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: BalsamicBaguettes
Members (names and Andrew IDs): Madison Teague (mteague), Eric Kumara (ekumara) Francisco Besoain (fbesoain)

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 best submission:

	Blockchain Service	QR Code Service	Twitter Service
score	35	25	0.0
submission id	2153013	2226709	0.0
throughput	41016	84138	0.0
latency	11.2	6.1	0.0
correctness	100.0	100.0	0.0
error	0.0	0.0	0.0

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 blockchain service to analyze because in our development, we made the transition from net/http framework to fasthttp while fine tuning the performance of our Blockchain Service.

- 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 worked with net/http and fastthttp frameworks in go. Our best net/http submission was used a m8a.large instances with 7 nodes, had a latency of 27.6 and an RPS of 18666
Our best fasthttp submission was used a m8a.large instances with 7 nodes, had a latency of 11.2, and an RPS of 41016.39.

- For the two frameworks with the highest RPS, please list and compare their functionality, configuration, and deployment.

Part of the appeal of using net/http and fasthttp frameworks is that they are very similar in syntax and implementation. The net/http framework requires no external dependencies and supports HTTP client and server implementation. It allows for connection and requet pulling and basic in-out routing. The fasthttp framework has similar functionality but has a reduced memory footprint and is more optimized for high-throughput, low latency scenarios. Fasthttp allows for more specific configuration than net/htttp (which helps improve its performance). This includes specification of keep alives and max body sizes. Both frameworks support relatively easy deployments. The main difference between the two is that net/http has more requires little to no custom configuration for deployment while fasthttp may require some depending on the cloud service used.

- Which one would you like to choose for other microservices? What are the most important differentiating factors?

We will continue to use fasthttp for the other microservices. We are doing this because if significantly brought down the latency in our blockchain implementation, and it is as simple to implement as the net/http framework.

- Did you have to do any trade-offs while picking one of the multiple frameworks you selected?

Since we were all inexperienced with go at the beginning of this project, the simplicity of implementation was a major factor in our decision for web framework.

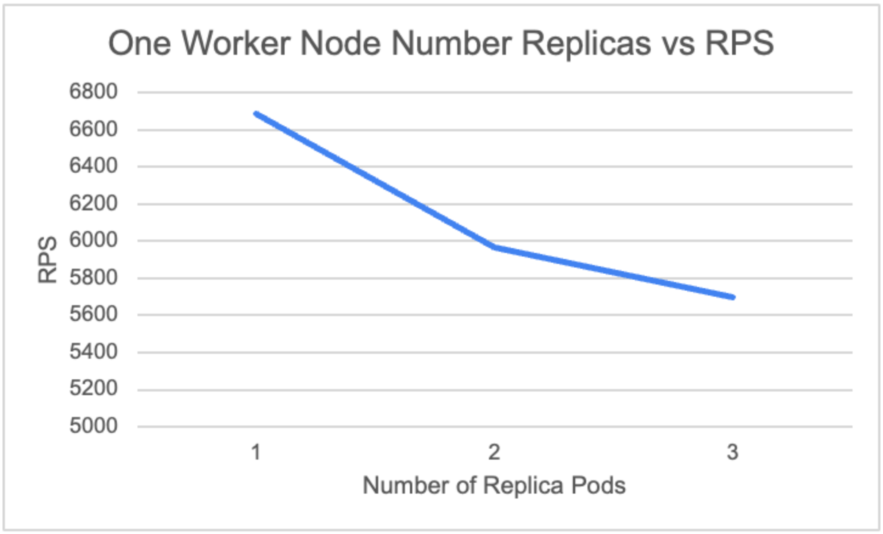
Question 2: Load balancing and Scaling

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.

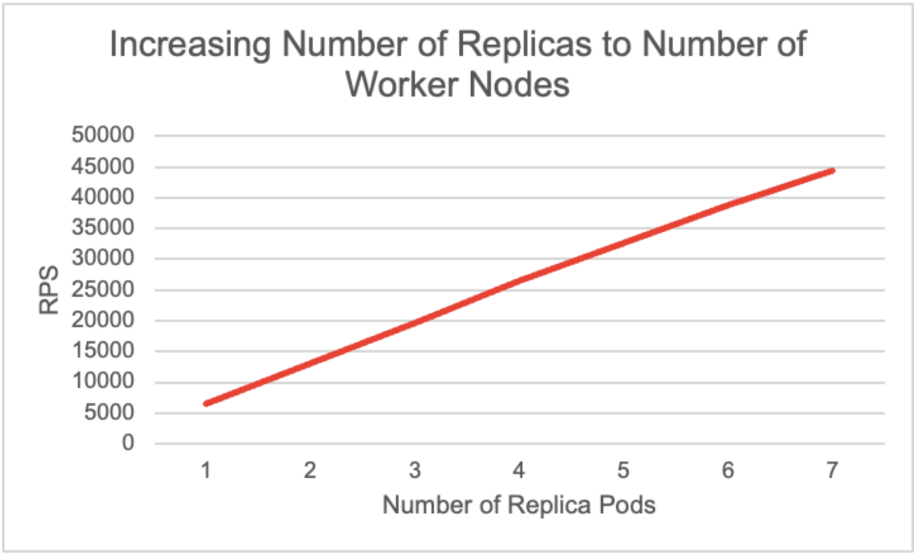
Load balancing is a critical component for cloud computing which distributes incoming network traffic across multiple servers or instances. They allow applications, which run on a cloud infrastructure, to easily scale in and out depending on various client loads. This helps the owner of the service balance customer satisfaction with cost optimization. Our web services uses a NGNX Load balancer to distribute incoming requests to different services, and network load balancers for distributing load within each service.

- 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.



The throughput for this decreases as the number of replicas increases. This is due to the fact that the replicas are sharing resources which causes the increase in latency. As more replicas are added the overhead of running additional pods over extends the one worker node.

- Provision a cluster with as many worker nodes as possible while staying under the $0.72/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.



The throughput increases linearly as the number of replicas increases to the number of worker nodes. This makes sense because each node can allocated resources more efficiently rather than spreading its self too thin over replicas like in the last experiment.

- 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?

To check which pod is running on which worker node, you could use the command: kubectl get pods -o wide —all-namespaces. To ensure workload is evenly distributed across the cluster, you can use appropriate resource limits (CPU Utilization, memory usage…) in the deployment, add afinity rules, which encourage the scheduler to place pods in different noes, and use a HPA (horizontal pod autoscaler) which helps manage the scaling in and out of nodes.

- Compare the difference between Application Load Balancer and Network Load Balancer. For each type, use the best cluster configuration you can have under the $0.72/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: m8.Large 1 master node/7 worker nodes
The number of instances in your cluster: 8
Based on the results below, there is a clear performance advantage for using the network locad balancer over the application load balancer. However performance is more stable with the Application Load balancer than with the network load balancer.

Submission ID	Application Load Balancer RPS	Network Load Balancer RPS
2237941/2237423	11465	41247
2237987/2237489	27023	36259
2238016/2237611	26037	33051
2238113/2237714	25581	31561
2238175/2237820	27593	30532
2238196/2237835	26609	23198

- We mentioned ALB warm-up in individual project P1. Explain the warm-up and why it is necessary based on the experiment above.

The warm up is conducted by providing simulated load to the cluster before running the actual tests. It gives time for the cluster to scale up Warm ups give load balancers an opportunity to vertically scale before they can handle the real test load. Based on the significant increase in performance between submission one and two for the ALB, we see that the warm up helped the load balancer establish optimal connections and internal resource allocation.

- 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 use CPU Utilization as the metric which tells our Horizontal Pod Autoscaler when to add or remove replicas without explicitly specifying the number of replicas.

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 to use gRPC for our QR code authentication implementation. We chose to use gRPC because its performance benefits align with the other optimizations of our system like memory pooling and parallel processing. The protocol buffers provide strict typing and validation compared to JSON in REST. Additionally, the user of timeouts, retries and fallbacks are well-supported with the gRPC API, which are essential to our authentication implementation.

- Have you implemented any optimizations for the authentication API call? For example, try to make asynchronous API calls to increase efficiency.

We implemented several optimizations for the authentication. We attempt to connect to multiple URLs (attempting to connect to multiple endpoints) which enhances the reliability of the system, and decreases the chances of a failure. Next we implement connection timeouts and retries, the timeouts ensure that out system continues working without spinning too long on one connection attempt. Finally, we do resource clean up in the case of timeout which prevents resource leaks.

- List one or more API protocols besides REST and gRPC that you think are suitable for this scenario and provide your rationale.

One alternative to gRPC or REST API protocols that we could have used for this scenario is JSON-RPC. This protocol provides a straightforward version of request-response interaction between the two web services using JSON data. This is suitable as it is lighter weight than the gRPC protocol, simplicity of JSON data structure, but mains the same efficiency.

- 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 did not conduct profiling to measure the network latency between the QR code and authentication services. However, we did a batch scrape for the different deployment configurations and monitored the throughput for each.

- (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?

We only implemented the API in gRPC protocol. We chose to stick with gRPC because after reading the primers, we knew it would perform the fastest for the authentication service.

- (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?

We found this task useful in terms of understanding microservice communication. We learned a lot about resilience and error handling and how they effect performance optimization for distributed services.

