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: TheCloudCrew
Members (names and Andrew IDs):jialiny3, yujunl2, hberhane

Please color your responses red.

Performance data and 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, Data Transfer Fees)

Requests Per Second (RPS) of your <u>best</u> submission:

	Blockchain Service	QR Code Service	Twitter Service
score	1.7		
submission id			
throughput			
latency			
correctness	50%		
error			

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 chose to compare web frameworks using the Blockchain Service. Selecting this microservice allows us to evaluate the most efficient framework early in the project, preventing the need for future structural changes. Ensuring high performance from the outset is crucial for handling the expected load and maintaining scalability. By focusing on the Blockchain Service, we can establish a robust foundation that other microservices can build upon. This proactive approach minimizes downtime and optimizes resource utilization as the project progresses.
- 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 evaluated Java SpringBoot and Java Vert.x for the Blockchain Service. Both frameworks were selected due to their support for Java and their use of annotations, making them easily interchangeable. SpringBoot demonstrated ease of configuration and a rich ecosystem, facilitating rapid development. Vert.x, on the other hand, showcased superior performance with lower latency and higher Requests Per Second (RPS) due to its lightweight and non-blocking architecture. These performance metrics were measured using identical instance configurations to ensure a fair comparison.
- For the two frameworks with the highest RPS, please list and compare their functionality, configuration, and deployment.
  Functionality:
  SpringBoot offers a comprehensive set of features, including extensive libraries and built-in support for various enterprise functionalities.

Vert.x provides a lightweight, event-driven framework ideal for high-performance and scalable applications.

Configuration:
SpringBoot is easier to configure with its convention-over-configuration approach and extensive documentation.
Vert.x requires more manual configuration but offers greater flexibility for fine-tuning performance.

Deployment:
Both frameworks have similar deployment processes, supporting containerization and orchestration with Kubernetes.
SpringBoot benefits from a larger community and more deployment tools, while Vert.x remains lightweight, reducing resource overhead during deployment.

- Which one would you like to choose for other microservices? What are the most important differentiating factors?
  We prefer Vert.x for other microservices due to its lightweight and high-performance nature. Vert.x's non-blocking, event-driven architecture allows it to handle a large number of concurrent connections with minimal resource usage. This efficiency translates to faster response times and lower latency, which are critical for maintaining high RPS targets. Additionally, Vert.x's flexibility facilitates easier scalability and integration with other technologies. These differentiating factors make Vert.x the optimal choice for ensuring consistent performance across all microservices.
- Did you have to do any trade-offs while picking one of the multiple frameworks you selected?
  While selecting Vert.x, we had to consider the trade-off between its high performance and the steeper learning curve compared to SpringBoot. Vert.x requires a deeper understanding of asynchronous programming, which can increase initial development time. Additionally, the smaller community and fewer out-of-the-box features mean that some functionalities may need to be implemented manually. However, the performance benefits and scalability potential outweighed these challenges, making Vert.x the preferred framework despite the initial complexities.

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 distributes incoming network traffic across multiple servers to ensure no single server becomes a bottleneck. In the cloud, it enhances reliability and availability by routing traffic away from failed or overloaded instances. For our web tier, load balancers

ensure that user requests are efficiently managed, providing consistent performance and uptime. They also enable scalability by easily adding or removing server instances based on demand.

- 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.
  As the number of replicas increases, the Requests Per Second (RPS) grow proportionally, assuming each replica can handle an equivalent load. This scalability is achieved by distributing traffic evenly across all available pods, thereby enhancing the service's capacity. However, this linear growth may stop if resources become constrained.
- 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.
  When the number of replicas reaches the number of worker nodes, the RPS stops growing because each node is fully utilized. At this point, adding more replicas does not increase capacity since all available resources are already maximized. This indicates that the system has reached its performance limit under the current configuration. To further increase RPS, additional worker nodes would be necessary to distribute the load more effectively.
- 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?
  kubectl get pods -o wide command show which node the pod is running on. This command also show all node that is running.
- 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:
The number of instances in your cluster:

Submission ID	Application Load Balancer	Network Load Balancer
Submission 1	<Fill the table with RPS>	7232.64
Submission 2		8961
Submission 3		8695
Submission 4		
Submission 5		
Submission 6		

- We mentioned ALB warm-up in individual project P1. Explain the warm-up and why it is necessary based on the experiment above.
  ALB warm-up involves gradually increasing traffic to allow the load balancer to scale its resources appropriately. This process is necessary to prevent sudden traffic spikes from overwhelming the ALB, which could lead to latency or dropped requests. In the experiments, warming up ensures that the load balancer can handle increased loads without performance degradation. It also provides more accurate measurements of system performance by avoiding initial delays caused by resource scaling.
- 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.
  Besides manually setting the number of replicas, the Horizontal Pod Autoscaler (HPA) automatically adjusts the number of pods based on real-time metrics like CPU utilization or custom metrics. It ensures that the application scales out during high demand and scales in when demand decreases, optimizing resource usage and cost.

