Team 3 - Milestone 3

Containerization

As the project grew in complexity and needs, we switched to using docker-compose to build and deploy our service with several containers instead of just one. So, in addition to the container for the main flask API inference service, Docker hosts containers for

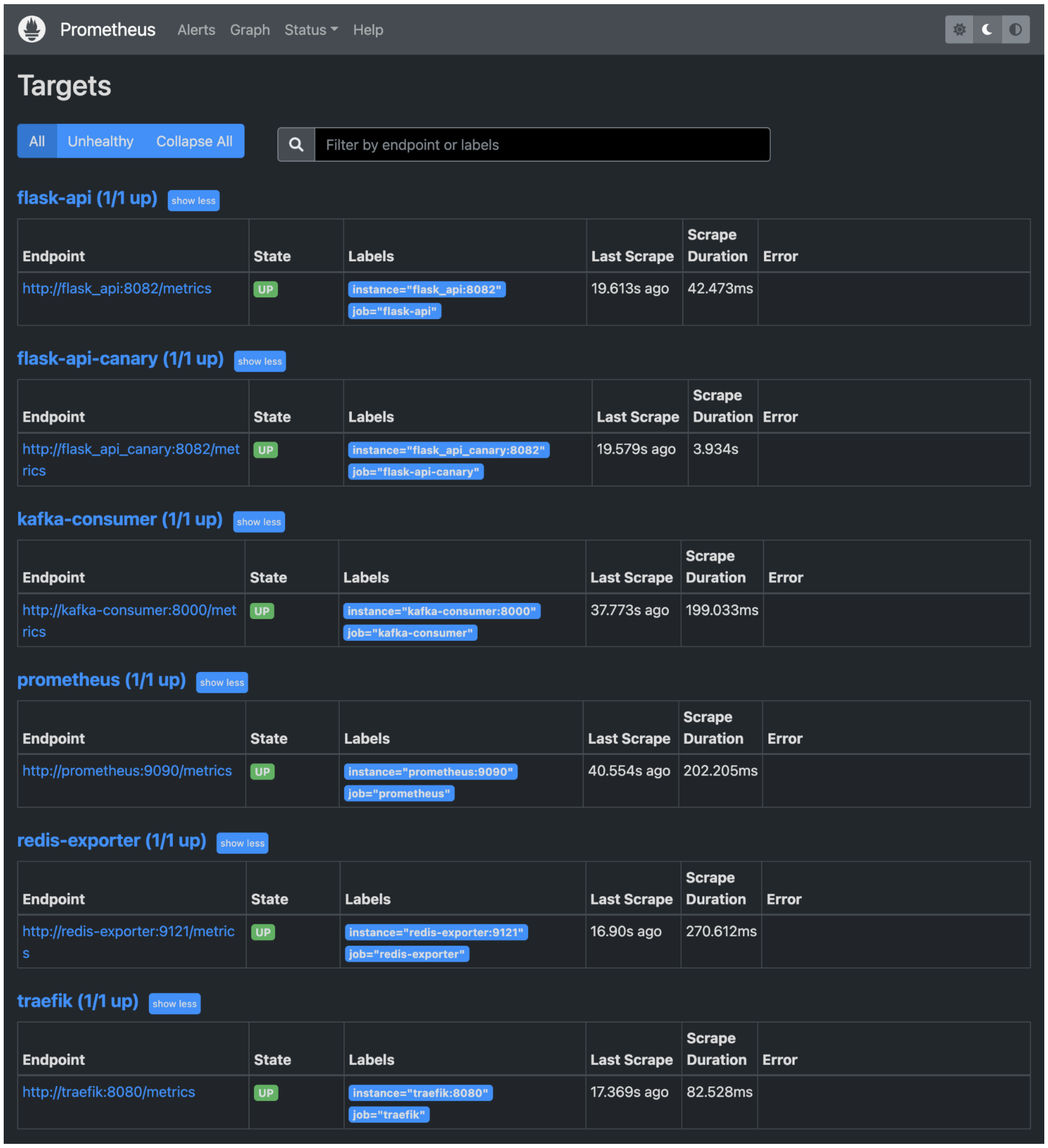
1. A Canary release [code]: for releasing canary versions of our API
2. Nginx fallback service [code, config]: for handling requests when our API is down, ensuring maximum uptime of our service
3. Traefik load balancer (fall2022-comp585-3.cs.mcgill.ca:8080/) [code, config]: for guiding traffic between our containers based on the configuration provided and the health of each service
4. A Kafka consumer [code]: for continuously monitoring the Kafka stream for errors
5. Redis [code]: for storing metrics data in the application
6. Grafana (fall2022-comp585-3.cs.mcgill.ca:3000/) [code]: for monitoring metrics across the entire application stack
7. Prometheus (fall2022-comp585-3.cs.mcgill.ca:9090/) [code]: for scraping the metrics from the relevant services
8. Alermanager (fall2022-comp585-3.cs.mcgill.ca:9093/) [code]: for sending out automated Slack alerts during outages
9. Redis-exporter [code]: for letting us monitor the database health from Prometheus

These containers are created when we run build tests [here], create canary releases [here], and deploy the project to the team server [here] from GitHub Actions in our CI/CD pipelines. The only stage in our CI/CD pipelines where the docker containers aren't built is for running unit/lint tests since they can run in isolation even when the rest of the services aren't active.

Monitoring

As briefly discussed, our monitoring stack consists of Grafana visualizations created with data from Prometheus and Redis and active alerting using Alermanager (with Slack integration).

