Team Project Report - Phase 1

Note: We Strongly recommend reading through both checkpoint and final report template before starting the project

Phase 1 summary

Team name: CCprojectFor3

Members (names and Andrew IDs):

Kewen Zhao A15937908 kewenz, Zhitong Guo zhitongg, Yuwei yuwei2

Please color your responses red.

Performance data and configurations

Number and types of instances: 7 m7g.large

Cost per hour of the entire system: 0.0816 EC2 * 7 + 0.008 ELB + 0.02 EBS = 0.599

(Cost includes on-demand EC2, EBS, and ELB costs. Ignore S3, Network, and Disk I/O costs, Data Transfer Fees)

Requests Per Second (RPS) of your best submission:

	Blockchain Service	QR Code Service	Twitter Service
score	35	25.00	
submission id	925175	928952	
throughput	70729.81	115654.56	
latency	5.25	3.48	
correctness	92.45	85.96	
error	0.00	0.00	

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: Web-tier

Question 1: Comparing web frameworks

Blockchain Service and QR Code Service do not involve integration with a storage tier and are a good opportunity for you to compare and choose a suitable web framework. Please try at least two combinations of programming and web frameworks, and answer the questions below. To save cost during your comparison, we recommend that you deploy your testing code to a single EC2 virtual machine instead of a Kubernetes cluster.

- You are free to pick either Blockchain Service and QR Code Service (or both) to compare web frameworks. Which Microservice did you choose and why?
- We picked QR Code Service because it involved both receiving and sending HTTP request, which test the web performance more thoroughly.
- What frameworks have you tried? How did each framework perform? Please include the instance (or cluster) configuration, latency, and RPS of each framework you tried.
- We tried axum, the most popular Rust framework, and Ntex, the fastest Rust framework. We used 6 Worker and 1 Master with m7g.large instance.
- Axum: latency 3.66, RPS 108985.02
- Ntex: latency 3.48, 115654.56
- For the two frameworks with the highest RPS, please list and compare their functionality, configuration, and deployment.
- Functionality:
- We only implement two functions for each of these two frameworks with one simply replying “hello world” to test connections and another one implementing the web tier logic to respond to requests from tests with appropriate responses.
- Configuration:
- We just use the default configuration from each of these two frameworks with routing to the functions that handle requests or test connections. We bind the address to localhost and port to 8080 for TCP connections.
- Deployment:
- Which one would you like to choose for other microservices? What are the most important differentiating factors?

- I'd like to use Ntex for block-chain service as with respect to the efficiency of json serialization and parsing, Ntex performs far better than Axum.
- Did you have to do any trade-offs while picking one of the multiple frameworks you selected?
- As we selected two frameworks implemented by Rust, we shared the code for handling requests for these two frameworks. Furthermore, through testing, Ntex always performs better than Axum in all aspects, so we did not make any trade-off and decided to use Ntex as our main web framework.

Question 2: Load balancing and Scaling

Our target RPS for Blockchain Service and QR Code Service are 30K and 55K, respectively, which can be hard to reach with a single pod. We have learned about Kubernetes and ELBs in individual projects, so let us explore the knowledge and skills you have developed here.

- Discuss load-balancing in general, why it matters in the cloud, and how it is applicable to your web tier.
- Load balancing is a technique used to distribute workloads across multiple computing resources. With respect to our web tier, load balancing is primarily focussed on distributing incoming Internet traffic across multiple servers hosting the same application or service and ensures that no single server becomes overwhelmed or starved to improve the reliability, availability and performance of applications and services.
- Provision a single EC2 instance and deploy your Blockchain Service using a Kubernetes deployment with one replica. Provision a Network Load Balancer by using Kubernetes service resources. Make a 120-second submission to the Sail() Platform and record the RPS. Add one replica to your deployment while keeping your instance count to only 1. Wait until the target group shows the new pod as healthy, make another submission to the Sail() Platform, and record the RPS. Repeat this process 2 times, ending with 3 replicas running on 1 instance.
  As the number of replicas increases, does the RPS grow accordingly? Please show the graph with the data points you just collected and write down a possible explanation.
  1 deployment: 10162.56
  2 deployment: 11179.66
  3 deployment: 10364.03

  No, the RPS doesn't seem grow, because they all share the same CPU resources, increasing deployment doesn't increase CPU utilization.

- Provision a cluster with as many worker nodes as possible while staying under the $0.70/hr submission budget. Deploy your Blockchain Service using a Kubernetes deployment with only one replica and provision a Network Load Balancer by using Kubernetes service resource. Make a 120-second submission to the Sail() Platform and record the RPS. Add one replica to your deployment at a time, wait until the target group shows the new replica as healthy, make another submission to the Sail() Platform, and record the RPS. Repeat this process until your number of replicas reaches your number of worker nodes.

As the number of replicas increases, does the RPS grow accordingly? Please show the graph with the data points you just collected and write down a possible explanation.
1 deployment: 10364.03
2 deployment: 21073.43
3 deployment: 34928.19

Yes the RPS grows accordingly. This is because one pod only uses one instance. In order to achieve full utilization, each instance should at least have one pod running.

- How do you check which pod is running on which worker node? Give a specific command. How would you ensure that the workload is distributed evenly across your cluster?