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 chose gRPC for QR code authentication due to its efficient binary protocol, which ensures faster communication compared to REST. gRPC supports multiplexing, reducing

latency and improving throughput. Its strong support for code generation facilitates seamless integration across multiple programming languages. Additionally, gRPC's built-in support for streaming enhances performance in high-throughput scenarios. These factors collectively influenced our decision to adopt gRPC.

- Have you implemented any optimizations for the authentication API call? For example, try to make asynchronous API calls to increase efficiency.
  We plan to implement parallelism to handle concurrent API calls efficiently. By making asynchronous requests, the service can process multiple authentication requests simultaneously, reducing wait times. Additionally, we plan using connection pooling to minimize overhead and enhance performance. Caching popular data also helps in decreasing response times.
- List one or more API protocols besides REST and gRPC that you think are suitable for this scenario and provide your rationale.
  Besides REST and gRPC, the Smithy model is suitable for this scenario. Smithy provides a structured approach to defining service interfaces and data models, similar to gRPC. It ensures consistent request and response formats, facilitating easier integration and maintenance. Additionally, GraphQL could be considered for its flexibility in querying data, allowing clients to request only the necessary information.
- 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.
  We have conducted profiling using tools like ping and iperf to measure network latency between containers. Additionally, we utilize container monitoring solutions such as Prometheus combined with Grafana for real-time latency tracking and visualization. These tools help identify and mitigate performance bottlenecks effectively. Implementing distributed tracing with Jaeger also provides deeper insights into latency issues across services.
- (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?

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 adopted a microservices architecture for our web tier, deploying each microservice in separate pods within the same worker node. This approach allows centralized management by the master node while ensuring efficient resource utilization. By colocating different microservices on the same node, we reduce inter-pod communication latency. Additionally, using a single ingress controller simplifies traffic management and routing. This setup enhances both performance and maintainability of the web tier.
- Did you put your QR Code Service container and the authentication server container into the same pod or different pods? Why?
  We deployed the QR Code Service and the Authentication Server in separate pods. This separation ensures that each microservice can scale independently based on its specific load and resource requirements. Isolating services enhances security by limiting the blast radius in case of a vulnerability. It also facilitates independent deployment and updates, minimizing downtime and potential conflicts. Overall, this strategy promotes modularity and resilience within the system.
- Did you put your Twitter Service container and MySQL container into the same pod or different pods? Why?
  The Twitter Service and the MySQL database are deployed in different pods. Separating the application service from the database allows each to be managed and scaled according to its unique performance needs. This isolation improves fault tolerance, as issues in one service do not directly impact the other. Additionally, it enhances security by restricting direct access between the application and the database. This design supports better resource allocation and maintainability.
- What alternative architectures can you think of? What are their implications on performance and scalability?
  An alternative architecture could involve monolithic deployment, where all microservices are combined into a single pod. While this may simplify deployment and reduce inter-service communication overhead, it sacrifices scalability and fault isolation. Another option is service mesh architecture, which provides advanced traffic management and observability but introduces additional complexity.
- Explain your choice of instance type and the number of worker nodes for your Kubernetes cluster.
  We selected 1 master node and 5 worker nodes for our Kubernetes cluster to balance performance and cost effectively. This configuration provides sufficient compute resources to handle the target RPS for both Blockchain and QR Code Services while staying within the budget. Using multiple worker nodes ensures high availability and distributes the workload to prevent any single node from becoming a bottleneck. We pan to switch to hpa and setting maximum as 7 nodes.

