. You might even find it necessary to re-design your ETL pipeline if your previous one is insufficient for your needs. For the following bullet points, please briefly explain:

- The programming model/framework used for the ETL job and justification
  Pyspark is used for the ETL job.

- The number and type of instances used during ETL and justification
  Using 5 Standard_E4ads_V5 as worker node in Azure HDInsight

- The execution time for the ETL process, excluding database load time
  3 hours

- The time spent loading your data into the database
  Depending on the database requirements. If need to unhex the hexed string, then it take more than 2 hours. If just loading data from tsv, then it takes .5 hours.

- The overall cost of the ETL process
  $ 30 dollars

- The number of incomplete ETL runs before your final run
  1

- The size of the resulting database and reasoning
  The dumped database has a size of 27G, because it has string in hex representation. It will increase the string size a lot, make the database larger.

- The size and format of the backup
  The back up of the database is mysql dump in .sql format. It is 27G.

- Discuss the difficulties encountered
  Encoding and decoding data in hex format is very difficult. We are considering changing it to base64 or find other ways to solve the \t and \n text problem.

- If you had multiple database schema iterations, did you have to rerun your whole ETL pipeline between schema iterations?
  The ETL pipeline is relatively stable. Firstly, I tried to unnest all the tweets, then I found that only unnest the most and secondary outermost layer is enough.

- How did you load data into your database? Are there other ways to do this, and what are the advantages of each method? Did you try any of these alternate methods of loading?
  Using TSV files stored in the s3 bucket.

- Did you store any intermediate ETL results and why? If so, what data did you store and where did you store the data?
  No, we use a entire ETL process.

- We would like you to produce a histogram of hashtag frequency on a **log** scale among all the valid tweets **without hashtag filtering**. Given this histogram, describe why we asked you to filter out the top hashtags when calculating the hashtag score. [Clarification: We simply need you to plot the hashtag frequencies, sorted on the x-axis by the frequency value on the y-axis]
- Before you run your Spark tasks in a cluster, it would be a good idea to apply some tests. You may choose to test the filter rules, or even test the entire process against a small dataset. You may choose to test against the mini dataset or design your own dataset that contains interesting edge cases. **Please include a screenshot of the ETL test coverage report here.** (We don't have a strict coverage percentage you need to reach, but we do want to see your efforts in ETL testing.)
  We have ETL test locally on the mini dataset. The twitter service is hosting locally on the mini set. Then we have python script to compare the result from the local service with the reference server.

```python
import requests
import re
import difflib

# Define the endpoints
local_url = "http://localhost:8080/twitter"
reference_url = "http://reference.sailplatform.org/mini/twitter"

# Function to send request and get response
def get_response(url, params):
    try:
        response = requests.get(url, params=params)
        return response.text[response.text.find("\n")+1:]
    except requests.exceptions.RequestException as e:
        print(f"Error fetching from {url}: {e}")
        return None

def parse_trace_file(file_path):
    parameters = []
    with open(file_path, 'r') as file:
        for line in file:
            match = re.search(r'user_id=\d+&type=\w+&phrase=[^&]+&hashtag=\w+', line)
            if match:
                parameters.append(match.group())
    return parameters

def run_test(file_path):
    parameter_list = parse_trace_file(file_path)
    nomatch=0
    for params in parameter_list:

        # Convert the concatenated parameters into a dictionary
        params_dict = dict(param.split('=') for param in params.split('&'))

        # Fetch responses from both endpoints
        local_response = get_response(local_url, params_dict)
        reference_response = get_response(reference_url, params_dict)

        print(f"\nParameters: {params}")
        if local_response==reference_response:
            print("match")
        else:
            nomatch+=1
            print("Localhost Response:")
            print(local_response)
            print("Reference Response:")
            print(reference_response)

    print(nomatch)
    print(len(parameter_list))

trace_file_path = "trace8-mini"
run_test(trace_file_path)

```

The trace file comes from the previous feedback of failed twitter submission.

Note: Don't forget to put your code for ETL testing under the **ETL** folder in your team's GitHub repo **master** branch.

Question 4: Optimizations and Tweaks

Frameworks and databases do not come in the best shape out-of-the-box. Each of them has tens of parameters to tune. Once you are satisfied with the schema, you can start learning about the configuration parameters to see which ones might be most relevant and gather data and evidence with each individual optimization. You have probably tried a few in Phase 1. To perform even better, you probably want to continue with your experimentation.