1. kubectl describe pods - check which pod is running on which node
2. kubectl top node - check the CPU utilization of each node. They should have about the same utilization.

- Compare the difference between Application Load Balancer and Network Load Balancer. For each type, use the best cluster configuration you can have under the $0.70/hr submission budget. Use the ingress resource to provision an ALB and service resource for NLB. Wait until the load balancers are active and the target groups show the pods as healthy. Submit the ALB/NLB endpoint to the Sail() Platform (QR Code Service, 600s) 6 times consecutively and record the RPS. Finish the following table.
  Note: Each type requires 6 submissions * 10 minutes consecutively, so it will take at least 1 hour; please plan your time and budget accordingly.

The instance type of your master/worker nodes: m7g.large
The number of instances in your cluster:

Submission ID	Application Load Balancer	Network Load Balancer
929087/ 928952	13361.26	115654.56
929097/ 929073	17872.71	139642.20
929100/ 929074	37004.89	114329.40
929103/ 929078	66561.78	86068.86
929107/ 929082	101703.37	144783.60
929111/ 929083	100379.07	127443.52

- We mentioned ALB warm-up in individual project P1. Explain the warm-up and why it is necessary based on the experiment above.
- From P1, “if the traffic load increases rapidly over a short period of time, it becomes difficult for the ALB to handle this load fast enough”, Therefore, warmup allows the ALB to scale up to handle the expected workload

- We discussed scaling Kubernetes deployment in P2. Besides manually specifying the number of replicas in your deployment resource, describe one other technique that scales the web tier of your microservice.
- We can set an autoscaling policy with min and max instance, cpu utilization to achieve auto-scaling.

Question 3: Comparing REST/gRPC APIs

This question applies to QR Code Service. In QR Code Service, you need to implement REST/gRPC APIs to communicate with the authentication service and request an authentication token. Use the following questions as a guide to implement your APIs.

- Which API protocol did you choose for QR code authentication, REST or gRPC? What were the key factors influencing your choice of API protocol?
- We used REST because REST APIs are generally simpler to design, implement, and use compared to gRPC.
- Have you implemented any optimizations for the authentication API call? For example, try to make asynchronous API calls to increase efficiency.
- No, because no other jobs can be run while waiting for the response, so we think it's not beneficial.
- List one or more API protocols besides REST and gRPC that you think are suitable for this scenario and provide your rationale.
- SOAP: SOAP's built-in standards for security, reliability can provide a more comprehensive framework for secure communications compared to REST
- Have you conducted any profiling to measure the network latency between the QR code service and the authentication service? List one or more tools/techniques that can be used to measure the network latency between containers.
- Yes, we used a timer to measure the amount of time, by taking the difference between request sent and response received. Another tool to measure network latency is to use Ping or Postman to send request directly to the auth service.
- (Optional) Did you try to implement APIs in both protocols? Are there any performance differences you observed between REST and gRPC? In terms of scalability and efficiency, did you find one API protocol to be more suitable for the QR code authentication task?
- (Optional) Do you find this task useful in terms of understanding microservice communication? If so, share with us the lessons you learned from this task. If not, what content do you believe should be included in future semesters?
- I learned that if there are dependencies between microservices, the latency adds up. So it's important that in a large-scale project consisting of many microservices, each team to be responsible for optimizing their latency.

Question 4: Web-tier architecture

This question applies to all microservices. For each microservice, we need a web tier that interprets requests and does processing to generate responses. For QR Code Service, the QR code server needs to request the authentication token from the authentication server. For Twitter Service, the web tier

needs to retrieve user data from the MySQL database. How would you set up your web tier for these purposes? You can use the following questions as a guide.

Remember that, although you can test different microservices individually for now, in later phases, your 3 microservices will need to be under the same public DNS and respond to requests simultaneously. Make sure to consider this in your architecture and cluster design.

- Which web-tier architecture did you choose? Did you put pods of different microservices into the same worker node?
- We chose Rust with Ntex as it's very efficient to handle API calls and data serialization. We did not put pods of different microservices into the same worker node.
- Did you put your QR Code Service container and the authentication server container into the same pod or different pods? Why?
- We put it into the same pod because it's easy to orchestrate and the authentication server can be accessed by localhost.
- Did you put your Twitter Service container and MySQL container into the same pod or different pods? Why?
- We put it into the same pod because it's easy to orchestrate and the authentication server can be accessed by localhost.
- What alternative architectures can you think of? What are their implications on performance and scalability?
- An alternative choice is to put the services into independent pods.
- Scalability: More granular control over scaling. The Twitter Service and MySQL can be scaled independently based on demand.
- Explain your choice of instance type and the number of worker nodes for your Kubernetes cluster.
- We chose m7g.large since we are using arm devices and fit the budget plan better. We use 7 nodes total to stay within the budget
- (Optional) Have you tried more than one architecture or cluster configuration? What did their performance look like? Can you reason about the difference?

Question 5: Web-tier deployment orchestration

We know it takes dozens (or hundreds) of iterations to build a satisfactory system, and the deployment process takes a long time to complete. Setting servers/clusters up automatically saves developers from repetitive manual labor and reduces errors.