At the top of our monitoring stack, we're using Prometheus to scrape metrics from our core services. This includes the Docker containers for the main flask API, the canary API, Kafka consumer, Prometheus itself, Redis, and Traefik load balancer:



Targets

flask-api (1/1 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://flask_api:8082/metrics	UP	instance="flask_api:8082" job="flask-api"	19.613s ago	42.473ms	

flask-api-canary (1/1 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://flask_api_canary:8082/metrics	UP	instance="flask_api_canary:8082" job="flask-api-canary"	19.579s ago	3.934s	

kafka-consumer (1/1 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://kafka-consumer:8000/metrics	UP	instance="kafka-consumer:8000" job="kafka-consumer"	37.773s ago	199.033ms	

prometheus (1/1 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://prometheus:9090/metrics	UP	instance="prometheus:9090" job="prometheus"	40.554s ago	202.205ms	

redis-exporter (1/1 up) show less

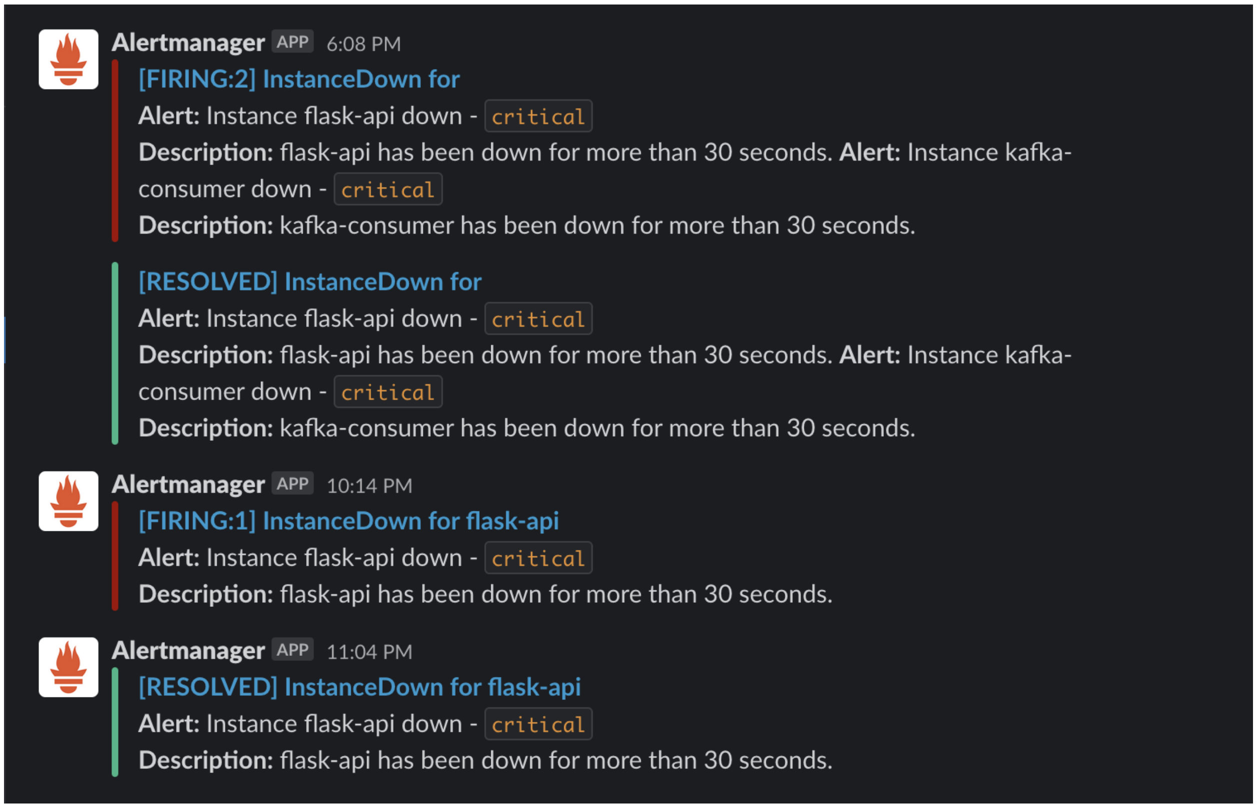
Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://redis-exporter:9121/metrics	UP	instance="redis-exporter:9121" job="redis-exporter"	16.90s ago	270.612ms	

traefik (1/1 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://traefik:8080/metrics	UP	instance="traefik:8080" job="traefik"	17.369s ago	82.528ms	

Prometheus scrapes health data from each of these services and monitors uptime at 30s intervals; full Prometheus configuration here.

In cases of service outages, Prometheus notifies Alertmanager, which then sends out Slack notifications to #group_3-alerts channel:



Alertmanager APP 6:08 PM

[FIRING:2] InstanceDown for

Alert: Instance flask-api down - `critical`

Description: flask-api has been down for more than 30 seconds. **Alert:** Instance kafka-consumer down - `critical`

Description: kafka-consumer has been down for more than 30 seconds.

[RESOLVED] InstanceDown for

Alert: Instance flask-api down - `critical`

Description: flask-api has been down for more than 30 seconds. **Alert:** Instance kafka-consumer down - `critical`

Description: kafka-consumer has been down for more than 30 seconds.

Alertmanager APP 10:14 PM

[FIRING:1] InstanceDown for flask-api

Alert: Instance flask-api down - `critical`

Description: flask-api has been down for more than 30 seconds.

Alertmanager APP 11:04 PM

[RESOLVED] InstanceDown for flask-api

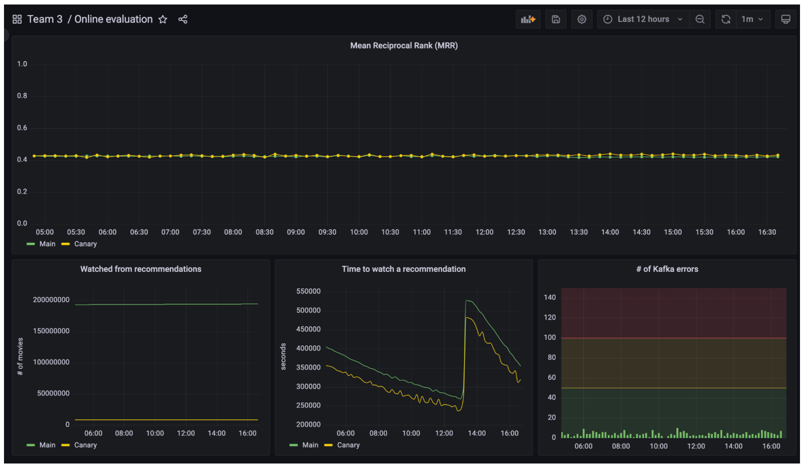
Alert: Instance flask-api down - `critical`

Description: flask-api has been down for more than 30 seconds.

