SSEPW24

- *https://microservices.io/* Chris Richardson
  - *https://microservices.io/presentations/index.html* PRESENTATIONS
  - *https://microservices.io/microservices/2020/02/04/jfokus-geometry-of-microservices.html* INTRODUCTION
  - *https://microservices.io/patterns/index.html* A pattern language for microservices
- *https://chrisrichardson.net/training-architecting-for-fast-flow.html* a pagamento exipred

Software Systems Engineering Project Work M 2024

Progettazione e costruzione di microservizi con oggetti e/o attori

Lo svolgimento comprende le seguenti fasi:

1. Cosa sono i nicroservizi e quali sono le problematiche progettuali e gestionali ad essi connesse. Come si porgetta un sistema a microservizi (*Microservices Patterns: With Examples* in Java *https://www.amazon.it/gp/product/1617294543/ref=ppx_yo_dt_b_search_asin_title?ie=UTF8&psc=1*)
2. Come si costruisce e si distribuisce un sistema a microservizi (*Microservices with Spring Boot 3 and Spring Cloud* *https://www.amazon.it/gp/product/1805128698/ref=ppx_yo_dt_b_search_asin_title?ie=UTF8&psc=1*)
3. Quale è il ruolo dei framework come Spring (o Node o altro).
4. Come si imposta la business logic di un ms usando oggetti (java) nel quadro delle clean architectures
5. Come si imposta la business logic di un ms usando attori. Gli attori potrebbero anche fungere da enti sostitutivi di piattaforme come Spring (con particolare riferimento a sistemi 'small' di tipo IOT)?
6. Sperimentazione di deploy mediante **Kubernetes** (tema coperto nel testo riportato in 3)

- I punti 1)-3) sono ovviamente propedeutici e forse già affrontati in qualche corso
- **I punt 4) e 5) sono il 'core'** della attività che potrebbe (auspicabilmente, ma non necessariamente) estendersi al punto 6.

Si tratta di capire / sperimentare le motivazioni che possono indurre un progettista a usare attori come elementi costitutivi primari dei microservizi o (forse meglio) come componenti interni di framework come Spring (o Node o altro).

Si tratta anche di costruire un sistema-demo da discutere nel corso del prossimo a.a. Durante l'avvità progettuale si potrebbe cooperare per lo sviluppo di questo sistema, magari sperimentando anche metodologie e metodologie/tools di **DevOps**.

Il sito è **https://github.com/anatali/mcrsv24**.

Sullo sfondo, sta nascendo una tematica che oggi avanza in ambiente industriale, ma che risulta comunque subordinata alla parte precedente.

- L'impatto del machine learning sui microservizi (*Machine Learning in Microservices* *https://www.amazon.it/gp/product/1804617741/ref=ppx_yo_dt_b_asin_title_o00_s00?ie=UTF8&psc=1*)

Materiale

- Video *ISS25 Primi passi* relativo al software da installare: *https://unibo.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=7ab5ed80-3df7-488c-b59d-b1dd00e8583b*
- Video *ISS25 Un primo servizio* relativo al servizio **CargoProduct**

Case study: cargo system

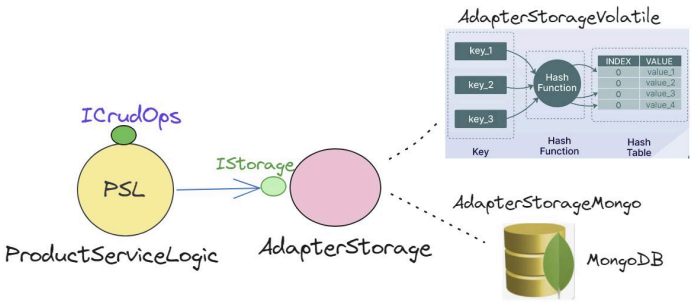
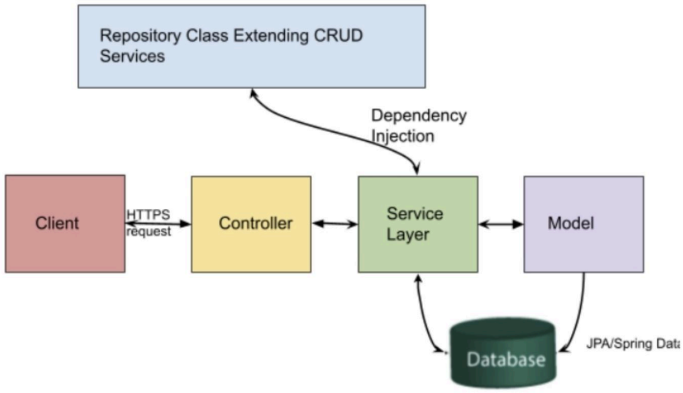
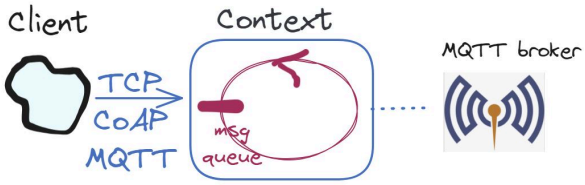
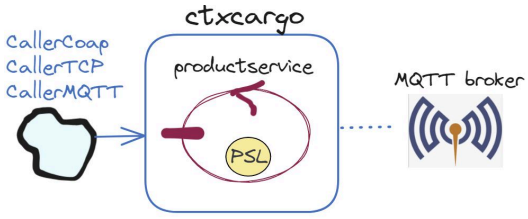
Un nave trasporto (cargo) permette a clienti di caricare prodotti da trasportare a un porto di destinazione. I prodtti vengono caricati da un trolley nella stiva della nave. Occorre realizzare un sistena software che gestisce i prodotti, la loro collocazione nella stiva e il loro trasporto dalla zona di carico/scarico alla stiva e viceversa.