- We require you to use Terraform, Kubernetes, and Helm to orchestrate the deployment of your web tier. What steps were taken every time you deployed your system? Please include your Terraform script, Helm chart, and Kubernetes manifest in your submission per the requirements stated in the Phase 1 write-up. If you use kOps to provision your Kubernetes cluster, please also include the cluster definition YAML files in your submission. You must provide the automation scripts and documentation that enable teaching staff to replicate a functioning web tier using a clean local machine that has Terraform, kOps, kubectl, and Helm installed.

- We use kOps to provision our cluster
- If Terraform, Kubernetes, and Helm were not required, what are some other ways to automate your web-tier deployment? Compare their advantages and disadvantages.
- AWS CloudFormation:
  - Advantage: Supported by AWS and includes the latest feature
  - Disadvantages: Provider Lock-in: These tools are not portable across different clouds.
- (Optional) How can you further automate the entire deployment process? If you implemented any other automation tools apart from Terraform/kOps/Helm, like Shell scripts, please describe them. We would also like to see your scripts in your code submission.

Task 2: Extract Transform Load (ETL)

Question 6: Your ETL process

We have 1TB of raw data, which takes a non-trivial amount of time and money to process. It is strongly advised to test, run, and debug your E & T on a small portion of the data locally, and plan wisely before booting expensive clusters! Please briefly explain:

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

scala/spark

We used Databrick notebook since its powerful and popular, and we chose scala to run spark

- The number and type of instances used during ETL and justification

5 worker nodes and 1 master node; we have tested on 2 worker nodes and 1 master node before, and run into many incomplete executions due to limitations of memory.

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

1hr 33min for reading process, the cleaning process took about 1 hour

- The time spent loading your data into the database

~2 hour

- The overall cost of the ETL process

~$130

- The number of incomplete ETL runs before your final run

10

- The size of the resulting database and reasoning

40G, filtering reduces size by 10x, and we only select a few columns which further reduce by 8x; then more features are created on top of the original ones but takes up very few space, up to 40G in total.

- The size and format of the backup

40G, we dumped the mysql database locally

- Discuss the difficulties encountered

JDBC Driver not found, we had to download the jar and connect the driver manually; unable to write the dataframe due to memory constraints, so we switched to using azure databricks cluster

- If you had multiple database schema iterations, did you have to rerun your ETL between schema iterations?

We don't have multiple iterations

Question 7: ETL considerations

Historically, some teams ran out of the budget because they chose AWS for ETL and had multiple runs before they got their final dataset. Sometimes they simply used unnecessarily fancy (expensive) hardware. We believe you have compared and contrasted all the options and made smart decisions. Here are a few more questions about your understanding of the different choices:

- Which cloud service provider (AWS, GCP, Azure) and which service (EMR, Dataproc, HDInsight, Databricks, etc.) did you use, and what was the difference?
- We use Databricks on Azure since they are well integrated and cost-effective; we tried Azure HDinsight before, but the zeppelin notebook is not very user-friendly, and it does not have additional features such as autoscaling that come with Databricks.
- What are the most effective ways to speed up ETL?
- Reduce table size if you can before join, cache re-used dataframe
- Besides Spark, list two other programming models you can use for ETL. Which approach would be more efficient and why? The term efficient can refer to many different aspects, such as development time and actual run time, but as a result, high efficiency should reduce the final ETL cost.
- MapReduce: Data is read from disk and slower than spark
- Apache Flink: it provides real-time processing, but it's much steeper learning curve and provides less mature ecosystem
- Did you encounter any error when running the ETL job in the provisioned cluster? If so, what are those errors? What did you do to investigate and recover from the error?
- String too long for mysql column: we change the column type from TEXT to LONGTEXT
- 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 some other ways to facilitate loading?
- We loaded the data into mysql database with jdbc drivers, and we write dataframes directly to the database. This method is fast and does not need intermediate steps that may cause additional overhead. There may be other ways such as writing the dataframe to a csv first, but we did not try them because writing also takes time and additional storage.
- Did you store any intermediate ETL results and why? If so, what data did you store, and where did you store the data?
- No, because the whole process is not too long.
- 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]
- The top hashtag contributes to the hashtag scores heavily and produces bias towards the popular hashtags, which makes the similarity scores between users less meaningful and more reflective of the relationship between users. By removing the popular hashtags, we made the similarity scores more niche and more specific towards each user.
- Before you run your Spark tasks in a cluster, it would be a good idea to apply some tests. You may test the filter rules or the entire process against a small dataset. You may 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 want to see your efforts in ETL testing.)
Note: Don't forget to put your code for ETL testing under the **ETL** folder in your team's GitHub repo **master** branch.

