Rapport de Projet
Architectures Orientées Services
CultureBox

AUGERAUD Lorelei
RAVANEL Tom
ZIOLKOWSKI Paul

Le projet

L'objectif du projet était de réaliser une API permettant de renseigner diverses collections de livres, séries ou films possédés par les utilisateurs. Il est, en effet, parfois compliqué d'avoir un aperçu global de ses collections et de tenir à jour ce que l'on a et ce qu'il nous manque. C'est là que notre API entre en jeu. Au-delà de cela, nous sommes convaincus que la culture devrait être accessible au plus grand nombre, pour soutenir cela, nous permettons aux utilisateurs d'avoir accès aux collections des autres et de faire des demandes d'emprunt pour profiter, eux aussi, de cette culture.

D'un point de vue plus technique, nous avons utilisé diverses APIs et technologies pour arriver à nos fins. L'API CultureBox se base en effet sur les résultats des recherches Google Book pour retourner des résultats concernant les livres ainsi qu'IMDB pour les séries et les films. Nous avons aussi décidé d'utiliser le C# et le framework ASP.NET Core afin de réaliser notre projet. Celui-ci est largement plébiscité pour créer des APIs. Ainsi, on ne manque pas de documentation ni d'exemples.

L'API tourne dans un conteneur Docker sur un serveur Linux. Celui-ci comprend aussi un Grafana et un Prometheus pour faire du monitoring.

Afin d'avoir une intégration continue, et du déploiement continu, nous avons décidé de la configuration de GitHub Actions. Cela permet de mettre en place des actions exécutées à chaque commit (dans notre cas), afin de tenter de faire le build de l'application, et de lancer les tests unitaires déjà écrits.

Pour la partie déploiement, nous avons décidé de garder un aspect manuel. Pour cela, il suffit de renseigner la chaîne "[Deploy]" dans un message de commit afin que le déploiement se fasse. Grâce à Docker, c'est très simple : il suffit de créer une image docker, de l'envoyer sur le docker Hub, puis de se connecter en SSH sur la machine hébergeant notre application, et lancer la mise à jour de l'image Docker.

Vous pouvez retrouver les différentes ressources du projet aux adresses suivantes :

Lien du projet sur GitHub :
https://github.com/Fraaktal/CultureBox

Lien de l'API :
http://server-fraaktal.ddns.net:4208/book

Lien du swagger contenant la documentation de l'API :
http://server-fraaktal.ddns.net:4208/swagger/index.html#/

Les problèmes rencontrés

Pour le monitoring de notre infrastructure, nous voulions à l'origine monitorer l'instance de docker présente sur le serveur, sans passer par l'exportateur Prometheus docker. Cela passait par la récupération des métriques exportées par l'instance, directement via le docker. Cela s'est avéré très compliqué, car cela impliquait de récupérer les données, accessibles sur le localhost de la machine, depuis un container (notre Prometheus). Après plusieurs heures, nous avons changé notre fusil d'épaule, car cela était contraire à la politique de docker, qui vise à mettre de côté chaque instance, indépendamment de la machine physique.

À la place, nous avons mis en place un container docker_exporter qui exporte l'ensemble des données de notre docker. Prometheus a ensuite accès à ces données grâce aux options que docker-compose met à disposition.
Toujours sur le thème du monitoring, nous avons eu une difficulté au niveau de la persistance des données de notre Grafana. Au départ, nous ne sauvegardions pas nos données, au moyen du volume. Cela a donc été compliqué par la suite de le faire. Grâce à votre aide, nous avons pu le mettre en place, et sans que cela ne compromette nos données.

Concernant le développement en lui-même de l'API, il n'y a pas eu de problème particulier. Le framework permettant de facilement tout mettre en place. Nous nous sommes cependant imposé un certain niveau de contrainte en termes de code coverage sur les requêtes de l'API en liaison directe avec les utilisateurs afin de s'assurer que celle-ci se comportaient comme on l'attendait.

La partie APIControllers est testée quasi-entièrement à l'exception de cas purement théoriques qui ne peuvent pas être reproduit facilement avec des tests unitaires.

Certains ont eu un peu de mal à comprendre comment fonctionne le C#, pour la rédaction des tests et sont donc, petit à petit, montés en compétences pour réussir à rédiger les tests demandés.

Conclusion

Le projet a dans l'ensemble été bien conduit. Nous avons mis en place les différents types de requêtes (GET, POST, PUT, DELETE), ajouté des filtres de pagination et de recherche ainsi que bien traité les différentes erreurs avec des codes d'erreur explicites. La documentation est complète, nous avons des tests unitaires qui tournent, de la CI/CD et du monitoring. On peut regretter l'absence de valeur par défaut sur certaines énumérations ce qui aurait rendu plus contraignant mais plus sur l'utilisation de l'API lors de requêtes d'emprunts. L'application tournant sur le serveur personnel et physique de Tom, il faut aussi espérer qu'aucune coupure de courant intempestive ne vienne couper l'accès au service. Nous n'avons en effet pas prévu de PRA. Hormis cela, nous sommes plutôt satisfaits et nous avons tous beaucoup appris avec ce projet que ce soit les technologies de developpement (ASP / C#) que les outils comme Docker ou la CI/CD.

Concernant le cours, nous sommes satisfait des différents thèmes abordés ainsi que le format et le rythme de celui-ci que cela soit pour Docker, Kubernetes, …