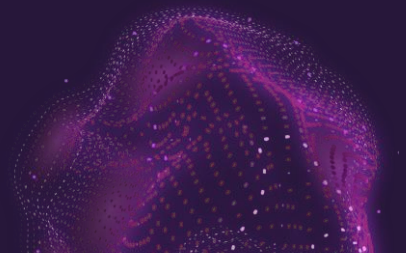
Strumento di refactoring automatico per architetture a microservizi

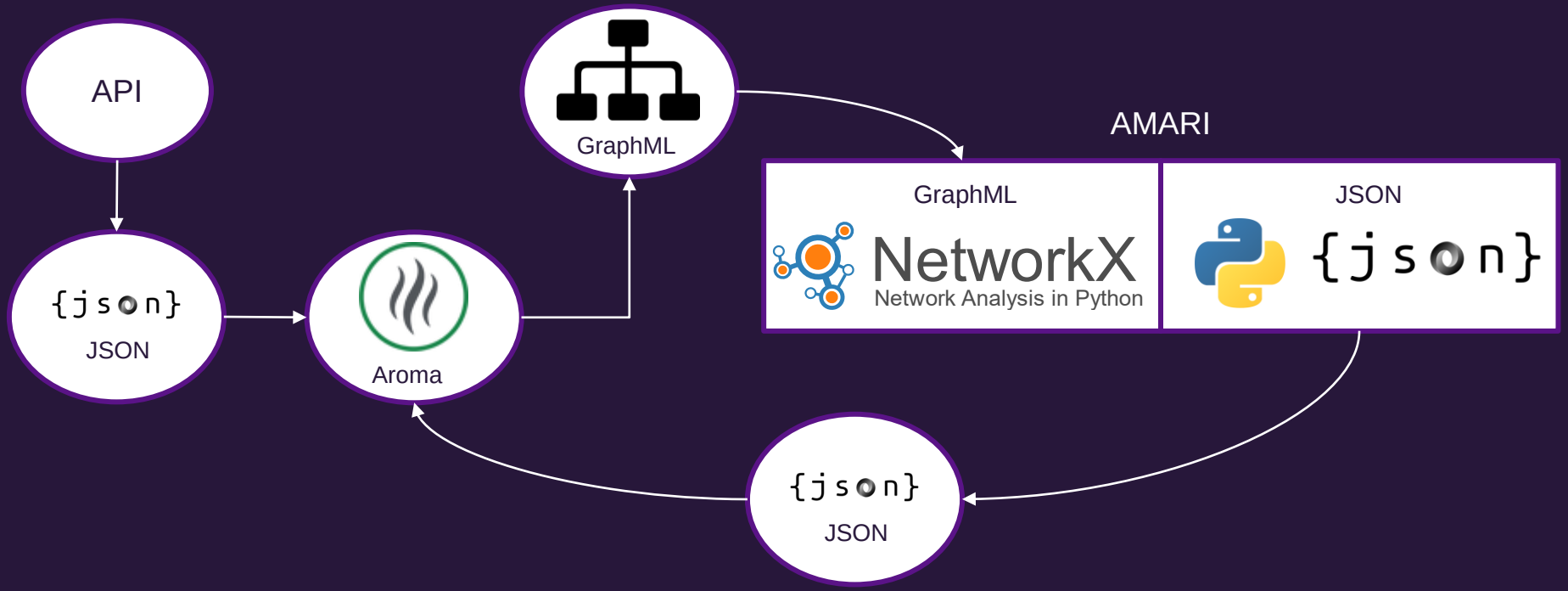
Francesco Ambrogio Marinoni
Matricola 869276
Anno Accademico 2022-2023
Relatore: Prof.ssa Francesca Arcelli Fontana
Correlatore: Dott. Paolo Bacchiega



Contenuto

- Introduzione ad AMARI
- Descrizione dei sistemi a microservizi
- Descrizione grafi e correlazione con i sistemi a microservizi
- GraphML e JSON
- Microservice smells(MS) trattati da AMARI
- Esempi di progetti analizzati
- Conclusioni e Sviluppi futuri

AMARI: an Automatic Microservices Architecture Refactoring Instrument



SISTEMI A MICROSERVIZI

- I sistemi a microservizi sono un'architettura software in cui le applicazioni vengono suddivise in piccoli servizi **indipendenti**, ognuno con una sua specifica funzionalità, facilitando la **scalabilità** e la **manutenzione**.

