Motor meghibásodás előrejelzése

Szoftver rendszerek modellezése

Sapientia EMTE
Szoftverfejlesztés I.
Ballai Tamás-Attila

1. Bevezetés

A RUL (Real Useful Lifes), azaz valódi hasznos élettartam előrejelzése egyre inkább elterjedt alkalmazott módszer arra, hogy bizonyos motorok élettartamát előre megjósolják, így megspórolva egy jelentős költséget, ami a meghibásodott motorok kicserélésére és az általuk okozott károkra menne el. A motorok meghibásodása számos költséggel járhat, mint például a gépek üzemen kívül kerülése, egy új motor vásárlása, beszerelése, üzembe helyezése, az eltelt szünidő alatt keletkezett veszteség stb. Egy motor élettartamának előrejelzése segíthet abban, hogy fogalmunk legyen róla, mennyi ideig fog működni, kell-e jelentősebb karbantartásra számítani a közeljövőben, meg lehet-e javítani, mielőtt mégjobban meghibásodik, így

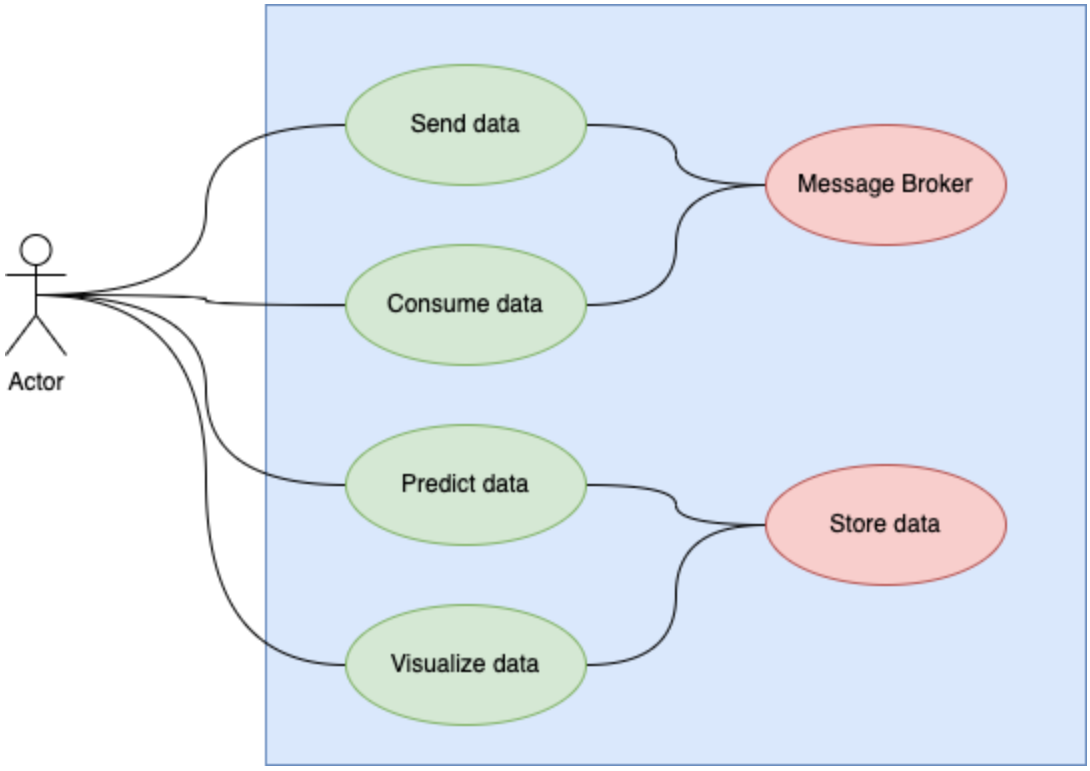
növelhetjük az élettartamát is és elkerülhetünk egy esetleges üzemen kívülre kerülést, ami súlyos veszteségeket okozhat.