- We compared the counts of the etl result of the first partition of the dataset, with the counts of the example result provided in the writeup, and discovered that they are the same, meaning we have the correct filters.

```
sampleDf_2mb.count()
```

```
res21: Long = 6500
```

Took 1 hrs 6 min 56 sec. Last updated by anonymous at March 14 2024, 6:24:57 PM.

```
import org.apache.spark.sql.functions._
import org.apache.spark.sql.SparkSession

val spark = SparkSession.builder.appName("Filter Tweets").getOrCreate()
import spark.implicits._

val tweets = sampleDf_2mb.cache

val validTweets = tweets.filter(
    ($"id".isNotNull || $"id_str".isNotNull) &&
    ($"user.id".isNotNull || $"user.id_str".isNotNull) &&
    $"created_at".isNotNull &&
    $"text".isNotNull && $"text" =!= "" &&
    $"entities.hashtags".isNotNull && size($"entities.hashtags") =!= 0 &&
    array_contains(array(lit("ar"), lit("en"), lit("fr"), lit("in"), lit("pt"), lit("es"), lit("tr"), lit("ja")), $"lang")
).cache

val uniqueTweets = validTweets.dropDuplicates("id_str").cache
```

```
import org.apache.spark.sql.functions._
import org.apache.spark.sql.SparkSession
spark: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@7ea17d2e
import spark.implicits._
tweets: sampleDf_2mb.type = [contributors: string, coordinates: struct<coordinates: array<double>, type: string> ... 36 more fields]
validTweets: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [contributors: string, coordinates: struct<coordinates: array<double>, type: string> ... 36 more fields]
uniqueTweets: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [contributors: string, coordinates: struct<coordinates: array<double>, type: string> ... 36 more fields]
```

Took 1 hrs 6 min 34 sec. Last updated by anonymous at March 14 2024, 6:24:58 PM.

```
validTweets.count()
```

```
res22: Long = 773
```

Took 9 sec. Last updated by anonymous at March 14 2024, 6:25:06 PM.

Task 3: Databases

Question 8: Schema

We first extract and transform the raw data and then store it in databases in a good format. Various schemas can result in a very large difference in performance! In Phase 1, we only ask you to reach 10% of the target RPS (200 out of 2000) of Twitter Service for receiving a checkpoint score. If you achieve 2000 RPS, you will get the Twitter Service Early Bird Bonus. These targets can be achieved with a basic but functional schema. In Phase 2, your Twitter Service 3 will need to reach 100% of the target (10,000) RPS which would require a more efficient schema.

When you are optimizing your schema in Phase 2, you will want to look at different metrics to understand the potential bottlenecks and come up with a better solution (before trying to tune parameters!). These iterations take a lot of time, and a lot of experimentation, so keep this in mind while designing your schema in Phase 1.

- What is your schema for Twitter Service in Phase 1? Did you already have multiple iterations of schemas? If so, please describe the previous schemas you have designed and explain why you decide to iterate to a new schema.

We have 1 schema with 2 tables. One table contains the interactions score and keyword score for each pair on contacted user, the other table contains the latest_screen_name and latest_description for each user.

- List 2 potential optimization that can be applied to your schemas. How might they improve your performance?
- We can apply index on user 1 for faster retrieval
- We can apply index on user 2 for faster retrieval

Question 9: 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 web framework, you can start learning about the configuration parameters to see which ones might be the most relevant and gather data and evidence with each individual optimization. Although you might not have arrived at an optimized database schema yet, feel free to learn about tunable parameters in your database system. No need to feel obliged to test all of them now. In future phases, you will continue to try to improve the performance of your microservices.

To plan better for future phases, you should consider the following: 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 tweak as you can find that might impact performance in terms of Kubernetes config, web framework, your code, DBs, DB connectors, OS, etc. Choose at least three optimizations, 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.
- Use autoscaling for all 3 services, some services might receive more work than others
- Get rid of print statements in the code
- Deploy more clusters that fit the budget
- For every tweak you listed above, try to explain how it contributes to performance.
- Autoscaling can scale the number of service pod base on number of request
- Removing print statement makes the program run faster
- Add more clusters makes the program run faster

Question 10: 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.

- We require you to use Terraform, Kubernetes, and Helm to orchestrate the deployment of your storage tier. What steps were taken every time you deployed your system? Include your Helm chart and Kubernetes manifest in your code submission on GitHub. Starting with a Kubernetes cluster that has your web tier running, we should be able to have a functional storage tier that is reachable from your web tier after running your deployment scripts.
- We first deploy mysql service and load the mysqldump file from s3 bucket
- Making snapshots of your database can save time importing your ETL results into the database if you need to re-deploy your storage tier. What are some methods to make snapshots of your database? Did you implement any of them? How much time do you save each time you revert to the snapshot as opposed to re-importing the ETL results?
- We used mysql dump, it saved ~3 hours
- **Task 4: Site Reliability**

Question 11: Monitoring

Once the service is set up and running, it generates a large amount of status data. They help us monitor if the system is operational and also give us insights into its performance.

