Tesina di Automazione dei Processi di Business

Galli Antonio - Mat. M63/0721 Gravina Michela - Mat. M63/0708
Valletta Paolo - Mat. M63/0723

8 giugno 2018

Indice

1 Introduzione **1**

2 ServiziREST **2**
2.1 Web Service Pagine Bianche . . . 2
2.2 Web Service ASL . . . 3
2.3 Web Service Collocamento . . . 3
2.4 Scopo . . . 4

3 SpringBoot e Docker **6**
3.1 REST Services . . . 6
3.2 Springboot . . . 6
3.3 Docker . . . 7
3.4 Tutorial . . . 7
3.4.1 Requisiti . . . 8
3.4.2 Download del boilerplate . . . 8
3.4.3 Realizzazione del servizio . . . 8
3.4.4 La nostra prima Applicazione SpringBoot . . . 9
3.4.5 Esecuzione del servizio . . . 13
3.4.6 Personalizzazione del servizio . . . 13
3.4.7 Deploy su docker . . . 13
3.5 References . . . 13

4 Netflix Conductor **15**
4.1 Lavorare con Conductor . . . 18
4.1.1 Workflow . . . 18
4.1.2 Tasks . . . 18
4.1.2.1 System Tasks . . . 19
4.1.2.2 Worker tasks . . . 19
4.2 Conductor nel nostro progetto . . . 20
4.2.1 Tasks . . . 20
4.2.2 Workflows . . . 22
4.2.2.1 Workflow Get . . . 24
4.2.2.2 Workflow Post . . . 27
4.2.3 Esempi di esecuzione . . . 32
4.2.3.1 Richiesta informazioni Salute . . . 32

4.2.3.2 Richiesta ai servizi Asl, Pagine Bianche e Collocamento: 34
4.2.3.3 Richiesta di memorizzazione nuovo utente al servizio Asl: 36
4.2.3.4 Richiesta di memorizzazione nuovo utente ai servizi Asl, Pagine Bianche e Collocamento : 38

Capitolo 1

Introduzione

In tale elaborato è documentato il lavoro svolto per la creazione di un tutorial che comprenda:

- RESTful web services
- Docker Container
- Netflix Conductor

Capitolo 2

ServiziREST

Il progetto prevede la creazione di tre RESTful web services. In particolare sono stati realizzati i seguenti servizi web:

- Pagine Bianche
- Asl
- Collocamento.

I servizi sono tra di loro indipendenti e sono stati implementanti in Java, con l'ausilio di Spring Boot.

2.1 Web Service Pagine Bianche

Il web service "Pagine Bianche" espone due servizi:

- Dato il codice fiscale restituisce tutte le informazioni di un dato utente: tale servizio viene richiamato tramite il metodo GET di una richiesta HTTP con il seguente path /utente/CodiceFiscale.
- Aggiunge un utente all'elenco date le informazioni su quest'ultimo: tale servizio viene richiamato tramite il metodo POST di una richiesta HTTP con il seguente path /aggiungi , specificando il body in formato JSON.

Le informazioni associate ad un utente sono le seguenti:

- Codice Fiscale
- nome
- cognome
- indirizzo
- numero fisso
- cellulare
- email

2.2 Web Service ASL

l web service “ASL” espone due servizi:

- Dato il codice fiscale restituisce tutte le informazioni di un dato utente: tale servizio viene richiamato tramite il metodo GET di una richiesta HTTP con il seguente path /user/CodiceFiscale.
- Aggiunge un utente all’elenco date le informazioni su quest’ultimo: tale servizio viene richiamato tramite il metodo POST di una richiesta HTTP con il seguente path /user/add , specificando il body in formato JSON.

Le informazioni associate ad un utente sono le seguenti:

- Codice Fiscale
- nome
- cognome
- sesso
- data di nascita
- Medico di base

2.3 Web Service Collocamento

Il web service “Collocamento” espone tre servizi:

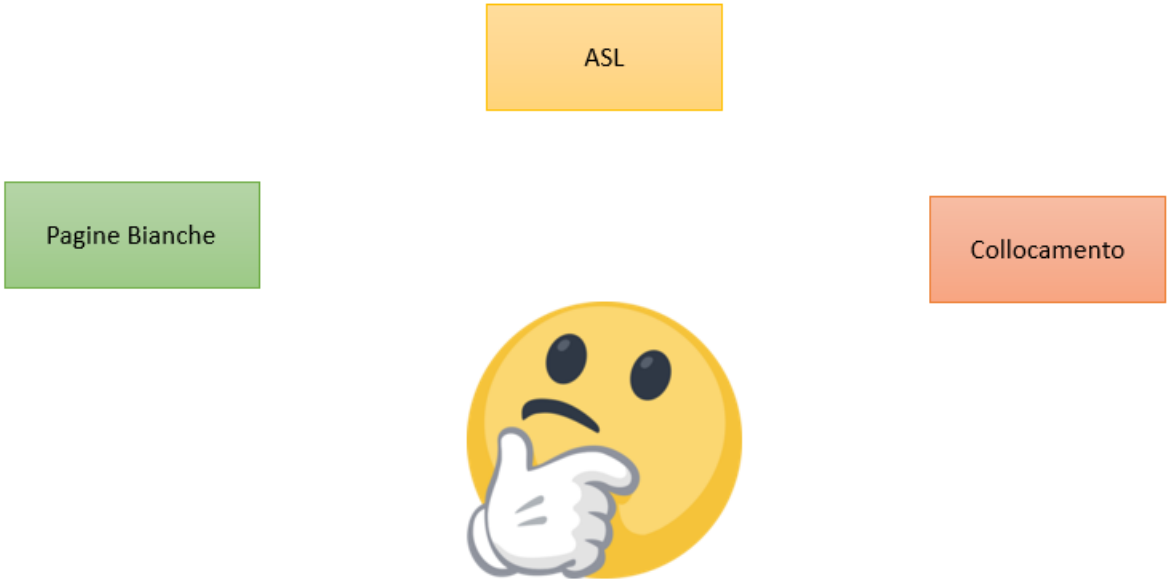
- Dato il codice fiscale restituisce tutte le informazioni di un dato utente: tale servizio viene richiamato tramite il metodo GET di una richiesta HTTP con il seguente path /users/CodiceFiscale.
- Aggiunge un utente all’elenco date le informazioni su quest’ultimo: tale servizio viene richiamato tramite il metodo POST di una richiesta HTTP con il seguente path /users/add , specificando il body in formato JSON.
- Restituisce la lista di tutti gli utenti: tale servizio viene richiamato tramite il metodo GET di una richiesta HTTP con il seguente path /users.

Le informazioni associate ad un utente sono le seguenti:

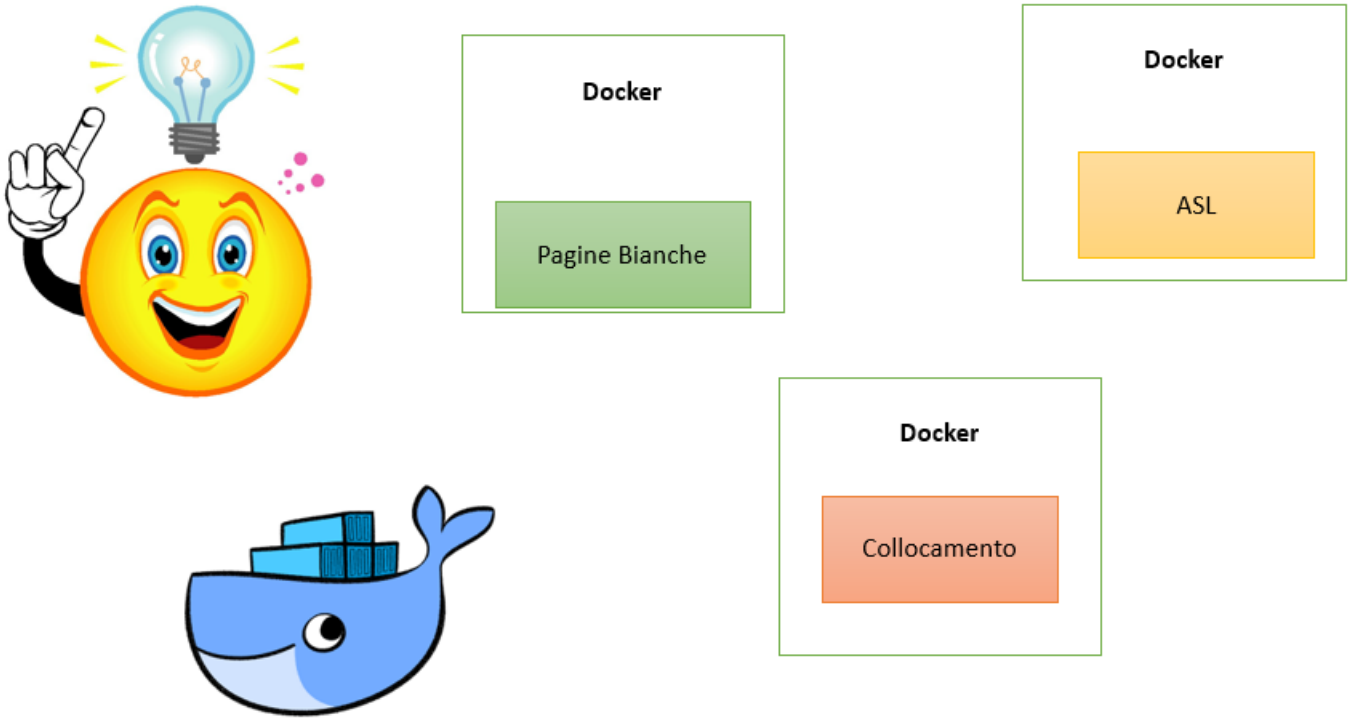
- Codice Fiscale
- nome
- cognome
- data di nascita
- lavoro attuale
- lavoro precedente

2.4 Scopo

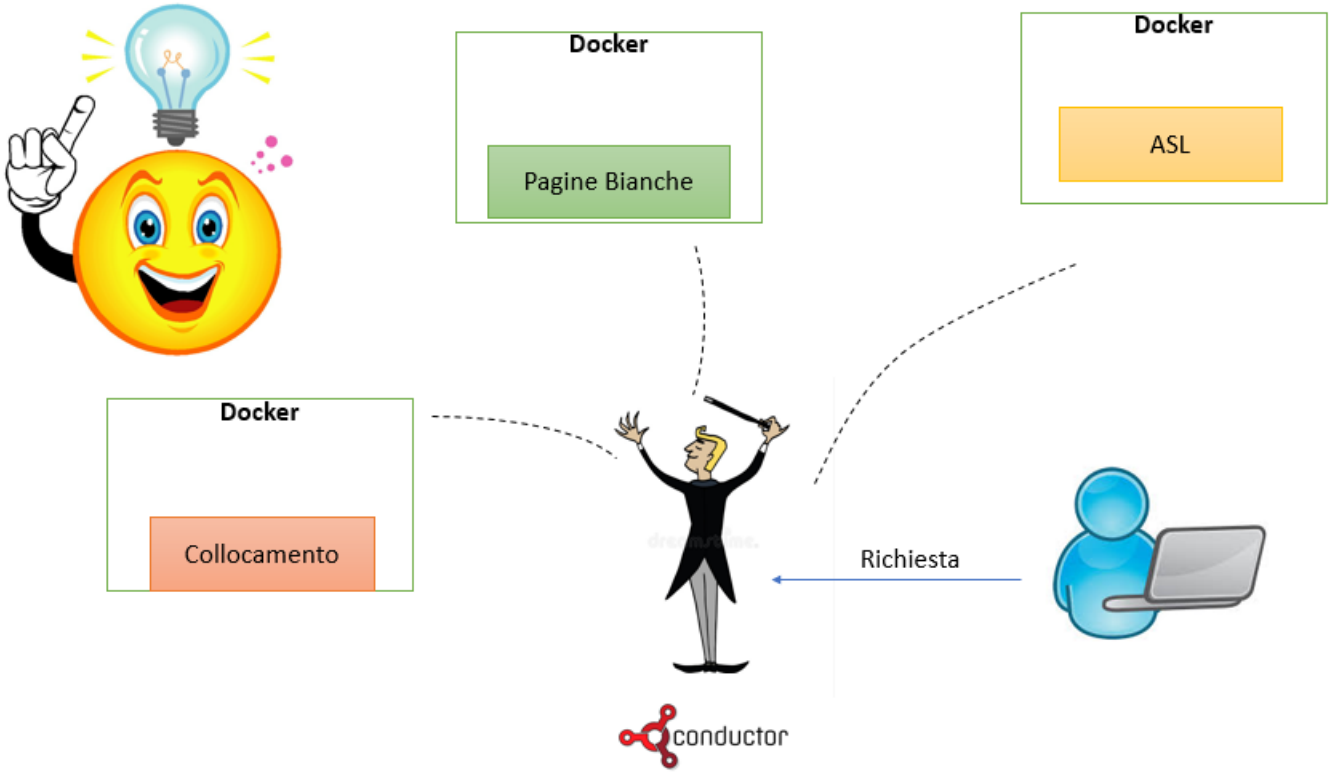
Dopo aver creato i tre Rest web services, lo scopo del progetto consiste nell'orchestrarli in maniera opportuna.



Innanzitutto i web services sono stati deployati con Docker.



Infine, sono stati ochestrati con conductor.



Nei capitoli successivi, sono spiegati del dettaglio l'implementazione, il deploy e l'orchestrazione dei web services realizzati, mediante la creazione di alcuni workflow di esempio.

Capitolo 3

SpringBoot e Docker

3.1 REST Services

Nati in contrapposizione, inizialmente, ai web services SOAP, i REST services si sono ritagliati sempre di più una fetta di mercato importante, tanto da essere oggi preferiti in molte situazioni. La leggerezza dell'architettura e i framework nati che abilitano al loro uso, sono stati i volano della loro diffusione. Negli scorsi anni erano spesso preferiti ai "cugini" SOAP, trend che oggi invece sta andando a scemare, data la consapevolezza che le due facce della stessa medaglia possono coesistere in un sistema, e non schiacciarsi a vicenda. Il fulcro del pensiero di tale architettura, nata da un lavoro di tesi del promotore, è il concetto di risorsa, che viene intesa come qualcosa che non deve essere solo al centro dei modelli dei database del sistema, ma anche ciò che un utente al di fuori deve vedere e poter accedere per sfruttare i servizi offerti. Su ogni risorsa, in particolare, deve essere possibile agire tramite le quattro operazioni classiche CRUD, create read update e delete. La sinergia con i metodi HTTP e tale protocollo, snello e preesistente, ha fatto si che le quattro operazioni venissero mappate sui quattro metodi di richiesta GET, POST, PUT e DELETE. Il poco overhead caratterizzato da HTTP, e la sua già ampia diffusione ha permesso ai developer di approcciarsi in maniera semplice, veloce ed indolore all'architettura REST.

3.2 Springboot

Un ruolo importante in questo panorame lo hanno giocato i framework. Uno di questi, per linguaggio Java, affermatosi di recente è SpringBoot. Tramite un sistema di annotazioni, permette in pochi passi di mettere su una serie di servizi REST robusti ed efficiente, così da realizzare API e route del proprio sistema in un tempo brevissimo. Ma springboot non è solo REST services, infatti esso permette di:

- avviare in pochi secondi un proprio progetto tramite Spring Initializr;
- costruire qualsiasi architettura: REST API, WebSocket, Web, Streaming, Tasks, e altro;
- sicurezza semplificata;

- ampio supporto per database SQL e NoSQL;
- supporto Runtime incluso (ad es. Tomcat, Jetty, e Undertow)
- strumenti a supporto del developer per aumentarne la produttività, come live reload e auto restart
- dipendenze ben realizzate che funzionano adeguatamente;
- feature pronte per la produzione, come tracing, metrics e health status;
- compatibilità con i propri IDE preferiti: Spring Tool Suite, IntelliJ IDEA e NetBeans.

