DevOps-Prozesse mit Tools unterstützen

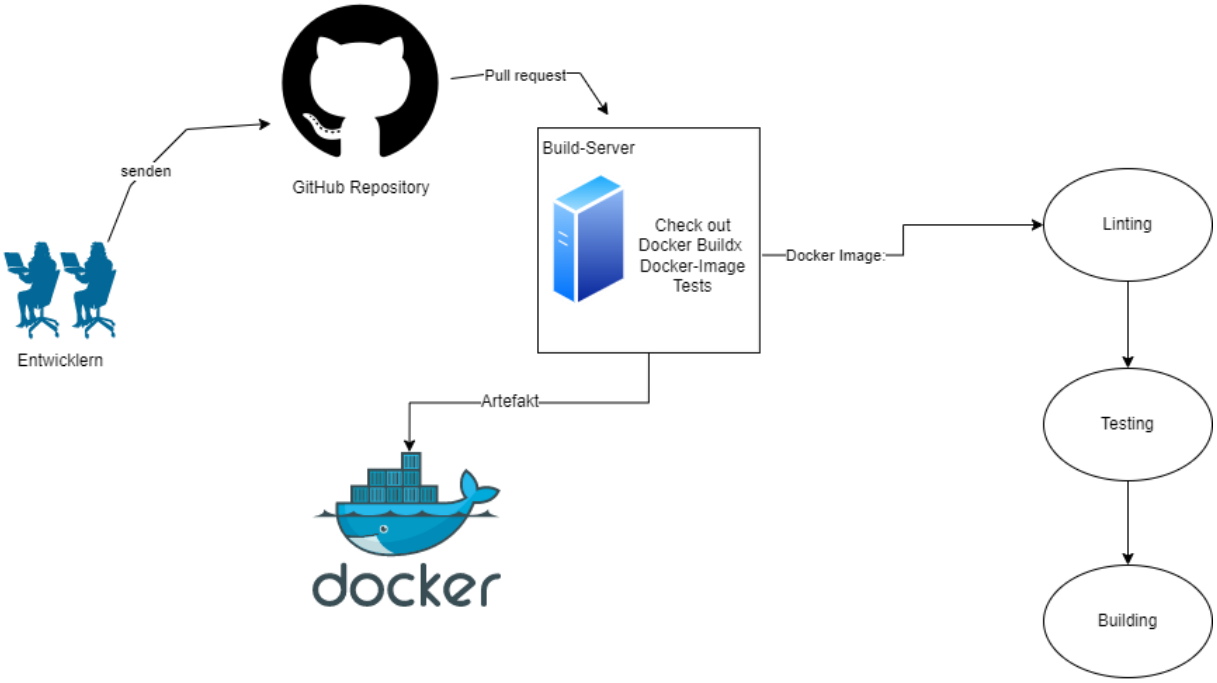
Morhaf Alnhlawe
Mouayad Alnhlawe
Inf21_A
CsBe (Computerschule Bern)
Version: 1.5