- What CloudWatch metrics are useful for monitoring the health of your system? What metrics help you understand performance?
- For health monitoring :
- CPUUtilization indicates the percentage of allocated EC2 compute units that are currently in use reflecting the load of EC2 instances.
- StatusCheckFailed_Instance indicates whether the instance has failed a status check with non-zero values representing a problem.
- For performance monitoring:
- NetworkIn/NetworkOut represents the volume of incoming/outgoing network traffic to an instance which is useful for understanding network performance.
- DiskReadOps/DiskWriteOps represents the number of read/write operations from/to disk reflecting the I/O performance of instances.
- What statistics can you collect if you have SSH access to the VM? What tools did you use? What did they tell you?
- Top command in terminal: this shows a real-time view of the system's resource usage including CPU, memory utilization per process to track if any of the processes is taking up too many resources.
- Sar command with sysstat installed: it collects, reports and saves system activity information. This can report on various system loads such as CPU activity, memory usage, device load and network.
- Nload command with nload installed: this provides a visual representation of incoming and outgoing traffic separately, which is helpful for quickly assessing network load.
- What statistics can you collect if you have kubectl access but not SSH access into the nodes? What tools did you use? What did they tell you?
- Built-in Kubernetes commands:

- Kubectl top node/ pod is useful for tracking the CPU and memory utilization of nodes and pods to help me identify resource-intensive workloads.
- Kubectl describe nodes/ pods is useful if we want detailed information about Kubernetes objects including events, conditions, usage statistics for troubleshooting.
- Kubernetes dashboard is a web-based user interface that lets us manage and monitor applications running in the cluster and troubleshoot them accordingly.
- How do you monitor the status of your database? What noteworthy insights did you gain?
- Native database monitoring tools such as Mysql Workbench, command-line tools ("mysqladmin") and third-party monitoring tools like Datadog: these can give us key performance indicators such as query execution time, number of connections, transaction rates, and resource utilization.
- INSIGHT GAINED: try and identify slow queries with these tools and optimize these with index created or queries rewritten. What's more, we can also use these tools to monitor the connection counts to prevent overloading the database with too many connections.

.

Question 12: Profiling

We expect you to know the end-to-end latency of your system. This helps us to focus on optimizing the parts of the system that contribute to most of the latency.

- Given a typical request-response for Twitter Service, for each latency below, think about if you can measure it, how you can measure it, and write down the tools/techniques that can be used to measure it:
    a Load Generator to Load Balancer

Answer: AWS X-ray can be used to trace and analyze requests as they travel the AWS infrastructure.

    b Load Balancer to Worker Node

Answer:ditto

    c Worker Node to Web-tier Pod

Answer: Using application instrumentation directly. For client-side instrumentation, in the worker node, instrument the part of the application that sends the request to the web-tier pod. Record immediately before the request is sent and immediately after the response is received. For server-side instrumentation, log the timestamp as soon as the request is received and right before the response is sent.

    d Parsing request

Answer: ditto, we need to surround the request parsing code with timing functions to capture the start and end time of the parsing process.

    e Web Service to DB

Answer: still we need to surround the database operation code with timing functions to capture the start and end time of database operations such as update, delete and query. However, to get the network latency from web service to DB and DB to web service we need to subtract the whole database operation latency with DB execution time which can be assessed by database monitoring tools such as AWS Performance Insights or Mysql workbench or enabling performance schema.

f At DB (execution)

Answer: Using the Mysql workbench, enabling performance schema.