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 implemented a horizontally scaled microservice architecture with separate deployments for each services and external Load Balancers for the QR Code service. Our system has service isolation which means that each service has its own deployment and service replicas of the same type can share a node, but different microservices do not.

- Did you put your QR Code Service container and the authentication server container into the same pod or different pods? Why?

We placed our QR Code Service and authentication in different pods. We did this to allow for independent scaling of the different services. Our architecture also ensures resource isolation which prevents faults in the QR service from affecting the authentication service and vice versa.

- Did you put your Twitter Service container and MySQL container into the same pod or different pods? Why?

We did not deploy the twitter service, however if we had, we would have placed our Twitter Service and MYSQL database in different pods to allow for independent scaling and to ensure resource isolation.

- What alternative architectures can you think of? What are their implications on performance and scalability?

Other web-tier architectures might include a node-specialized architecture where each node pool has a unique instance type, or a service mesh architecture, where each service has its own proxy server for handling those requests. The mesh architecture would be more sophisticated to scale because of the complexity of request routing, and also might reduce performance (increase latency) due to an additional resource overhead for each service pod. The node-specialization, would likely increase performance and scalability. Specializing resources allocated to each service would allow us to petter optimize performance of each service despite differences in operations, this means that we would not necessarily have to scale horizontally as much for the different services and would allow us to handle more requests while staying on budget. However it would also likely increase the operational complexity of the system.

- Explain your choice of instance type and the number of worker nodes for your Kubernetes cluster.

Our final implementation for phase one includes m8 large instances (one master and 7 worker nodes). We came to this decision after trying different methods for improving performance (code efficiency review and load balancing changes). When we adjusted our instance type from m5 to m8 we got almost double the performance. This architecture allows us to stay within the specified budget, but has sufficient resources to handle our microservice workload. It allows us high availability and fault tolerance, which increases the load traffic we can handle.

- (Optional) Have you tried more than one architecture or cluster configuration? What did their performance look like? Can you reason about the difference?

We did experiments with different numbers of worker nodes and pod replicas. After analyzing the CPU usage, we saw that we were not getting high enough CPU utilization (<100%) when the number of pod replicas was less than the number of cores of the cluster . The amount of replicas we have should be a multiple of the number of cores cpu has, This means that one process will be running per core which maximizes parallelism.

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.

Each time we deployed our system, we use the bash script (deploy.sh) to deploy. It takes the following steps

1) Provision infrastructure resources
2) Set up kubernetes cluster with kops (k8s/cluster)
3) Deploy applications (k8s/helm)

- If Terraform, Kubernetes, and Helm were not required, what are some other ways to automate your web-tier deployment? Compare their advantages and disadvantages

If Terraform, Kubernetes, and Helm were not required, we could automate our webservice using services like the AWS cloud development kit, which is an open source framework which helps model and provision cloud application resources. Some of the advantages to this are that it supports common programming languages (java and python) and allows for different abstractions of constructs for developer control of the deployment. One major disadvantage is that this is an AWS only tool, so we could not combine different cloud services for different applications if we wanted to. Another automation tool is to use Dockers native container orchestration tools only. These are easier to learn and can be integrated with different cloud service providers. They also have low operational overhead. Some of the disadvantages are limited scaling capabilities for larger deployments and less robust orchestration features than using Kubernetes directly.

- (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.

We used shell scripts (seen on our REPO) for deployment and test running for our web service development. Additionally a CI/CD pipeline is an essential tool for having automated deployment that adjusts to changes published to the master repository. Adding automated monitoring would be another step that we could take to improve the functionality of our project.

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

We modeled our framework for ETL off the P3 and P4 individual projects. We used scala spark to process the raw twitter data, cleaning and transforming the interaction and hashtag scores. before loading it into a mysql database. To stream the data for processing, we used aws api to get the data from the s3 bucket. Since we had to use scala spark for this process, we had to use the net.http package to deploy our web endpoint for this part of the project.

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

We were unsuccessful at developing a deployable recommendation service so we do not have these results. However, we likely would have used r series instances since they are memory efficient.

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

We were unsuccessful at developing a deployable recommendation service so we do not have these results.

- The time spent loading your data into the database

We were unsuccessful at developing a deployable recommendation service so we do not have these results.

- The overall cost of the ETL process

We were unsuccessful at developing a deployable recommendation service so we do not have these results.

- The number of incomplete ETL runs before your final run

We were unsuccessful at developing a deployable recommendation service so we do not have these results.

- The size of the resulting database and reasoning

We were unsuccessful at developing a deployable recommendation service so we do not have these results.

- The size and format of the backup

We were unsuccessful at developing a deployable recommendation service so we do not have these results.

- Discuss the difficulties encountered

We worked locally on the E&T phases of the data before attempting to move onto the larger dataset. The streaming portion of the process for getting the data loaded was more difficult because in the local implementation, we did a transformation using the fully cleaned tweet table but to match that, we would need to do it with the streaming, which was not as easy or efficient.

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

We only loaded the cleaned tweet table into the database once, the other tables were results of transformations using the existing table.

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 were unsuccessful at developing a deployable recommendation service so we do not have these results. However, we planned to use HDInsight, as we already have a reference from P3.

- What are the most effective ways to speed up ETL?

The two most effective ways to speed up ETL are to improve data architecture and parallelization strategies. Proper data partitioning and using incremental processing allows for better parallelization as smaller chunks can be processed in parallel. Additionally, you can manually control parallelization when using frameworks Spark or databricks.

- 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.

Apache Flink and Beam are alternate programming models that can be used for ETL. Apache beam would likely be more efficient than Flink because it provides higher development and runtime efficiency based in SQL skills. If we had to do real time streaming (adding new batches of data points to the database with each query) then Flink would be better for this project.

- 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?

We were unsuccessful at developing a fully functioning recommendation service so we do not have these results. We did not make it to running the ETL job in the cluster.

- 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 tried to process and load the tweets into the database in a stream. We used spark's built in JDBC connector to write processed data directly to the database. Alternatively, we could have build a temporary load data file as the result of each processing batch and the data into the database using LOAD DATA INFILE. Advantages of this method are that it provides faster row-by-row jdbc inserts and reduces the database connection overhead. Disadvantages are that it required a n additional storage block for the staging files, and a more complex transaction management. Another method for loading is to batch our data inserts into the database. This would require lower memory than writing the full data frame and no intermediate file storage, however it is still slower than the bulk loading method and requires more careful tuning of the batch sizes to optimize performance.

- Did you store any intermediate ETL results and why? If so, what data did you store, and where did you store the data?

When we first attempted ETL locally, we stored the full results (cleaned tweets and temporary scores [interaction * hashtag]) in tsvs and then loaded them into the database. However, this would take up too much space to do the full data set so we transitioned to a streaming approach.

- 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]

We were unsuccessful at developing a fully functioning recommendation service so we do not have these results. We were unable to fully process the full dataset for hashtag frequency

- 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 were unsuccessful at developing a fully functioning recommendation service, so we do not have these results. We were unable to process the dataset or build tests for coverage.

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 were unsuccessful at developing a fully functioning recommendation service. However, our database will likely have the schema:

```
TABLE `tweets` (
  `tweet_id` VARCHAR(30) NOT NULL,
  `user_id` VARCHAR(30) DEFAULT NULL,
  `created_at` TIMESTAMP DEFAULT NULL,
  `text` LONGTEXT DEFAULT NULL,
  `in_reply_to_user_id` VARCHAR(30) DEFAULT NULL,
  `retweeted_user_id` VARCHAR(30) DEFAULT NULL,
  `hashtags` LONGTEXT DEFAULT NULL,
  PRIMARY KEY (tweet_id)
);

TABLE `scores` (
  `id` INT AUTO_INCREMENT PRIMARY KEY,
  `user_id1` VARCHAR(30) DEFAULT NULL,
  `user_id2` VARCHAR(30) DEFAULT NULL,
  `score` DECIMAL(10,5),
  UNIQUE KEY user_pair (user_id1, user_id2)
);
```

- List 2 potential optimization that can be applied to your schemas. How might they improve your performance?

One optimization we could have is to add strategic indexing (by timestamp and interaction tuple) and partitioning of the tweets table. This would make it faster to access specific tweets for certain user interactions:

```
ALTER TABLE tweets ADD INDEX idx_user_interactions (user_id, in_reply_to_user_id,
retweeted_user_id);
PARTITION BY RANGE (UNIX_TIMESTAMP(created_at))
```

Another optimization might be to add a table that is indexed by hashtag, This would also provide us with a faster way of executing the recommendation queries.

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.