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

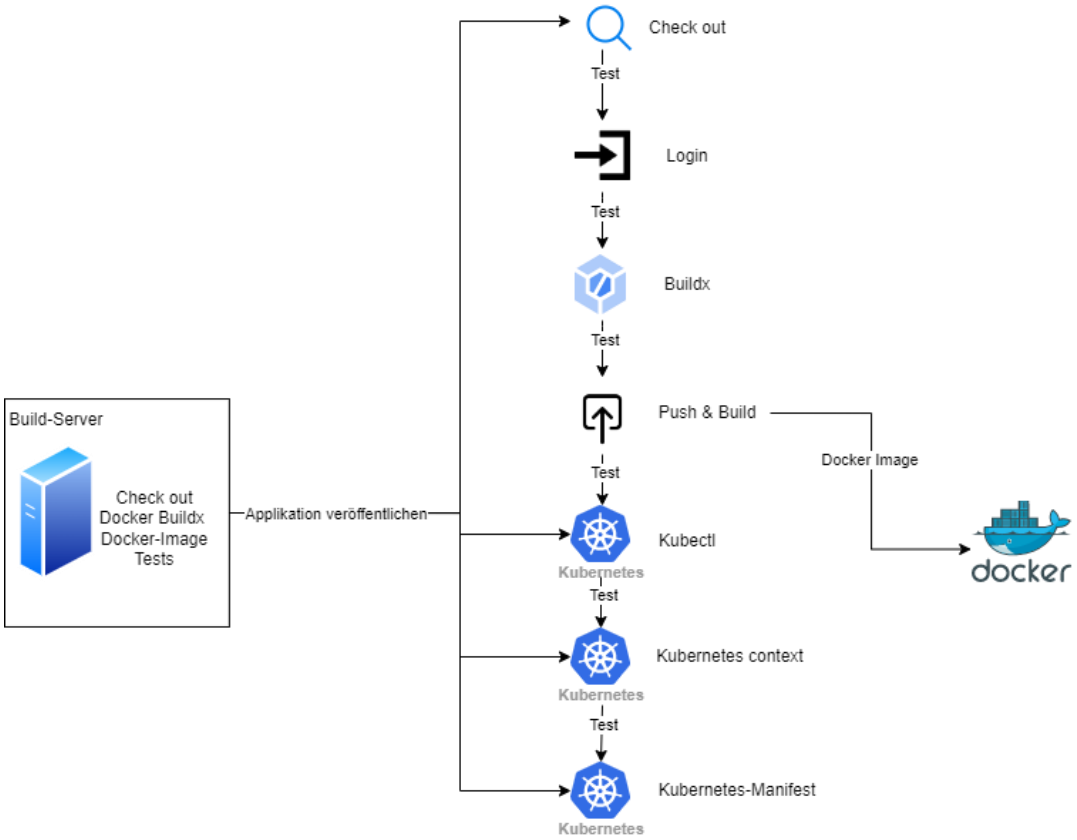
CI-Pipeline



1 CI-Pipeline

So sieht unsere CI-Pipeline aus: Am Anfang werden die Änderungen an GitHub hochgeladen. Es wird nichts unternommen, bis ein Pull Request auf Main angefragt wird. Sobald die Pull Request angefragt ist, wird die CI-Pipeline ausgeführt. Zuerst wird der Code ausgecheckt, dann wird Buildx eingerichtet. Danach wird das Docker-Image gepusht und getaggt. Im Dockerfile werden Lint, Test und Build ausgeführt. Schliesslich wird das Docker-Image ausgeführt und getestet. Das Resultat bzw. das Artefakt wird in Docker Hub gespeichert und ist bereit für die CD-Pipeline.