Insomma, SpringBoot è un vero e proprio coltellino svizzero che supporta il developer a 360°, aiutandolo in ogni fase di sviluppo, deploy e manutenzione di un sistema, anche complesso. Tramite esso, uno sviluppatore può metter su un'architettura con davvero poco sforzo.

3.3 Docker

In tale scenario, come si posizione Docker? Partiamo da un quoting preso dal loro sito:

"Docker is the company driving the container movement and the only container platform provider to address every application across the hybrid cloud. "

Il concetto di container era già ampiamente diffuso su linux fin dalle sue origini. L'idea è quella di creare un ambiente di esecuzione indipendente dalla macchina host, isolato, e comprensivo di tutto ciò che è necessario per, ad esempio, eseguire una determinata applicazione. Per comprendere meglio tale concetto, possiamo fare un parallelismo con quello più consolidato di virtual machine: un container è come una vm in esecuzione sul nostro sistema, ma altamente più leggera e semplice da avviare, cancellare, clonare e deployare. Questo vuol dire che Docker è un diretto concorrente o, addirittura, più efficiente di una vm? Assolutamente no, perchè vi è ovviamente un tradeoff: salvo alcune eccezioni, tutti i container avviati su una stessa macchina condividono il kernel, e altre parti, del sistema operativo sottostante, il che si traduce in una flessibilità di gran lunga inferiore ad una VM. Dunque, possiamo affermare che Docker non è una panacea contro tutti i mali dell'uso di una VM, ma una valida alternativa in determinati scenari, che oggi stanno diventando sempre più diffusi e che stanno portando tale piattaforma sempre più in auge. Un container Docker è possibile vederlo come un'istanza di una cosiddetta immagine, che viene riempita con tutto ciò il necessario per mandarla in esecuzione e offrire determinate funzionalità.

3.4 Tutorial

In questa sezione andremo ad avviare la nostra prima applicazione springboot, e la andremo a deployare su un container Docker. Grazie a dei boilerplate già pronti e messi a disposizione dalla community, il tutto sarà possibile farlo in pochissimi minuti, permettendoci così di concentrarci sulla logica dell'applicazione, piuttosto che su operazioni di contorno.

3.4.1 Requisiti

Per arrivare in fondo a questa guida, è necessario:

- Qualche minuto;
- Un IDE a scelta (suggerimento: Eclipse);
- JDK 1.8 or later;
- Gradle 4+ or Maven 3.2+ (la presente guida seguirà l'uso di Gradle, ma i passaggi sono simili);
- Sistema Operativo Linux;
- docker (installabile seguendo la guida online del software).

3.4.2 Download del boilerplate

Per realizzare questa guida adopereremo un boilerplate già pronto, nel quale dovremo solamente mettere il nostro codice. Per scaricarlo, abbiamo due alternative. La prima è clonare la repo su github messa a disposizione da springboot tramite il comando:

```
1 git clone https://github.com/spring-guides/gs-spring-boot-docker.git
```

Se tale operazione dovesse fallire (nella nostra esperienza, a volte git diceva che non avevamo i permessi per accedere alla repo), si può procedere con il download dei file dalla repo github tramite browser. Il file sarà uno zip, che potrà poi essere scompattate sul disco (tramite comando unzip). Ultima nota, per evitare problemi di permessi futuri, si consiglia di dare il seguente comando sulla cartella unzippata:

```
1 sudo chmod 777 gs-spring-boot-docker -R
```

per essere sicuri di poter operare liberamente.

3.4.3 Realizzazione del servizio

Una volta scaricato tutto il materiale, ciò che ci interessa è solamente la cartella complete. Il resto è possibile eliminarlo tranquillamente. Questa cartella ha pre-impostati sia gli script gradle e maven, per la gestione di dipendenze e compilazione, che il Dockerfile per poter deployare il servizio. Inoltre, vi è presente in springboot una versione leggera di Tomcat, per cui non servirà altro per avviare il tutto. La cartella complete è possibile importarla nel nostro IDE preferito come "gradle project". Una volta fatto ciò, possiamo eliminare dalla cartella src tutta la cartella test, che si riferisce al codice di default presente. Inoltre, possiamo procedere ad eliminare la cartella "hello" in "src/java", che contiene il codice del progetto di esempio. Ora possiamo procedere a creare i nostri package e il nostro codice sotto la directory src/java. Nella successiva sezione vediamo le parti salienti di una applicazione SpringBoot e alcune annotazioni. Si rimanda al sito SpringBoot nelle referenze per una guida completa in merito.

3.4.4 La nostra prima Applicazione SpringBoot

Un'applicazione SpringBoot è caratterizzata da una classe entry point, realizzata tramite annotazioni. Vediamone qui un esempio:

```
1  package CollocamentoREST;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6  @SpringBootApplication
7  public class SpringBootRestApplication {
8
9    public static void main(String[] args) {
10     SpringApplication.run(SpringBootRestApplication.class, args);
11   }
12 }
```

Qui possiamo notare un main, necessario come entry point all'applicazione, formato da una sola riga, che carica il servizio. L'annotazione @SpringBootApplication caratterizza tale classe come entry point.

Volendo seguire un pattern MVC per la realizzazione del servizio, possiamo posizionare questa classe nel pckage Controller (nel nostro esempio chiamato CollocamentoREST), in cui inseriamo anche la classe che presenta le route delle nostre API:

```
1  package CollocamentoREST;
2
3  import java.util.List;
4
5  import org.springframework.web.bind.annotation.*;
6
7  import MVC.User;
8  import MVC.UserDAO;
9
10 @RestController
11 public class UserController {
12
13   private UserDAO udao = new UserDAO();
14
15   @GetMapping("/users/all")
16   public List<User> retrieveAllStudents() {
17     return udao.getUsers();
18   }
19
20   @GetMapping("/users/{cf}")
21   public User getUserById(@PathVariable("cf") String cf) {
22     return udao.getUterByCf(cf);
23   }
```

```
24
25    @PostMapping("/users/add")
26    public boolean addUser(@RequestBody User user) {
27      return udao.addUser(user);
28    }
29  }
```

Da questo esempio notiamo:

- @RestController: annotazione per definire il Controller del nostro servizio;
- @GetMapping("..."): annotazione per definire una route del nostro servizio, che risponderà con il codice del metodo annotato in caso di richiesta GET (analogamente @PostMapping per una POST);
- senza indicare nulla di più, l'output prodotto sarà json, altrimenti modificabile con una apposita annotazione;
- @RequestBody, per prendere il body della richiesta e mapparlo direttamente ad un oggetto del nostro model, senza aggiungere alcun che. Il body dovrà essere un json con i campi di pari nome di quello della classe User del Model. Provvederà SpringBoot a fare il mapping;
- @PathVariable: annotazione per prendere una variabile nell'url della richiesta, riconoscibile tramite le {} che la circondano.