Here's an extensive list of potential performance optimizations across various layers of the technology stack: We cannot provide a table, as we did not come up with a deployable twitter service

Kubernetes & Infrastructure

1. **Resource requests/limits tuning** - Adjust CPU/memory allocations to better match workload patterns
2. **HorizontalPodAutoscaler configuration** - Optimize scaling thresholds and metrics
3. **Storage class selection** - Choose appropriate storage performance tiers
4. **Cluster autoscaler settings** - Optimize node provisioning behavior
5. **Node pool configuration** - Use appropriate instance types

Web Framework & Application Code

6. **Request timeout settings** - Prevent long-running requests
7. **Caching strategies** - Add in-memory caches for frequent data
8. **Batch processing** - Consolidate multiple similar operations
9. **Code hot paths optimization** - Refactor critical sections

Database & Storage

10. **Index optimization** - Add or refine database indexes
11. **Query optimization** - Rewrite inefficient queries
12. **Database partitioning** - Split large tables by logical boundaries
13. **Prepared statements** - Use instead of dynamic queries
14. **Database schema optimization** - Denormalize or normalize as needed

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

Kubernetes & Infrastructure

1. **Resource Requests/Limits Tuning**: Setting optimal resource requests/limits can improve throughput by 10-30% by preventing both resource starvation and over-allocation

2. **HorizontalPodAutoscaler Configuration:** HPAs help maintain consistent response times by ensuring adequate resource capacity is available

3. **Storage Class Selection:** Database operations can be 3-10x faster on optimized storage classes due to lower I/O wait times

4. **Cluster Autoscaler Settings:** Can prevent performance degradation during growth periods by having capacity ready before it's critically needed

5. **Node Pool Configuration**: Can improve price/performance ratio by 20-40% by matching hardware characteristics to workload requirements

Web Framework & Application Code

6. **Request Timeout Settings:** Prevents cascading failures under load by ensuring resources are freed up

7 Caching Strategies**:** Reduces database load for frequently accessed data; Can reduce response times

8. Batch Processing: Can improve throughput by 3-10x for operations that can be batched due to reduced per-item overhead

9. Code Hot Paths Optimization**:** Reduces computational complexity in critical sections

Database & Storage

10. Index Optimization: Eliminates runtime sorting when indexes match query order

11. Query Optimization: Restructuring queries to use more efficient plans; Can reduce query execution time

12. Database Partitioning: Enables multiple CPUs to process different partitions simultaneously

13. Prepared Statements**:** Eliminates repeated query parsing and planning

17. Database Schema Optimization: Reduces joins for read-heavy workloads, Using appropriate data types reduces storage and improves comparison speed

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 were unsuccessful at developing a fully functioning recommendation service. So we did not have a worked out system for deploying the storage 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?

Methods for making database snapshots are to use “mysqldump” which backs up the database at a given point. You can restore from that point using a command like “mysql twitter_db < twitter_db_snapshot.sql”. If we were to use an EBS volume to create a snapshot, using a command like this: “aws ec2 create-snapshot --volume-id vol-1234567890abcdef0 --description "Twitter DB Snapshot"” You can also restore the volume from this point. We didn’t run the etl process on the database so we do not know how much timw sould be saved exactly, but it would be calculated by (time to import[ETL] – time to restore[snapshot])

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?

CloudWatch metrics that are useful for monitoring the health of the system are pod status, load balancer health or server errors which can help identify where failures are occurring in the system and causing lags in performance. CloudWatch metrics that are useful for monitoring the performance of the system are resource utilization and request processing metrics like CPU or memory utilization and request count or latency.

- What statistics can you collect if you have SSH access to the VM? What tools did you use? What did they tell you?

When you have SSH access you can collect detailed system-level statistics. These include resource utilization (CPU or memory), disk and I/O performance, and network performance. We collected these statistics using tools like **top/htop** which told us about the per-core utilization, memory usage, and process completion statistics, **iostat** which told us about the speed or read write operations and i/o requests, **df** which informed the disk usage, and **iftop** which provides information about bandwidth usage and connection activity

- 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 you have Kubectl access but not SSH access you can collect statistics on resource utilization, pod health and status, and specific error logging information. When we were developing out system, we looked at these statistics using Kubernetes command line operations like kubectl describe, kubectl get, kubectl logs, and kubectl top. These helped us identify issues with scheduling and container stopping and starting, the limitations of our resources (cpu/memory limits) and specific application errors and container output. All of this informed our decisions about service architecture and resource choices.

- How do you monitor the status of your database? What noteworthy insights did you gain?

We were unsuccessful in deploying the database, so we did not do monitoring of the database for this phase.

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

Measurement: AWS CloudWatch metrics for ALB/NLB latency
Tool: aws elbv2 describe-load-balancers --names <your-lb-name>

  b Load Balancer to Worker Node –

Measurement: AWS CloudWatch load balancer metrics for target response time
Tool: aws elbv2 describe-target-health

  c Worker Node to Web-tier Pod

Measurement: Check kubelet logs for scheduling issues
Tool: kubectl logs -n kube-system kubelet-<instance-id>

  d Parsing request

Measurement: Server logging with timing information
Tool: kukbectl logs <web-tier-pod> | grep "request_processing_time"

  e Web Service to DB

Measurement: Application connection pool monitoring
Tool: time mysql -h <db-host> -u <user> -p<password> -e "SELECT 1"

  f At DB (execution)

Measurement: Performance schema metrics in MySQL
Tool: kubectl exec -it <db-pod> -- mysql -e "SET GLOBAL slow_query_log = 'ON'; SET GLOBAL long_query_time = 0.5;"

  g DB to Web Service

Measurement: Network utilization metrics on database server
Tool: kubectl exec -it <web-tier-pod> -- tcpdump -i eth0 host <db-host> -w /tmp/db-traffic.pcap

  h Parsing DB response

Measurement: Memory utilization during result processing
Tool: kubectl top pod <web-tier-pod> --containers

  i Web-tier Pod to Worker Node

Measurement: Container runtime network metrics
Tool: kubectl exec -it <web-tier-pod> -- iftop

  j Worker Node to Load Balancer

Measurement: CloudWatch metrics for EC2 network performance
Tool: aws elbv2 describe-load-balancer-attributes --load-balancer-arn <lb-arn>

  k Load Balancer to Load Generator

Measurement: CloudWatch load balancer metrics for processing time
Tool: aws elbv2 modify-load-balancer-attributes --load-balancer-arn <lb-arn> --attributes Key=access_logs.s3.enabled,Value=true

- For each part above, think about whether you can improve the latency or not, and list possible ways to reduce the latency.

a Load Generator to Load Balancer

Cannot Optimize with project constraints

b Load Balancer to Worker Node –

Optimization: Optimize load balancer health check intervals

c Worker Node to Web-tier Pod

Optimization: Optimize container network interface config.

d Parsing request

Optimization: Implement more efficient parsing algorithms

e Web Service to DB

Optimization: Implement database proxies with query caching

f At DB (execution)

Optimization: Optimize query structure and JOINs

g DB to Web Service

Optimization: Scale network interface capacity on database instances

h Parsing DB response

Optimization: Use more efficient data structures for response handling

i Web-tier Pod to Worker Node

Optimization: configure appropriate pod resource limits

j Worker Node to Load Balancer

Optimization: Optimize response size and compression

k Load Balancer to Load Generator

Cannot Optimize with project constraints

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.

In our CD workflow, we run the deployment of the services. The steps of this job are to configure the AWS credentials and then build and push all docker images for the micro services. Next, the job installs the Kops dependency so that we can deploy our cluster with our helm chart. The steps in this workflow are primarily sequential, however updates to the cluster state are made automatically if changes are made to the cluster.yaml or any helm chart file.
In the CI workflow, we run the service tests. This is a simple workflow that uses the go test command to run the tests we wrote.

- Briefly describe how you implement test cases for each microservice and how to execute these tests without manual operations

We use golang to write our micro services. This allows us to run tests suites using the “go test” command. We developed tests for our services using that built in testing package.

- 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?

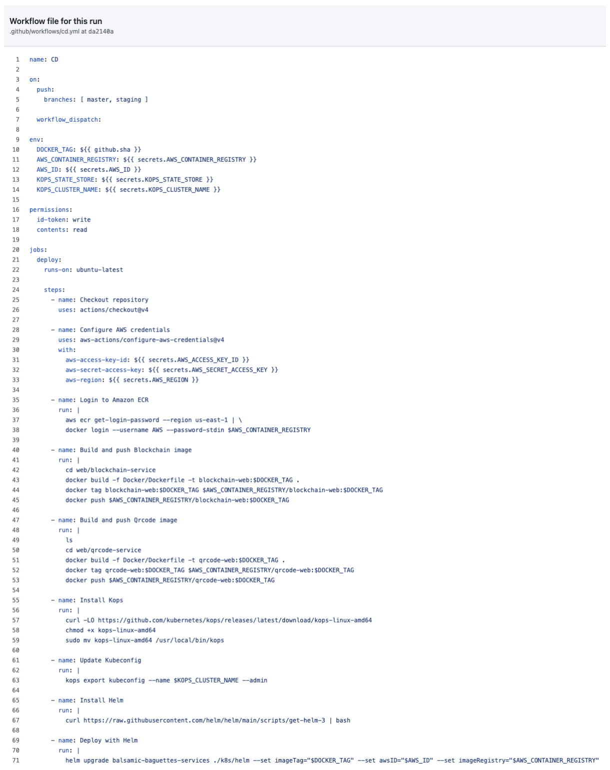
We developed a bash script found in scrips/deploy.sh that allows us to initialize the cluster and the helm release. This way we can delete the cluster every time we are going to to take a long break, saving us budget.

- Did you encounter any difficulties while working on the CI/CD tasks? If so, please describe the challenges and how you eventually resolved them.

Since the th-actons/changed-file action was comprised, we had to build and update the cluster each time an action is run rather than just restarting specific changed services.