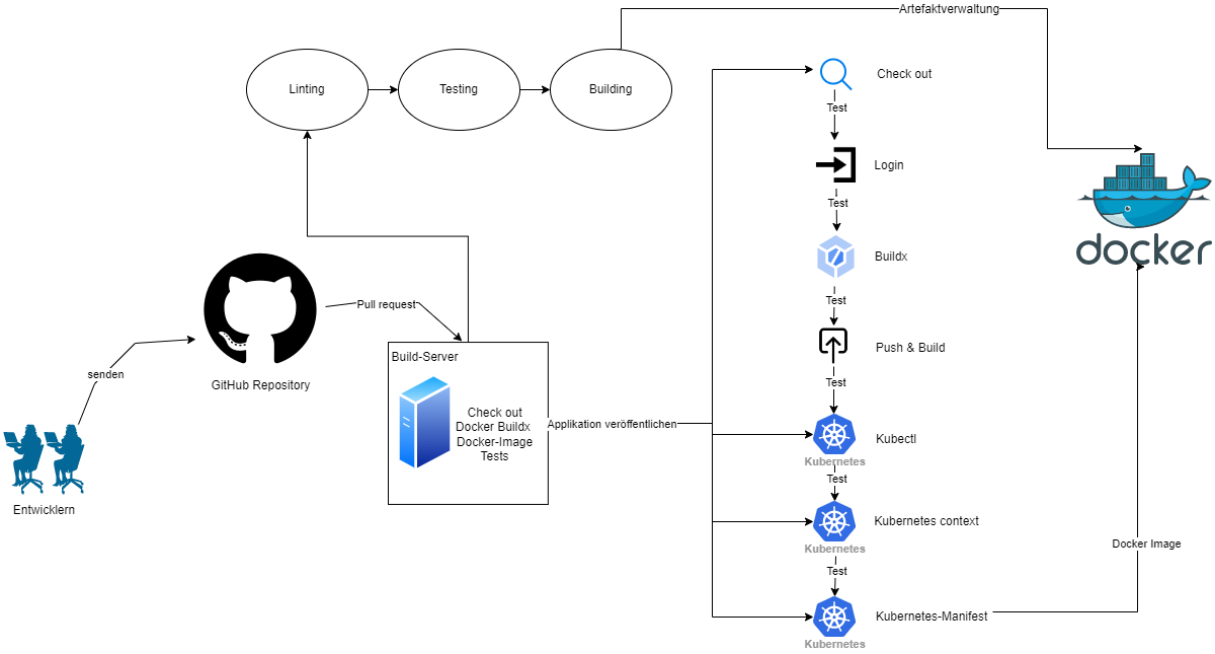
CD-Pipeline



2 CD-Pipeline

Nachdem das Artefakt in Docker Hub gespeichert und für das Deployment bereit ist, wird CD-Pipeline ausgeführt. Ähnlich wie bei der CI-Pipeline wird auch CD-Pipeline nach ein Pull Request auf Main ausgeführt. Am Anfang wird der Code erneut ausgescheckt, danach wird eine Verbindung zu Docker Hub hergestellt, für die wir unseren Docker Hub Username und Token brauchen. Um sicherzustellen, dass Username und Token nicht öffentlich zugänglich sind, müssen sie im GitHub Actions Secret gespeichert werden. Wenn die Verbindung erfolgreich hergestellt wurde, wird Buildx konfiguriert und Docker Image gebaut und gepusht. Anschliessend wird Kubectl installiert, der Kubernetes Context wird eingesetzt und der Code wird aus dem Actions Secret abgerufen und angewendet. Schliesslich werden die Kubernetes Manifeste aktualisiert und angewendet.