g DB to Web Service

Answer: same as e

h Parsing DB response

Answer: surround the parsing code with timing functions

i Web-tier Pod to Worker Node

Answer: same as c

j Worker Node to Load Balancer

Answer: same as b

k Load Balancer to Load Generator

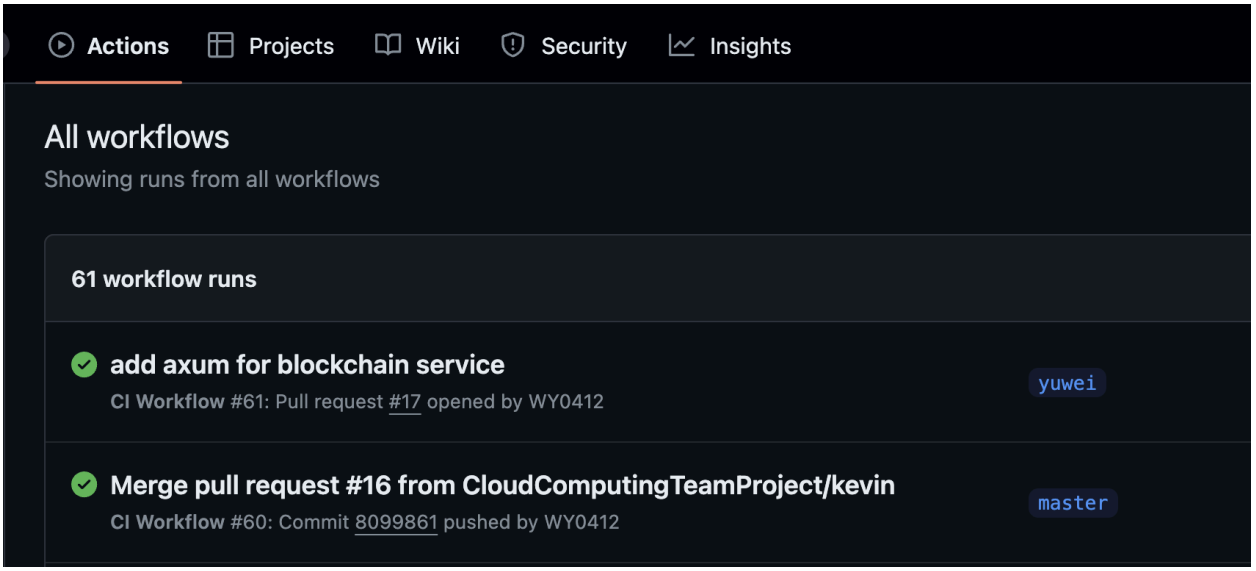
Answer: same as a

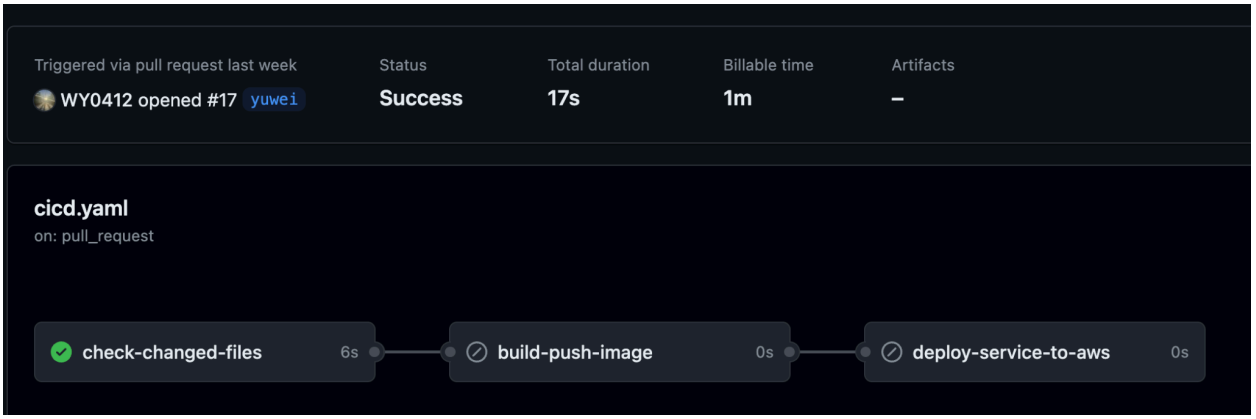
- For each part above, think about whether you can improve the latency or not, and list possible ways to reduce the latency.
- a) Consider the geographical proximity of these two. Also, we can optimize the network path by using direct connect provided by AWS.
- b) Optimize the load balancer configuration such as changing load balancer type or enabling HTTP/2.
- c) leverage proximity-based routing such as ingress controllers to route requests to the closest available service endpoint based on various criteria.
- d) optimize the parsing algorithm by using more efficient parsing libraries for json, xml or other formats.
- e) consider using connection pooling instead of opening and closing a database connection for request. Furthermore, caching or batch operations can also significantly reduce the latency for database connections.
- f) optimize the sql queries such as avoiding fetching unnecessary data and ensuring JOIN operations are efficient. What's more, indexing can also dramatically reduce the latency for queries.
- g) same as e
- h) same as d
- i) same as c
- j) same as b
- k) same as a

Task 5: Deployment Automation

Question 13: Deployment Automation

- Briefly describe the different jobs in your workflow YAML files. Describe their purpose and functionality, and also if they are executed sequentially or in parallel.
- Cluster.yaml set the the physical cluster on aws
- helm/*yaml set the the services (blockchain, qrcode, twitter)

- Briefly describe how you implement test cases for each microservice and how to execute these tests without manual operations.
- We run cargo run test in github workflow and check the error code. Unit test are under each rust files.
- Did/would you explore additional CI/CD tasks besides the requirements? If not, what kinds of CI/CD tasks do you think will be helpful to save your time and prevent failures in the team project?
- Integration test can be helpful, we currently only have unit test
- Did you encounter any difficulties while working on the CI/CD tasks? If so, please describe the challenges and how you eventually resolved them.
- We need to set up appropriate roles for git action to orchestrate.
- Please attach your workflow YAML files and the following screenshots to this question: (1) the "Actions" tab of your CloudComputingTeamProject GitHub repository; (2) the "Summary" tab of a workflow; (3) Execution results for each job in a workflow.
- 