Hint: A good schema will easily double or even triple the target RPS of your Twitter Service without any parameter tuning. It is advised to first focus on improving your schema and get an okay performance with your best cluster configuration. If you are 10 times (an order of magnitude) away from the target, then it is very unlikely to make up the difference with parameter tunings.

- List as many possible items to configure or tweak that might impact performance, in terms of web framework, your program, DBs, DB connectors, OS, etc.
- Apply them one at a time and show a table with the microservice you are optimizing, the optimization you have applied and the RPS before and after applying it.
- For every configuration parameter or tweak, try to explain how it contributes to performance.

Task 2: Development and Operations

Question 5: Web-tier orchestration development

We know it takes dozens or hundreds of iterations to build a satisfactory system, and the deployment process takes a long time to complete. We asked about the automated deployment process in the Phase 1 Final Report. Answer the following questions to let us know how your automation has changed in Phase 2.

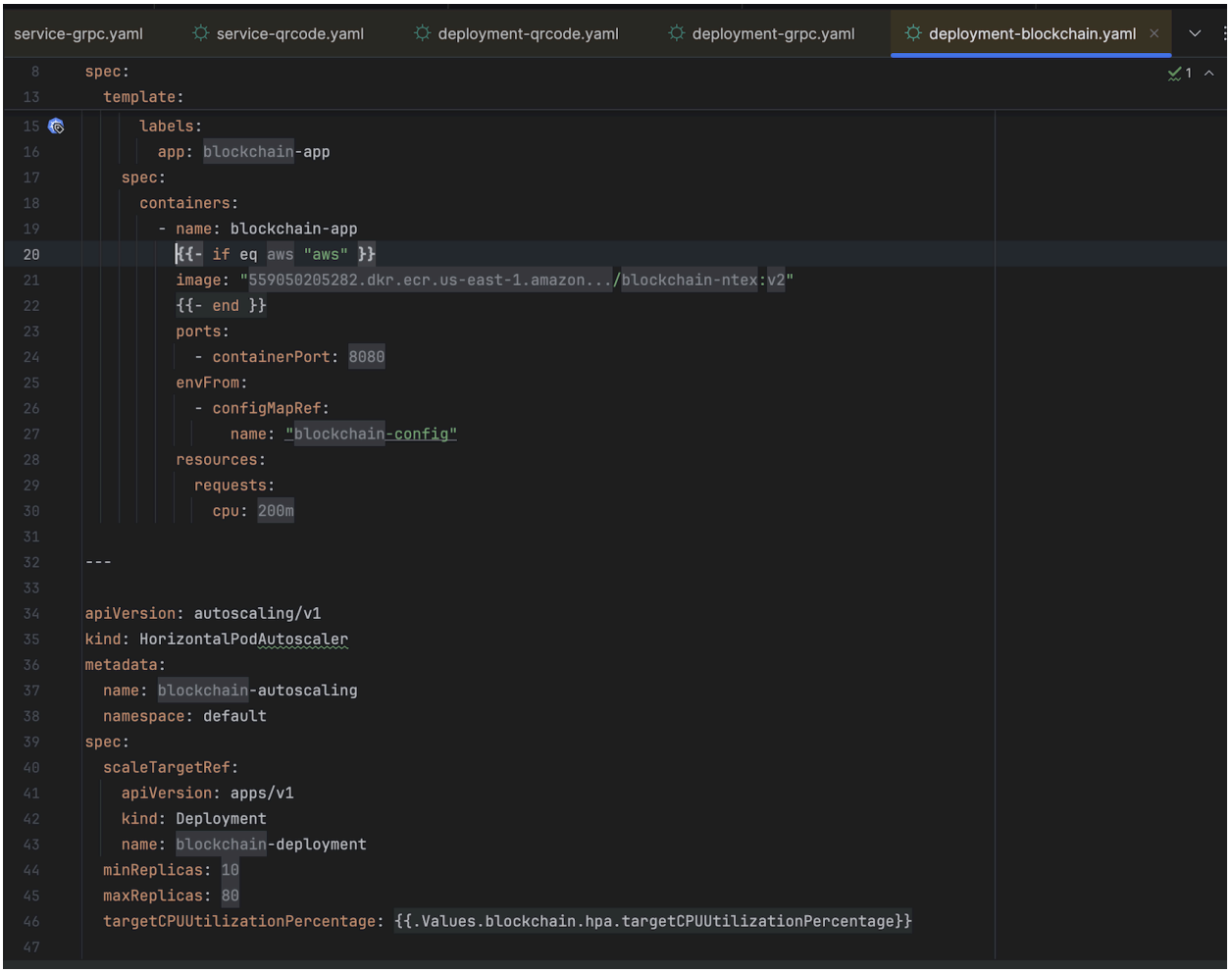
- Did you improve your web-tier deployment process?
  Yes we improve the web-tier deployment
  What were the improvements or changes you made?
  In the helm chart, we changed the deployment configuration for each microservice. In that way, we include the use of CPU requests for each pod and then diminish the number of pods by setting a CPURequest limit, and also the "`targetCPUUtilizationPercentage`" for autoscaling. This helps to control and use 8 m6a.large instances to reach the target RPS for the blockchain service in test and, exploit the same strategy for the for QRCode Service (Here combine 4 onDemand nodes + 10 spots instances) to deploy the gRPC pods (5) and QRCode pods(120) to reach the high throughput.