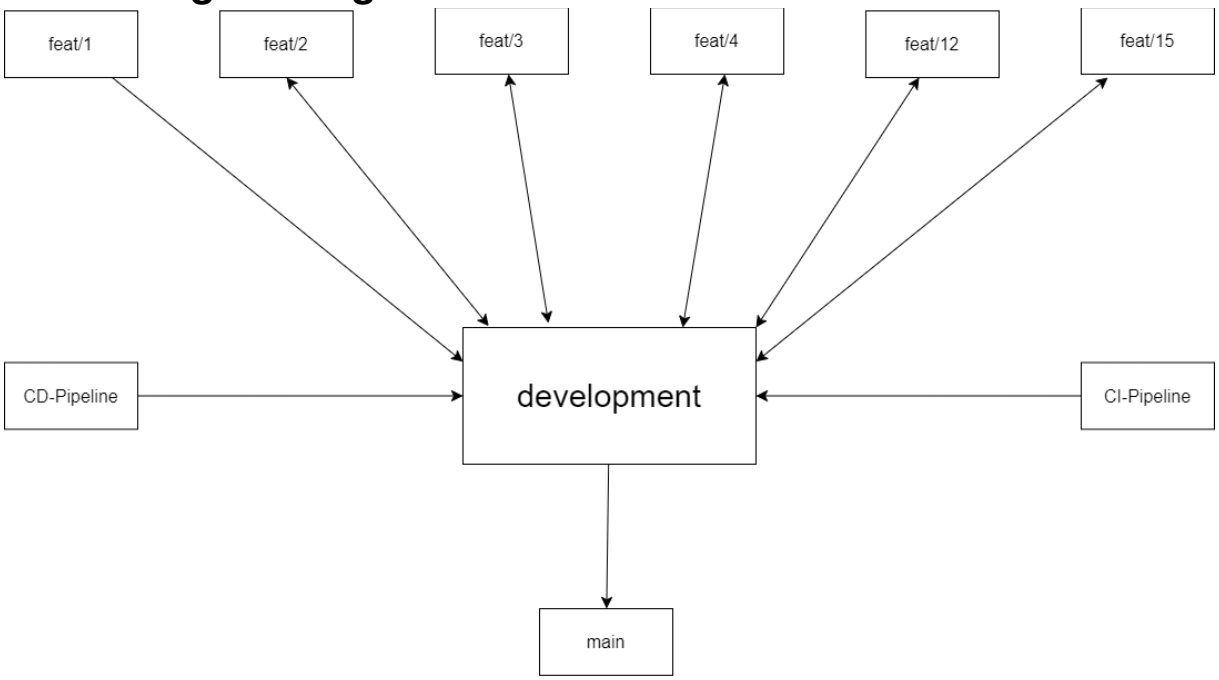
CI&CD-Pipeline



3 CI&CD-Pipeline

Am Anfang wird CI ausgeführt und dann CD. Wenn das Prozess erfolgreich abgeschlossen wurde, ist die Anwendung der Applikation bzw. der Webseite bereit. Man kann es testen, indem man den Link öffnen, den in Kubernetes Manifest definiert ist (https://mbrothers.m324.hackerman.ch).

Branching-Strategie



4 Branching-Strategie

Branchs	Wofür es steht?
Main	Hauptcode
Development	Arbeitsbereich
Feat/1	Darkmode für den Applikation
Feat/2	«Schreibt gerade ...» Funktion
Feat/3	Aktiven Benutzern auflisten
Feat/4	«/healthcheck» API
Feat/12	Abhängigkeiten installieren und konfigurieren
Feat/15	Formatierung der Code

Tabelle 1 Branching-Strategie

Am Anfang haben wir die Vorlage auf Main hochgeladen. Danach haben wir Development und alle Issues erstellt. Für jede Issue haben wir einen Branch erstellt, der feat/issue-nummer heisst. Herr Alnhlawe Morhaf hat an feat/1 gearbeitet und einen Pull Request zu Development erstellt. Herr Alnhlawe Mouayad hat den Code überprüft und in Development gemerged. Dann hat Herr Alnhlawe Mouayad Development in feat/2 gemerged und daran gearbeitet, gepusht und einen Pull Request von feat/2 zu Development erstellt. Herr Alnhlawe Morhaf hat den Code überprüft und in Development gemerged. Anschliessend hat Herr Alnhlawe Morhaf Development in feat/3 gemerged und daran gearbeitet, gepusht und einen Pull Request von feat/3 zu Development erstellt. Herr Alnhlawe Mouayad hat den Code überprüft und in Development gemerged. Dann hat Herr Alnhlawe Mouayad Development in feat/4 gemerged und daran gearbeitet, gepusht und einen Pull Request von feat/4 zu Development erstellt. Herr Alnhlawe Morhaf hat den Code überprüft und in Development gemerged. Herr Alnhlawe Mouayad hat dann dasselbe für feat/12 und feat/15 getan und für sie einen Pull Request erstellt. Herr Alnhlawe Morhaf hat den Code überprüft und sie dann in Development gemerged. Alles wurde am Ende von Development zu Main gemerged.

Für CI/CD mussten wir keine Branches erstellen und konnten sie direkt in Development mergen. Herr Alnhlawe Morhaf hat die CI-Pipeline erstellt und einen Pull Request erstellt. Herr Alnhlawe Mouayad hat den Code überprüft und in Main gemerged. Nachdem die CI abgeschlossen war, hat Herr Alnhlawe Mouayad an der CD gearbeitet und einen Pull Request erstellt. Herr Alnhlawe Morhaf hat den Code überprüft und in Main gemerged.