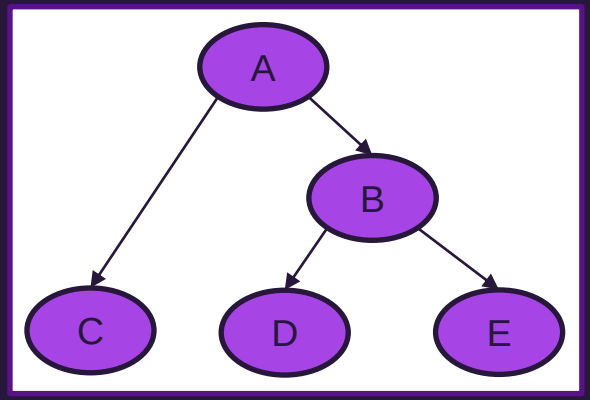
- Questi servizi comunicano tra loro tramite API e possono essere sviluppati, distribuiti e aggiornati in modo autonomo.

Caratteristiche sistemi a microservizi

DAG (Directed Acyclic Graphs)

- Un DAG(Directed Acyclic Graph) è un grafo diretto senza cicli.
- Scopo principale dei DAG:

  Rappresentare relazioni di dipendenza o di precedenza senza cicli.



GraphML e JSON : Definizione e Utilizzo

- **GraphML** offre una rappresentazione standardizzata per la descrizione di grafi, i file GraphML sono costituiti da elementi XML che definiscono i nodi, gli archi e gli attributi associati.
- **Scopo**: facilitano lo scambio di dati tra diverse applicazioni e piattaforme.
- Due caratteristiche principali sono:
  1. **Leggibilità umana**: Il formato GraphML è basato su XML, il che lo rende facilmente leggibile anche per gli esseri umani.
  2. **Estendibilità**: È possibile definire attributi personalizzati per nodi e archi, consentendo una rappresentazione dettagliata.

- **JSON**, acronimo di **JavaScript Object Notation**, è un formato di scambio dati leggero e utilizzato per rappresentare strutture dati in un formato testuale.
- **Scopo**: Rappresentare dati strutturati in un formato facilmente leggibile e trasmissibile tra sistemi diversi.
- Tre caratteristiche principali sono:
  1. **Sintassi Simile a JavaScript**: Sintassi chiara e leggibile ispirata da JavaScript.
  2. **Coppie Chiave/Valore**: Dati organizzati in coppie chiave/valore.
  3. **Tipi di Dati**: Può rappresentare stringhe, numeri, booleani, array, oggetti e null.