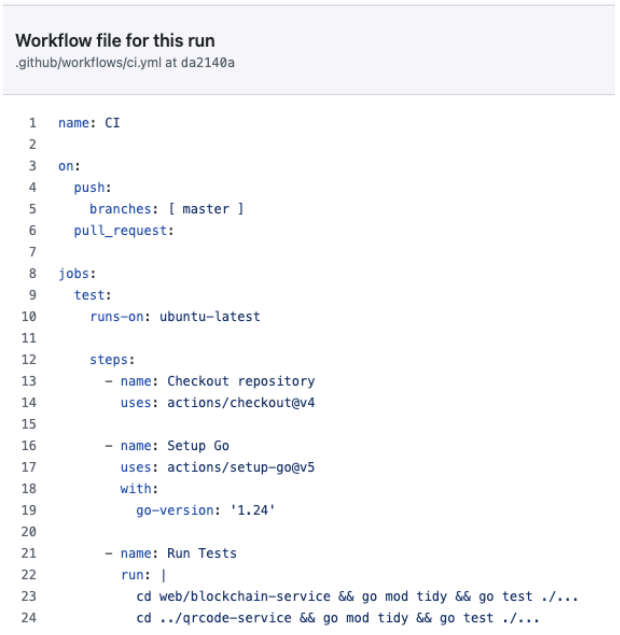
- Please attach your workflow YAML files and the following screenshots to this question: (1) the "Actions" tab of your CloudComputingTeamProject GitHub repository; (2) the "Summary" tab of a workflow; (3) Execution results for each job in a workflow.

YAML:



Workflow file for this run
.github/workflows/cd.yml at da2140a

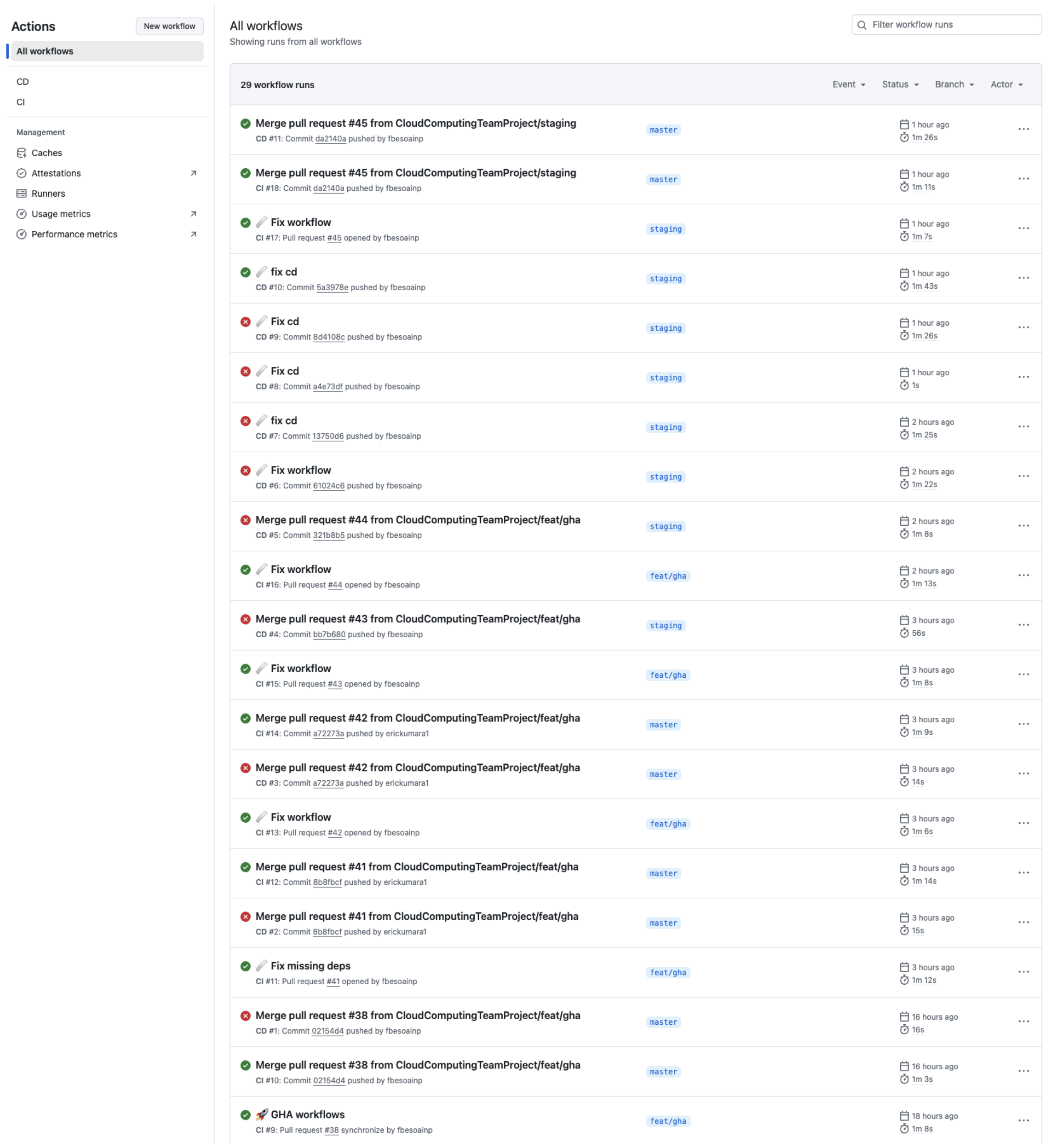
```yaml
name: CD

on:
  push:
    branches: [ master, staging ]

  workflow_dispatch:

env:
  DOCKER_TAG: ${{ github.sha }}
  AWS_CONTAINER_REGISTRY: ${{ secrets.AWS_CONTAINER_REGISTRY }}
  AWS_ID: ${{ secrets.AWS_ID }}
  KOPS_STATE_STORE: ${{ secrets.KOPS_STATE_STORE }}
  KOPS_CLUSTER_NAME: ${{ secrets.KOPS_CLUSTER_NAME }}

permissions:
  id-token: write
  contents: read

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Configure AWS credentials
        uses: aws-actions/configure-aws-credentials@v4
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: ${{ secrets.AWS_REGION }}

      - name: Login to Amazon ECR
        run: |
          aws ecr get-login-password --region us-east-1 | \
          docker login --username AWS --password-stdin $AWS_CONTAINER_REGISTRY

      - name: Build and push Blockchain image
        run: |
          cd web/blockchain-service
          docker build -f Docker/Dockerfile -t blockchain-web:$DOCKER_TAG .
          docker tag blockchain-web:$DOCKER_TAG $AWS_CONTAINER_REGISTRY/blockchain-web:$DOCKER_TAG
          docker push $AWS_CONTAINER_REGISTRY/blockchain-web:$DOCKER_TAG

      - name: Build and push Qrcode image
        run: |
          ls
          cd web/qrcode-service
          docker build -f Docker/Dockerfile -t qrcode-web:$DOCKER_TAG .
          docker tag qrcode-web:$DOCKER_TAG $AWS_CONTAINER_REGISTRY/qrcode-web:$DOCKER_TAG
          docker push $AWS_CONTAINER_REGISTRY/qrcode-web:$DOCKER_TAG

      - name: Install Kops
        run: |
          curl -LO https://github.com/kubernetes/kops/releases/latest/download/kops-linux-amd64
          chmod +x kops-linux-amd64
          sudo mv kops-linux-amd64 /usr/local/bin/kops

      - name: Update Kubeconfig
        run: |
          kops export kubeconfig --name $KOPS_CLUSTER_NAME --admin

      - name: Install Helm
        run: |
          curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash

      - name: Deploy with Helm
        run: |
          helm upgrade balsamic-baguettes-services ./k8s/helm --set imageTag="$DOCKER_TAG" --set awsID="$AWS_ID" --set imageRegistry="$AWS_CONTAINER_REGISTRY"
```



Workflow file for this run
.github/workflows/ci.yml at da2140a

```yaml
name: CI

on:
  push:
    branches: [ master ]
  pull_request:

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Setup Go
        uses: actions/setup-go@v5
        with:
          go-version: '1.24'

      - name: Run Tests
        run: |
          cd web/blockchain-service && go mod tidy && go test ./...
          cd ../qrcode-service && go mod tidy && go test ./...
```

ACTIONS:



Actions New workflow

All workflows

CD

CI

Management

Caches

Attestations

Runners

Usage metrics

Performance metrics

All workflows

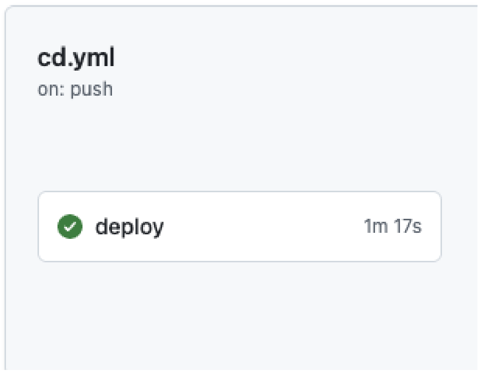
Showing runs from all workflows

Filter workflow runs

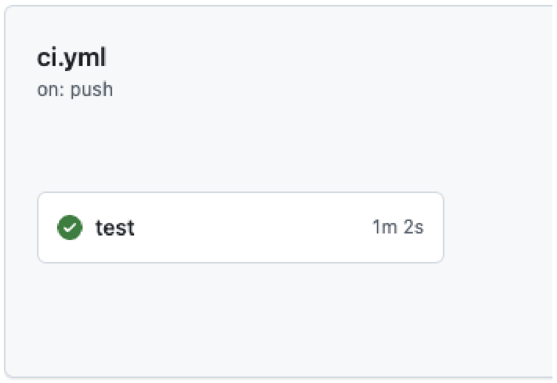
29 workflow runs Event Status Branch Actor

