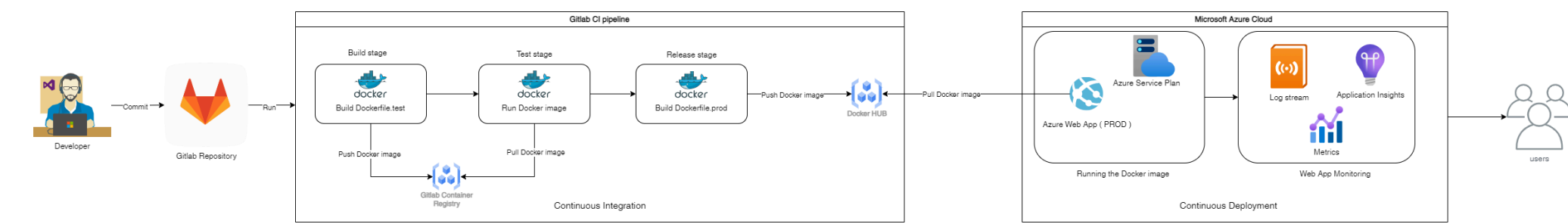


CI CD Pipeline

Last edited by **Naceur Sayedi** 7 months ago

Wir haben die CI/CD-Pipeline implementiert, um eine schnellere Softwarebereitstellung zu ermöglichen, die Codequalität zu verbessern und die Zusammenarbeit im Team zu stärken.

Hier ist ein Überblick, wie wir die CI CD Pipeline in unserem Projekt konfiguriert haben:



1. Entwickler Commit zum GitLab Repository

Die Entwickler senden ihre Codeänderungen an das GitLab-Repository.

2. CI-Pipeline (GitLab)

Nachdem der Entwickler eine Commit an das Gitlab-Repository gemacht hat, wird die CI-Pipeline automatisch in Gitlab ausgeführt.

2.1 Einrichtung der CI-Pipeline

Hier sind unsere gitlab ci-Dateien für Backend und Frontend:

- [.gitlab-ci.yml](#) (Backend)
- [.gitlab-ci.yml](#) (Frontend)

Um unsere CI-Pipeline richtig zu laufen bringen, müssen wir also sicherstellen, dass die Variablen, die wir in der CI-Pipeline verwendet werden, auch in unserem Repository-Projekt in Gitlab gespeichert sind. Hier sehen Sie Schritt für Schritt, wie wir die Variablen speichern:

Schritt 1:

- Erzeugen wir sowohl im Repository-Backend als auch im Frontend ein Deploy-Token. Dieses Token können wir generieren, wenn wir zu Einstellungen -> Repository -> Deploy Tokens gehen und dann auf Token hinzufügen klicken. Mit diesem Token können wir die Anmeldung am Projekt-Repository sicherstellen und dann das Docker-Image zum Testen in die Gitlab-Container-Registry pushen

New deploy token

Create a new deploy token for all projects in this group. [What are deploy tokens?](#)

Name

connectandexplorefrontendtoken

Enter a unique name for your deploy token.

Expiration date (optional)

Enter an expiration date for your token. Defaults to never expire.

Username (optional)

connectandexplorefrontend

Enter a username for your token. Defaults to `gitlab+deploy-token-{n}`.

Scopes (select at least one)

☐ read_repository

Allows read-only access to the repository.

☒ read_registry

Allows read-only access to registry images.

☒ write_registry

Allows read and write access to registry images.

☐ read_package_registry

Allows read-only access to the package registry.

☐ write_package_registry

Allows read, write and delete access to the package registry.

Create deploy token

Cancel

Schritt 2:

Wir müssen zunächst ein [Docker Hub-Konto](#) erstellen (die kostenlose Version ist ausreichend), dann müssen wir zu Mein Konto -> Sicherheit -> Access Tokens gehen und dann auf New Access Token klicken und ein Access Token erstellen, um sich damit in Ihrem Docker Hub anzumelden und die Backend- und Frontend-Images zu pushen, um sie anschließend in der Azure Cloud bereitzustellen

General

Security

Default Privacy

Notifications

Convert Account

Deactivate Account

New Access Token

A personal access token is similar to a password except you can have many tokens and revoke access to each one at any time. [Learn more](#)

Access Token Description *

connectandexplore

Access permissions

Read, Write, Delete

Read, Write, Delete tokens allow you to manage your repositories.