ESEMPI

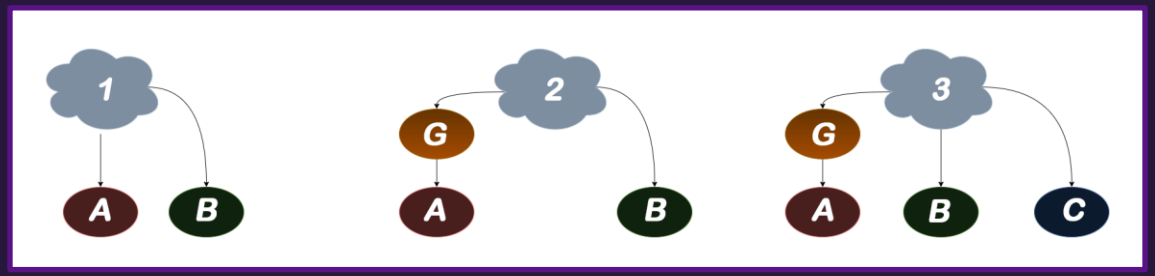
GraphML

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
  xmlns:java="http://www.yworks.com/xml/yfiles-common/1.0/java"
  xmlns:sys="http://www.yworks.com/xml/yfiles-common/markup/primitives/2.0"
  xmlns:x="http://www.yworks.com/xml/yfiles-common/markup/2.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:y="http://www.yworks.com/xml/graphml"
  xmlns:yed="http://www.yworks.com/xml/yed/3"
  xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
    http://www.yworks.com/xml/schema/graphml/1.1/ygraphml.xsd">
<!--Created by yEd 3.22-->
<key for="port" id="d0" yfiles.type="portgraphics"/>
<key for="port" id="d1" yfiles.type="portgeometry"/>
<key for="port" id="d2" yfiles.type="portuserdata"/>
<key attr.name="labelV" attr.type="string" for="node" id="d3"/>
<key attr.name="name" attr.type="string" for="node" id="d4"/>
<key attr.name="url" attr.type="string" for="node" id="d5"/>
<key attr.name="description" attr.type="string" for="node" id="d6"/>
<key for="node" id="d7" yfiles.type="nodegraphics"/>
<key for="graphml" id="d8" yfiles.type="resources"/>
<key attr.name="labelE" attr.type="string" for="edge" id="d9"/>
<key attr.name="weight" attr.type="int" for="edge" id="d10"/>
<key attr.name="isDB" attr.type="boolean" for="edge" id="d11"/>
<key attr.name="url" attr.type="string" for="edge" id="d12"/>
<key attr.name="description" attr.type="string" for="edge" id="d13"/>
<key for="edge" id="d14" yfiles.type="edgegraphics"/>
<graph edgedefault="directed" id="G">
```

JSON

```
{
  "traceId": "trace1",
  "id": "api-gateway_id",
  "timestamp": 1647943646909605,
  "duration": 377,
  "localEndpoint": {
    "serviceName": "api-gateway"
  }
},
```

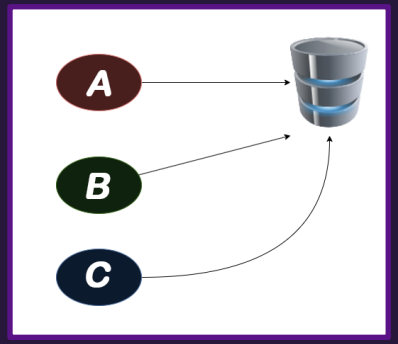
MS trattati da AMARI



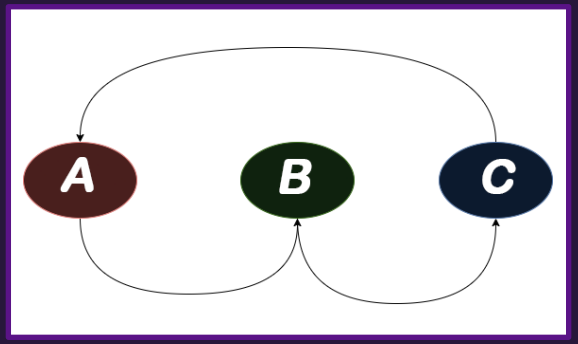
No API Gateway(NAG) :
I microservizi interagiscono direttamente tra loro senza un **API Gateway** come punto centrale.
Questo può accadere in due scenari:

- **L'API Gateway è assente** (non presente per progettazione).
- **L'API Gateway è presente**, ma alcune richieste client lo sorpassano.

1. Più di un nodo radice: A, B
2. É presente un nodo B isolato dalla radice G
3. É l'insieme della prima casistica e della seconda



Shared Persistence(SP):
Più microservizi **condividono il medesimo database o una risorsa generica.** Talvolta possono accedere alle stesse entità dello stesso database.

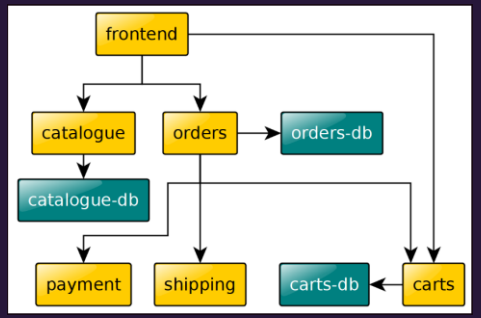


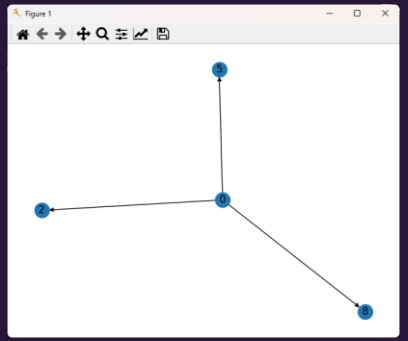
Dipendenza Ciclica(CD) :
I servizi **dipendono circolarmente l'uno dall'altro**, creando una serie di dipendenze cicliche.

Questa **dipendenza ciclica** può **danneggiare la scalabilità** dei servizi e inoltre **violare il principio di dipendenza aciclica.**

Ricostruzione architetturale

1. Grafo architetturale fornito dai proprietari di Sock Shop





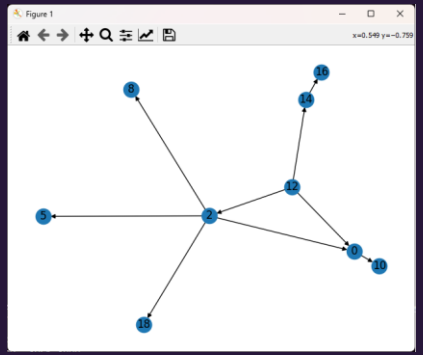
2. GraphML in input letto da AMARI.
Nodi visualizzati:
0 = carts
2 = orders
5 = payment
8 = shipping

Il Grafo 2 è evidentemente incompleto

Rispetto al grafo di partenza (Figura 2) sono stati modificati gli archi:

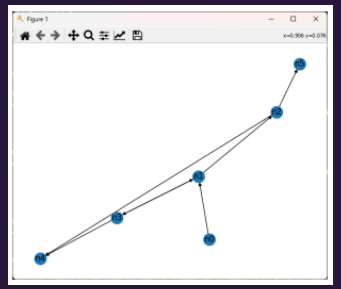
- (0,2) → (2,0)
- (0,8) → (2,8)
- (0,5) → (2,5)

Poi sono stati aggiunti tutti gli archi e nodi mancanti al punto 2



3. Tramite il sistema di aggiunta, rimozione e modifica di AMARI
Nuovi nodi aggiunti:
10 = carts-db
18 = orders-db
12 = frontend
14 = catalogue
16 = catalogue-db

Esempio di rimozione di uno smell



Struttura architetturale di un progetto sintetico con dipendenza ciclica