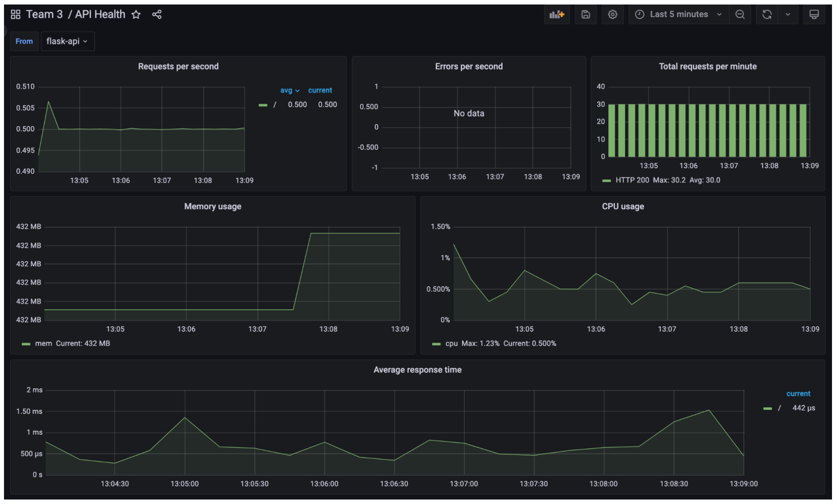
See the full configuration of Alermanager here.

In addition to alerts, we also monitor various metrics across our services using Grafana (http://fall2022-comp585-3.cs.mcgill.ca:3000/, admin, pass@123). These metrics primarily consist of the following:

1. Online evaluation metrics from Milestone 2 allow us to visualize the model performance from both the main and the canary containers.


Team 3 / Online evaluation
Last 12 hours
1m
Mean Reciprocal Rank (MRR)
Main
Canary
Watched from recommendations
# of movies
Time to watch a recommendation
seconds
# of Kafka errors

2. API health metrics allow us to see the system performance of the main Flask API as well as the canary API (switchable between canary/main using the top-left “From” dropdown)



We also have dashboards for monitoring our Redis database health, as well as for monitoring our Cache layer.

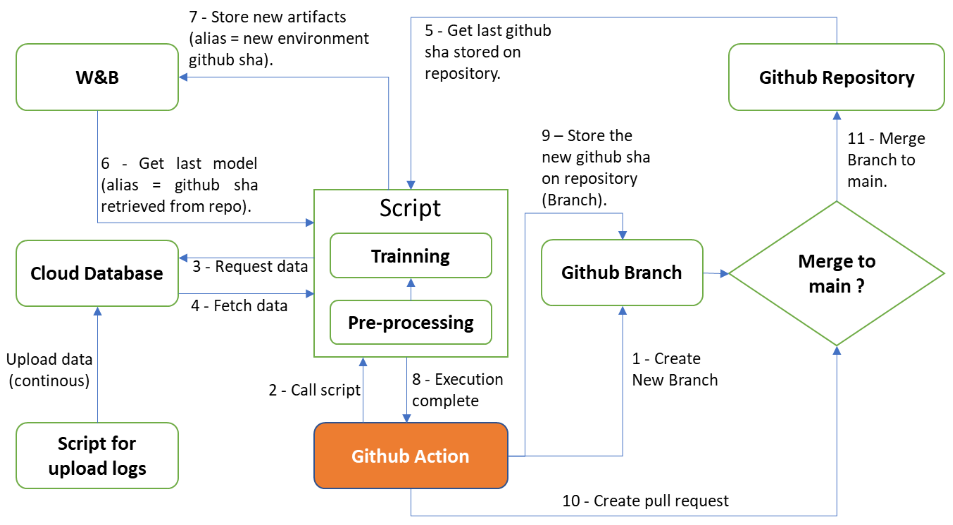
All configurations related to these services are stored in our repository within the Monitoring folder [here]. Combining all these services helps us efficiently debug issues, investigate bottlenecks, and maintain service reliability across all aspects of the project.

Automated model updates

The designed pipeline continuously fetches Kafka logs and updates the data in a cloud database. When the new training process is triggered, all new data (after the last training process) is collected from the cloud database and pre-processed according to the model definitions. The pipeline uses the util w&b (Weights and Biases) to track the dataset used and model versions, so at this point, it opens a new w&b session to retrieve the latest model version. Actual model weights are preserved, and the new pre-processed data is used to continue the training process for performance improvement. Using the w&b session, the process stores as artifacts the data used in the training step, also the new version of the model using the environment variable "GitHub SHA" as an alias, which is stored (updated) in the repository to be used as a reference and allow the pipeline to track the latest version from the w&b model.

A YML file was developed to automate the whole process through Github actions. When the action starts, it creates a new GitHub branch and runs the feedback pipeline script (which runs the training steps). This branch will automatically be associated with a new environmental GitHub SHA, which will be used to store and track the new model as described above. Once the training script has done its execution, the Github action automatically creates a pull request for the administrators to review the new w&b artifacts and decide whether or not to merge the new repository version and consequently deploy the new model or not. In the data processing phase, some metadata of data analysis with respect to gender will be stored for the preprocessed dataset. It plays a significant role in whether we would like to merge it to guarantee it contains some kind of fairness from the gender perspective.