✓	Merge pull request #45 from CloudComputingTeamProject/staging CD #11: Commit da2140a pushed by fbesoainp	master	1 hour ago	1m 26s
✓	Merge pull request #45 from CloudComputingTeamProject/staging CI #18: Commit da2140a pushed by fbesoainp	master	1 hour ago	1m 11s
✓	Fix workflow CI #17: Pull request #45 opened by fbesoainp	staging	1 hour ago	1m 7s
✓	fix cd CD #10: Commit 5a3978e pushed by fbesoainp	staging	1 hour ago	1m 43s
✗	Fix cd CD #9: Commit 8d4108c pushed by fbesoainp	staging	1 hour ago	1m 26s
✗	Fix cd CD #8: Commit a4e73df pushed by fbesoainp	staging	1 hour ago	1s
✗	fix cd CD #7: Commit 13750d6 pushed by fbesoainp	staging	2 hours ago	1m 25s
✗	Fix workflow CD #6: Commit 61024c6 pushed by fbesoainp	staging	2 hours ago	1m 22s
✗	Merge pull request #44 from CloudComputingTeamProject/feat/gha CD #5: Commit 321b8b5 pushed by fbesoainp	staging	2 hours ago	1m 8s
✓	Fix workflow CI #16: Pull request #44 opened by fbesoainp	feat/gha	2 hours ago	1m 13s
✗	Merge pull request #43 from CloudComputingTeamProject/feat/gha CD #4: Commit bb7b680 pushed by fbesoainp	staging	3 hours ago	56s
✓	Fix workflow CI #15: Pull request #43 opened by fbesoainp	feat/gha	3 hours ago	1m 8s
✓	Merge pull request #42 from CloudComputingTeamProject/feat/gha CI #14: Commit a72273a pushed by erickumara1	master	3 hours ago	1m 9s
✗	Merge pull request #42 from CloudComputingTeamProject/feat/gha CD #3: Commit a72273a pushed by erickumara1	master	3 hours ago	14s
✓	Fix workflow CI #13: Pull request #42 opened by fbesoainp	feat/gha	3 hours ago	1m 6s
✓	Merge pull request #41 from CloudComputingTeamProject/feat/gha CI #12: Commit 8b8fbcf pushed by erickumara1	master	3 hours ago	1m 14s
✗	Merge pull request #41 from CloudComputingTeamProject/feat/gha CD #2: Commit 8b8fbcf pushed by erickumara1	master	3 hours ago	15s
✓	Fix missing deps CI #11: Pull request #41 opened by fbesoainp	feat/gha	3 hours ago	1m 12s
✗	Merge pull request #38 from CloudComputingTeamProject/feat/gha CD #1: Commit 02154d4 pushed by fbesoainp	master	16 hours ago	16s
✓	Merge pull request #38 from CloudComputingTeamProject/feat/gha CI #10: Commit 02154d4 pushed by fbesoainp	master	16 hours ago	1m 3s
✓	🚀 GHA workflows CI #9: Pull request #38 synchronize by fbesoainp	feat/gha	18 hours ago	1m 8s

SUMMARY:

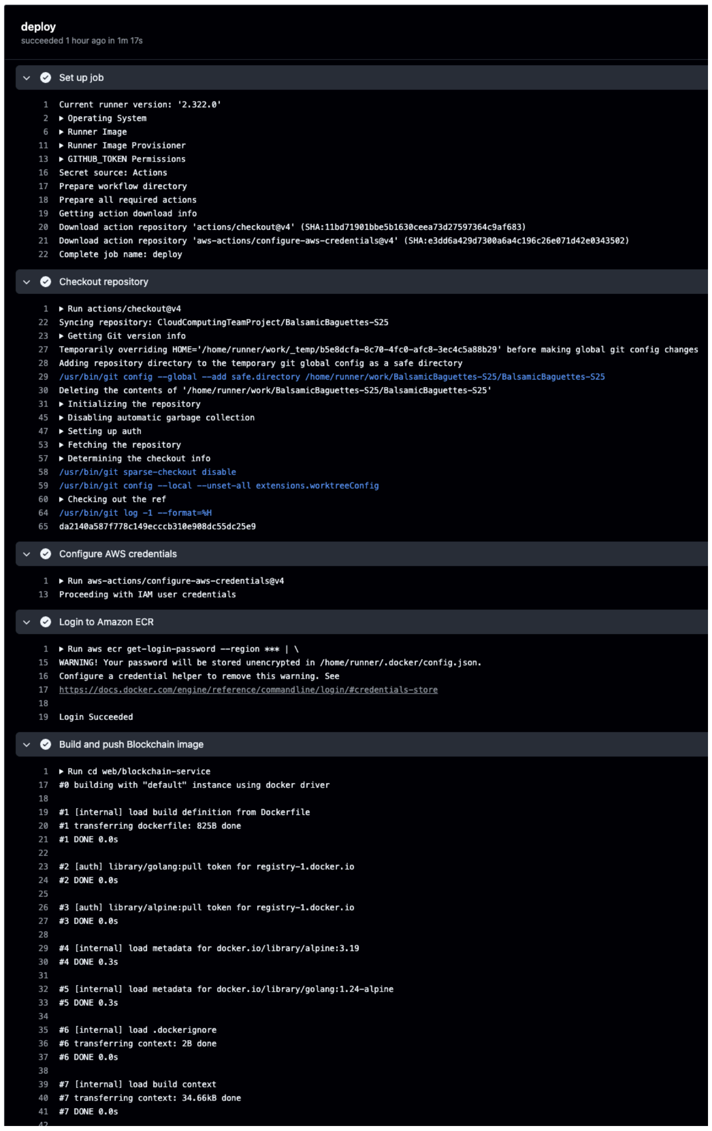

cd.yml
on: push

✓ deploy 1m 17s


ci.yml
on: push

✓ test 1m 2s

Execution Results:



deploy
succeeded 1 hour ago in 1m 17s

Set up job

```
Current runner version: '2.322.0'
▶ Operating System
▶ Runner Image
▶ Runner Image Provisioner
▶ GITHUB_TOKEN Permissions
Secret source: Actions
Prepare workflow directory
Prepare all required actions
Getting action download info
Download action repository 'actions/checkout@v4' (SHA:11bd71901bbe5b1630ceea73d27597364c9af683)
Download action repository 'aws-actions/configure-aws-credentials@v4' (SHA:e3dd6a429d7300a6a4c196c26e071d42e0343502)
Complete job name: deploy
```

Checkout repository

```
▶ Run actions/checkout@v4
Syncing repository: CloudComputingTeamProject/BalsamicBaguettes-S25
▶ Getting Git version info
Temporarily overriding HOME='/home/runner/work/_temp/b5e8dcfa-8c70-4fc0-afc8-3ec4c5a88b29' before making global git config changes
Adding repository directory to the temporary git global config as a safe directory
/usr/bin/git config --global --add safe.directory /home/runner/work/BalsamicBaguettes-S25/BalsamicBaguettes-S25
Deleting the contents of '/home/runner/work/BalsamicBaguettes-S25/BalsamicBaguettes-S25'
▶ Initializing the repository
▶ Disabling automatic garbage collection
▶ Setting up auth
▶ Fetching the repository
▶ Determining the checkout info
/usr/bin/git sparse-checkout disable
/usr/bin/git config --local --unset-all extensions.worktreeConfig
▶ Checking out the ref
/usr/bin/git log -1 --format=%H
da2140a587f778c149ecccb310e908dc55dc25e9
```

Configure AWS credentials

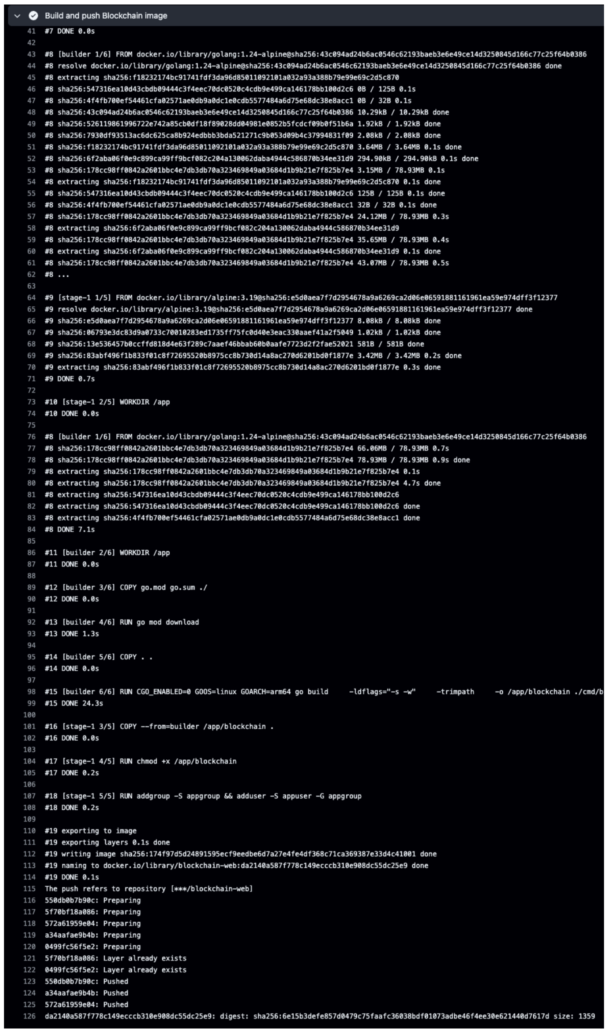
```
▶ Run aws-actions/configure-aws-credentials@v4
Proceeding with IAM user credentials
```