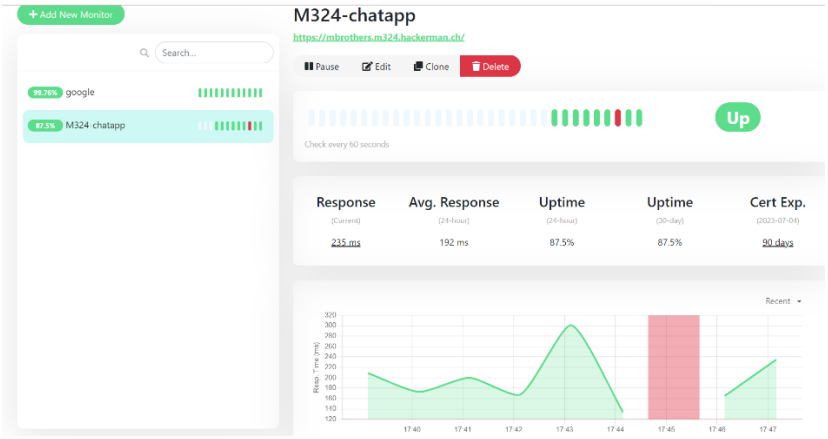
Schliesslich haben wir ein Release für den ganzen Applikation gemacht.

Uptime Kuma

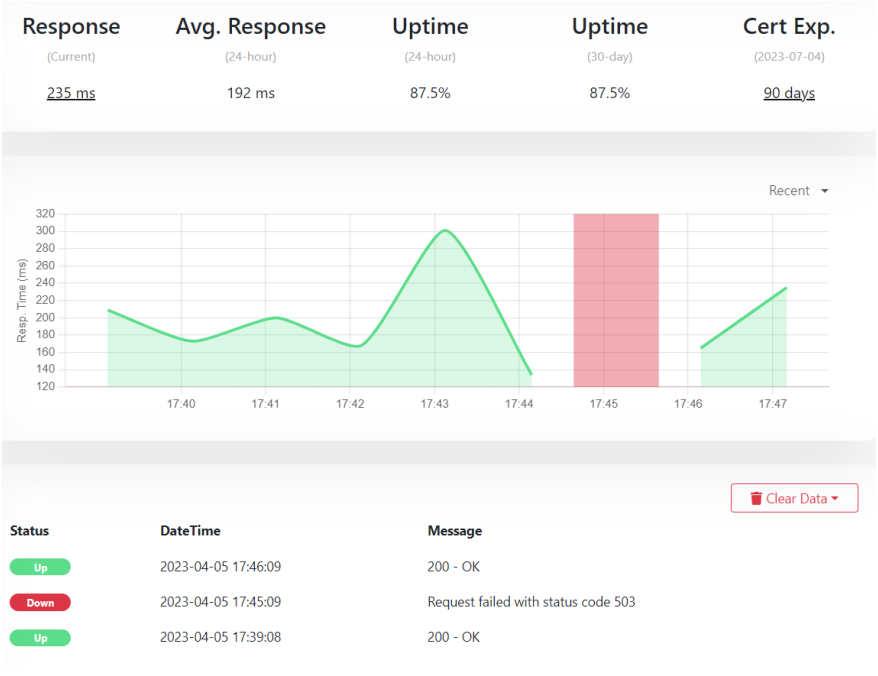
Zuerst haben wir einen Bot in Telegram erstellt, um Nachrichten dorthin gesendet werden. Der Bot heisst @MBrothersBot, wenn man diesem Chat beitreten, kann er den Status der Website sehen.

Nachdem wir den Bot erfolgreich erstellt hatten, starteten wir Uptime Kuma in Docker Desktop, damit wir ihn im Browser ausführen konnten. Bei Uptime Kuma haben wir einen neuen Monitor erstellt und ihn M324-chatapp genannt. Die URL, die wir verwendet haben, ist dieselbe von der CD, die wir dort definiert haben (mbrothers.m324.hackerman.ch). Für die Notifikation haben wir uns für Telegram entschieden, weil wir bereits Erfahrung damit haben. Bei den Bot-Token mussten wir unseren Bot-Token eingeben. Man findet seinen Bot-Token in Telegram unter @BotFather. Wir haben die Chat-ID des Bots von @RawDataBot erhalten und mussten sie bei Chat-ID hinzufügen.

Jetzt können wir immer den Status der Website sehen. Bis wir überprüfen können, ob alles funktioniert, haben wir die Seite gestürzt. Wie man in den folgenden Bildern sehen kann, haben wir es für eine Minute gestürzt, nur um zu prüfen, ob wir eine Nachricht in Telegram erhalten werden oder nicht.