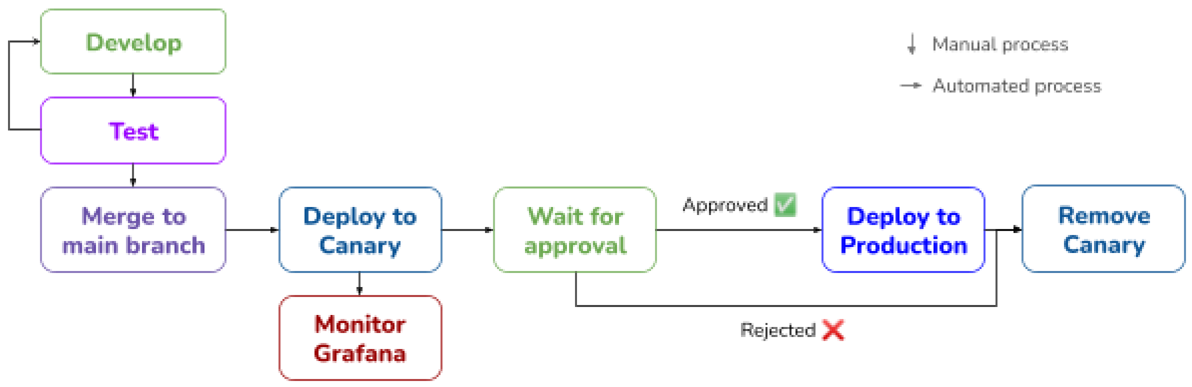
When the model is deployed to the server, the stored GitHub SHA reference will be used to retrieve the latest version of the w&b model to use for inference. In general, tracking GitHub SHA model versions allows us to identify some process metadata, for example, what branch it is from, what pull request it is related to, and when we deployed the model.



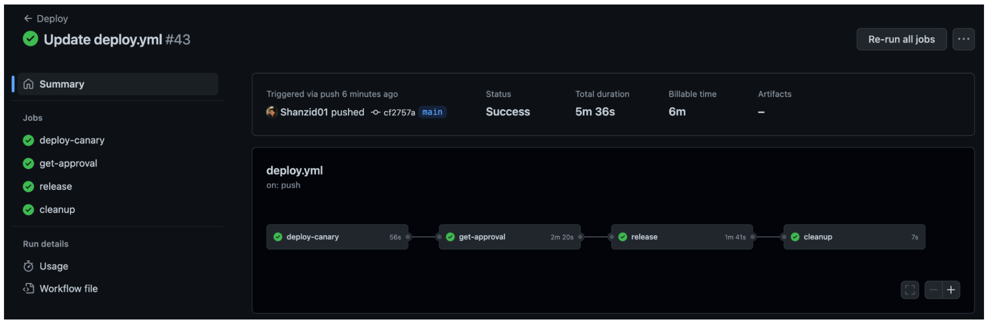
This process runs automatically every 3 days using a CRON job on GitHub Actions. It can also be triggered manually using the GitHub interface: https://github.com/Shanzid01/Team-3/actions/workflows/feedback.yml

Releases

The control flow for development and creating releases can be summarized into this flow chart:



The deployment process is handled by GitHub Actions workflows [config], including automatically deploying new code in the main branch as Canary releases to the team server, waiting for the team to verify and approve the Canary release, and finally - when approved - deploying the new code as a production release.



← Deploy
Update deploy.yml #43
Re-run all jobs

Summary

Jobs
- deploy-canary
- get-approval
- release
- cleanup

Run details
- Usage
- Workflow file

Triggered via push 6 minutes ago	Status	Total duration	Billable time	Artifacts
Shanzid01 pushed cf2757a main	Success	5m 36s	6m	–

deploy.yml
on: push

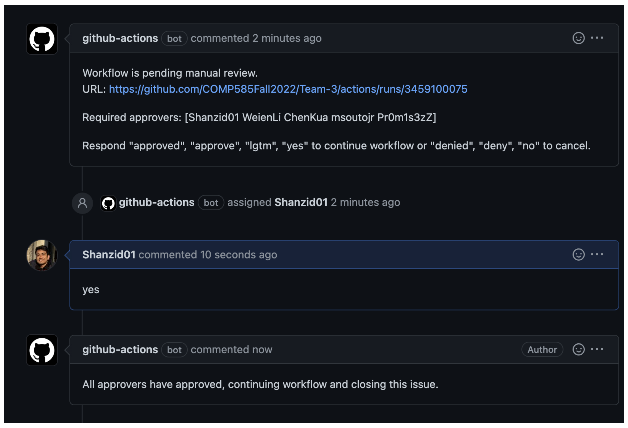
deploy-canary 56s → get-approval 2m 20s → release 1m 41s → cleanup 7s

Fig: Successful deploy workflow in GitHub Actions

To avoid service disruptions between updates, we configured the server with Traefik load balancing, which, in the presence of a Canary container, splits the incoming traffic at an 80:20 ratio between the Production container and the Canary container, respectively; 100% of requests go to the Production container otherwise [config]. Additionally, to ensure our service can serve requests during container updates, we have also set up a fallback service that serves a static list of recommendations for any incoming request [config]. This helps us maintain maximum uptime, albeit with a static list of recommendations [this], even when our containers are momentarily unavailable.

For every new Canary release, our GitHub action creates a new Issue that allows us to approve/reject the Canary release [example]. The team manually verifies whether the release is fit for full deployment to production. To assess this, we have set up Grafana dashboards to monitor the health of the services, as described above in the Monitoring section. Our assessment primarily depends on the Online Evaluation and API Health dashboards. Both these dashboards have been configured to display metrics from the Canary and Production containers separately, helping us make comparisons. These dashboards help us understand important metrics about the model performance (MRR and the number of Kafka errors observed most notably) and the overall service health (CPU/memory consumption, errors in API, etc.). We make our assessments based on whether any of the observed metrics deteriorate or improve. The approval process typically lasts 1 hour, which is usually enough time for all the metrics to aggregate and help us decide on the release's viability and whether it is fit for production.

Once the Canary release is determined to be viable, we comment “yes” on the created Issue. Our workflow automatically deploys the latest code to the Production container and removes the Canary container. On the other hand, suppose we determine that a Canary release had problems. In that case, we can reply “no” to the Issue, which simply removes the Canary container deployed, thereby aborting the release. Team members can also be tagged into the thread on the Issues to investigate the root cause of why the release wasn’t fit for production and, eventually, create patches.



github-actions bot commented 2 minutes ago

Workflow is pending manual review.
URL: https://github.com/COMP585Fall2022/Team-3/actions/runs/3459100075

Required approvers: [Shanzid01 WeienLi ChenKua msoutojr Pr0m1s3zZ]

Respond "approved", "approve", "lgtm", "yes" to continue workflow or "denied", "deny", "no" to cancel.

github-actions bot assigned **Shanzid01** 2 minutes ago

Shanzid01 commented 10 seconds ago

yes

github-actions bot commented now — Author

All approvers have approved, continuing workflow and closing this issue.