```
Insert the folder in which is the graphml: synthetic
Nodes:  ['n0', 'n1', 'n2', 'n3', 'n4', 'n5']
Edges:  [('n0', 'n1'), ('n1', 'n2'), ('n1', 'n3'), ('n2', 'n4'), ('n2', 'n5'), ('n3', 'n1'), ('n3', 'n4')]
What do you want to check in the graph? (nga(No Gataway API), sp(Shared Persistance), cd(Cyclic Dependency), e(exit))cd
There is a Cyclic Dependency
Problematic nodes:  [('n1', 'n3'), ('n3', 'n1')]
Do you want to modify the problematic nodes? (y/n)y
```

Ciclo individuato sui nodi n1-n3.

1. Eliminazione arco diretto (n3,n1)

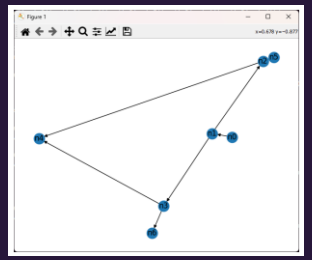
```
Do you want to add/delete/modify a node/edge? (y/n) (n to exit)y
Do you want to add a node or an edge or delete a node or an edge or modify a node or an edge? (an/ae/dn/de/mn/me)de
Enter the name of the first node: n3
Enter the name of the second node: n1
```

2. Aggiunta nodo n6

```
Do you want to add a node or an edge or delete a node or an edge or modify a node or an edge? (an/ae/dn/de/mn/me)an
Enter the ID of the node: n6
Enter the name of the node: tester
Enter the label of the node: n6
Nodes:  ['n0', 'n1', 'n2', 'n3', 'n4', 'n5', 'n6']
Edges:  [('n0', 'n1'), ('n1', 'n2'), ('n1', 'n3'), ('n2', 'n4'), ('n2', 'n5'), ('n3', 'n4')]
```

3.Aggiunta arco diretto (n3,n6)