Cancel

Generate

New Access Token

maximum of 5 auto-generated tokens

created

an 23, 2024 22:18:58

ec 04, 2023 00:05:58

Schritt 3: Nun müssen wir unsere Variablen in Backend- und Frontend-Repositories speichern. Hier sind also alle Variablen, die wir speichern müssen:

- **CI_IMAGE_DOCKERHUB :**
 - Backend = naceursayed34/connectandexplore:backend
 - Frontend = naceursayed34/connectandexplore:frontend
- **CI_USERNAME_DOCKERHUB** = naceursayed34 (Schritt 2)
- **CI_PASSWORD_DOCKERHUB** = Generiertes Token von Docker Hub (Schritt 2)
- **CI_REGISTRY :**
 - Backend = registry.bht-berlin.de:443/s85975/connectandexplore
 - Frontend = registry.bht-berlin.de:443/s86156/connectandexplore
- **CI_REGISTRY_USER** = gitlab+deploy-token-106 (Schritt 1)
- **CI_REGISTRY_PASSWORD** = Generiertes Token von Gitlab (Schritt 1)

Um diese Variablen zu speichern, müssen wir zu Einstellungen -> CI/CD -> Variablen gehen und dann auf Variable hinzufügen klicken.

PHT

1

2

Build

Secure

Deploy

Operate

Monitor

Analyze

Settings

General

Integrations

Webhooks

Access Tokens

Repository

Merge requests

CI/CD

Packages and regist...

Monitor

Naceur Sayedi > ConnectAndExplore > CI/CD Settings

Variables

Variables store information, like passwords and secret keys, that you can use in job scripts. Each project can define a maximum of 8000 variables. [Learn more.](#)

Variables can have several attributes. [Learn more.](#)

- **Protected:** Only exposed to protected branches or protected tags.
- **Masked:** Hidden in job logs. Must match masking requirements.
- **Expanded:** Variables with `$` will be treated as the start of a reference to another variable.

CI/CD Variables </> 7

Reveal values

Add variable

| ↑ Key | Value | Attributes | Environments | Actions |
|---------------------------|-------|------------|---------------|-----------------------------------|
| CI_IMAGE_DOC KERHUB | ***** | Expanded | All (default) | <div><div></div><div></div></div> |
| CI_PASSWORD_ DOCKERHUB | ***** | Expanded | All (default) | <div><div></div><div></div></div> |
| CI_REGISTRY | ***** | Expanded | All (default) | <div><div></div><div></div></div> |
| CI_REGISTRY_I MAGE | ***** | Expanded | All (default) | <div><div></div><div></div></div> |

2.2. Build Stage

Zielsetzung: Build und Pushen von Docker-Images an GitLab Container Registry zum Testen.

Schritte:

1. Build des Docker-Images aus Dockerfile.test
2. Tagge das Image als "main"
3. Das Image in die GitLab Container Registry pushen
4. Expose Ports 8080 (Backend) und 3030 (Frontend) zu Testzwecken

Links:

- [Dockerfile.test](#) (Backend)
- [Dockerfile.test](#) (Frontend)
- [GitLab Container Registry](#) (Backend)
- [GitLab Container Registry](#) (Frontend)

2.3. Test Stage

In diesem Stage werden nur die Tests vom Backend ausgeführt, da wir keine Tests für das Frontend implementiert haben.

Unsere Tests im Backend sind in Jest geschrieben und wir haben alle Testdateien in diesem Ordner:

- [Backend-Tests](#)

Zielsetzung: Durchführung von Unit-Tests mit dem in der Build-Phase erstellten Docker-Image.

Schritte:

1. Pull vom Docker-Image mit dem Tag "main" aus der GitLab Container Registry
2. Ausführen von **npm test** für die Ausführung von Jest-Unit-Tests

2.4. Release Stage

Es hat bei der Deployment in Azure Cloud nicht funktioniert, Images aus der Gitlab-Container-Registry zu pullen, daher haben wir als Lösung Docker Hub verwendet.

https://gitlab.bht-berlin.de/s86156/connectandexplore/-/wikis/CI/CI-CD-Pipeline

3/8

Zielsetzung: Build und Push von Docker-Images an Docker Hub für die Produktion.

Schritte:

- 1. Build eines Docker-Images aus Dockerfile.prod
- 2. Push das Image an Docker Hub

Links:

- [Dockerfile.prod](#) (Backend)
- [Dockerfile.prod](#) (Frontend)
- [Docker Hub Repository](#)

3. CD-Pipeline (Azure Cloud)

Um unsere Webanwendung in Produktion gehen zu lassen, müssen wir unsere Docker-Images von Frontend und Backend als Container laufen lassen. Dafür haben wir herausgefunden, dass wir mit Azure Cloud den Service Web App for Containers nutzen können, um unsere Webanwendung in Produktion zu bringen.