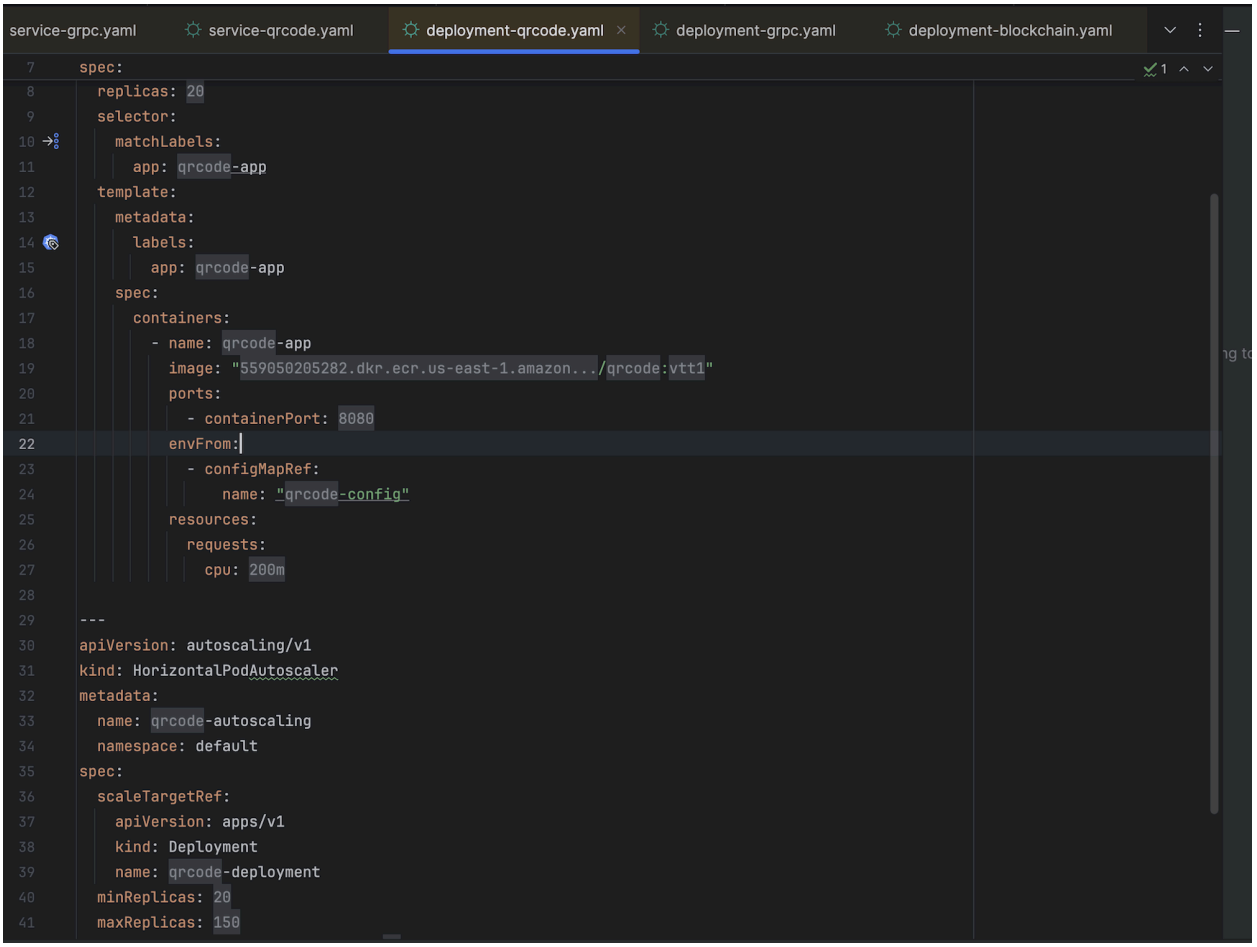
  Include your improved/changed Terraform scripts, Kubernetes manifest or Helm charts under the folders referring to Phase 1 folder structures.

```yaml
blockchain:
  # TODO: Specify values used in blockchain service manifests
  # TODO: Specify values used in deployment.yaml
  releaseName: "blockchain"
  replicaCount: 10
  image: "blockchain-ntex"
  # TODO: Specify values used in service.yaml
  service:
    type: "ClusterIP"
#    type: "LoadBalancer"
    port: "80"
    targetPort: "8080"
#    nodePort: "30009"
  # TODO: Specify values for Horizontal Pods Autoscaler
  hpa:
    minReplicas: 10
    maxReplicas: 80
#    minReplicas: 150
#    maxReplicas: 200
    targetCPUUtilizationPercentage: 100
    cpuRequest: 200m

qrcode:
  releaseName: "qrcode"
  replicaCount: 20
  image: "qr-code"
#  tag: "v0"
  # TODO: Specify values used in service.yaml
  service:
#    type: "LoadBalancer"
    type: "ClusterIP"
    port: "80"
    targetPort: "8080"
  #    nodePort: "30009"
  # TODO: Specify values for Horizontal Pods Autoscaler
```

```
27    qrcode:
33      service:
34    #   type: "LoadBalancer"
35        type: "ClusterIP"
36        port: "80"
37        targetPort: "8080"
38      #   nodePort: "30009"
39      # TODO: Specify values for Horizontal Pods Autoscaler
40      hpa:
41        minReplicas: 20
42        maxReplicas: 150
43        targetCPUUtilizationPercentage: 95
44        cpuRequest: 200m
45
46    grpc:
47      # Release name and container image for the gRPC service
48      releaseName: "grpc"
49      grpcPort: "50051"
50      grpcTargetPort: "9000"
51      grpcImage: "public.ecr.aws/o1d3p3m0/cmucc-f24-teamproject-auth-grpc:amd64"
52      replicaCount: 2
53      service:
54    #   type: "LoadBalancer"
55        type: "ClusterIP"
56        port: "50051"
57        targetPort: "9000"
58      # Horizontal Pod Autoscaler values for the gRPC service
59      hpa:
60        minReplicas: 5
61        maxReplicas: 10
62        targetCPUUtilizationPercentage: 100
63        cpuRequest: 200m
64
65
```