```
Do you want to add a node or an edge or delete a node or an edge or modify a node or an edge? (an/ae/dn/de/mn/me)ae
Enter the name of the first node: n3
Enter the name of the second node: n6
Enter the weight of the edge: 300
Is the edge a DB edge? (y/n)n
Nodes:  ['n0', 'n1', 'n2', 'n3', 'n4', 'n5', 'n6']
Edges:  [('n0', 'n1'), ('n1', 'n2'), ('n1', 'n3'), ('n2', 'n4'), ('n2', 'n5'), ('n3', 'n4'), ('n3', 'n6')]
```



Struttura architetturale di un progetto sintetico senza la dipendenza ciclica

```
Insert the folder in which is the graphml: synthetic
Nodes:  ['n0', 'n1', 'n2', 'n3', 'n4', 'n5', 'n6']
Edges:  [('n0', 'n1'), ('n1', 'n2'), ('n1', 'n3'), ('n2', 'n4'), ('n2', 'n5'), ('n3', 'n4'), ('n3', 'n6')]
What do you want to check in the graph? (nga(No Gataway API), sp(Shared Persistance), cd(Cyclic Dependency), e(exit))cd
There is no Cyclic Dependency
```

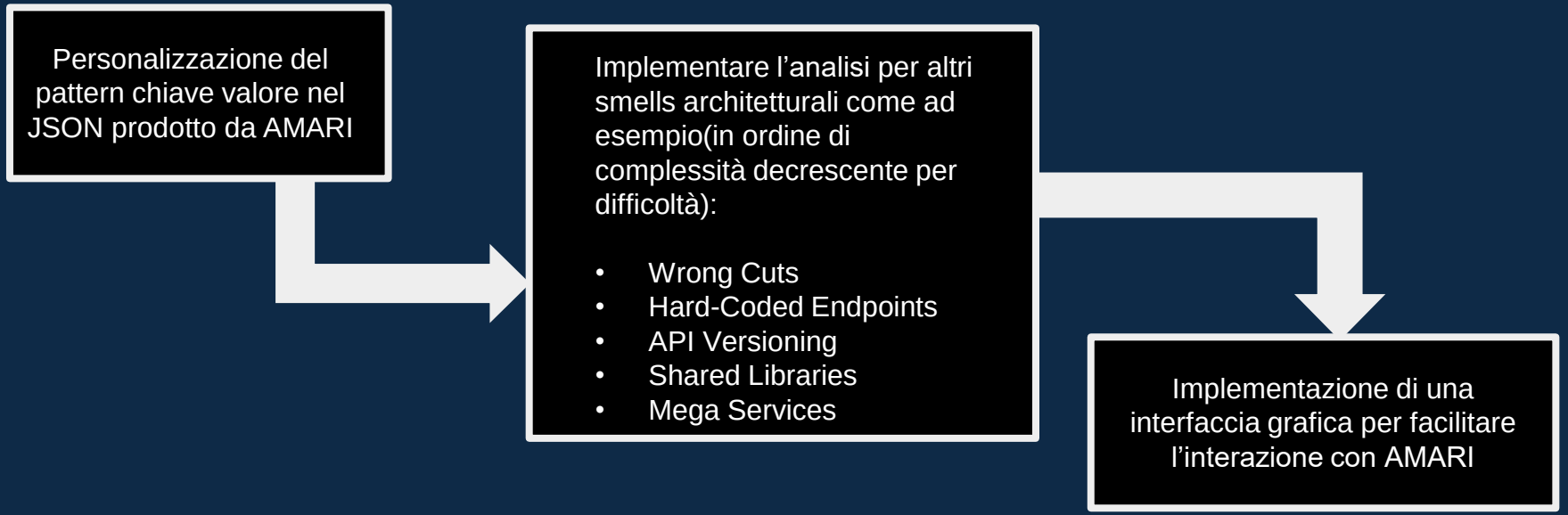
Conclusioni

- AMARI offre la possibilità di lavorare alla ricostruzione e rimozione di tre tipologie di smells (NAG, CD e SP) riconoscendone le caratteristiche nei grafi di sistemi a microservizi.

- In totale sono stati analizzati due progetti open-source(Sock Shop e Online Boutique) e un progetto sintetico.

- AMARI garantisce compatibilità in output, con qualsiasi strumento software che utilizzi file input in formato JSON(Zipkin V2), e in input con qualsiasi strumento software che utilizzi file output in formato GraphML.

Sviluppi futuri



Grazie per l’attenzione

Francesco Ambrogio Marinoni

Matricola 869276

Anno accademico 2022 – 2023

24 Ottobre 2023