Login to Amazon ECR

```
▶ Run aws ecr get-login-password --region *** | \
WARNING! Your password will be stored unencrypted in /home/runner/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

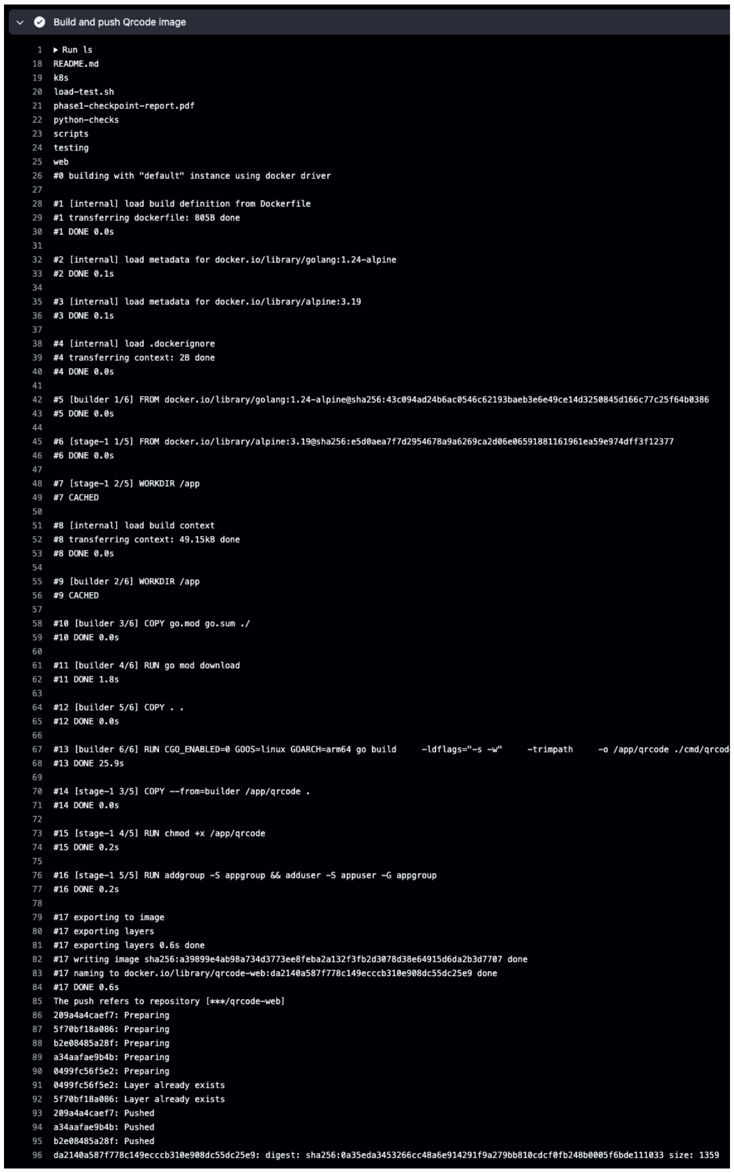
Build and push Blockchain image

```
▶ Run cd web/blockchain-service
#0 building with "default" instance using docker driver

#1 [internal] load build definition from Dockerfile
#1 transferring dockerfile: 825B done
#1 DONE 0.0s

#2 [auth] library/golang:pull token for registry-1.docker.io
#2 DONE 0.0s

#3 [auth] library/alpine:pull token for registry-1.docker.io
#3 DONE 0.0s

#4 [internal] load metadata for docker.io/library/alpine:3.19
#4 DONE 0.3s

#5 [internal] load metadata for docker.io/library/golang:1.24-alpine
#5 DONE 0.3s

#6 [internal] load .dockerignore
#6 transferring context: 2B done
#6 DONE 0.0s

#7 [internal] load build context
#7 transferring context: 34.66kB done
#7 DONE 0.0s
```



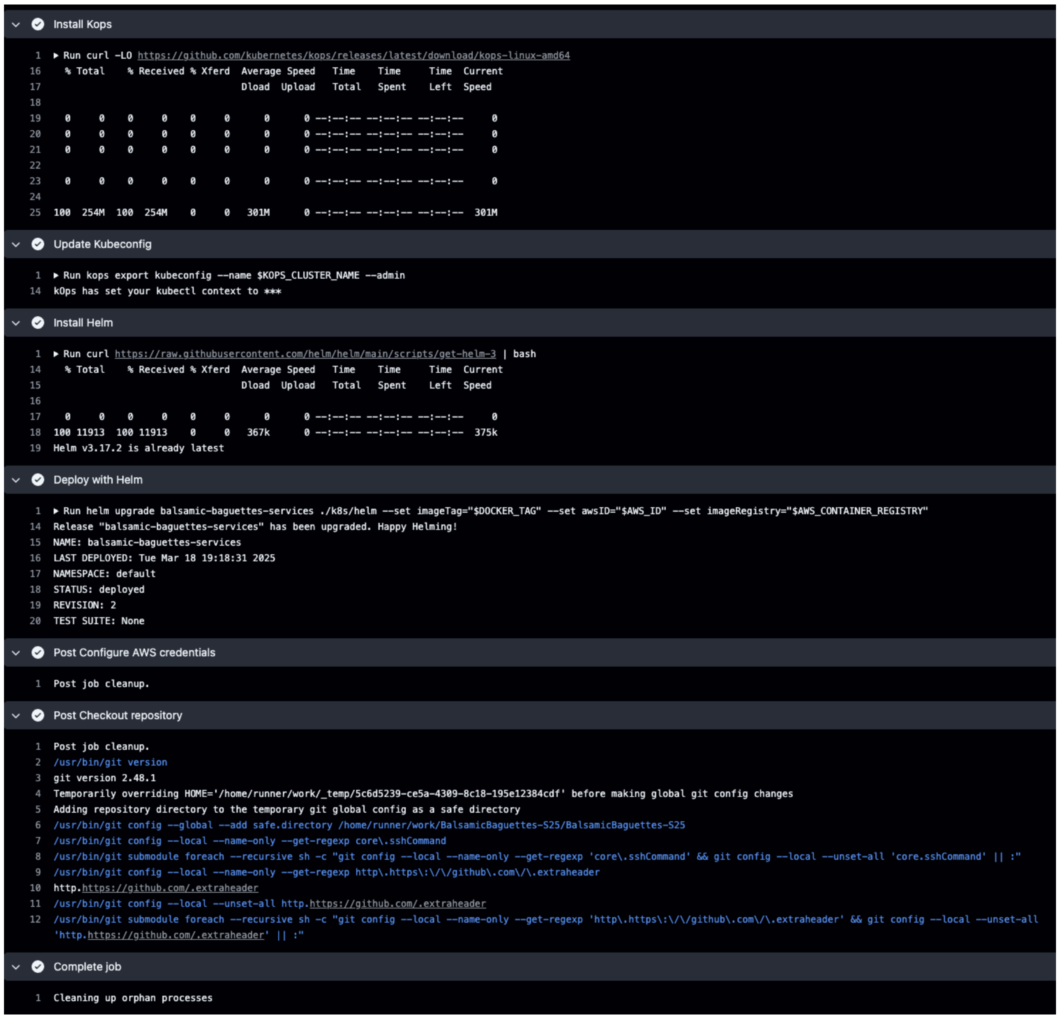
Build and push Blockchain image