Il resto, inserito qui nel package MVC sono classi normali, classiche del Java, senza annotazioni ne altro. Come vediamo, in maniera facile e veloce (e soprattutto indolore) il servizio è stato realizzato. Riportiamo qui le altre due classi presenti per completezza:

```
1   package MVC;
2
3   import java.util.ArrayList;
4   import java.util.HashMap;
5   import java.util.List;
6
7   public class UserDAO {
8     private HashMap<String, User> users = new HashMap<>();
9     UserDAO instance = null;
10
11    public UserDAO getInstance() {
12      if (instance==null)
13        instance = new UserDAO();
14      return instance;
15    }
16
17    public UserDAO() {
18      users.put("VLLTPL947", new User("Paolo", "Valletta", "VLLTPL947", "25/02/1994",
          "Impiegato", "Disoccupato"));
```

```
    users.put("GLLNTN94", new User("Antonio", "Galli", "GLLNTN94", "11/05/1994", "
        Manager", "Disoccupato"));
    users.put("GRVMHL94", new User("Michela", "Gravina", "GRVMHL94", "30/11/1994", "
        Prof.re Ordinario", "Disoccupato"));
    users.put("VFGVN94", new User("Virginia", "Venezia", "VFGVN94", "02/08/1994", "
        Soldato Semplice", "Disoccupato"));

  }

  public List<User> getUsers() {
    return new ArrayList<User>(users.values());
  }

  public User getUterByCf(String cf) {
    return (User)users.get(cf);
  }

  public boolean addUser(User u) {
    if (users.containsKey(u.getCfis()))
      return false;
    else
      users.put(u.getCfis(), u);
    return true;
  }
}
```

```
package MVC;

public class User {
  private String nome = "";
  private String cognome = "";
  private String cfis = "";
  private String data_nascita = "";
  private String lavoro_attuale = "";
  private String lavoro_precedente = "";

  public User(){
    cfis="";
  }

  public User(String nome, String cognome, String cfis, String data_nascita, String
      lavoro_attuale,
       String lavoro_precedente) {
    super();
    this.nome = nome;
    this.cognome = cognome;
```

```
20      this.cfis = cfis;
21      this.data_nascita = data_nascita;
22      this.lavoro_attuale = lavoro_attuale;
23      this.lavoro_precedente = lavoro_precedente;
24    }
25
26    public String getNome() {
27      return nome;
28    }
29    public void setNome(String nome) {
30      this.nome = nome;
31    }
32    public String getCognome() {
33      return cognome;
34    }
35    public void setCognome(String cognome) {
36      this.cognome = cognome;
37    }
38    public String getCfis() {
39      return cfis;
40    }
41    public void setCfis(String cfis) {
42      this.cfis = cfis;
43    }
44    public String getData_nascita() {
45      return data_nascita;
46    }
47    public void setData_nascita(String data_nascita) {
48      this.data_nascita = data_nascita;
49    }
50    public String getLavoro_attuale() {
51      return lavoro_attuale;
52    }
53    public void setLavoro_attuale(String lavoro_attuale) {
54      this.lavoro_attuale = lavoro_attuale;
55    }
56    public String getLavoro_precedente() {
57      return lavoro_precedente;
58    }
59    public void setLavoro_precedente(String lavoro_precedente) {
60      this.lavoro_precedente = lavoro_precedente;
61    }
62
63
64
65  }
```

NOTA: la classe UserDAO è in realtà un Singleton che ha dati dummy inseriti di default, che conserva per tutta l'esecuzione del servizio. La classe User presenta un costruttore senza parametri necessario per fare funzionare il mapping del RequestBody fatto da SpringBoot.

3.4.5 Esecuzione del servizio

Scritto il codice, siamo pronti per provare ed avviare il nostro servizio. Basterà semplicemente fare tasto destro nell'IDE sulla classe SpringBootRestApplication e ciccare Run As/-Java Application. Nella console potremo vedere l'avvio di springboot e del tomcat integrato. Una volta completato, potremo testare le nostre route, ad esempio localhost:8080/users/all (il servizio realizzato nel nostro codice è, in realtà, sulla porta 9090).

3.4.6 Personalizzazione del servizio

Lasciando gli script di gradle di default, il servizio e il docker creato avranno dei nomi di default, e soprattutto saranno lanciati sulla porta 8080. Se si vuole avere più servizi, sarà innanzitutto necessario modificare tale porta. Per farlo, basterà aprire il file src/main/resources/application.yml e modificare la riga "port:" con ciò che si preferisce. Inoltre, modificando il file build.gradle della root complete sarà possibile modificare il nome che docker darà al servizio. La riga da modificare è la numero 36 "name": possiamo scrivere una qualsiasi stringa, seppur senza lettere maiuscole o spazi.

3.4.7 Deploy su docker

Per realizzare l'immagine docker, lanciare il container e avviare il servizio, possiamo posizionarci da shell nella root del progetto e dare i seguenti comandi:

```
1 sudo ./gradlew build docker
2 docker run -p 8080:8080 -t nome_container_scelto
```

La prima riga prepara il container e lo aggiunge alla libreria docker, mentre il secondo lo lancia, indicando la porta modificata eventualmente precedentemente, e il nome dato prima nella modifica del file build.gradle. Fatto ciò è tutto pronto, docker è avviato e deployato e all'indirizzo localhost giusto potremo trovare il nostro servizio in esecuzione!

3.5 References

RESTServices: https://en.wikipedia.org/wiki/Representational_state_transfer)

SpringBoot: https://spring.io/

Docker: https://www.docker.com/

Docker-container: https://www.docker.com/what-container

Guida docker e springboot: https://spring.io/guides/gs/spring-boot-docker/

Capitolo 4

Netflix Conductor

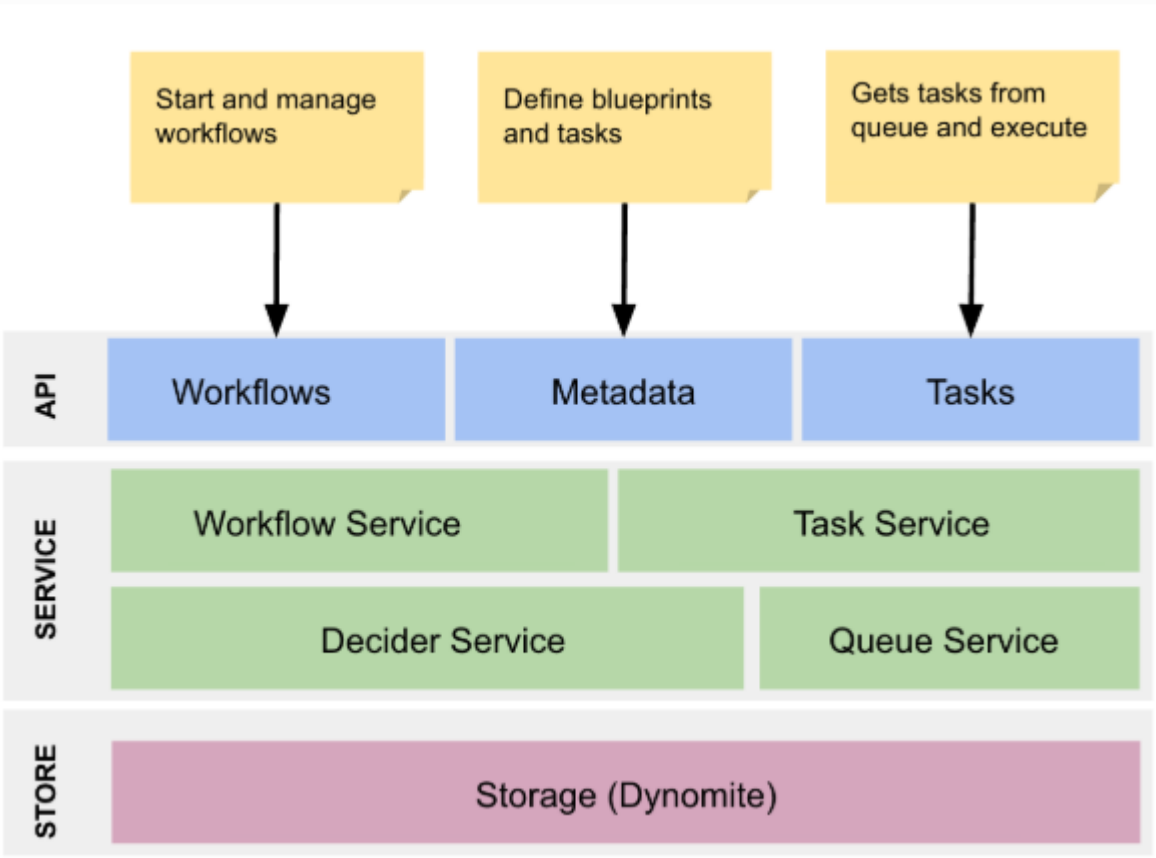
Conductor è un orchestratore di servizi, creato principalmente per orchestrare i servizi offerti da Netflix. Presenta le seguenti caratteristiche:

- permette la creazione di processi complessi in cui ogni task viene implementato come un microservizio
- il flusso di esecuzione è definito in JSON
- garantisce tracciabilità e visibilità all'interno dei flussi di esecuzione
- permette il riuso di microservizi esistenti
- mette a disposizione un'interfaccia grafica per visualizzare il flusso di esecuzione
- permette la sincronizzazione dei task
- è scalabile in quanto è capace di gestire molti flussi di esecuzione contemporaneamente

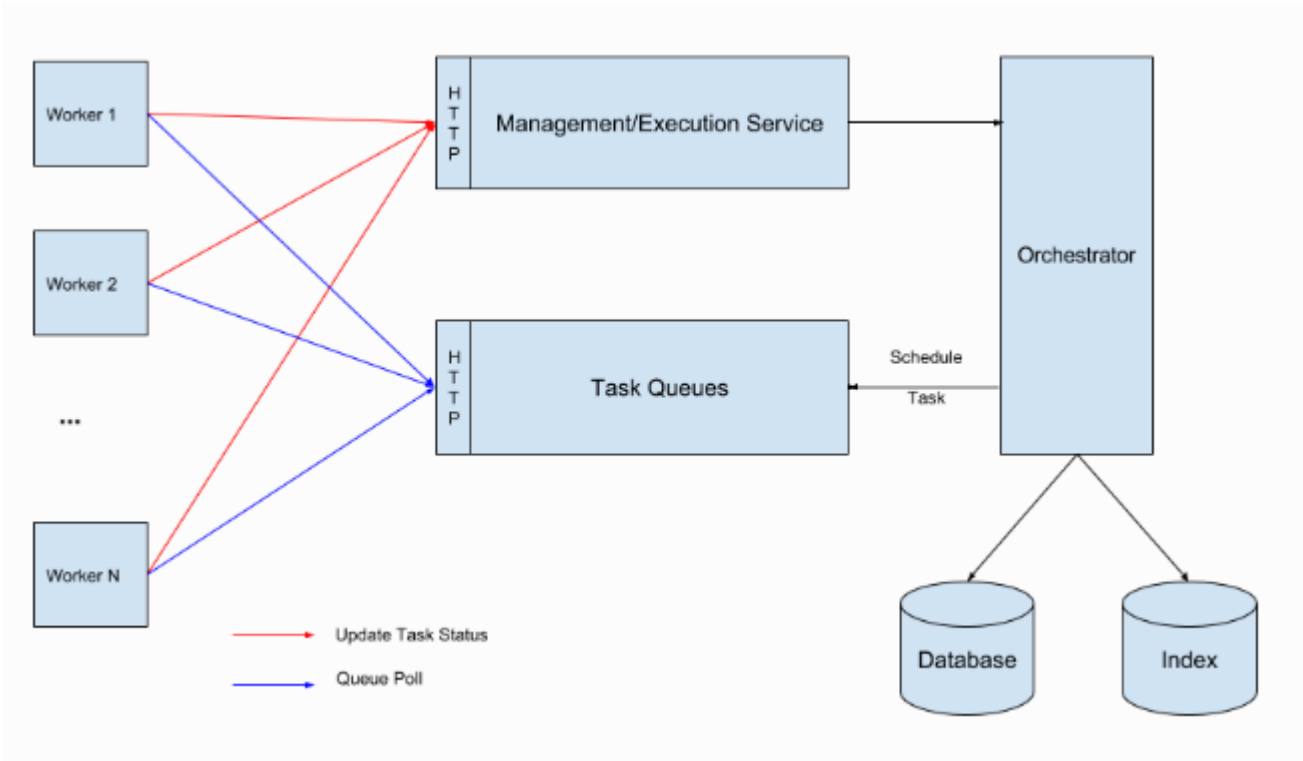
- la cominucazione è realizzata mediante richieste HTTP oppure RPC.

Nella realizzazione di Conductor è stata preferita l'orchestrazione alla coreografia in quanto la prima permetteva una scalabilità migliore.

Di seguito è mostrata l'architettura ad alto livello di Conductor.



Conductor segue il modello di comunicazione basato su RPC in cui i workers sono in esecuzione su una macchina separata dal server. I workers comunicano con il server mediante richieste HTTP.



Quindi:

- I workers sono remote systems e comunicano con conductor server mediante richieste HTTP (oppure tramite qualsiasi meccanismo che supporti RPC)
- Le Task Queues sono usate per schedulare i tasks.
- Per la persistenza dei dati viene utilizzato DynomiteDB.

Per il nostro progetto Conductor è stato installato su una macchina Linux. In particolare è stato installato un "In-Memory Server" e quindi i dati non vengono veramente memorizzati sul disco, ma sono persi una volta spento il server.

In particolare i passi seguiti sono:

- **https://github.com/Netflix/conductor** per clonare la repo scaricando in formato zip
- **sudo unzip nomefile.zip**
- **sudo chmod 777 conductor-master -R** : in modo da avere tutti i permessi sulla cartella.
- Dal sito **https://search.maven.org/#search%7Cga%7C1%7Cnetflix%20conductor** scaricare il file con estensione all.jar relativo a:
  - GroupID: com.netflix.conductor
  - artifact: conductor-server-all
  - versione: 1.6+

- Creare nella directory conductor-master/server la cartella build
- Creare nella directory build la cartella libs
- copiare nella directory conductor-master/server/build/libs il file scaricato precedentemente
- installare docker-compose (sudo apt-get install docker-compose);
- a questo punto effettuare i comandi:
  - **cd docker**
  - **sudo docker-compose build (attendere la terminazione)**
  - **sudo docker-compose up**

Per interagire con l'interfaccia grafica è necessario aprire un browser e digitare localhost:5000, mentre per interagire con Swagger, è necessario digitare localhost:8080.

Tramite l'interfaccia grafica è possibile osservare i task e i workflow presenti in Conductor e il loro stato di esecuzione (Running, Complete, Failed, Terminate), mentre con Swagger è possibile gestire i task e i workflow per crearne di nuovi o modificarli.

4.1 Lavorare con Conductor

Conductor si basa principalmente su due concetti principali:

- Workflow
- Task.

4.1.1 Workflow

I workflows sono definiti utilizzando un linguaggio di dominio basato su JSON (JSON based DSL) e includono i task che sono eseguiti come parte del workflow.

4.1.2 Tasks

I task devono essere necessariamente registrati prima di essere eseguiti in un workflow. Si hanno due categorie di task:

- System Task
- Worker Task

4.1.2.1 System Tasks

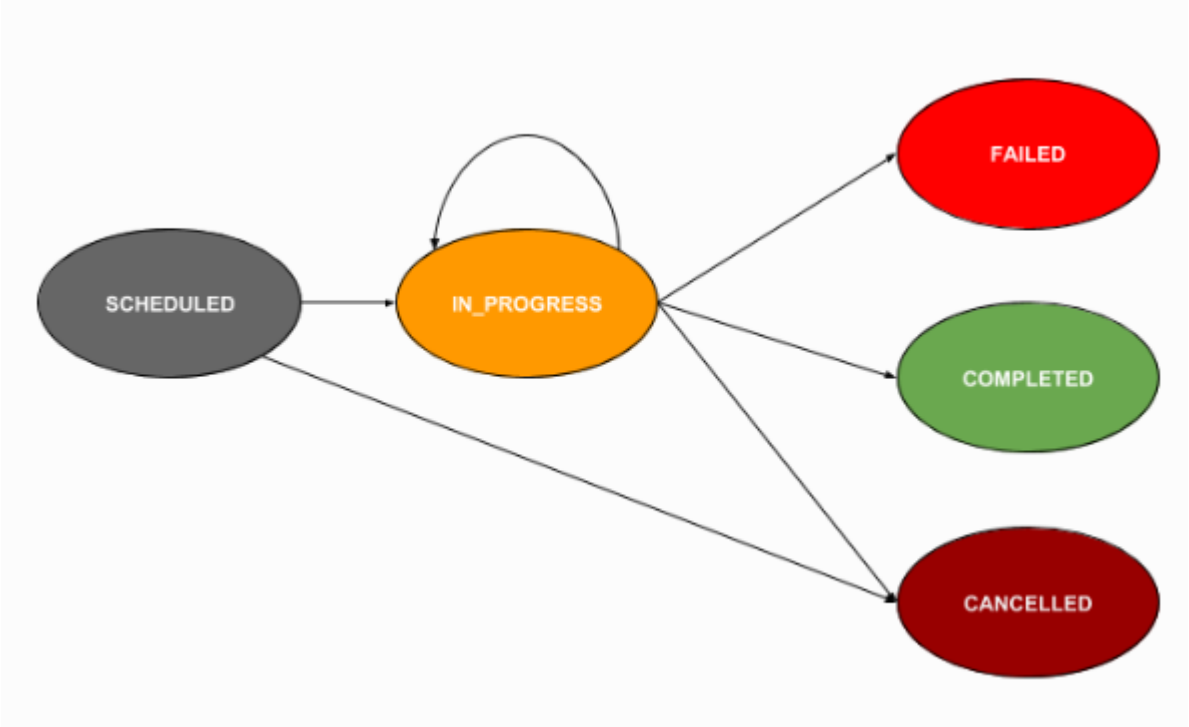
I System Tasks sono eseguiti all'interno della JVM di Conductor e sono gestiti da quest'ultimo. Abbiamo vari tipi di System Tasks:

Task	Scopo
Dynamic Task	definito in maniera dinamica in base all'input
Decision	Implementa uno switch case
Fork	Definisce l'esecuzione parallela di un insieme di task.
Dynamic Fork	I task da eseguire in parallelo sono decisi in base all'input.
Join	Unione di rami paralleli
Sub Workflow	Necessario per la realizzazione di workflow innestati
Wait	task che rimane nello stato di IN_PROGRESS fino ad un trigger esterno
HTTP	usato per chiamare microservizi tramite richieste HHTP
Event	Produce un evento

4.1.2.2 Worker tasks

I worker tasks sono implementati dalle applicazioni ed eseguono in un ambiente separato dal Conductor. Possono essere implementanti in qualsiasi linguaggio di programmazione, comunicanco con il Conductor server tramite delle REST ASPI. Nel modello sono definiti "SIMPLE".

Di seguito è riportato uno schema dei possibili stati di un task o di un workflow.



4.2 Conductor nel nostro progetto

4.2.1 Tasks

Come già detto in precedenza per poter realizzare un workflow è necessario definire a priori i task che fanno parte di quest'ultimo. In particolare nel nostro progetto sono stati definiti i seguenti task per la realizzazione dei workflow:

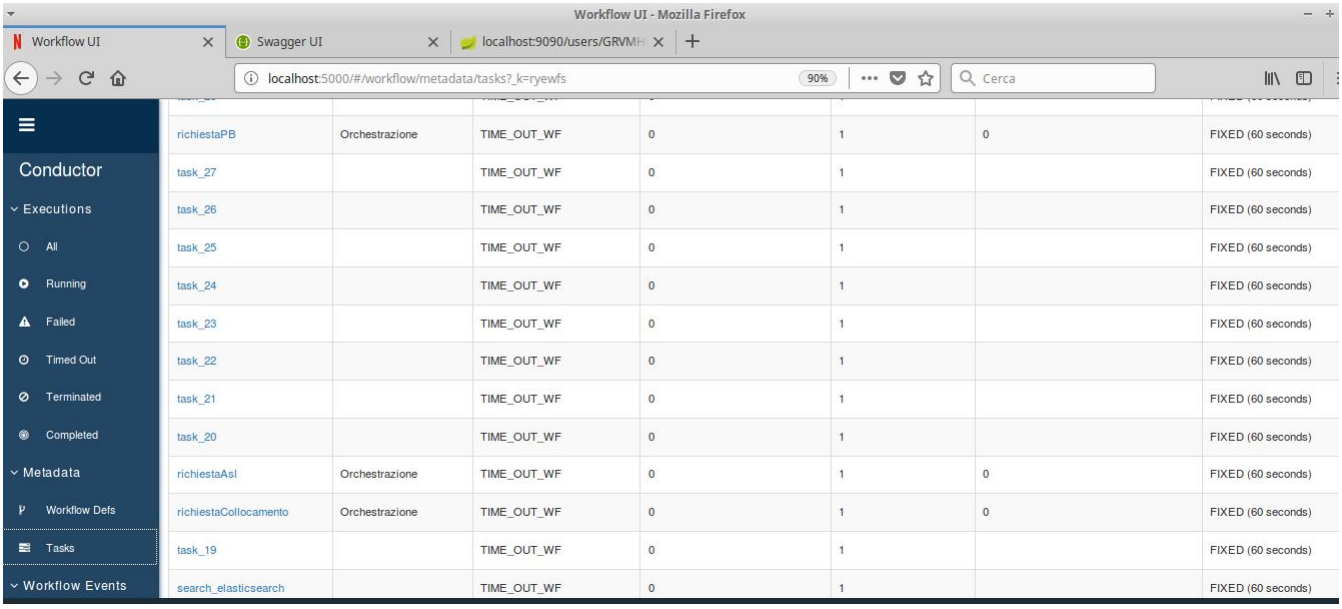


Figura 4.1: Tasks definiti

- richiestaPB : tale task è stato realizzato per poter effettuare una richiesta al servizio Pagine bianche.
- richiestaAsl : tale task è stato realizzato per poter effettuare una richiesta al servizio Asl.
- richiestaCollocamento : tale task è stato realizzato per poter effettuare una richiesta al servizio Collocamento.

Di seguito sono riportati i codici JSON relativi alla realizzazione di ogni task:

Per creare i task si è utilizzata l'interfaccia grafica Swagger, in paricolare i passi eseguiti sono i seguenti :

- Collegarsi all'interfaccia Swagger localhost:8080
- Tra le varie liste di operazioni disponibili selezionare il menù Metadata Management
- Selezionare "Create new task definition(s)"

Un'alternativa all'interfaccia grafica consiste nell'eseguire la richiesta HTTP suggerita dall'operazione (in questo caso una PUT).

```
[
  {
      "ownerApp": "Orchestrazione",
      "createTime": 0,
      "updateTime": 0,
      "createdBy": "MAP",
      "updatedBy": "MAP",
      "name": "richiestaPB",
      "description": "richiesta al servizio PB",
      "retryCount": 1,
      "timeoutSeconds": 0,
      "inputKeys": [
          "cf"
      ],
      "outputKeys": [
          "utente"
      ],
      "timeoutPolicy": "TIME_OUT_WF",
      "retryLogic": "FIXED",
      "retryDelaySeconds": 60,
      "responseTimeoutSeconds": 60,
      "concurrentExecLimit": 0,
      "inputTemplate": {}
  }
]
```

```
[
  {
      "ownerApp": "Orchestrazione",
      "createTime": 0,
      "updateTime": 0,
      "createdBy": "MAP",
      "updatedBy": "MAP",
      "name": "richiestaAsl",
      "description": "richiesta al servizio Asl",
      "retryCount": 1,
      "timeoutSeconds": 0,
      "inputKeys": [
          "cf"
      ],
      "outputKeys": [
          "utente"
      ],
```

```
      "timeoutPolicy": "TIME_OUT_WF",
      "retryLogic": "FIXED",
      "retryDelaySeconds": 60,
      "responseTimeoutSeconds": 60,
      "concurrentExecLimit": 0,
      "inputTemplate": {}
  }
]
```

```
[
  {
      "ownerApp": "Orchestrazione",
      "createTime": 0,
      "updateTime": 0,
      "createdBy": "MAP",
      "updatedBy": "MAP",
      "name": "richiestaCollocamento",
      "description": "richiesta al servizio Collocamento",
      "retryCount": 1,
      "timeoutSeconds": 0,
      "inputKeys": [
          "cf"
      ],
      "outputKeys": [
          "utente"
      ],
      "timeoutPolicy": "TIME_OUT_WF",
      "retryLogic": "FIXED",
      "retryDelaySeconds": 60,
      "responseTimeoutSeconds": 60,
      "concurrentExecLimit": 0,
      "inputTemplate": {}
  }
]
```