service-grpc.yaml
service-qrcode.yaml
deployment-qrcode.yaml
deployment-grpc.yaml
deployment-blockchain.yaml
spec:
template:
labels:
app: blockchain-app
spec:
containers:
- name: blockchain-app
{{- if eq aws "aws" }}
image: "559050205282.dkr.ecr.us-east-1.amazon.../blockchain-ntex:v2"
{{- end }}
ports:
- containerPort: 8080
envFrom:
- configMapRef:
name: "blockchain-config"
resources:
requests:
cpu: 200m
---
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
name: blockchain-autoscaling
namespace: default
spec:
scaleTargetRef:
apiVersion: apps/v1
kind: Deployment
name: blockchain-deployment
minReplicas: 10
maxReplicas: 80
targetCPUUtilizationPercentage: {{.Values.blockchain.hpa.targetCPUUtilizationPercentage}}


service-grpc.yaml
service-qrcode.yaml
deployment-qrcode.yaml
deployment-grpc.yaml
deployment-blockchain.yaml
spec:
replicas: 20
selector:
matchLabels:
app: qrcode-app
template:
metadata:
labels:
app: qrcode-app
spec:
containers:
- name: qrcode-app
image: "559050205282.dkr.ecr.us-east-1.amazon.../qrcode:vtt1"
ports:
- containerPort: 8080
envFrom:
- configMapRef:
name: "qrcode-config"
resources:
requests:
cpu: 200m
---
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
name: qrcode-autoscaling
namespace: default
spec:
scaleTargetRef:
apiVersion: apps/v1
kind: Deployment
name: qrcode-deployment
minReplicas: 20
maxReplicas: 150

cicd.yml | test-samza – GenerateFollowsTask.ja | 15619-cloud-computing-clou

data.sh | values.yaml | service-grpc.yaml | service-qrcode.yaml

```yaml
spec:
  template:
    metadata:
      labels:
        app: grpc-app
    spec:
      containers:
        - name: grpc-app
          image: public.ecr.aws/o1d3p3m0/cmucc-f24-tea...
          ports:
            - containerPort: 50051
          envFrom:
            - configMapRef:
                name: "grpc-config"
          resources:
            requests:
              cpu: 200m

---
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: grpc-autoscaling
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: grpc-deployment
  minReplicas: 5
  maxReplicas: 10
  targetCPUUtilizationPercentage: 100
```

Question 6: Storage-tier deployment orchestration

In addition to your web-tier deployment orchestration, it is equally important to have automation and orchestration for your storage-tier deployment in order to save labor time and avoid potential human errors during your deployment. Answer the following questions to let us know how your storage-tier deployment orchestration has changed in Phase 2 as compared to Phase 1.

- Did you improve your storage-tier orchestration? What were the improvements or changes you made? Include your improved/changed Terraform scripts, Kubernetes manifest or Helm charts under the folders referring to Phase 1 folder structures.

Question 7: Live test and site reliability

Phase 2 is evaluated differently than how we evaluated Phase 1. In Phase 2, you can make as many attempts as you want before the deadline. However, all these attempts do not contribute to your score. Your team's Phase 2 score is determined by the live test. It is a one-off test of your web service during a specified period of time, and so you will not want anything to suddenly fail; if you encounter failure, you (or some program/script) probably want to notice it immediately and respond!

- What CloudWatch metrics are useful for monitoring the health of your system? What metrics help you understand performance?
  CloudWatch metrics like CPU Utilization, Memory Usage, Disk I/O, and Network I/O are useful for monitoring the health of your system, indicating if resources are being overused or if there are bottlenecks. For performance, metrics like Request Latency, Request Count, and Error Rates (4xx/5xx errors) help identify slow or failed requests, guiding performance optimizations.
- Which statistics did you collect when logged into the EC2 VM via SSH? Which tools did you use? What do the statistics tell you?
  When logged into the EC2 instance, tools like top for CPU usage, free -m for memory, df -h for disk usage, and netstat for network performance are used. These tell you if resources are under heavy usage, which could lead to performance issues, helping you spot potential bottlenecks or areas requiring optimization.
- Which statistics did you collect using kubectl? What do the statistics tell you?
  Using kubectl, metrics like pod status (kubectl get pods), resource usage (kubectl top pod), and logs (kubectl logs <pod-name>) help monitor the health of services, check resource consumption, and diagnose errors or failures. These statistics are crucial for understanding whether services are running smoothly or facing issues.
- How did you monitor the status of your storage tier? What interesting facts did you observe?
  For storage, AWS CloudWatch metrics track EBS volume performance, like Disk I/O and Volume Queue Length, helping to identify slow storage. Kubernetes uses kubectl get pvc to ensure volumes are correctly bound to pods. These tools help monitor storage health and performance, ensuring reliable data access.
- How would your microservices continue to serve requests (possibly with slower response times) in the event of a system failure? Consider various scenarios, including program crashes, network outages, and even the termination of your virtual machines.

In case of system failure, microservices can continue to serve requests using mechanisms like auto-scaling, health checks, and retry logic. In scenarios like crashes or network outages, Kubernetes restarts failed pods, and services can scale to maintain availability, though with potentially slower response times due to resource limitations.

- During the live test, did anything unexpected happen? If so, how big was the impact on your performance, and what was the reason? What did you do to resolve these issues? Did they affect your overall performance?
  During the live test, unexpected issues like resource exhaustion, network outages, or volume attachment failures may occur, impacting performance. By monitoring with CloudWatch and kubectl, you can quickly identify the root cause, apply automated fixes like scaling or restarting pods, and minimize downtime or slow response times.