2. Célkitűzések

- Felhasználni egy RUL előrejelző módszert és alkalmazni más, hasonló adatokra, más RULok előrejelzésére
- Felépíteni egy rendszert, ahol az adatok egy service-től érkeznek, szenzort szimulálva, egy másik service megkapja és elvégzi az előrejelzést
- A service-ek valamilyen aszinkron protokollon keresztül kommunikáljanak
- A valódi RUL és előrejelzett RUL adatok tárolása későbbi vizualizációk céljából
- A valódi RUL és előrejelzett RUL adatok vizualizálása összehasonlítás céljából valamilyen felhasználói felületen
- A service-ek külön konténerekben fussanak egy Kubernetes clusterben.

3. Felhasználói követelmények

Mivel a rendszerhez nem készült külön felhasználói felület, a service endpointokat Postman nevű programból lehet meghívni és utána, a Kibana saját UI-án lehet ábrázolni az adatokat.



1. Ábra - Use case

Postmanból 3 endpointot kell meghívni, egyet a sender service-től, ami elküldi az adatokat, kettőt pedig a prediction service-től, egyikkel fogadja, majd a másikkal elvégzi az előrejelzést. A küldésnél és az előrejelzésnél meg kell adni egy számot, ami az adathalmazt mutatja, 4 különböző van, 1-től 4-ig számozva:

- Sender: 10.10.132.193:8010/send/3
- Prediction: 10.10.132.196:8020/consume, 10.10.132.196:8020/predict/3.

Az előrejelzés befejezte után fel kell lépnie a felhasználónak a Kibana oldalára, ott létrehozni egy Index patternt, természetesen egy Index már kell létezzen, ahova az Elasticsearch az adatokat mentette, ezután pedig egy dashboardon lehet ábrázolni az adatokat.

4. Rendszerkövetelmények

4.1. Funkcionális követelmények

A rendszernek kell egy betanított modell és egy skálázó, ami illesztve van az adatokhoz, ezek kigenerálását egy szkript végzi el, amit egyszer le kell futtatni, ha még nincsenek jelen. A

sender service egy fájlból olvassa ki a szimulált szenzor adatokat, amelyeket el kell küldjön egy protokollon keresztül a másik service-nek. Ezeket az adatokat soronként küldi, tehát felbontja az adathalmazt sorokra. A protokoll ebben az esetben egy Kafka Message Broker, ami tárolja az adatokat mindaddig, amíg a prediction service ki nem olvassa. A prediction az endpoint meghívásával elkezdi olvasni az adatokat a már előtte rácsatlakozott protokollon keresztül. Miután megérkezett minden adat, ezeket egy másik endpoint meghívására feldolgozza és előrejelzést készít a motorok élettartamáról. Ezeket az adatokat elmenti egy indexen az Elasticsearch kereső motor.

4.2. Nem funkcionális követelmények

A service-ek Python nyelvben vannak megírva, Docker segítségével készültek az image-ek, amelyekből majd containerek lettek indítva Kubernetes környezetben. Az adatok átküldéséhez Kafka Message Brokert használtam, amelyben topicokra lehet küldeni adatokat és egy topic megadásával ki lehet olvasni, ha van adat az adott topicon. Az adatok tárolására és vizualizására az Elastic Stack csoportból az Elasticsearchot és a Kibanat használtam.

5. Tervezés

A rendszer megtervezését egy adathalmaz kiválasztásával kezdtem. Kezdetben az adatokat saját magam akartam kigenerálni valamilyen szkript megírásával, viszont a módszer, ahogy ezeket az előrejelzéseket végzik, eléggé komplex lehet és az adathalmaz, amiket felhasználnak, jól elkülönített train, test és RUL adathalmazokba van sorolva. Ezért egyszerűbbnek tűnt áttérni és felhasználni egy adathalmazt a hozzátartozó előrejelzést végző módszerrel együtt.