4.2.2 Workflows

Dopo aver definito i vari task abbiamo definito due diversi workflow :

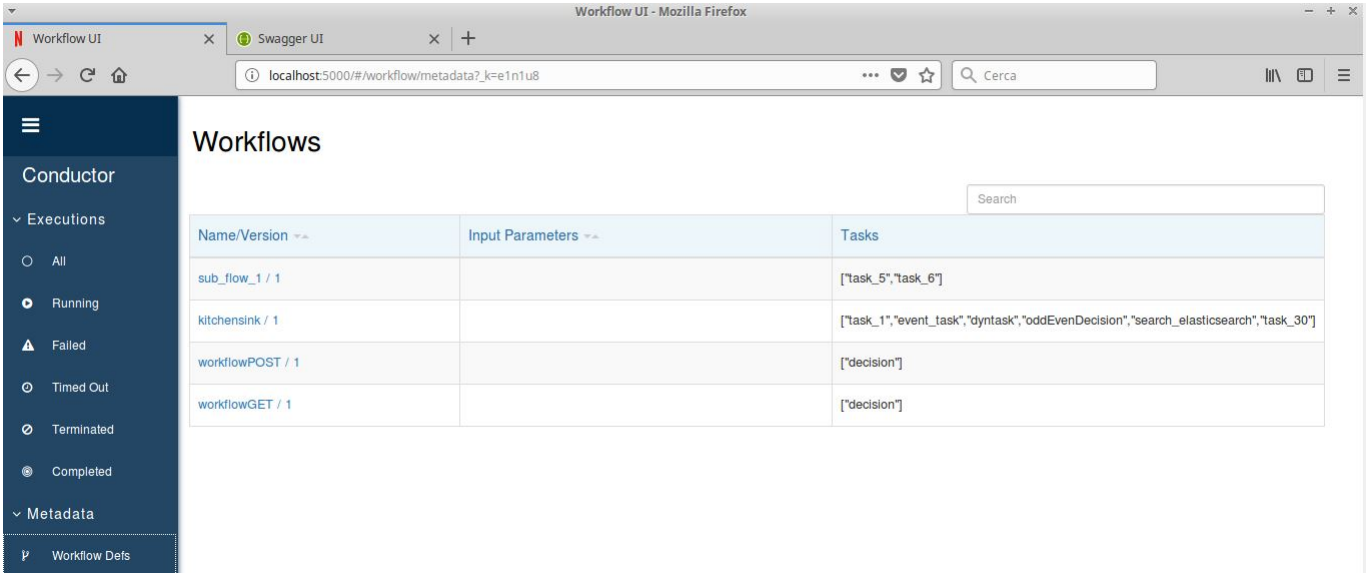


Figura 4.2: Workflow definiti

- workflowGet : per la realizzazione di tale workflow si è fatto uso non solo di task da noi definiti, ma anche di alcuni system task ovvero “Decision” , “Fork” e “Join”. In particolare questo workflow definisce il flusso per poter realizzare le seguenti funzionalità :
  - Effettuare una chiamata ad un singolo servizio tra Salute-Info-Lavoro per ricevere informazioni relative ad un utente.
  - Effettuare delle chiamate in maniera parallela ai tre servizi disponibili per ricevere tutte le informazioni disponibili relative ad un utente.
- workflowPost : tale workflow definisce il flusso per poter realizzare le seguenti funzionalità :
  - Effettuare una chiamata ad un singolo servizio tra Salute-Info-Lavoro per poter registrare le informazioni relative ad un utente.
  - Effettuare delle chiamate in maniera parallela ai tre servizi disponibili, distrubuendo le varie informazioni relative ad un utente al fine registrarle.

Di seguito è riportato il codice e lo schema relativo ad ogni workflow:

Per creare i workflows si è utilizzata l’interfaccia grafica Swagger, in paricolare i passi eseguiti sono i seguenti :

- Collegarsi all’interfaccia Swagger localhost:8080
- Tra le varie liste di operazioni disponibili selezionare il menù Metadata Management
- Selezionare “Create new workflow definition”

Un’alternativa all’interfaccia grafica consiste nell’eseguire la richiesta HTTP suggerita dall’operazione (in questo caso una PUT).

4.2.2.1 Workflow Get

```
{
  "ownerApp": "Orchestrazione",
  "createTime": 0,
  "updateTime": 0,
  "createdBy": "MAP",
  "name": "workflowGET",
  "description": "orchestrazione dei servizi",
  "version": 1,
  "tasks": [
  {
  "name": "decision",
  "taskReferenceName": "decision",
  "inputParameters": {
    "scelta": "${workflow.input.scelta}"
  },
  "type": "DECISION",
  "caseValueParam": "scelta",
  "decisionCases": {
    "info": [{
          "name": "richiestaPB",
              "taskReferenceName": "richiestaPB",
            "inputParameters": {
            "http_request":{
                "uri":  "http://172.17.0.1:8081/utente/${workflow.input.cf}",
                  "method": "GET"
                }
                  },
            "type": "HTTP"
    }],
    "salute": [{
          "name": "richiestaAsl",
            "taskReferenceName": "richiestaAsl",
            "inputParameters": {
            "http_request":{
                "uri":  "http://172.17.0.1:8082/user/${workflow.input.cf}",
                  "method": "GET"
                }
                  },
            "type": "HTTP"
    }],
    "lavoro": [{
          "name": "richiestaCollocamento",
```

```
            "taskReferenceName": "richiestaCollocamento",
          "inputParameters": {
          "http_request":{
              "uri":  "http://172.17.0.1:9090/users/${workflow.input.cf}",
                "method": "GET"
              }
                },
          "type": "HTTP"
    }],
    "all": [{
      "name": "fork_join",
         "taskReferenceName": "forktask",
       "type": "FORK_JOIN",
       "forkTasks": [
       [{
         "name": "richiestaPB",
         "taskReferenceName": "richiestaPBRef",
         "inputParameters": {
           "http_request":{
           "uri":  "http://172.17.0.1:8081/utente/${workflow.input.cf}",
            "method": "GET"
             }
         },
         "type": "HTTP"
       }],
      [{
        "name": "richiestaAsl",
            "taskReferenceName": "richiestaAslRef",
            "inputParameters": {
            "http_request":{
            "uri":  "http://172.17.0.1:8082/user/${workflow.input.cf}",
              "method": "GET"
            }
                },
             "type": "HTTP"
            }],
      [{
        "name": "richiestaCollocamento",
              "taskReferenceName": "richiestaCollocamentoRef",
              "inputParameters": {
              "http_request":{
            "uri":  "http://172.17.0.1:9090/users/${workflow.input.cf}",
              "method": "GET"
                }
```

```
                          },
              "type": "HTTP"
                }]
        ]},
          {
               "name": "joint",
              "taskReferenceName": "joint",
              "type": "JOIN",
              "joinOn" :[
                    "richiestaPBRef",
                    "richiestaAslRef",
              "richiestaCollocamentoRef"
                ]
            }]
      }
    }],
    "outputParameters": {
      "infoU": "${richiestaPB.output.response.body}",
      "saluteU": "${richiestaAsl.output.response.body}",
    "lavoroU": "${richiestaCollocamento.output.response.body}",
      "infoAll": "${richiestaPBRef.output.response.body}",
      "saluteAll": "${richiestaAslRef.output.response.body}",
    "lavoroAll": "${richiestaCollocamentoRef.output.response.body}",

  },
    "schemaVersion": 2
}]
```