Als Student können wir ein Azure Cloud-Konto mit einem Subscription [Azure for Students](#) erstellen. Diese Subskription gibt uns ein Guthaben für 100 \$, so dass wir sie verwenden können, um Services zu erstellen und sie zu verwenden

3.1. Konfiguration des Deployments

Für die Konfiguration unseres Deployments müssen wir zunächst eine Ressource Group in der Azure Cloud erstellen, dann müssen wir den Service (Web App for Containers) für Backend und Frontend erstellen (je 1 Service).

Hier ist eine Übersicht über die Schritte:

Schritt 1: Erstellen der Ressourcengruppe

- Hier muss eine Ressource Group erstellt werden: [Create Ressource Group in Azure](#)

Schritt 2: Konfiguration der Web App for Containers

- 1. Web App for Containers erstellen

Azure Portal -> Ressourcengruppe -> Web App for Containers

[Home](#) > [Resource groups](#) > [connectandexplore](#) > [Marketplace](#) >

Web App for Containers

Microsoft






Web App for Containers

Microsoft | Azure Service

★ 3.8 (173 ratings)

 [Add to Favorites](#)

Plan

Web App for Containers

▼

Create

- 2. Erstellung des Services

Die Details der Instanz:

- Name: connectandexplore
- Publish: Docker-Container
- Betriebssystem: Linux

Create Web App

1

Subscription * ⓘ

Azure for Students

Resource Group * ⓘ

connectandexplore

[Create new](#)

Instance Details

Name *

connectandexploreapp

.azurewebsites.net

Publish *

☐ Code

☒ Docker Container

☐ Static Web App

Operating System *

☒ Linux

☐ Windows

Region *

East US

ⓘ

Not finding your App Service Plan? Try a different region or select your App Service Environment.

Review + create

< Previous

Next : Database >

3. Festlegen des Preisplans

Der Preisplan:

Basic B2 (3,5 GB RAM, 2 vCPU) für 24,82 USD pro Monat.

Create Web App

2

Pricing plans

App Service plan pricing tier determines the location, features, cost and compute resources associated with your app.
[Learn more](#) ⓘ

Linux Plan (East US) * ⓘ

(New) ASP-connectandexploreappgroup-8178

[Create new](#)

Pricing plan

Basic B1 (100 total ACU, 1.75 GB memory, 1 vCPU)

[Explore pricing plans](#)

Home > Web App for Containers > Create Web App >

Select App Service Pricing Plan

×

| | | | | | | | |
|--|-------------------|-----|------|------|-----|-----------|-------------|
| Premium v3 P4mv3 | 195* | 16 | 128 | 250 | 30 | 1,488 USD | 1086,24 USD |
| Premium v3 P5mv3 | 195* | 32 | 256 | 250 | 30 | 2,976 USD | 2172,48 USD |
| Isolated v2 I1V2 | 195 | 2 | 8 | 1000 | N/A | 0,386 USD | 281,78 USD |
| Isolated v2 I2V2 | 195 | 4 | 16 | 1000 | N/A | 0,772 USD | 563,56 USD |
| Isolated v2 I3V2 | 195 | 8 | 32 | 1000 | N/A | 1,544 USD | 1127,12 USD |
| ▼ Dev/Test (For less demanding workloads) | | | | | | | |
| Free F1 | 60 minutes/day... | N/A | 1 | 1 | N/A | Free | Free |
| Basic B1 | 100 | 1 | 1.75 | 10 | 3 | 0,017 USD | 12,41 USD |
| <input checked="" type="checkbox"/> Basic B2 | 100 | 2 | 3.5 | 10 | 3 | 0,034 USD | 24,82 USD |
| Basic B3 | 100 | 4 | 7 | 10 | 3 | 0,067 USD | 48,91 USD |

Hinweis: Es kann sein, dass diese Konfiguration nur für eine begrenzte Anzahl von Nutzern funktioniert. Um sie für eine große Anzahl von Nutzern zu verwenden, ist es sinnvoller, einen anderen Preisplan zu wählen oder einen anderen Service wie Kubernetes zu nutzen, damit die Anwendung skaliert werden kann, wenn die Nachfrage auf der Website steigt.

4. Konfiguration des Docker-Images

Konfiguration des Docker-Images, um es vom Docker Hub abzurufen.

Create Web App ...

BasicsDatabaseDeploymentDockerNetworkingMonitoringTagsReview + create

Pull container images from Azure Container Registry, Docker Hub or a private Docker repository. App Service will deploy the containerized app with your preferred dependencies to production in seconds.