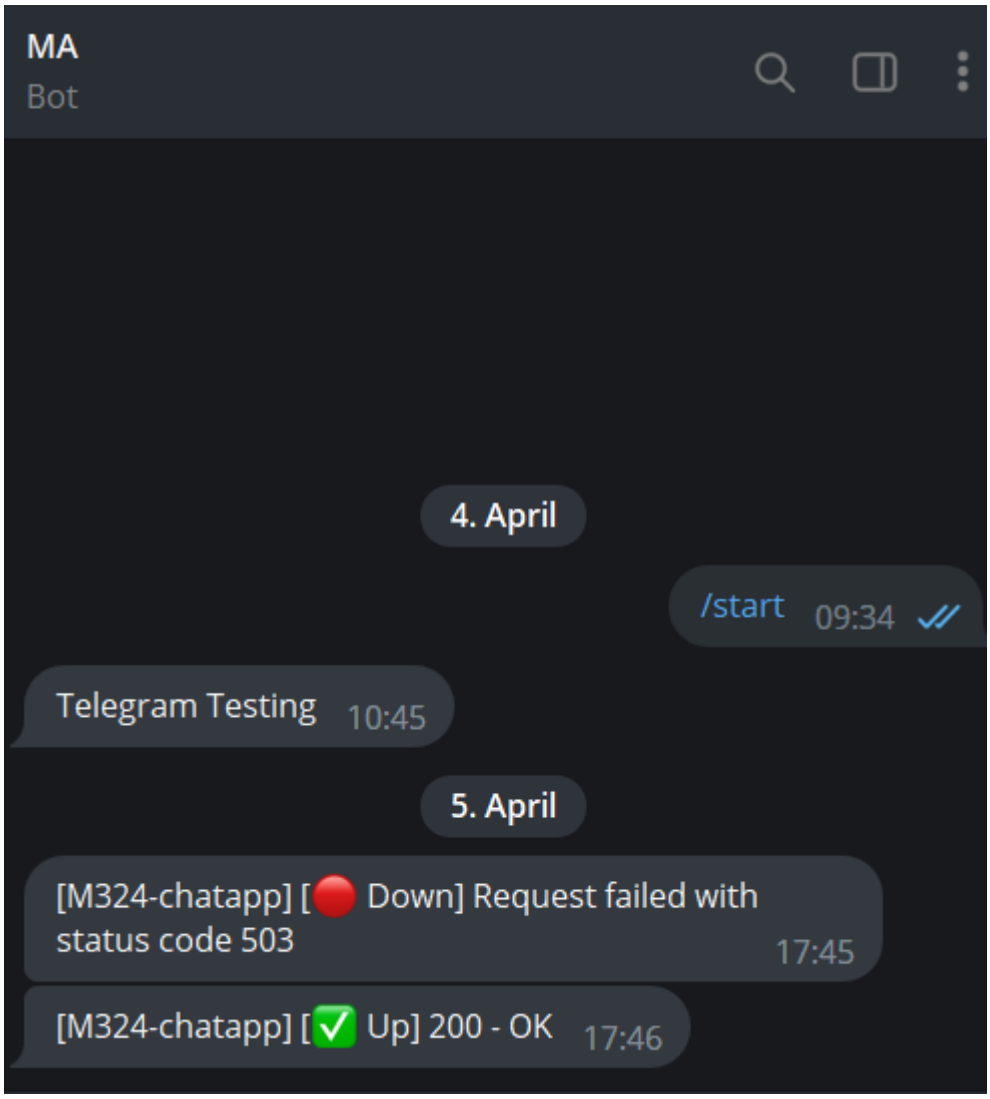
5 Uptime-Kuma



6 Uptime-Kuma

Telegramm-Chats

Folgend ist ein Bild von Telegramm-Chats und hier kann man sehen, dass wir eine automatische Nachricht erhalten haben, als wir die Website gestürzt haben und als die Website wieder funktionierte, haben wir eine Nachricht erhalten, dass alles wieder in Ordnung ist.



7 Telegram-Chat

Quellen

- Hausaufgaben
- Google Übersetzer
- Uptime Kuma für Monitoring
- Draw.io