Fig: Example issue created from a new Canary release

The team gets email notifications from GitHub when a new Issue is created or changed. These emails help us get alerted when there are recent changes in Canary, when these changes are approved, or even when they are denied directly through GitHub’s notification system. Paired with Slack alerts from Alertmanager, the team can always stay on top of issues and new updates.

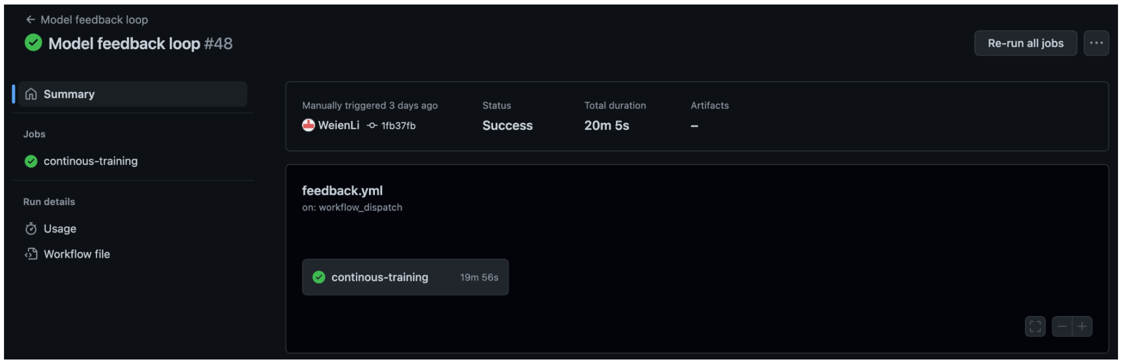
Example of a successful release:
https://github.com/Shanzid01/Team-3/actions/runs/3556140816 [Issue#100]
Example of an aborted release:
https://github.com/Shanzid01/Team-3/actions/runs/3545965753 [Issue#93]

In addition to being able to trigger the model training/deployment process manually (using workflow_dispatch trigger on GitHub Actions), or GitHub Actions are also configured to auto-run the training and deployment process every 3 days with new data using a CRON schedule [here].

Provenance

As described briefly in Automatic Model Updates, we introduced the concept of “SHA” for our models, which are essentially just aliases for different versions. In conjunction with Weights and Biases, the SHA identifier helps us identify and retrieve all data associated with a model and the entire pipeline. The provenance process is structured as follows:

1. At the start of model training in GitHub Actions, we assign a unique SHA identifier to the training process. This identifier is directly collected from the GitHub Action environment variable *github.sha* and is unique to every workflow run [here]. For example, let’s consider this workflow run for model retraining, which carries the SHA 1fb37fb95a96829bf09e2740ad71edaf5e9bcdc7.

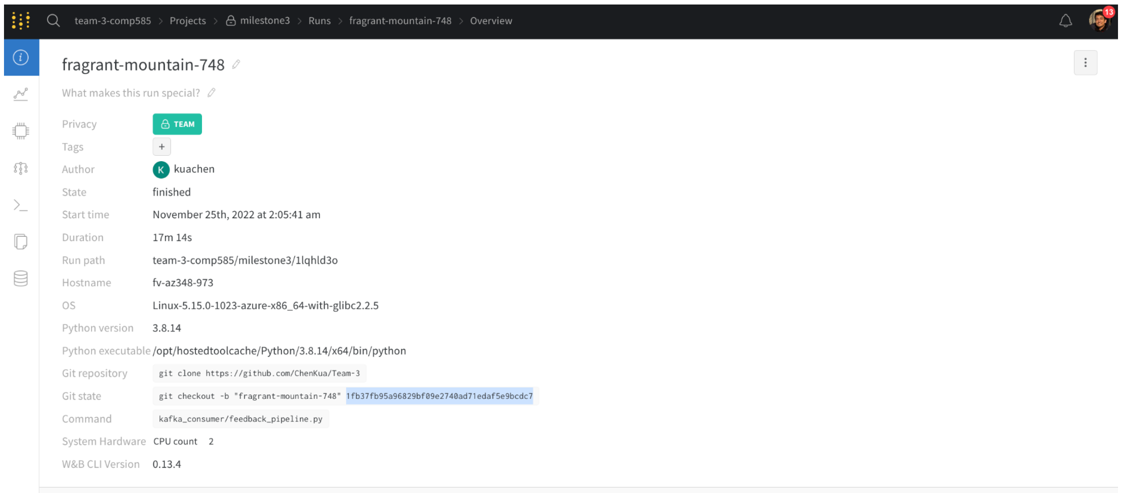


Manually triggered 3 days ago	Status	Total duration	Artifacts
WeienLi 1fb37fb	Success	20m 5s	–

feedback.yml
on: workflow_dispatch

continous-training 19m 56s

2. During the training, our scripts use the SHA provided to label all pieces of information used for training, including the dataset and the model weights generated [here]. For the above example, this is the associated run on W&B:

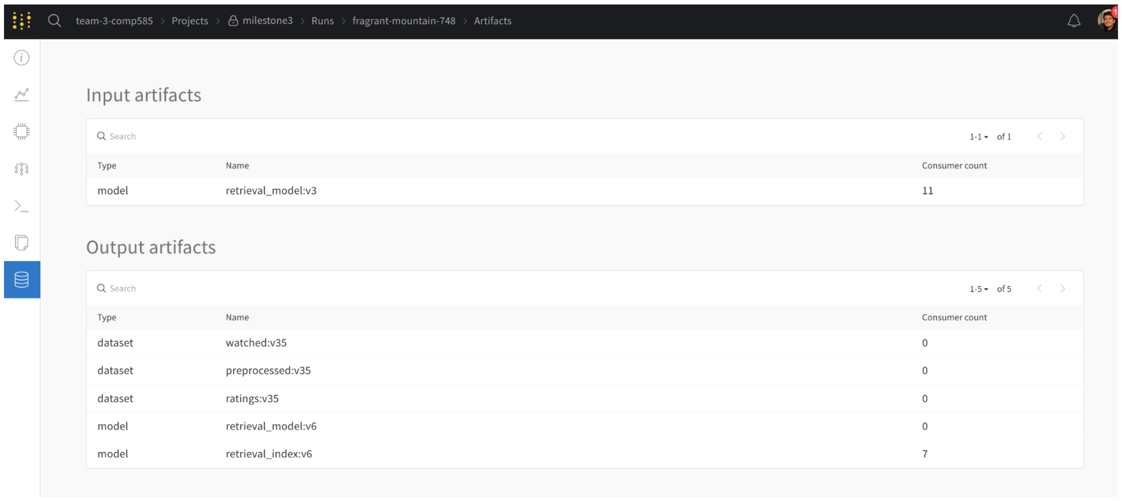

team-3-comp585 > Projects > milestone3 > Runs > fragrant-mountain-748 > Overview
What makes this run special?