A Kaggle oldalán találni lehet néhány ilyen adathalmazt és forráskódok is vannak különböző személyektől. A NASA turbóventilátoros sugárhajtómű adathalmazát használtam.

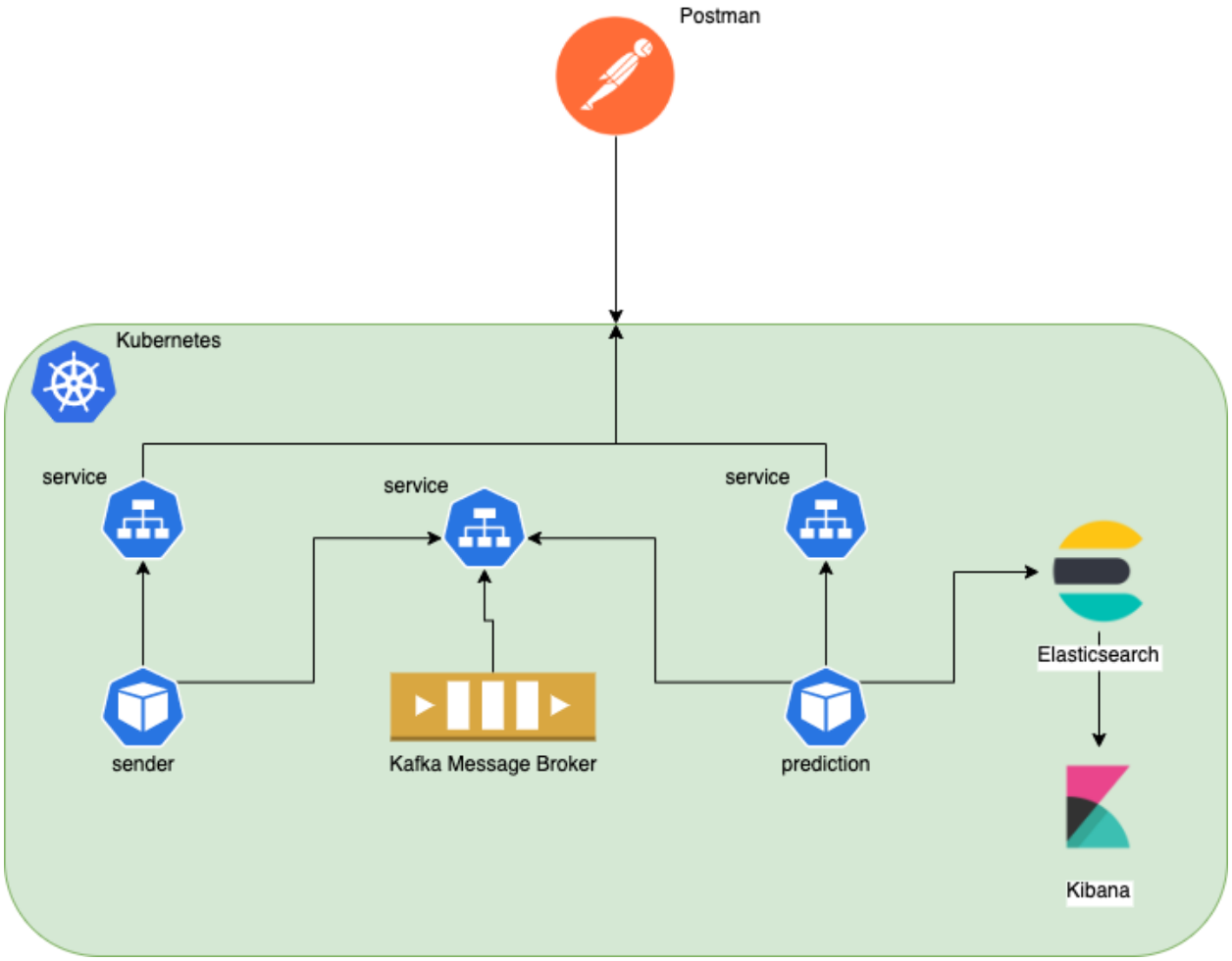
Az adathalmaz megválasztása mellett még el kellett dönteni, hogy milyen programozási nyelvet haszálok. Mivel az adatfeldolgozás többnyire Pythonban történik, ezért ezt választottam.