Task 3: General questions

Question 8: Scalability

In this phase, we serve read-only requests from tens of GBs of processed data. Remember this is just an infinitesimal slice of all the user-generated content on Twitter. Can your servers provide the same quality of service when the amount of data grows? What if we realistically allow updates to tweets? Here are a few good questions to think about after successfully finishing this phase.

- Would your design work as well if the quantity of data was 10 times larger? What about 100x? Why or why not? (Assume you get a proportional amount of machines.)
  Scaling the system to 10 or 100 times the data size would likely impact performance unless the infrastructure is designed to handle such growth. If you simply scale the number of machines proportionally without considering how the data is distributed, there could be bottlenecks due to inefficient data retrieval or storage. Caching, data partitioning, and load balancing would need to be optimized for performance as data grows.
- Would your design work if your web service also implemented insert/update (PUT) requests? Why or why not?
  If insert/update (PUT) requests were added, the system would need to handle not just read operations but also modifications to data, which introduces complexity. Write operations can cause performance bottlenecks unless the database is optimized for fast writes, and careful handling of data consistency is required. Ensuring durability and consistency (e.g., using transactions) might introduce latency or affect scalability if not designed correctly.

- What kind of changes in your schema design or system architecture would help you serve insert/update requests? Are there any new optimizations that you can think of that you can leverage in this case?
  To handle insert/update requests, you might need to redesign your database schema to support efficient writes, like using indexing or partitioning. Additionally, switching to a distributed database or adding replication and sharding can help scale out. Implementing

an event-driven architecture or using message queues could help handle traffic spikes and improve performance during updates.

Question 9: Postmortem

- How did you set up your local development environment? Did you install the databases locally on your machines to experiment? Did you test all your programs before running them on your Kubernetes cluster on the cloud?
  After the ETL on the miniset, there is also a mysql database hosted locally for test. The local database is only stored miniset data.
  Before running twitter service on the cloud, the service will be hosted locally on the local dataset, and the output result is compared with the reference server.
- Did you attempt to generate a load to test your system on your own? If so, how? And why? (Optional)
- Describe an alternative design to your system that you wish you had time to try.
  Alternatively I would like to apply a cache to the database. Because there are few tweets that interacted with other majority users, which means they will be retrieved from the database multiple times. If there is a cache on those tweets, it can be much faster.
- What were the toughest roadblocks that your team faced in Phase 2?
  As mentioned above, we have local mysql hosted on miniset and we compare the result with the reference server before running on cloud. However, there are lots of problems only existed in the full dataset, make us have to working with them on the cloud.
- Did you do something unique (any cool optimization/trick/hack) that you would like to share?
  Encoding text field in url-safe base64 can easily avoid \t or \n to break the tsv structure.

Question 10: Spark and ETL Process

- Did the concepts and tasks from Project 3 (Spark project) assist you in your ETL process for the Team Project? If so, in what ways did they contribute to your understanding or implementation? How did the Spark tools or techniques help, if at all, in optimizing or managing the ETL pipeline? Would you approach the ETL process differently without Spark?
  Trying to avoid expensive operations can largely reduce the ETL time. It is very useful.

Question 11: Contribution

- In the Phase 1 Final reports, we asked about how you divided the exploration, evaluation, design, development, testing, deployment, etc. components of your system and how you partitioned the work. Were there any changes in responsibilities in Phase 2? Please show us the changes you have made and each member's responsibilities.
  Team is working in a cross functional way where we help each other in what we can. Specifically, Carmel is responsible for coordinating the team and helping Jeremey to design the twitter service while Zewei and Jeremey dig into the ETL service and how to

set it up and how to optimize the database schema. Moreover, each member is responsible for deploying services for the live test.
Instead having clear divided job, each member is collaborating and contributing to the team in what is needed.

- Please show Github stats to reflect the contribution of each team member. Specifically, include screenshots for each of the links below.
    https://github.com/CloudComputingTeamProject/<repo>/graphs/contributors
    https://github.com/CloudComputingTeamProject/<repo>/network

    https://github.com/CloudComputingTeamProject/ARandomName-F24/graphs/contributors

    https://github.com/CloudComputingTeamProject/ARandomName-F24/network