fragrant-mountain-748

Privacy	TEAM
Tags	+
Author	kuachen
State	finished
Start time	November 25th, 2022 at 2:05:41 am
Duration	17m 14s
Run path	team-3-comp585/milestone3/1lqhld3o
Hostname	fv-az348-973
OS	Linux-5.15.0-1023-azure-x86_64-with-glibc2.2.5
Python version	3.8.14
Python executable	/opt/hostedtoolcache/Python/3.8.14/x64/bin/python
Git repository	`git clone https://github.com/ChenKua/Team-3`
Git state	`git checkout -b "fragrant-mountain-748" 1fb37fb95a96829bf09e2740ad71edaf5e9bcdc7`
Command	`kafka_consumer/feedback_pipeline.py`
System Hardware	CPU count 2
W&B CLI Version	0.13.4

3. Then we upload the labeled model weights and datasets used in the training process to the Weights&Biases run as artifacts [here]. These artifacts can be accessed from our W&B dashboard later by simply querying the SHA that was used during training. For the

example above, these were the artifacts stored:


team-3-comp585 > Projects > milestone3 > Runs > fragrant-mountain-748 > Artifacts

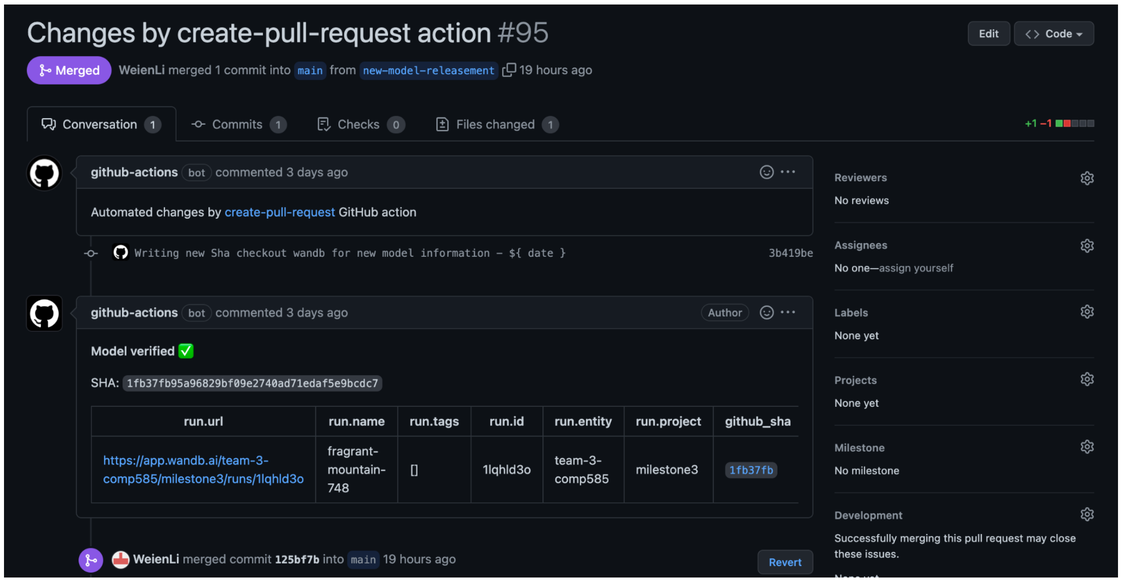
Input artifacts

Type	Name	Consumer count
model	retrieval_model:v3	11

Output artifacts

Type	Name	Consumer count
dataset	watched:v35	0
dataset	preprocessed:v35	0
dataset	ratings:v35	0
model	retrieval_model:v6	0
model	retrieval_index:v6	7

4. Once the training completes, we evaluate the new model offline against the used SHA. This helps us verify that the new model is an improvement over the previous version. This is the verification step for the example above:


Changes by create-pull-request action #95

Merged WeienLi merged 1 commit into `main` from `new-model-releasement` 19 hours ago

Conversation 1 · Commits 1 · Checks 0 · Files changed 1

github-actions bot commented 3 days ago

Automated changes by create-pull-request GitHub action

```
Writing new Sha checkout wandb for new model information - ${ date }    3b419be
```

github-actions bot commented 3 days ago

Model verified ✅

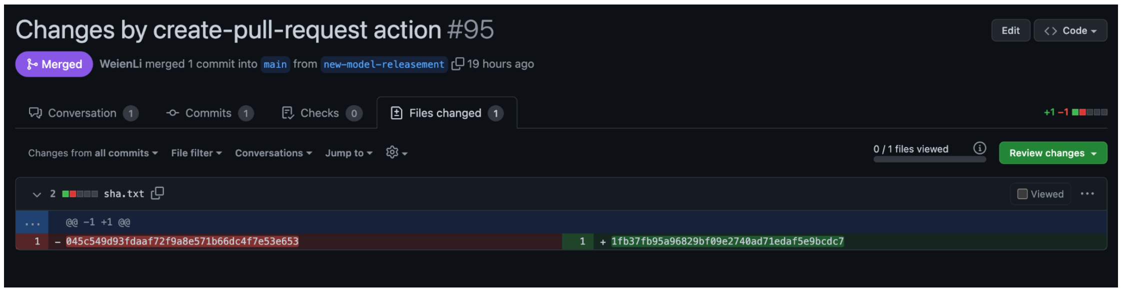
SHA: `1fb37fb95a96829bf09e2740ad71edaf5e9bcdc7`

run.url	run.name	run.tags	run.id	run.entity	run.project	github_sha
https://app.wandb.ai/team-3-comp585/milestone3/runs/1lqhld3o	fragrant-mountain-748	[]	1lqhld3o	team-3-comp585	milestone3	1fb37fb

WeienLi merged commit 125bf7b into `main` 19 hours ago

Reviewers: No reviews
Assignees: No one—assign yourself
Labels: None yet
Projects: None yet
Milestone: No milestone
Development: Successfully merging this pull request may close these issues.