A protokoll megválasztása azért esett a Kafkára, mivel egyrészt már dolgoztam vele, másrészt pedig nagyon gyors, nagyon sok service esetén is használható, jól skálázható és egyre elterjedtebb a használata.

Egy másik, egyre több területen alkalmazott eszköz az Elastic Stack, ezért a tárolást és vizualizációt ezzel oldottam meg.

Mindezeket Kubernetes konténer összehangoló rendszerben futtattam, mivel ehhez van hozzáférésem és nem ütközött semmilyen költségbe a használata.

5.1. Architektúra



2. Ábra - Architektúra

Felhasználói felület híján, a service-eket Postman vagy hasonló eszköz segítségével tudjuk meghívni. A hívások a service-ekhez érkeznek, ezekhez pedig a tényleges applikáció konténere van hozzákötve a megfelelő egy-egy Kubernetes Pod elem használatával. A konténerek a Kafka service-án keresztül tudják elérni a Kafkát írásra vagy olvasásra. A prediction konténer pedig csatlakozva van egy Elasticsearch kereső motorhoz, ahova tárolja az adatokat, ebből pedig a Kibana majd ki tudja olvasni.

Postman - egy API platform APIk építéséhez és használatához.

Kubernetes Container Orchestration System - a Kubernetes egy konténer összehangoló rendszer, ami segítségével nem a lokális gépünkön futtatjuk az alkalmazásokat,

hanem külön konténerekben, más számítógépen, amiket monitorizálhatunk skálázhatunk, biztonságossá tehetünk különféle Kubernetes elemekkel.

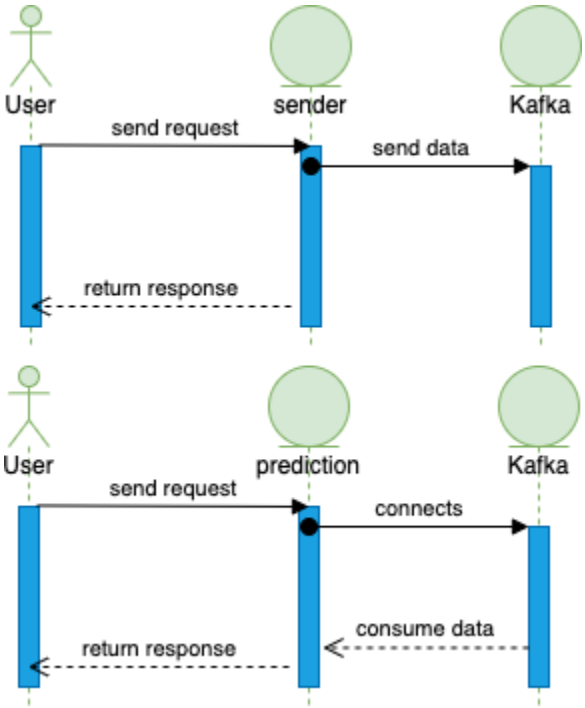
Pod - olyan Kubernetes elem, ami a legkisebb egység egy konténer indítására és egy réteget húz fel a konténerre. Több konténert is indíthatunk egy Podban, viszont semmilyen alkalmazás nem indulhat Podon kívül.

Service - egy absztrakciós réteg, ez nem tényleges applikáció, hanem a Pod láthatóságát határozzuk meg attól függően, hogy milyen típust használunk.

Kafka - a Kafka egy osztott event streaming platform, amelyet világszerte használnak nagyobb és nagyobb vállalatok. Nagyon gyors, aszinkron kommunikációt biztosít, elbír több millió üzenetet másodpercenként anélkül, hogy lelassúlna.