- 

Task 6: General questions

Question 14: Scalability

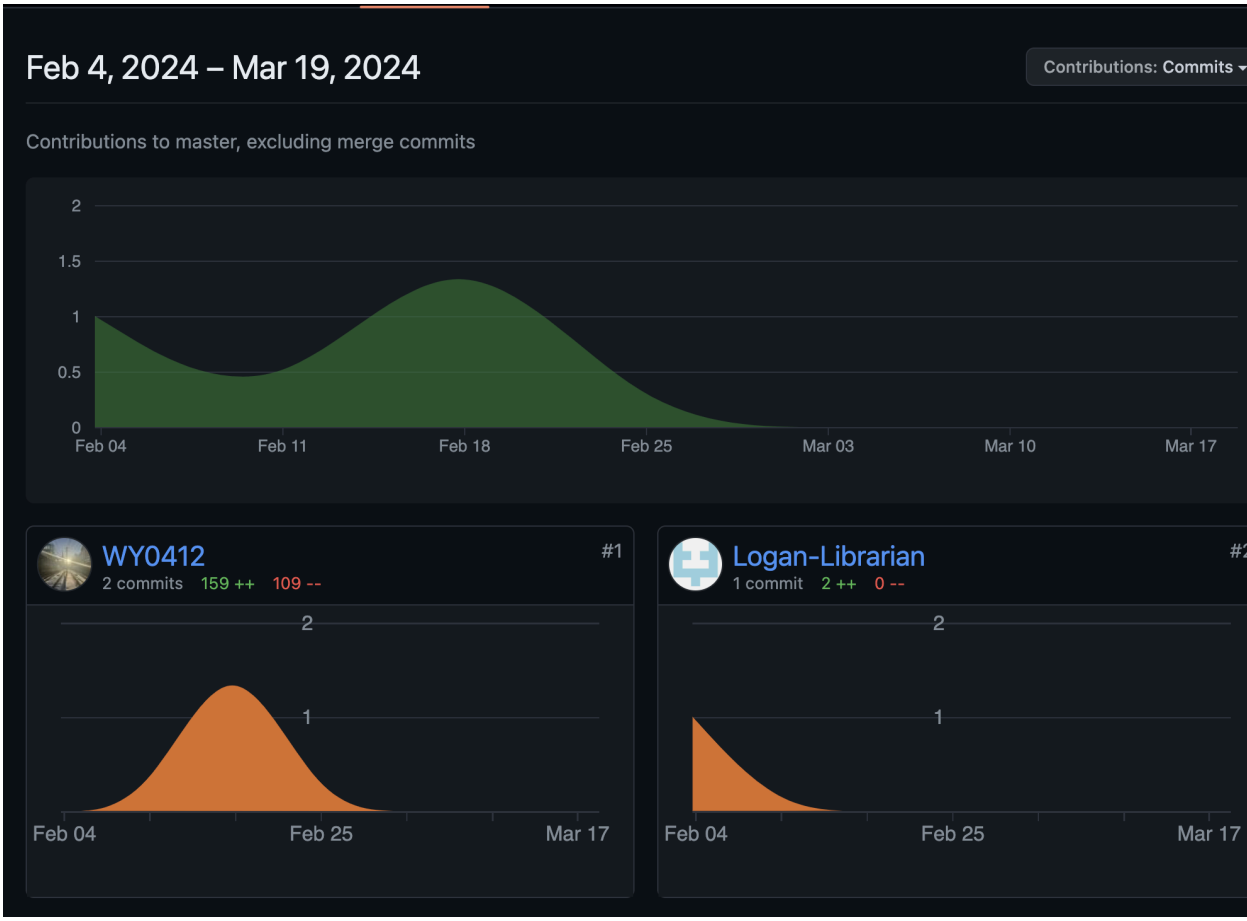
In this phase, you serve read-only requests from tens of GBs of processed data. Remember, this is just an infinitesimal slice of all the user-generated content on Twitter. Can your servers provide the same quality of service when the amount of data grows? What if we allow updates to tweets? These are good questions to discuss after successfully finishing this phase. It is okay if you answer no, but we want you to think about them and share your understanding.

- 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.)
- No. Because the size of database instances needs to be increased through vertical scaling to handle higher loads or using horizontal scaling/ sharding to spread the load across multiple instances. Also, caching layers like Redis may be necessary to cache frequent queries.
- Would your design work if your web service also implemented insert/update (PUT) requests? Why or why not?
- No. As long as we take insert/update into consideration. Synchronization is a must for us to consider so as to keep the ACID quality of databases when it comes to multithreading. However, we did not implement synchronization, which makes our service not able to work when faced with PUT requests.