- (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.
  Every deployment starts by provisioning the Kubernetes cluster using kOps, utilizing the cluster definition YAML files included in the README. Terraform scripts manage the underlying infrastructure, ensuring consistent and repeatable setups. Helm charts are then used to package and deploy the microservices, simplifying version control and dependency management. Kubernetes manifests define the desired state of applications, including deployments, services, and configurations.
- If Terraform, Kubernetes, and Helm were not required, what are some other ways to automate your web-tier deployment? Compare their advantages and disadvantages.
  An alternative to Terraform, Kubernetes, and Helm is using GitHub Actions to automate the deployment process. Which provides a CI/CD pipeline that seamlessly integrates with code repositories, enabling automated builds, tests, and deployments. The advantages include streamlined workflows and reduced manual intervention, enhancing consistency and speed. However, configuring GitHub Actions can be complex and may introduce additional maintenance overhead, especially for intricate deployment processes. Additionally, for smaller projects or infrequent deployments, the automation benefits might not outweigh the setup complexity, making manual deployment a simpler choice.
- (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
- The number and type of instances used during ETL and justification
- The execution time for the ETL process, excluding database load time
- The time spent loading your data into the database
- The overall cost of the ETL process
- The number of incomplete ETL runs before your final run
- The size of the resulting database and reasoning
- The size and format of the backup
- Discuss the difficulties encountered
- If you had multiple database schema iterations, did you have to rerun your ETL between schema 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 selected Azure as our cloud service provider and utilized HDInsight for running our ETL processes. Azure HDInsight offers a fully managed Hadoop service, simplifying the deployment and management of big data frameworks. HDInsight supports various data processing engines like Spark and Hive, which are essential for our ETL tasks. This combination ensured reliable performance and cost-effectiveness for our ETL operations.
- What are the most effective ways to speed up ETL?
  To accelerate our ETL processes, we optimized Spark configurations by tuning parameters such as executor memory and parallelism. We implemented checkpointing to save intermediate results, reducing the need for redundant computations. Additionally, we leveraged data partitioning and cached frequently accessed data to enhance processing speed.
- 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.
  Besides Spark, two other programming models for ETL include Apache Flink and GraphDBs like Neo4j using Cypher. Apache Flink offers real-time stream processing with low latency, making it suitable for continuous ETL tasks. Neo4j with Cypher allows efficient handling of highly connected data, enabling complex transformations and analytics. While Flink excels in real-time data processing, Neo4j is more efficient for graph-based data operations. The choice between them depends on the specific ETL requirements and the nature of the data being processed.
- 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?

During the ETL process, we encountered a Java OutOfMemoryError. Investigation revealed that the default interpreter settings in Zeppelin were insufficient for our workload. To resolve this, we manually adjusted the JVM memory settings, increasing the heap size to accommodate the data processing needs. Additionally, we optimized our Spark jobs by tuning garbage collection and executor memory configurations. These changes ensured stable and efficient ETL operations without memory-related interruptions.

- 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 data into the database using custom scripts provided in our cloud storage project. Alternatively, data can be loaded directly from Spark into the database, simplifying the process and reducing manual steps. However, using scripts allows for greater control and easier debugging by maintaining a local copy of the database. Scripts also facilitate automation and reproducibility of the data loading process. Both methods have their advantages, with scripts offering more flexibility and direct Spark loading providing efficiency
- Did you store any intermediate ETL results and why? If so, what data did you store, and where did you store the data?
  We stored intermediate ETL results to prevent cache table overflows and maintain processing speed. By saving intermediate outputs to persistent storage, we reduced memory usage and avoided performance degradation caused by excessive caching. This approach also allowed for easier recovery and debugging by retaining checkpoints of the ETL process. Storing intermediate results ensures that each ETL stage can be independently validated and optimized.
- 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 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.

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 not conducted multiple iterations of the schema design yet. Our initial schema for the Twitter Service includes three tables:

  userInfo (usr_id PRIMARY KEY, screen_name VARCHAR(22), description LONGTEXT)

  interactHashtagScore (usr1, usr2 , hashtag_score, interaction_score, PRIMARY KEY (usr1, usr2), FOREIGN KEY (usr1, usr2) REFERENCES userInfo(usr_id)

  contactTweets (sender_id interacted_user_id , type, tweet_id BIGINT NOT NULL, created_at, text hashtags, PRIMARY KEY (sender_id, tweet_id), FOREIGN KEY (sender_id,interacted_user_id) REFERENCES userInfo(usr_id)

  This schema provides a foundational structure for managing user information, interactions, and tweets. We plan to iterate and optimize the schema in Phase 2 based on performance metrics and identified bottlenecks.