Elastic Stack - az Elastic Stack csomagba tartozik az **Elasticsearch** és a **Kibana**, amelyek szintén nagyon elterjedtek. Az Elasticsearch tulajdonképpen a motorja az egész csomagnak, ez egy kereső motor, amellyel nagyon gyorsan kereshetünk rengeteg adat között is. A Kibana egy vizualizációs felületet biztosít, amellyel az Elasticsearch indexeiben tárolt adatokat érjük el.

5.2. Alkalmazás működése



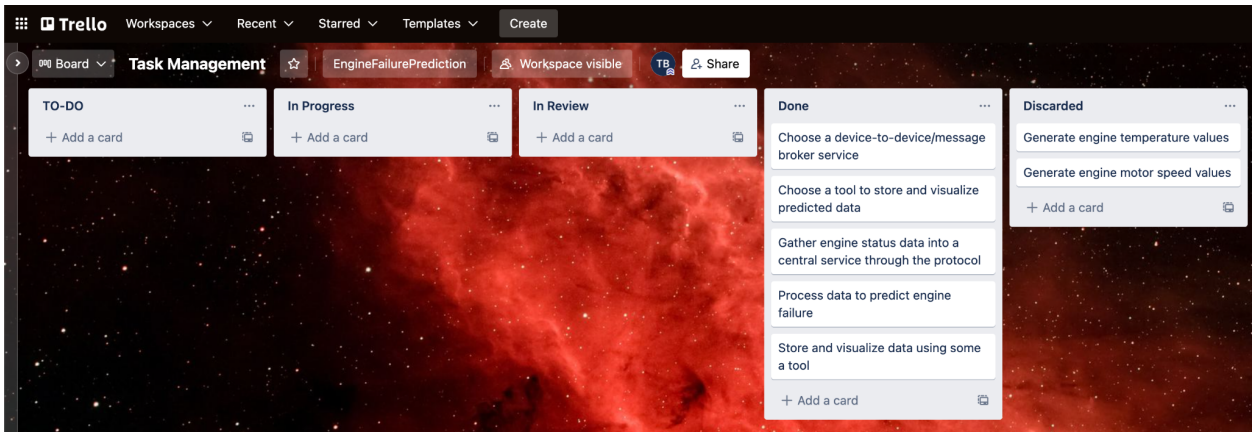
3. Ábra - Szekvencia diagram

A felhasználó kérést küld a sender service-nek, az elküldi az adatot a Kafkának és visszatér egy válasszal. A prediction esetében elkezdi olvasni a Kafkából az adatokat a service, a végén pedig küld egy választ a felhasználónak.