Si intende organizzare il sistema software seguendo quanto indicato in *https://microservices.io/microservices/2020/02/04/jfokus-geometry-of-microservices.html*

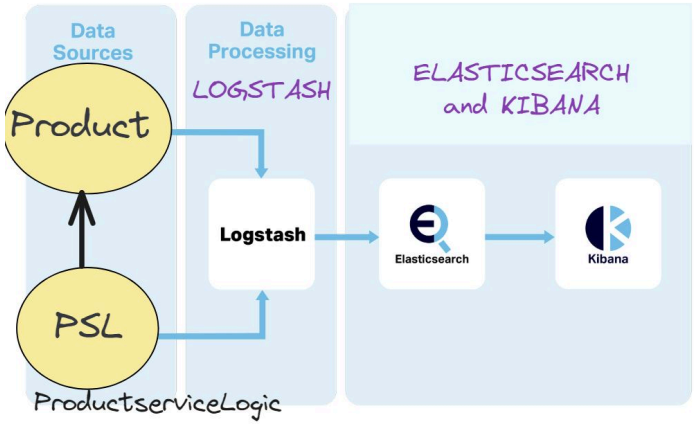
Primo microservizio

1. Microservizio di memorizzazione prodotti:
2. Come actor attorno a un POJO **ProductServiceLogic**

3. **ProductServiceLogic** usa **AdapterStorage** per realizzare persistenza in memoria (**AdapterStorageList**) oppure su *MongoDB* (**AdapterStorageMongo**)
4. Che usa *logback.xml*, *elasticseacrh* e *logstash*, attivati usando **docker-compose-EFKOnly.yml**
5. Che si avvale di **ProductServiceCallerCoap** come client
6. Deployed mediante Dockerfile e **docker-compose.yml**
7. Che fa parte di un insieme più ampio di microservizi

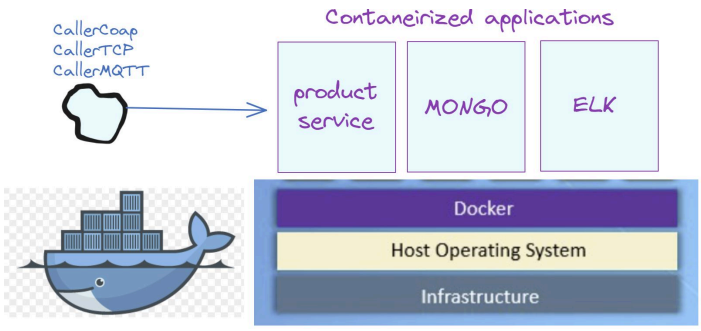
ProductServiceLogic	 Test Unit: *TestProduct* in **cargo\src\main\java\test\TestProduct.java**
SptingBoot	
Actor (qak)	
productservice Actor	

ProductService and Logging



Test Unit: *TestCargo* in **cargo\src\main\java\test\TestCargo.java** using *Cargo logback.xml*

Dockerized ProducttService



Situazione codice 1 Ottobre 2024

1. Progetto **cargoproduct** completo con logging, *AdapterStorageMongo*, *StorageVolatile-singleton*,

   ProductServiceLogic.createProduct ha un delay di 4sec tra **get==null** e **dataStore.createItem** per aumentare la probabilità di avere un errore di duplicazione di prodotti dovuto all'Interlacciamento.