OptionsSingle Container

Image SourceDocker Hub

Docker hub options

Access Type *Public

Image and tag *naceursayed34/connectandexplore:frontend

Startup Command ⓘ

Review + create

< Previous

Next : Networking >

Und dann auf Review + Create klicken, um den Service zu erstellen.

5. Konfiguration der kontinuierlichen Bereitstellung

Einstellung der kontinuierlichen Bereitstellung, um immer dieselbe Docker-Image-URL abzurufen.

connectandexplore | Deployment Center ☆ ...

Web App

Search

Save Discard Browse Manage publish profile Leave Feedback

Diagnose and solve problems

Microsoft Defender for Cloud

Events (preview)

Log stream

Deployment

Deployment slots

Deployment Center

Settings

Configuration

Authentication

Application Insights

Identity

Backups

Custom domains

Certificates

Networking

Scale up (App Service plan)

Scale out (App Service plan)

SettingsLogs

Use these settings to configure your container app deployment model and registry. We recommend GitHub Actions for greater operational efficiency. Learn more

Source *

Container Registry: Set up your app to pull the container image from a registry.

GitHub Actions: Build, deploy, and manage your container app automatically with GitHub Actions.

Azure Pipelines: Configure a robust deployment pipeline for your application using Azure Pipelines, part of Azure DevOps Services (formerly known as VSTS).

Registry settings

Container typeSingle Container

Registry sourceDocker Hub

Repository Access *Public

Full Image Name and Tag *naceursayed34/connectandexplore:fr...

Startup file or command

Continuous deployment

On Off

Webhook URL



.....

6. Konfiguration der .env-Datei im Frontend

Hinzufügen der Backend-Service-URL zur [.env-Datei](#) im Frontend-Code, damit das Frontend mit der Backend-API kommunizieren kann.

https://gitlab.bht-berlin.de/s86156/connectandexplore/-/wikis/CI/CI-CD-Pipeline

6/8

 .env  462 B

1

2

3

4

5

6

7

8

9

10

11

12

```
# Wir verwenden das Frontend als Proxy, siehe package.json
# Daher ist hier der API-Server das Frontend
REACT_APP_API_SERVER_URL_PROD=https://connectandexplore-backend.azurewebsites.net
REACT_APP_API_SERVER_URL=https://localhost:5000
# cf. https://github.com/facebook/create-react-app/issues/11762
# related to proxy-setting in package.json
DANGEROUSLY_DISABLE_HOST_CHECK=true

NODE_ENV="production"

# REACT_APP_GOOGLE='AIzaSyBhUFUQzKibQWvVtskL4vVzZvfHkEaHWlc'
```

3.2. Deployment Stage

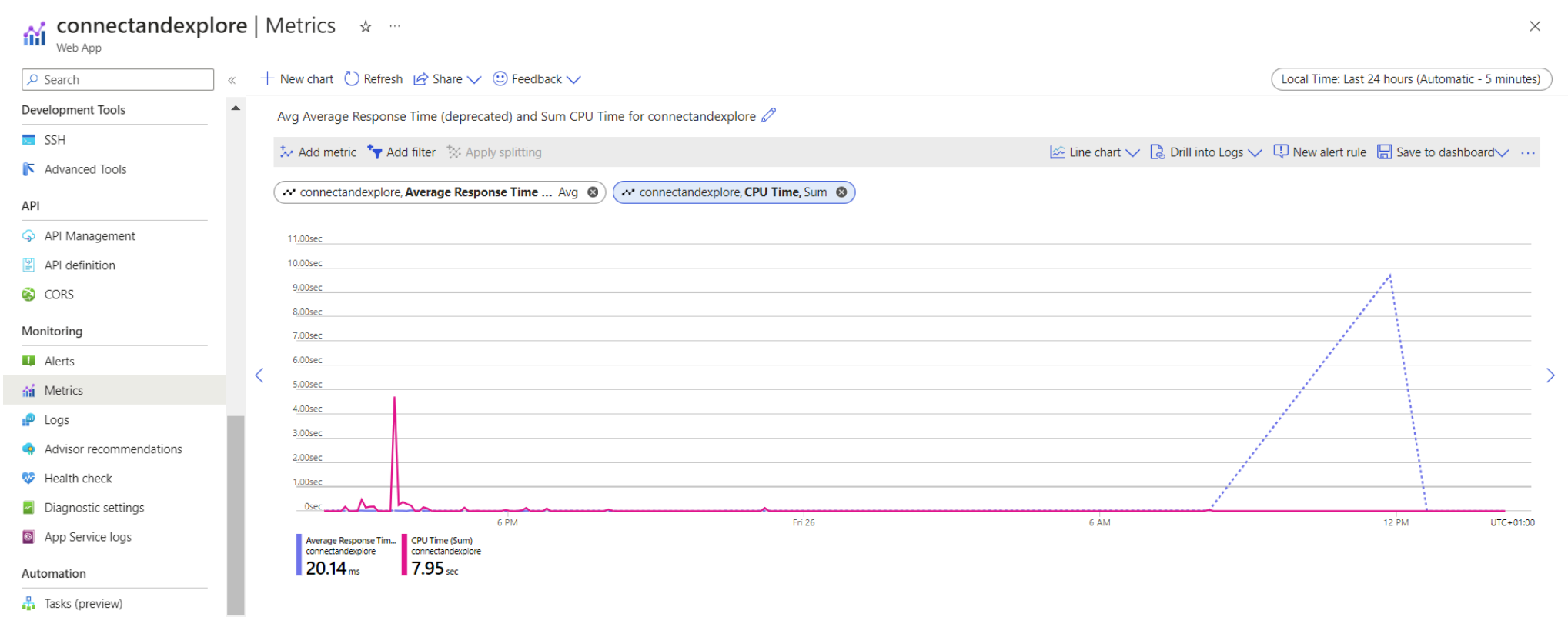
Zielsetzung: Deployment der Anwendung in der Produktion mit Azure Cloud (Web App for Containers).