5. Once approved for release, the PR can be merged, saving the SHA to our repo in a sha.txt file. For our example, this is the saved file:


Changes by create-pull-request action #95

Merged WeienLi merged 1 commit into `main` from `new-model-releasement` 19 hours ago

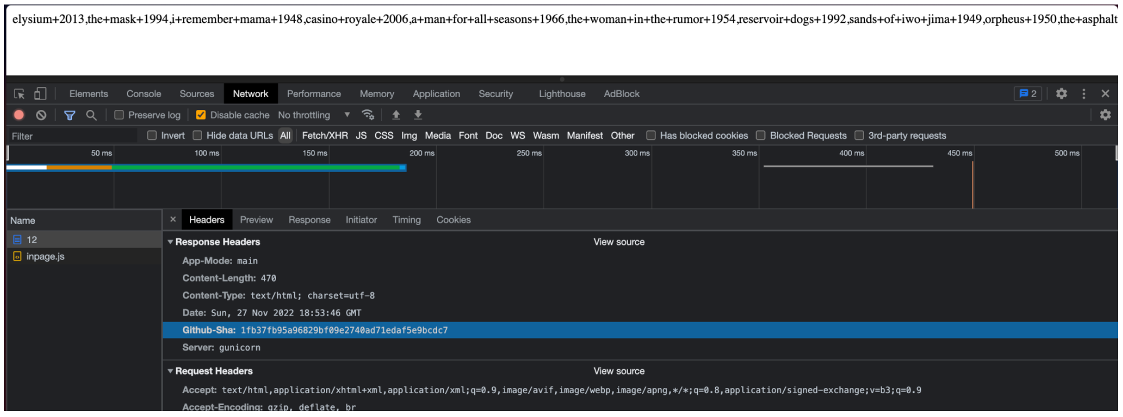
Conversation 1 · Commits 1 · Checks 0 · Files changed 1

0 / 1 files viewed

```
sha.txt
@@ -1 +1 @@
- 045c549d93fdaaf72f9a8e571b66dc4f7e53e653
+ 1fb37fb95a96829bf09e2740ad71edaf5e9bcdc7
```

6. After deployment, the inference service uses the new SHA in sha.txt to download the model weights from Weights&Biases to generate responses [here]. Every response from the API sends a new HTTP header, *Github-Sha*, which is the identifier of the model (and datasets) used to generate the response. Additionally, an App-Mode HTTP header

indicates whether the response was made from a *canary* container or the *main* container. For the example above, when we request a recommendation for user#12:



```
elysium+2013,the+mask+1994,i+remember+mama+1948,casino+royale+2006,a+man+for+all+seasons+1966,the+woman+in+the+rumor+1954,reservoir+dogs+1992,sands+of+iwo+jima+1949,orpheus+1950,the+asphalt
```

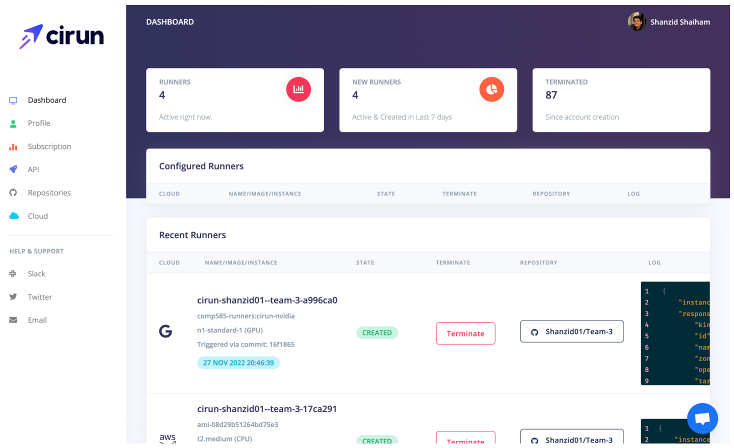
```
Response Headers
  App-Mode: main
  Content-Length: 470
  Content-Type: text/html; charset=utf-8
  Date: Sun, 27 Nov 2022 18:53:46 GMT
  Github-Sha: 1fb37fb95a96829bf09e2740ad71edaf5e9bcdc7
  Server: gunicorn
Request Headers
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
  Accept-Encoding: gzip, deflate, br
```

Therefore, using just the SHA in the response header for any recommendation, we can identify the version of the model, the dataset, and the pipeline with ease.

Obstacles and resolutions (extra)

During the development process, our primary repository, COMP585Fall2022/Team-3, ran out of credits for running GitHub Actions. Although the issue may have been fixed by making our repository public, we decided not to take that route since we would need to change our code significantly to hide secrets like database passwords, Slack webhook URLs, etc. As a result, we moved our code to a private repository, Shanzid01/Team-3, which helped us continue running GitHub Actions.

Additionally, our workflow for model training was taking significantly long (~30mins) on the default GitHub Actions servers since these instances do not support GPUs. As a way to address this, we run our training process in self-hosted Runners using Google Cloud Platform instances with NVIDIA T4 GPUs. However, if we were to keep the Runner instance active even when it is not being used, we would incur a very large computing bill (~USD290/mo). So, we used Cirun.io to auto-provision these compute instances only when we need to run the training process. This auto-provisioning reduced our effective bill to just ~$25/mo (extrapolated). Similarly, we use AWS EC2 instances to run the rest of our non-GPU workflows (e.g., lint testing). The implementation and configuration of this can be found here.