```
#7 DONE 0.0s

#8 [builder 1/6] FROM docker.io/library/golang:1.24-alpine@sha256:43c094ad24b6ac0546c62193baeb3e6e49ce14d3250845d166c77c25f64b0386
#8 resolve docker.io/library/golang:1.24-alpine@sha256:43c094ad24b6ac0546c62193baeb3e6e49ce14d3250845d166c77c25f64b0386 done
#8 extracting sha256:f18232174bc91741fdf3da96d85011092101a032a93a388b79e99e69c2d5c870
#8 sha256:547316ea10d43cbdb09444c3f4eec70dc0520c4cdb9e499ca146178bb100d2c6 0B / 125B 0.1s
#8 sha256:4f4fb700ef54461cfa02571ae0db9a0dc1e0cdb5577484a6d75e68dc38e8acc1 0B / 32B 0.1s
#8 sha256:43c094ad24b6ac0546c62193baeb3e6e49ce14d3250845d166c77c25f64b0386 10.29kB / 10.29kB done
#8 sha256:526119861996722e742a85cb0df18f89028dd04981e0852b5fcdcf09b0f51b6a 1.92kB / 1.92kB done
#8 sha256:7930df93513ac6dc625ca8b924edbbb3bda521271c9b053d09b4c37994831f09 2.08kB / 2.08kB done
#8 sha256:f18232174bc91741fdf3da96d85011092101a032a93a388b79e99e69c2d5c870 3.64MB / 3.64MB 0.1s done
#8 sha256:6f2aba06f0e9c899ca99ff9bcf082c204a130062daba4944c586870b34ee31d9 294.90kB / 294.90kB 0.1s done
#8 sha256:178cc98ff0842a2601bbc4e7db3db70a323469849a03684d1b9b21e7f825b7e4 3.15MB / 78.93MB 0.1s
#8 extracting sha256:f18232174bc91741fdf3da96d85011092101a032a93a388b79e99e69c2d5c870 0.1s done
#8 sha256:547316ea10d43cbdb09444c3f4eec70dc0520c4cdb9e499ca146178bb100d2c6 125B / 125B 0.1s done
#8 sha256:4f4fb700ef54461cfa02571ae0db9a0dc1e0cdb5577484a6d75e68dc38e8acc1 32B / 32B 0.1s done
#8 sha256:178cc98ff0842a2601bbc4e7db3db70a323469849a03684d1b9b21e7f825b7e4 24.12MB / 78.93MB 0.3s
#8 extracting sha256:6f2aba06f0e9c899ca99ff9bcf082c204a130062daba4944c586870b34ee31d9
#8 sha256:178cc98ff0842a2601bbc4e7db3db70a323469849a03684d1b9b21e7f825b7e4 35.65MB / 78.93MB 0.4s
#8 extracting sha256:6f2aba06f0e9c899ca99ff9bcf082c204a130062daba4944c586870b34ee31d9 0.1s done
#8 sha256:178cc98ff0842a2601bbc4e7db3db70a323469849a03684d1b9b21e7f825b7e4 43.07MB / 78.93MB 0.5s
#8 ...

#9 [stage-1 1/5] FROM docker.io/library/alpine:3.19@sha256:e5d0aea7f7d2954678a9a6269ca2d06e06591881161961ea59e974dff3f12377
#9 resolve docker.io/library/alpine:3.19@sha256:e5d0aea7f7d2954678a9a6269ca2d06e06591881161961ea59e974dff3f12377 done
#9 sha256:e5d0aea7f7d2954678a9a6269ca2d06e06591881161961ea59e974dff3f12377 8.08kB / 8.08kB done
#9 sha256:06793e3dc83d9a0733c70010283ed1735ff75fc0d40e3eac330aaef41a2f5049 1.02kB / 1.02kB done
#9 sha256:13e536457b0ccffd818d4e63f289c7aaef46bbab60b0aafe7723d2f2fae52021 581B / 581B done
#9 sha256:83abf496f1b833f01c8f72695520b8975cc8b730d14a8ac270d6201bd0f1877e 3.42MB / 3.42MB 0.2s done
#9 extracting sha256:83abf496f1b833f01c8f72695520b8975cc8b730d14a8ac270d6201bd0f1877e 0.3s done
#9 DONE 0.7s

#10 [stage-1 2/5] WORKDIR /app
#10 DONE 0.0s

#8 [builder 1/6] FROM docker.io/library/golang:1.24-alpine@sha256:43c094ad24b6ac0546c62193baeb3e6e49ce14d3250845d166c77c25f64b0386
#8 sha256:178cc98ff0842a2601bbc4e7db3db70a323469849a03684d1b9b21e7f825b7e4 66.06MB / 78.93MB 0.7s
#8 sha256:178cc98ff0842a2601bbc4e7db3db70a323469849a03684d1b9b21e7f825b7e4 78.93MB / 78.93MB 0.9s done
#8 extracting sha256:178cc98ff0842a2601bbc4e7db3db70a323469849a03684d1b9b21e7f825b7e4 0.1s
#8 extracting sha256:178cc98ff0842a2601bbc4e7db3db70a323469849a03684d1b9b21e7f825b7e4 4.7s done
#8 extracting sha256:547316ea10d43cbdb09444c3f4eec70dc0520c4cdb9e499ca146178bb100d2c6
#8 extracting sha256:547316ea10d43cbdb09444c3f4eec70dc0520c4cdb9e499ca146178bb100d2c6 done
#8 extracting sha256:4f4fb700ef54461cfa02571ae0db9a0dc1e0cdb5577484a6d75e68dc38e8acc1 done
#8 DONE 7.1s

#11 [builder 2/6] WORKDIR /app
#11 DONE 0.0s

#12 [builder 3/6] COPY go.mod go.sum ./
#12 DONE 0.0s

#13 [builder 4/6] RUN go mod download
#13 DONE 1.3s

#14 [builder 5/6] COPY . .
#14 DONE 0.0s

#15 [builder 6/6] RUN CGO_ENABLED=0 GOOS=linux GOARCH=arm64 go build     -ldflags="-s -w"     -trimpath     -o /app/blockchain ./cmd/b
#15 DONE 24.3s

#16 [stage-1 3/5] COPY --from=builder /app/blockchain .
#16 DONE 0.0s

#17 [stage-1 4/5] RUN chmod +x /app/blockchain
#17 DONE 0.2s

#18 [stage-1 5/5] RUN addgroup -S appgroup && adduser -S appuser -G appgroup
#18 DONE 0.2s

#19 exporting to image
#19 exporting layers 0.1s done
#19 writing image sha256:174f97d5d24891595ecf9eedbe6d7a27e4fe4df368c71ca369387e33d4c41001 done
#19 naming to docker.io/library/blockchain-web:da2140a587f778c149ecccb310e908dc55dc25e9 done
#19 DONE 0.1s
The push refers to repository [***/blockchain-web]
550db0b7b90c: Preparing
5f70bf18a086: Preparing
572a61959e04: Preparing
a34aafae9b4b: Preparing
0499fc56f5e2: Preparing
5f70bf18a086: Layer already exists
0499fc56f5e2: Layer already exists
550db0b7b90c: Pushed
a34aafae9b4b: Pushed
572a61959e04: Pushed
da2140a587f778c149ecccb310e908dc55dc25e9: digest: sha256:6e15b3defe857d0479c75faafc36038bdf01073adbe46f4ee30e621440d7617d size: 1359
```



Build and push Qrcode image

```
▶ Run ls
README.md
k8s
load-test.sh
phase1-checkpoint-report.pdf
python-checks
scripts
testing
web
#0 building with "default" instance using docker driver

#1 [internal] load build definition from Dockerfile
#1 transferring dockerfile: 805B done
#1 DONE 0.0s

#2 [internal] load metadata for docker.io/library/golang:1.24-alpine
#2 DONE 0.1s

#3 [internal] load metadata for docker.io/library/alpine:3.19
#3 DONE 0.1s

#4 [internal] load .dockerignore
#4 transferring context: 2B done
#4 DONE 0.0s

#5 [builder 1/6] FROM docker.io/library/golang:1.24-alpine@sha256:43c094ad24b6ac0546c62193baeb3e6e49ce14d3250845d166c77c25f64b0386
#5 DONE 0.0s

#6 [stage-1 1/5] FROM docker.io/library/alpine:3.19@sha256:e5d0aea7f7d2954678a9a6269ca2d06e06591881161961ea59e974dff3f12377
#6 DONE 0.0s

#7 [stage-1 2/5] WORKDIR /app
#7 CACHED

#8 [internal] load build context
#8 transferring context: 49.15kB done
#8 DONE 0.0s

#9 [builder 2/6] WORKDIR /app
#9 CACHED

#10 [builder 3/6] COPY go.mod go.sum ./
#10 DONE 0.0s

#11 [builder 4/6] RUN go mod download
#11 DONE 1.8s

#12 [builder 5/6] COPY . .
#12 DONE 0.0s

#13 [builder 6/6] RUN CGO_ENABLED=0 GOOS=linux GOARCH=arm64 go build     -ldflags="-s -w"     -trimpath     -o /app/qrcode ./cmd/qrcod
#13 DONE 25.9s

#14 [stage-1 3/5] COPY --from=builder /app/qrcode .
#14 DONE 0.0s

#15 [stage-1 4/5] RUN chmod +x /app/qrcode
#15 DONE 0.2s

#16 [stage-1 5/5] RUN addgroup -S appgroup && adduser -S appuser -G appgroup
#16 DONE 0.2s

#17 exporting to image
#17 exporting layers
#17 exporting layers 0.6s done
#17 writing image sha256:a39899e4ab98a734d3773ee8feba2a132f3fb2d3078d38e64915d6da2b3d7707 done
#17 naming to docker.io/library/qrcode-web:da2140a587f778c149ecccb310e908dc55dc25e9 done
#17 DONE 0.6s
The push refers to repository [***/qrcode-web]
209a4a4caef7: Preparing
5f70bf18a086: Preparing
b2e08485a28f: Preparing
a34aafae9b4b: Preparing
0499fc56f5e2: Preparing
0499fc56f5e2: Layer already exists
5f70bf18a086: Layer already exists
209a4a4caef7: Pushed
a34aafae9b4b: Pushed
b2e08485a28f: Pushed
da2140a587f778c149ecccb310e908dc55dc25e9: digest: sha256:0a35eda3453266cc48a6e914291f9a279bb810cdcf0fb248b0005f6bde111033 size: 1359
```



Install Kops