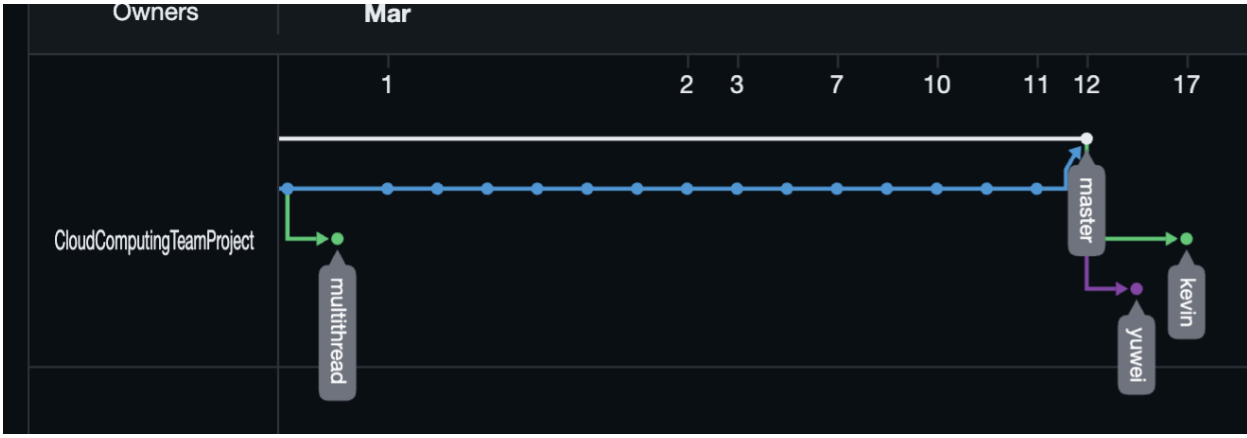
Question 15: Postmortem

- How did you set up your local development environment? Did you install the databases locally on your machines to experiment? Did you test all programs before running them on your Kubernetes cluster on the cloud?
- For web tier and storage tier, we indeed set our local development environment. As ntex and axum are relatively lightweight web frameworks, we set up http server locally on our PC for correctness testing. We also have a small subset of data stored in the local mysql database to do simple tests. Thus we did test our programs for correctness before running them on Kubernetes clusters on the cloud.
- Did you attempt to generate a load to test your system on your own? If so, how? And why?
- No, we decided to run the tests given on the cloud to get detailed performance of our service with the correctness ensured so that we won't waste too much budge both on improving performance and ensuring correctness.
- Describe an alternative design to your system that you wish you had time or budget to try.
- Use multiple thread to compute PoW in blockchain.
- Which were the toughest roadblocks that your team faced in Phase 1?
- ETL fails very often, costing significant budget
- Did you do something unique (any cool optimization/trick/hack) that you would like to share?
- Switching from HDinsight to Databricks is a life saver decision for us, we were able to better monitor and debug the etl process

Question 16: Non-technical

- How did you divide components in the system? How do the components interact? How did you partition the work and what was each team member's contribution?
- We divide into Web-tier, Backend-Tier, and Deployment. We rotate on each job
- How did you test each component? Did you implement Unit Testing or Integration Testing?
- We implemented Unit Testing
- Did you like the Git workflow, code review, and internal testing requirements? Were they helpful to your development?
- Yes
- Please show Github stats to reflect the contribution of each team member. Specifically, include screenshots for each of the links below.
  1. https://github.com/CloudComputingTeamProject/<repo>/graphs/contributors
  2. (some team members didn't set their identities in the commits"
  3. 

  4. https://github.com/CloudComputingTeamProject/<repo>/network

5. 

Task 6: Bonus

5% QR Code Service Early Bird Bonus

Fill in the form if your team was able to achieve the QR Code Service target by 03/03 and hence eligible for the 5% early bird bonus. Please push all the files you wrote to reach the target RPS and tag your git commit as "earlyBird-qrcode" so that we know you completed this bonus at that commit.

	QR Code Service
score	
submission id	
throughput	
latency	
correctness	
error	
date/time	

5% First Mover Bonus for QR Code Service

If your team was among the first 10 to reach the QR Code Service checkpoint by 03/03 and thus eligible for the first mover bonus, please provide the following information to allow the teaching staff to verify and grant you the bonus.

You need to fill the table below to describe the submission that achieved QR Code Service checkpoint, label the code that you used for the submission and push to your team's GitHub repository, and provide a screenshot of the complete scoreboard to serve as evidence of your team's ranking. Note that you need to take the screenshot immediately after you achieve the bonus, otherwise making more submissions may overwrite the scoreboard.

	QR Code Service
score	
submission id	
throughput	
correctness	
latency	
error	
date AND time of submission (e.g., 2024-02-17 19:11:08)	
ranking for first mover bonus (e.g., 1, 2, …, 10)	

Please push the code that you used to reach the target RPS and tag your git commit as "firstMover-qrcode" so that we know you completed this bonus at that commit.

In order to claim the bonus, you must submit a screenshot of the scoreboard at the moment of submission showing your team's ranking among the other submissions. The screenshot should include all present submissions on the scoreboard, not just your own. This will help us verify that your team was one of the top 10 teams on the scoreboard for the QR Code Service at the time of submission.

5% Twitter Service Early Bird Bonus

Fill in the form if your team was able to achieve the Twitter Service Phase 1 target (2000 RPS) by 03/17 and hence eligible for the 5% early bird bonus. Please push all the files you wrote to reach the target RPS and tag your git commit as "earlyBird-twitter" so that we know you completed this bonus at that commit.

	Twitter Service
score	

submission id	
throughput	
latency	
correctness	
error	
date/time	

Penalty Waiver for Twitter Service Correctness

Fill in the form if your team was able to make a 10-minute Twitter Service submission with above 95% correctness by 03/17 and hence eligible for waiving one most significant penalty for each team member.

	Twitter Service
score	
submission id	
throughput	
latency	
correctness	
error	
date/time	

5% First Mover Bonus for Twitter Service

If your team was among the first 10 to reach the Twitter Service checkpoint by 03/17 and thus eligible for the first mover bonus, please provide the following information to allow the teaching staff to verify and grant you the bonus.

You need to fill the table below to describe the submission that achieved the Twitter Service checkpoint, label the code that you used for the submission and push to your team's GitHub repository, and provide a screenshot of the complete scoreboard to serve as evidence of your team's ranking. Note that you need to take the screenshot immediately after you achieve the bonus, otherwise making more submissions may overwrite the scoreboard.

	Twitter Service
score	
submission id	
throughput	
correctness	
latency	
error	
date AND time of submission (e.g., 2024-02-17 19:11:08)	
ranking for first mover bonus (e.g., 1, 2, …, 10)	

Please push the code that you used to reach the target RPS and tag your git commit as “firstMover-twitter” so that we know you completed this bonus at that commit.

In order to claim the bonus, you must submit a screenshot of the scoreboard at the moment of submission showing your team's ranking among the other submissions. The screenshot should include all present submissions on the scoreboard, not just your own. This will help us verify that your team was one of the top 10 teams on the scoreboard for Twitter Service at the time of submission.