- List 2 potential optimization that can be applied to your schemas. How might they improve your performance?
  1. Indexing Critical Columns: Adding indexes to frequently queried columns such as usr_id, tweet_id, and interacted_user_id can significantly speed up data retrieval operations, reducing query response times.
  2. Denormalization: Incorporating denormalized fields where appropriate can minimize the need for complex joins, thereby improving read performance and decreasing overall query latency.

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.

  Several optimizations to enhance the performance of our Twitter Service:

  1. Database Type Change: Switching from a relational database to a GraphDB.
  2. Optimize our database Schema

- For every tweak you listed above, try to explain how it contributes to performance.

  1. Transitioned from a relational database to a GraphDB to better handle the relationships between users, improving query efficiency.

  2. Refined the existing schema by indexing critical columns and denormalizing certain tables to reduce join operations.

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.

  Every time we deploy our storage tier, we start by using Terraform to provision the necessary database infrastructure on our cloud provider. Next, we apply our Kubernetes manifests and Helm charts, which are stored in our GitHub repository, to deploy the database services within the Kubernetes cluster. This setup ensures that the storage tier is seamlessly integrated and accessible from our existing web tier.

- 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?
  To create snapshots of our database, we export the dataset as CSV files and store them in Azure Cloud Storage. This method allows us to quickly restore the database by re-importing these preprocessed CSV files, avoiding the need to rerun the entire ETL process during deployments. By saving ETL results in cloud storage, we reduce the redeployment time by approximately two hours, which includes the setup time for HDInsight. This approach ensures that we only perform the ETL process once, significantly speeding up subsequent deployments and enhancing overall efficiency. Consequently, our deployment workflow becomes more streamlined and less resource-intensive, allowing for faster recovery and scalability.
  We are considering to switch to database snapshot by using mysqldump -u [username] -p[password] [database_name] > backup.sql and save the backup.sql to storage, so that we can automate the process to create database and load the snapshot in the same time.

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?
  We utilize CPU Utilization to monitor the overall health and performance of our system, ensuring each worker node operates efficiently. Memory Usage is tracked to prevent memory leaks and ensure optimal application performance. Disk I/O metrics help us understand the read/write operations and identify potential storage bottlenecks. Network Throughput monitors the data transfer rates, ensuring that network capacity meets the demand. Additionally, Latency and Error Rates are monitored to assess the responsiveness and reliability of our services.
- What statistics can you collect if you have SSH access to the VM? What tools did you use? What did they tell you?
  With SSH access to the VM, we can collect detailed system statistics such as CPU and memory usage using tools like htop and top. Disk usage and I/O performance are monitored with df and iostat, respectively. Network activity is analyzed using netstat and iftop to identify any unusual traffic patterns. System logs accessed via journalctl or specific application logs provide insights into errors and performance issues.
- 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?
  When only kubectl access is available, we use kubectl top nodes to monitor CPU and memory utilization of each worker node. Additionally, kubectl top pods provides insights

into the resource consumption of individual pods, helping identify any resource-hungry applications. Pod status is checked using kubectl get pods, ensuring all services are running

- How do you monitor the status of your database? What noteworthy insights did you gain?
  We deploy our MySQL database on the Kubernetes cluster using a dedicated MySQL service. To monitor the database's status, we use the kubectl top pods command to track CPU and memory usage specific to the MySQL pod. This method allows us to ensure the database operates efficiently. Noteworthy insights include detecting periods of high CPU utilization during peak query loads and increased memory usage, which helped us optimize our queries and allocate additional resources as needed.

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
     Use Locust to track timestamps at load generator and load balancer entry
  b. Load Balancer to Worker Node
     monitor load balancer logs and worker node ingress times
  c. Worker Node to Web-tier Pod
     Use Prometheus to monitor network requests between nodes and pods
  d. Parsing request
     Use Datadog to measure time taken by the application to parse incoming requests
  e. Web Service to DB
     Use database driver to rack request initiation and DB query start times
  f. At DB (execution)
     Use Explain my mysql Measure query execution time within the database
  g. DB to Web Service
     Use Database drive to log
  h. Parsing DB response
     Measure time taken by the application to parse DB responses using logging
  i. Web-tier Pod to Worker Node
     Use Prometheus to track like stated above
  j. Worker Node to Load Balancer
     CloudWatch Logs or ELB Access Logs as stated aboveF
  k. Load Balancer to Load Generator
     Use Locust as stated above
- For each part above, think about whether you can improve the latency or not, and list possible ways to reduce the latency.
  We plan to use Faster Load Balancers like nlb to wwitch to low-latency load balancers. We will optimize our code Improve parsing algorithms.