6. Megvalósítás

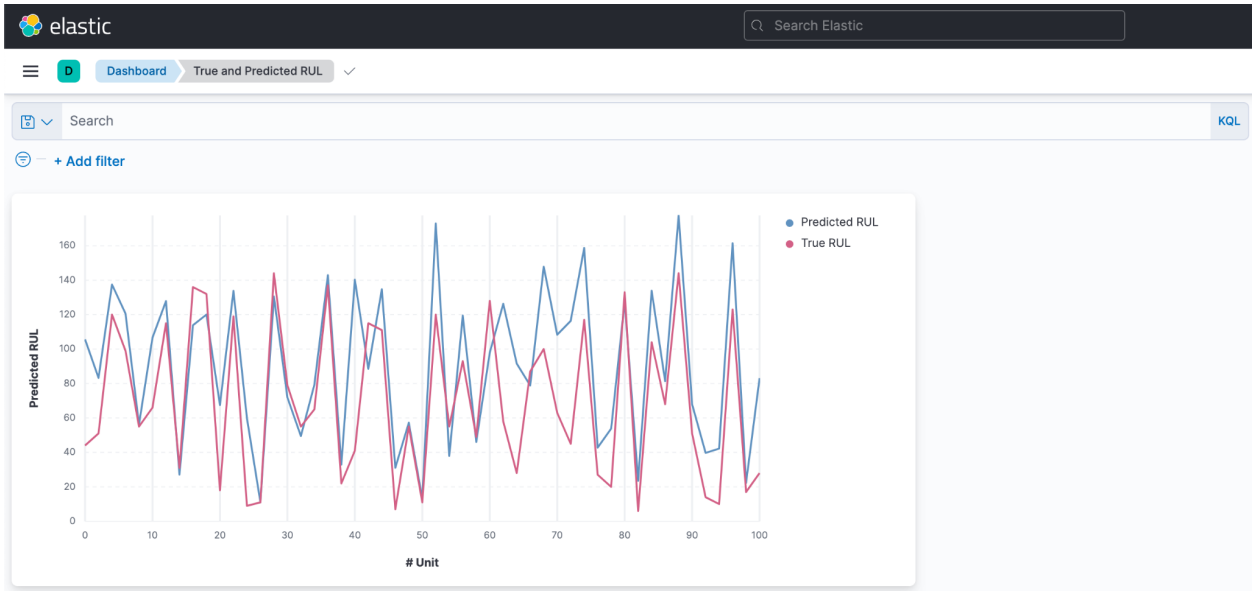
A célkitűzéseket sikerült megvalósítani, az adat előrejelzés külön szkriptben működött, tehát kimentettem a betanított modelt és skálázót, hogy a prediction service-ben tudjam felhasználni más adatok előrejelzéséhez. A service-ekről Docker image készült, amelyekből konténereket indítottam a Kubernetes clusterben. A service-ek kommunikálnak egymással a Kafkán keresztül, átjut az adat, amit a prediction service feldolgoz és új adatokat jósol. Ezeket az adatok elmentődnek az Elasticsearchben és a Kibana oldalán ezeket meg tudjuk tekinteni, vizualizálni tudjuk őket. A vizualizációkról dashboardokat készíthetünk.

A projekt egy Github repositoryban lett létrehozva, ezért verziókövetés is lehetséges. A különböző lépések megszervezésére és sorrendbe helyezésére létrehoztam egy Trello boardot és néhány minimális lépést abban vezettem le.



4. Ábra - Trello

7. Ábrázolás



5. Ábra - True és Predicted RUL

Az ábrán pirosban a valódi élettartam látszik, kékben pedig az előrejelzett. A vízszintes tengelyen az adott unit, azaz motor egyedi azonosítója látszik, a függőleges tengelyen pedig az élettartam ciklus számban kifejezve.

Az előrejelzett adatok mondhatni elég jól ráhelyezkednek a valódira, persze egy-egy helyen látszik az eltérés. További fejlesztésekkel talán javítani lehet még ezeken az eltéréseken.

8. Összegzés

A rendszert sokféleképpen lehet még bővíteni, ha csak az adatfeldolgozás részét is nézzük. Sokban lehet még javítani az osztályozót, kipróbálni más osztályozókat stb.

A rendszerben a valós RUL adatokat a prediction service beolvassa file-ból. Ezt is lehetne service-en keresztül küldeni, továbbá felbontani a másik adatokat is, több szenzort szimulálva.

A service-ek föle lehet alkalmazni egy Ingress Kubernetes elemet, így nem kell külön IP címeket beírni URL-be, hanem egy konstans címen érhetnénk el a service-eket.

Egy kezdetleges rendszernek tekinthető a projekt, ami eléggé megközelíti a valódi élettartam értékeket és továbbfejlesztve még többet lehet ezen javítani.

9. Bibliográfia

https://www.kaggle.com/datasets/behrad3d/nasa-cmaps
https://medium.com/@polanitzer/prediction-of-remaining-useful-life-of-an-engine-based-on-sensors-building-a-random-forest-in-ffad82c8a1c6
https://github.com/aio-libs/aiokafka
A projekt:
https://github.com/Tamas99/EngineFailurePrediction
https://trello.com/b/mOf8yTG8/task-management