   Distribuzione: **cargoproduct-1.0.jar** slim

   TODO (???): trovare un modo 'agile' per dare la possibilità di selezionate il tipo di Storage senza cablare la scelta nel codice.

2. Introduzione al tema *ComputareComunicare* come elemento fondamentale per i microservizi (productservicelogicNat/userDocs/ComputareComunicare .pdf)

3. Progetto **cargoservicespring**

TODO (Riccardi): completare con estensioni simili a quanto introdotto in **productservicelogicNat** per trattare i seguenti punti (legati al **tema Interazione/Interlacciamento**):

- RestController che opera con loggtong ELK
- RestController che gestisce POST su */createProductStr* in modo sincrono (*/createProductStr*)
- RestController che gestisce POST su */createProductAsynch* in modo **asincrono**
- client che invoca metodi in modo sincrono usando *HttpConnection* su HTTP (*PSLCallerHTTPSynch.java*)
- client che invoca metodi in modo sincrono usando *Interaction* su HTTP (*PSLAnotherCallerHTTPSynch.java*)
- client che invoca metodi in modo asincrono usando *HttpURLConnection* (*PSLCallerHTTPAsynch.java*)

TODO (Riccardi): verificare che, con esecuzione asincrona e con la nuova distribuzione di **cargoproduct-1.0.jar** , si potrebbero creare prodotti duplicati su storage volatile. VIDEO: ESPERIMENTIHTTPsynch/asynch: *https://unibo.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=17f063c3-a30d-4112-8dc1-b1f90110f4ae*

Discutere possibili soluzioni per superare il problema.

4. Impostazione del progetto **cargopservice** con l'idea che i qak siano un modo per esprimere nanoservizi (productservicelogicNat/userDocs/MicroNanoServizi.pdf) con comunicazioni efficienti in locale. Macro-cosmo riprodotto nel micro-cosmo: *Dal macro al micro*.

5. Discutere le nuove problematiche che emergono dall'uso degli attori: si veda *Problematiche usando attori* in. productservicelogicNat/userDocs/MicroNanoServizi.pdf.

   TODO (Riccardi): valutare se la realizzazione di uno *StorageRAMActor* potrebbe esser un utile esempio dei vantaggi che si hanno nella ripartizione di un monolite in microservizi.

6. TODO (Riccardi): SpringBoot come framework per microservizi che si avvale non dei POJO, ma dei *Nanoservizi* qak posti in escuzione in locale dal RestController oppure già attivi su un container docker. Discutere i pro-contro dei due approcci.

7. TODO (Natali): Impostazione di un nanoservizio per il carico/scarico di un prodoto nel cargo usando un tobot.

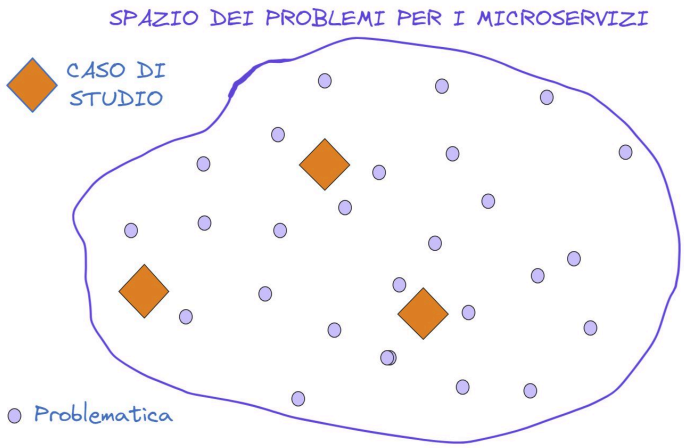
Video

```
PRIMI PASSI
https://unibo.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=7ab5ed80-3df7-488c-b59d-b1dd00e8583b

COMPLETAMENTO WORKSPACE
https://unibo.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=349d30ac-87b7-4890-a09b-b1e7008e50ee

CARGOPRODDUCT
https://unibo.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=d3598b13-35a6-4732-9924-b1e800f3eec0
verso metà: logback.xml CommUtils verso fine: JUnit e testlog + gradle test con report

PRODDUCTSERVICELOGICINTRODUCION
https://unibo.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=ff1c4610-af3f-423f-a09e-b1ea00f74ba1

PRODDUCTSERVICELOGICDISTRIBUTION
https://unibo.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=ea12ece8-ff3c-41df-b160-b1ea00f6944b

PRODDUCTSERVICELOGICIMPLEMENTATION
https://unibo.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=aafd07fe-f4cb-469a-9b68-b1ea00f4ce84
Verso la fine: test con report, distribuzione fat ed esecuzione

ELKINTRODUCTION&USAGE
https://unibo.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=31e7d7e8-89da-46da-838e-b1f200e81058

KIBANAUSAGE
https://unibo.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=d0a56e0d-5193-47da-809e-b1f800e857ed

ESPERIMENTIHTTPsynch/asynch
https://unibo.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=17f063c3-a30d-4112-8dc1-b1f90110f4ae
```