RUNNERS 4 — Active right now.
NEW RUNNERS 4 — Active & Created in Last 7 days
TERMINATED 87 — Since account creation

Configured Runners

CLOUD	NAME/IMAGE/INSTANCE	STATE	TERMINATE	REPOSITORY	LOG

Recent Runners

CLOUD	NAME/IMAGE/INSTANCE	STATE	TERMINATE	REPOSITORY	LOG
G	cirun-shanzid01--team-3-a996ca0 comp585-runners:cirun-nvidia n1-standard-1 (GPU) Triggered via commit: 16f1865 27 NOV 2022 20:46:39	CREATED	Terminate	Shanzid01/Team-3	
aws	cirun-shanzid01--team-3-17ca291 ami-08d29b51264bd75e3 t2.medium (CPU)	CREATED	Terminate	Shanzid01/Team-3	

The costs for consuming these additional services were fully subsidized by student discounts and credits from

1. Cirun.io for their Business plan (USD79/mo) [sent them a tweet]
2. Google Cloud Platform USD50 education credits [sent a request from their online portal]
3. Amazon Web Services student credits [AWS educate]
4. Lambdalabs.com for USD100 GPU credits for running ML workflows. [sent them an email at cloud@lambdalabs.com requesting credits]

Individual contributions

Our team organized ourselves by holding weekly hybrids before class meetings and having ad-hoc meetings when something urgent arose. The hybrid meeting style accommodates everyone's needs in case people can not attend campus. We also have frequent communications via slack, creating pull requests on GitHub and setting each other as reviewers to review published work. We split the work according to each one's specialty and background, similar to the previous two milestones. The responsibilities are split in the following way:

- Shanzid is in charge of AI Infra (details below)
- Kua is in charge of processes that continuously monitor the Kafka stream, update the database, and ensure quality with unit testing of the code, and data and model pipelines.
- Barry is in charge of doing data pipeline, data analysis with respect to gender, and setting up the GitHub actions to guarantee provenance can be tracked.
- Marcos is in charge of the model pipeline, the interface with w&b (weights and bias) to save and retrieve the models, and updating the inference code to guarantee that provenance is preserved.

Shanzid Shaiham (SWE)	Contributions: - Containerized entire application services into a single docker-compose file - Set up infrastructure around load balancing, fallback services, and overall site uptime and reliability - Set up the infrastructure to create canary releases of the API, including setting up GitHub Actions and bash scripts to trigger deployments with just a single command - Set up monitoring/alerting with Grafana+Prometheus+Alertmanager, and their integration with the rest of our containers and services - Set up dashboards in Grafana for monitoring KPIs - Strategized on how we can establish provenance within our pipeline, and partly built out the provenance solutions across the API, database and Grafana dashboards - Set up custom Runners in GCP+AWS for running GitHub Actions. See contributions authored with Pull Requests: https://github.com/Shanzid01/Team-3/pulls?q=is:pr+author:Shanzid01+ (new repo) and https://github.com/COMP585Fall2022/Team-3/pulls?q=is:pr+author:Shanzid01+ (old repo)
Kua Chen (SWE)	Contributions: ● Developed utilities for CRUD operations on the cloud database. ● Developed extra testing for Milestone 3. ● Set up a new GitHub repository to have extra resources. See contributions authored with Pull Requests: https://github.com/Shanzid01/Team-3/pulls?q=is%3Apr+author%3AChenKua+is%3Aclosed (new repo), and https://github.com/COMP585Fall2022/Team-3/pulls?q=is%3Apr+a

	uthor%3AChenKua+ (old repo)
Barry Li (ML)	Contributions: ● Write the script for processing data loaded from supabase for the model to train when feedback is triggered. Also storing it in wandb to guarantee provenance can be tracked ● Analyzed data from the database with respect to gender and store it as a metadata in wandb to guarantee fairness and make it provable ● Setup the cron Github actions for the feedback loop and store the appropriate Github sha for provenance. See contributions authored with Pull Requests: https://github.com/Shanzid01/Team-3/pulls?q=is%3Apr+author%3AWeienLi+is%3Aclosed (new) https://github.com/COMP585Fall2022/Team-3/pulls/WeienLi (old)
Marcos Souto Jr. (ML)	Contributions: - Implementation of the training python script. - Study and implementation of the interface with weights and bias (w&b) to store and retrieve the different versions of the model. - Study and implementation of the best way to save and retrieve models from tensorflow recommender systems. - Adjust the inference code to use the model stored in weights and bias. https://github.com/Shanzid01/Team-3/issues?q=author%3Amsoutojr (new repo) https://github.com/COMP585Fall2022/Team-3/pulls?q=is%3Apr+author%3Amsoutojr+ (old repo)
Zeyu Li (ML)	

Meeting Notes

Monday, Nov. 7, Hybrid

- The first meeting for milestone 3. The main purpose of the meeting was to distribute tasks and set up goals for this milestone.
- One reader read through Milestone 3 documentation on Github for everyone.
- Discussed what tasks had been done and distributed tasks that needed to be implemented.
- Shanzid had finished the majority of Monitoring and Provenance during Milestone 2.
- Shanzid would be responsible for Containerization.
- Barry and Kua would be responsible for the data pipeline.
- Marcos would be responsible for Automated model updates.
- Shanzid, Barry and Zeyu would be responsible for Releases and Provenance.

Monday, Nov. 14, Online

- Discussed progress during the last week.
- Containerization was mostly finished.
- A Supabase cloud database was set up for the data pipeline.
- Discussed technical details on accessing Weight and Bias. This was an important step in automated model updates.
- Barry would take responsibility for setting up the GitHub Actions workflow for the continuous training process

Saturday, Nov. 19, Online

- The meeting was about fixing the automated process and the interface between the tasks related to the pre-processing and training steps.
- - It also served to align some expectations related to the organization of the repository and the ways of saving and retrieving the w&b model.

Monday, Nov. 21, Online

- The project deadline was approaching in 4 days. The focus of this meeting was to update the current progress of each task and discuss technical difficulties and clean-ups.
- Each one updated the task progress that he was in charge of.
- Discussed task dependency, i.e. what people could do if they are waiting for other people's code.
- Addressed some environment settings because we “forked” our school GitHub repository to an individual repository. There were some issues with environmental variables that we had to resolve.
- Discussed why unit testing kept failing.
- Discussed a potential bug in automated code style reformatting.