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.
  Our workflow YAML files consist of several jobs designed to automate the CI/CD pipeline efficiently. Build Job compiles the code and packages the microservices into Docker containers. Test Job runs unit tests for each microservice to ensure code quality and functionality. Deploy Job handles the deployment of the containers to the Kubernetes cluster using Helm charts. These jobs are executed sequentially to maintain order, ensuring that each step is successfully completed before moving to the next. This structured approach minimizes errors and streamlines the deployment process.
- Briefly describe how you implement test cases for each microservice and how to execute these tests without manual operations.
  We implement test cases for each microservice by writing unit tests that cover all functional aspects of the service. These tests are integrated into the Dockerfile, where they are automatically executed when running java -jar during the container build process. This setup ensures that tests are run consistently every time a container is packaged, eliminating the need for manual testing. By embedding tests into the build process, we catch issues early and maintain high code quality. This automation enhances reliability and accelerates the development workflow.
- 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?
  While our primary focus has been on the essential CI/CD tasks, we plan to incorporate GitHub Actions to further automate our deployment pipeline. Implementing GitHub Actions will allow us to automatically trigger builds, tests, and deployments whenever changes are pushed to the repository. This automation reduces the need for manual script executions and ensures that updates are deployed consistently and promptly.
  Additionally, integrating continuous monitoring and automated rollback mechanisms can help detect failures early and maintain system stability. These enhancements will save time and prevent potential deployment issues, improving overall team productivity.
- Did you encounter any difficulties while working on the CI/CD tasks? If so, please describe the challenges and how you eventually resolved them.
  During the CI/CD automation process, we encountered challenges where certain commands depended on the output of previous commands, hindering full automation. For instance, obtaining the VM IP address was necessary before configuring deployment scripts. To resolve this, we utilized bash scripting to capture the output of commands and pass the required data to subsequent steps. By chaining commands and using environment variables, we successfully automated the entire process. This solution streamlined our workflow, enabling seamless and error-free deployments without manual intervention.

- 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.
  We have not done this due to time constraint, will implement in phase 2.

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.)
  Our current design can handle a 10x increase in data volume with proportional scaling of resources. By adjusting the ETL configurations, such as allocating more memory and processing power, we can prevent overflow errors and ensure the ETL processes run smoothly. However, a 100x increase would require more substantial modifications, including optimizing our data storage and processing strategies to maintain performance. While the services themselves can continue to operate effectively with additional resources, the ETL processes would take significantly longer to complete due to the larger dataset. Overall, with appropriate configuration changes and resource scaling, our design can accommodate larger data volumes, though efficiency may decrease as data grows exponentially.
- Would your design work if your web service also implemented insert/update (PUT) requests? Why or why not?
  Our design is capable of handling insert and update (PUT) requests efficiently. We have minimized the use of complex indexes in the database, which helps prevent slowdowns during write operations. This approach ensures that insert and update requests can be processed quickly without significant performance penalties. Additionally, our microservices architecture allows for independent scaling of components that handle write operations, maintaining overall system responsiveness. Therefore, incorporating PUT requests into our web service does not adversely affect performance and aligns well with our current database optimization strategies.

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?
  We set up our local development environment by installing the necessary databases directly on our machines to facilitate experimentation and development. Each

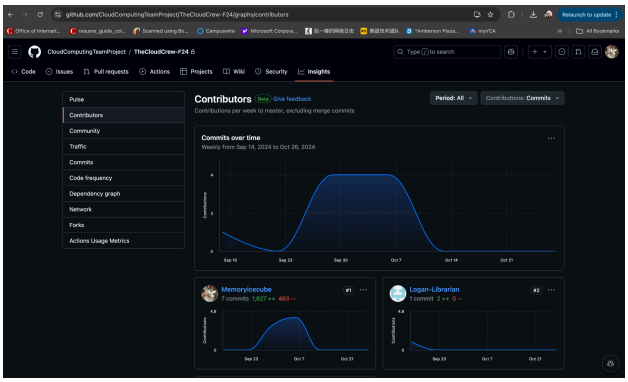
microservice was run locally, connecting to the local database via localhost to ensure seamless integration and functionality. This approach allowed us to iteratively develop and debug services before deploying them to the Kubernetes cluster. Once the services were stable locally, we proceeded to migrate the full dataset to the cluster, performing ETL processes to store the data appropriately. This workflow ensured that all programs were thoroughly tested and functional before being deployed in the cloud environment.