```
▶ Run curl -LO https://github.com/kubernetes/kops/releases/latest/download/kops-linux-amd64
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed

  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0

  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0

100  254M  100  254M    0     0   301M      0 --:--:-- --:--:-- --:--:--  301M
```

Update Kubeconfig

```
▶ Run kops export kubeconfig --name $KOPS_CLUSTER_NAME --admin
kOps has set your kubectl context to ***
```

Install Helm

```
▶ Run curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed

  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100 11913  100 11913    0     0   367k      0 --:--:-- --:--:-- --:--:--  375k
Helm v3.17.2 is already latest
```

Deploy with Helm

```
▶ Run helm upgrade balsamic-baguettes-services ./k8s/helm --set imageTag="$DOCKER_TAG" --set awsID="$AWS_ID" --set imageRegistry="$AWS_CONTAINER_REGISTRY"
Release "balsamic-baguettes-services" has been upgraded. Happy Helming!
NAME: balsamic-baguettes-services
LAST DEPLOYED: Tue Mar 18 19:18:31 2025
NAMESPACE: default
STATUS: deployed
REVISION: 2
TEST SUITE: None
```

Post Configure AWS credentials

```
Post job cleanup.
```

Post Checkout repository

```
Post job cleanup.
/usr/bin/git version
git version 2.48.1
Temporarily overriding HOME='/home/runner/work/_temp/5c6d5239-ce5a-4309-8c18-195e12384cdf' before making global git config changes
Adding repository directory to the temporary git global config as a safe directory
/usr/bin/git config --global --add safe.directory /home/runner/work/BalsamicBaguettes-S25/BalsamicBaguettes-S25
/usr/bin/git config --local --name-only --get-regexp core\.sshCommand
/usr/bin/git submodule foreach --recursive sh -c "git config --local --name-only --get-regexp 'core\.sshCommand' && git config --local --unset-all 'core.sshCommand' || :"
/usr/bin/git config --local --name-only --get-regexp http\.https\:\/\/github\.com\/\.extraheader
http.https://github.com/.extraheader
/usr/bin/git config --local --unset-all http.https://github.com/.extraheader
/usr/bin/git submodule foreach --recursive sh -c "git config --local --name-only --get-regexp 'http\.https\:\/\/github\.com\/\.extraheader' && git config --local --unset-all
'http.https://github.com/.extraheader' || :"
```

Complete job

```
Cleaning up orphan processes
```

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 architecture would likely scale well for a 10x increase in data because our services are independently scalable because of their containerization and we use load balancers to properly distribute traffic. However, we would likely face significant challenges if the data increased by 100 because at that point, simple scaling wouldn't address the access pattern issues.At that point, adding some kind of caching layer for the twitter service would be essential and we do not have that.

- Would your design work if your web service also implemented insert/update (PUT) requests? Why or why not?

We did not complete the twitter service, but our current implementation would need significant changes to support write operations. We currently don't have any mechanisms built in to add writes into the system. Additionally, concurrency would become an extreme problem if write operations were allowed for the system. We do not have anything implemented to manage concurrent writes because concurrent reads on distributed systems are significantly less complex.

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 did a lot of local testing before we deployed our services on cloud clusters. For local testing, we build dependency scripts to ensure we had all the necessary packages set up to mirror those which would be loaded into the cluster on deployment.

- Did you attempt to generate a load to test your system on your own? If so, how? And why?

We primarily used the load of the reference servers to test our services. This helped us save on development costs since we didn't have to submit to SAIL each time.

- Describe an alternative design to your system that you wish you had time or budget to try.

In an alternative system design. We might have used application load balancers to change routing capabilities. We may have also placed related applications in the same containers (like qr and authentication) rather than separating them. It is likely that this would not have improved performance since they could no longer be scaled independently, but it might have reduced communication time between those services. We simply did not have the time to test and build that. Additionally, if we had the budget for it, we would have used larger more powerful instances, reduced processing time for queries and request fulfillment.

- Which were the toughest roadblocks that your team faced in Phase 1?

Time was our biggest challenge for this project. All three of our team members have a heavy course load this semester, and as we got closer to the final phase1 deadline, requirements for other courses and the individual projects for this class increased significantly. This made it really difficult for us to get together and dedicate a lot of time to development.
In terms of technical problems, before transitioning from M5 instances to M8 instances, performance was a major roadblock for our team. We tried several different optimizations for performance like changing load balancer type, scaling policies , and web framework.

- Did you do something unique (any cool optimization/trick/hack) that you would like to share?

For the QR Service we were stuck with around 20k until we recorded the time for each function to execute locally with test cases. We noticed that some functions that did string conversions were extremely slow (~18 us) where other functions took 3 us. Finding alternative ways to do the string conversion without using sprintf functions decreased the execution time which led to a higher throughput in the end.

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 generally followed the same workload split in the second half of the project as we did in the first. Francisco focused on deployment optimization, Eric focused on logical operations for each service, and Madison set up web endpoints and query handlers. For the Recommendation service, Things were split slightly differently with Eric developing an initial logical framework for the data transformation and Madison working on the streaming and loading of that data into a database for submission.

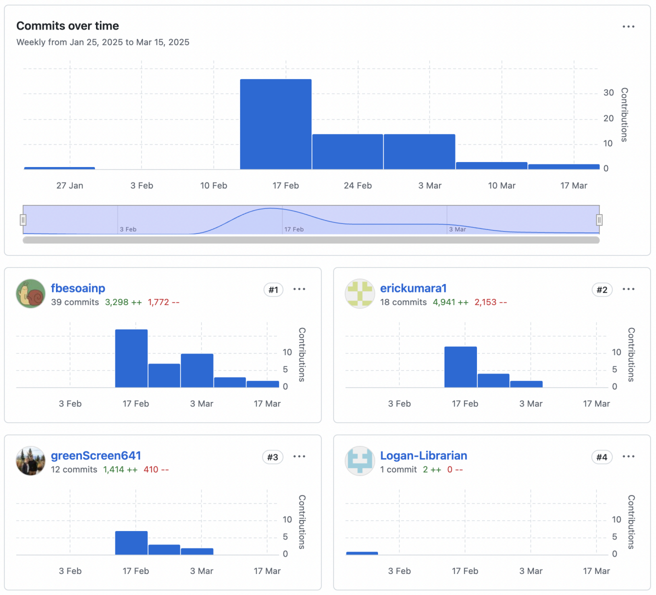
- How did you test each component? Did you implement Unit Testing or Integration Testing?

We did a lot of correctness validation testing. (Eric Here) I wasn't used to test development so when creating the logic I made unit tests in a main module to check if each function developed produced an expected output. However, after finding the testing module of go, I leaned more towards integration testing for blockchain and QR based on the reference sample. Creating new cases was difficult (especially for blockchain), so I fed back some of the wrong output in the sail feedback for future tests to see if new implementations would pass it.

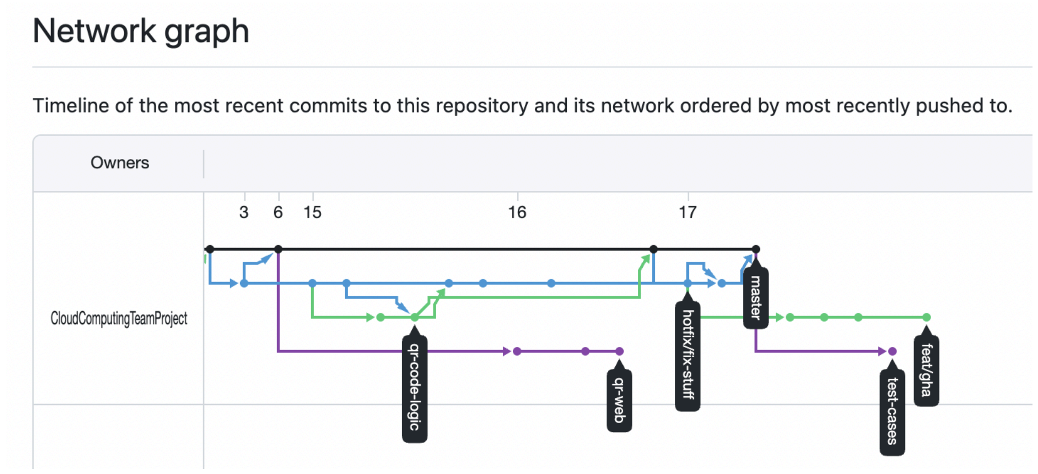
- Did you like the Git workflow, code review, and internal testing requirements? Were they helpful to your development?

Overall, the git workflow was good for keeping all the pieces of the system separate. It helped us ensure that there were no major bugs pushed into the main repository which would derail all the branches. The code review was also very useful for making sure our code was modular and written well.

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


Network graph
Timeline of the most recent commits to this repository and its network ordered by most recently pushed to.
Owners
CloudComputingTeamProject
3
6
15
16
17
qr-code-logic
qr-web
hotfix/fix-stuff
master
test-cases
feat/gha

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., 2025-02-17 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., 2025-02-17 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.