Schritte:

1. Pull des Docker-Images aus Docker Hub
2. Starten des Docker-Containers mit Port 443 (Backend) und Port 3000 (Frontend).
3. Benutzung vom Befehl npm run start:prod für die Produktionsbereitstellung

4. Monitoring in Azure

Wir können die Monitoring-Tools von Azure nutzen, um Container-Metriken wie die CPU- und RAM-Auslastung mit dem Metrics-Tool zu überwachen.



5. Logging in Azure

Wir können Azure Logging Tools wie Log Stream verwenden, um die Logs zu sehen, die aus dem Container kommen.

connectandexplore-backend | Log stream

Web App

Search

« Reconnect Copy || Pause × Clear

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Microsoft Defender for Cloud

Events (preview)

Deployment

Deployment slots

Deployment Center

Settings

Configuration

Authentication

Application Insights

Identity

Backups

Custom domains

Connecting...

2024-01-24T08:40:15 Welcome, you are now connected to log-streaming service.Starting Log Tail -n 10 of existing logs ---
-/appsvctmp/volatile/logs/runtime/3321454f8f10b56000ddc93eebd4fcc72adeb52e0f33c745ee359b41054c0236.log
2024-01-24T07:42:11.068765499Z: [INFO] fullpath image Event: /app/dist/app/uploads/events/8cec039f-584b-4ae0-9ba0-e628a2a41c99-20231231_231553.jpg
2024-01-24T07:42:42.684899978Z: [INFO] fullpath image Event: /app/dist/app/uploads/events/8a5cd730-7080-4c63-b7d2-ad256068a5e7-20231220_195708.jpg
2024-01-24T07:42:42.684936578Z: [INFO] fullpath image Event: /app/dist/app/uploads/events/bf10596d-95a3-4206-88b3-09cde332f03a-20231231_231553.jpg
2024-01-24T07:42:42.685284979Z: [ERROR] (node:56) UnhandledPromiseRejectionWarning: Error: ENOENT: no such file or directory, unlink
'/app/dist/app/uploads/events/bf10596d-95a3-4206-88b3-09cde332f03a-20231231_231553.jpg'
2024-01-24T07:42:42.685299179Z: [ERROR] at Object.unlinkSync (fs.js:1256:3)
2024-01-24T07:42:42.685358879Z: [ERROR] at t.deleteEventThumbnail (/app/dist/bundle.js:1436:50)
2024-01-24T07:42:42.685383879Z: [ERROR] at /app/dist/bundle.js:661:683
2024-01-24T07:42:42.685387579Z: [ERROR] at runMicrotasks (<anonymous>)
2024-01-24T07:42:42.685391579Z: [ERROR] at processTicksAndRejections (internal/process/task_queues.js:95:5)
2024-01-24T07:42:42.685396579Z: [ERROR] (node:56) UnhandledPromiseRejectionWarning: Unhandled promise rejection. This error originated either by throwing
inside of an async function without a catch block, or by rejecting a promise which was not handled with .catch(). To terminate the node process on
unhandled promise rejection, use the CLI flag --unhandled-rejections=strict (see https://nodejs.org/api/cli.html#cli_unhandled_rejections_mode).
(rejection id: 9)Ending Log Tail of existing logs ---Starting Live Log Stream ---