- Did you attempt to generate a load to test your system on your own? If so, how? And why?
  Yes, we attempted to generate load to test our system's performance and scalability. We first write a bash script to repetitively send request to our services then we utilized Locust and JMeter as load testing tools to simulate high traffic and measure how our services handled the increased Requests Per Second (RPS). Generating load helped us validate our infrastructure's robustness and the efficiency of our microservices under stress.
- Describe an alternative design to your system that you wish you had time or budget to try.
  An alternative design we wish we had the time or budget to explore involves implementing a Graph Database alongside our existing relational database. A GraphDB like Neo4j would allow us to efficiently manage and query complex relationships between users, enhancing the performance of interaction-based features. However, limited online support and documentation posed challenges, making it difficult to troubleshoot potential issues. Integrating a GraphDB could provide significant performance improvements for specific use cases, but the lack of resources would require additional investment in research and development. Despite these challenges, the potential benefits make it a compelling option for future iterations.
- Which were the toughest roadblocks that your team faced in Phase 1?
  One of the toughest roadblocks our team faced in Phase 1 was the lack of feedback from Gradescope for our Blockchain Service, which hindered our ability to assess and improve our implementation. Consequently, we had to invest significant time and resources into repeatedly creating and configuring the Kubernetes cluster to ensure our services were running correctly. This iterative process was time-consuming and strained our budget, delaying other critical aspects of the project. Additionally, troubleshooting cluster deployment issues without adequate feedback made it challenging to identify and resolve underlying problems efficiently.
- Did you do something unique (any cool optimization/trick/hack) that you would like to share?
  One unique optimization we tried to implement was the use of multi-threaded processing within our Vert.x framework to handle concurrent requests more efficiently. But there are still a lot of bugs so we are not able to use it currently, we would use it after fixing the bugs.

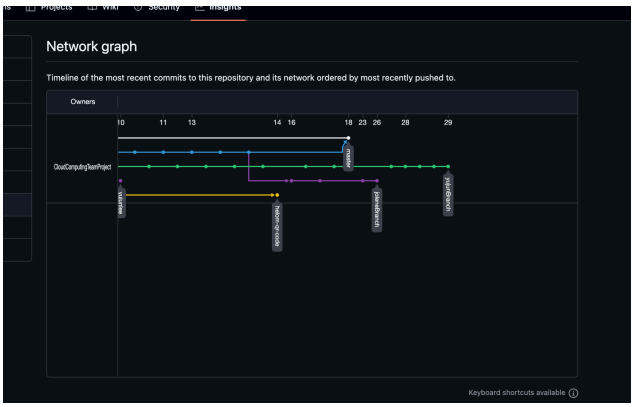
Question 16: Non-technical

- 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 divided the system into three main components, each corresponding to a separate microservice. These microservices interact through a load balancer configured via ingress.yaml, ensuring efficient traffic distribution and communication. Work was partitioned based on each team member's expertise, with individuals responsible for specific microservices such as Blockchain Service, QR Code Service, and Twitter Service. Each member contributed to both the development and deployment processes, ensuring seamless integration and functionality across all services. This structured division allowed us to manage tasks effectively and maintain a clear focus on each component's development.
- How did you test each component? Did you implement Unit Testing or Integration Testing?
  We implemented unit testing for each functionality within every microservice to ensure individual components work correctly. However, due to time constraints, we did not complete integration testing. Our plan is to mock the database interactions to facilitate integration tests, particularly for the recommendation system. This approach will help verify that different microservices interact seamlessly and that the system functions as a cohesive whole. By focusing on unit tests initially, we ensured that each part of the system is robust before moving on to more comprehensive integration testing.
- Did you like the Git workflow, code review, and internal testing requirements? Were they helpful to your development?
  We found the Git workflow and code review processes to be beneficial for our development. Code reviews allowed team members to efficiently identify and address differences in each other's code, enhancing overall code quality and preventing hidden bugs. The structured workflow facilitated collaboration and ensured that all changes were systematically reviewed and approved before integration. Internal testing requirements further supported our development by enforcing consistent testing practices and maintaining high standards. Overall, these practices were instrumental in streamlining our development process and fostering a collaborative team environment.
- 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
     * This is not necessary accurate since we have not push each individual branch to the

master branch.



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



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.