Flusso logico report finale Riccardi

Riporto uno schema di massima del flusso logico del report finale indicando il tipo di contenuto che si potrebbe inserire in ciascuna sezione. Le parti indicate con CORE sono quelle più rilevanti per l'attività progettuale e dovrebbero eseere sempre accompagnate dalla una evoluzione del progetto Cargo.

I progetti realtvi al caso di studio Cargo toccano un numero limitato di aspetti dei microservizi che si ritengono tra i più rilevanti e formativi.



1. **Cosa sono** i microservizi e perchè sono importanti (Survey esempio:productservicelogicNat/userDocs/MicroNanoServizi.pdf) Un altro utile riferimento: *https://medium.com/geekculture/introduction-to-microservices-9dcaafa5d882*

2. Quali sono le **problematiche** progettuali e gestionali ad essi connesse (Survey) Connesso con il punto 5 *Guida al progetto di un sistema a microservizi*.

3. **Computare non basta**: occorre anche comunicare e (inter)agire (Rilfessione opzionale) (productservicelogicNat/userDocs/ComputareComunicare .pdf)

4. **Framework che abilitano la comunicazione**: da *SpringBoot* (e interazione Restful) a *Interaction* (comunicazione come nuova forma contrattuale). (CORE)

5. **Guida al progetto di un sistema a microservizi** presentazione dei Microservices patterns coem quadro di riferimento egli spazi progettuali a livello di **Infrastructure**, **Application infrastructure** e **Application** (productservicelogicNat/userDocs/MicroservicePatternLanguage.pdf)

   - Un altro utile riferimento: *https://medium.com/geekculture/best-practices-for-microservices-architecture-9cd896fb41b5*
   - *https://microservices.io/patterns/decomposition/decompose-by-subdomain.html* : decomposizione per sottodominio (CORE)
   - *https://microservices.io/patterns/microservices.html* : Microservice Architecture (CORE)
   - Api Gateway, Service Discovery, Circuit Breaker, Event Sourcing, CQRS, Saga, etc. (CORE)
   - Il concetto di *aggregator*: un possibile riferimento *https://medium.com/nerd-for-tech/design-patterns-for-microservices-aggregator-pattern-99c122ac6b73*

6. **Come si progetto** (CORE) un sistema a microservizi alla luce delle *clean architectures*, partendo da oggetti Java

   Progetto **cargoproduct** che evolve in **cargoservicespring** e **cargoservice**

7. L'utilità di un **sistema di logging** come *ELK* in un sistema a microservizi (CORE)

8. Come si affronta il *problema della Gestione dello stato*: servizi *stateless* non sono sempre possibili. Il caso di studio implica servizi stateful (CORE)

9. Come si affronta il *problema dell' Interlacciamento* che nasce dai pattern asincroni Ripercussioni sui progetti precedenti ((CORE)).
10. Come si affrontano il *problema della consistenza dei dati*. ((CORE)). Il concetto di *saga*.
11. Il dilemma *orchestrazione/coreografia* ((FORSE))
12. Come si **costruisce** un sistema a microservizi ((DevOps?))
13. Come si **distribuisce** un sistema a microservizi: il ruolo di **Docker** ((CORE))
14. Microservizi e IOT ((Survey))
15. Dai microservizi ai *nanoservizi*
16. Gli attori come nanoservizi: il macro-mondo si rilflette/riproduce nel micro-mondo.
17. Aggiungiamo ((CORE)) al (**cargoservice**) un insieme di nano servizi che muovono robot per il carico/scarico dei prodotti
18. Come si fa il **testing** di un sistema a microservizi (alla fine, perchè può essere più conveniente fare esempi usando l'osservabilità degli attori qak).
19. La relazione tra i messaggi di **log** e il **testing**: da Kibana all'uso di **AI** ((FORSE/CENNO))