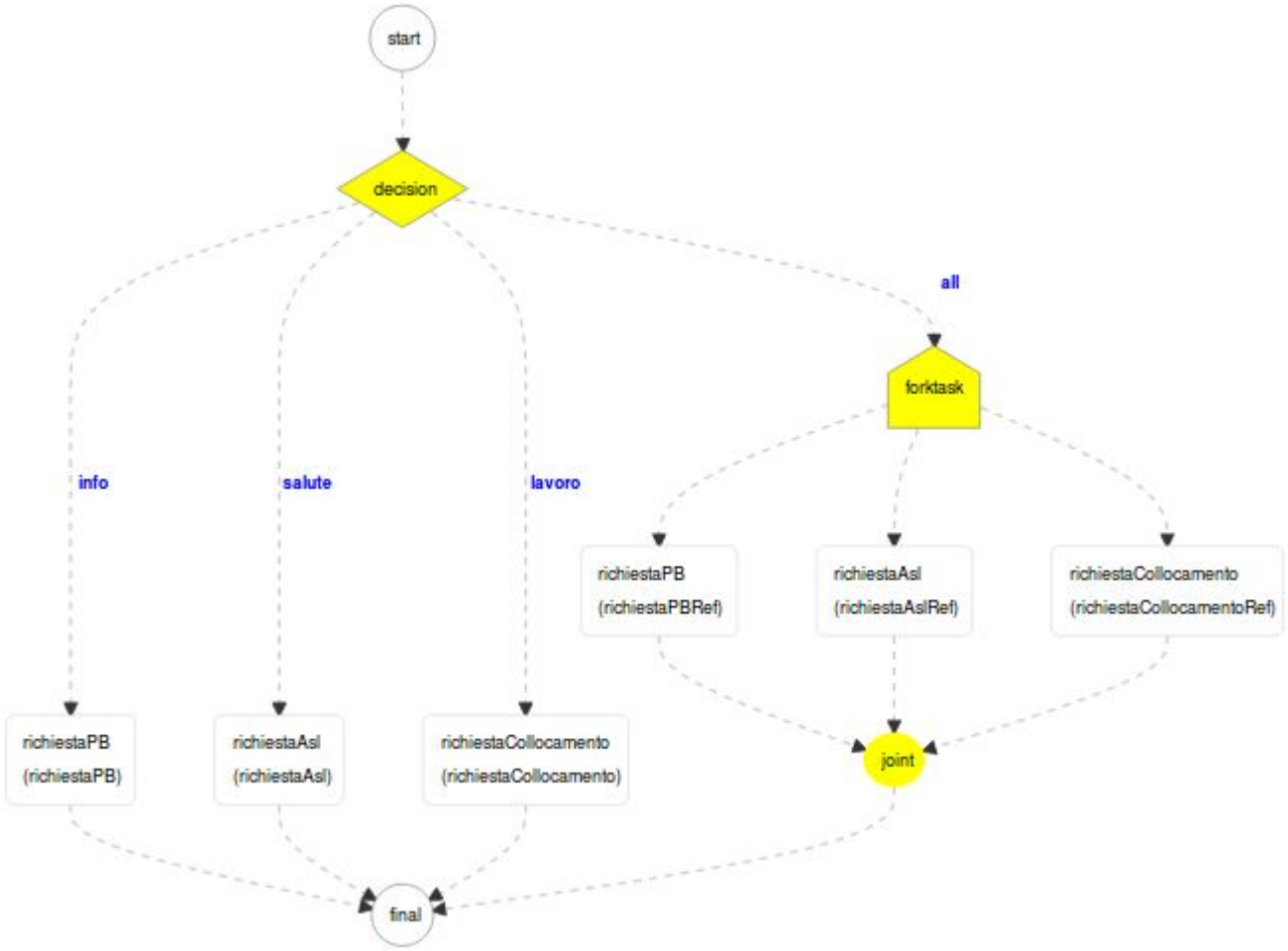


Figura 4.3: WorkflowGet

4.2.2.2 Workflow Post

```
{
  "ownerApp": "Orchestrazione",
  "createTime": 0,
  "updateTime": 0,
  "createdBy": "MAP",
  "name": "WorkFPostAPB",
  "description": "orchestrazione dei servizi",
  "version": 1,
  "tasks": [
    {
  "name": "decision",
  "taskReferenceName": "decision",
  "inputParameters": {
    "scelta": "${workflow.input.scelta}"
    },
```

```
  "type": "DECISION",
  "caseValueParam": "scelta",
  "decisionCases": {
    "info": [{
        "name": "richiestaPB",
        "taskReferenceName": "richiestaPB",
        "inputParameters": {
          "http_request":{
            "uri":  "http://172.17.0.1:8081/aggiungi",
              "method": "POST",
            "contentType": "application/json",
            "body":{
              "cf": "${workflow.input.codiceFiscale}",
              "nome": "${workflow.input.nome}",
              "cognome": "${workflow.input.cognome}",
              "indirizzo": "${workflow.input.indirizzo}",
              "numerofisso": "${workflow.input.numFisso}",
              "cellulare": "${workflow.input.cellulare}",
              "email": "${workflow.input.email}"
            }
          }
         },
         "type": "HTTP"
    }],
    "salute": [{
        "name": "richiestaAsl",
          "taskReferenceName": "richiestaAsl",
          "inputParameters": {
              "http_request":{
            "uri":  "http://172.17.0.1:8082/user/add",
            "method": "POST",
            "contentType": "application/json",
            "body":{
              "nome": "${workflow.input.nome}",
              "cognome": "${workflow.input.cognome}",
              "dataNascita": "${workflow.input.dataNascita}",
              "sesso": "${workflow.input.sesso}",
              "codiceFiscale": "${workflow.input.codiceFiscale}",
              "medicoBase": "${workflow.input.medicoBase}"
            }
          }
              },
              "type": "HTTP"
      }],
```

```
60      "lavoro": [{
61          "name": "richiestaCollocamento",
62          "taskReferenceName": "richiestaCollocamento",
63          "inputParameters": {
64                "http_request":{
65              "uri":  "http://172.17.0.1:9090/users/add",
66                "method": "POST",
67              "contentType": "application/json",
68              "body":{
69                "nome": "${workflow.input.nome}",
70                "cognome": "${workflow.input.cognome}",
71                "cfis": "${workflow.input.codiceFiscale}",
72                "data_nascita": "${workflow.input.dataNascita}",
73                "lavoro_attuale": "${workflow.input.lavoroAtt}",
74                "lavoro_precedente": "${workflow.input.lavoroPrec}"
75              }
76            }
77                },
78            "type": "HTTP"
79      }],
80      "all": [{
81         "name": "fork_join",
82        "taskReferenceName": "forktask",
83         "type": "FORK_JOIN",
84           "forkTasks": [
85          [{
86            "name": "richiestaPB",
87          "taskReferenceName": "richiestaPBPar",
88          "inputParameters": {
89            "http_request":{
90              "uri":  "http://172.17.0.1:8081/aggiungi",
91                "method": "POST",
92              "contentType": "application/json",
93              "body":{
94                "cf": "${workflow.input.codiceFiscale}",
95                "nome": "${workflow.input.nome}",
96                "cognome": "${workflow.input.cognome}",
97                "indirizzo": "${workflow.input.indirizzo}",
98                "numerofisso": "${workflow.input.numFisso}",
99                "cellulare": "${workflow.input.cellulare}",
100               "email": "${workflow.input.email}"
101             }
102           }
103           },
```

```
                "type": "HTTP"
      }],
      [{
        "name": "richiestaAsl",
          "taskReferenceName": "richiestaAslPar",
          "inputParameters": {
              "http_request":{
            "uri":  "http://172.17.0.1:8082/user/add",
            "method": "POST",
            "contentType": "application/json",
            "body":{
              "nome": "${workflow.input.nome}",
              "cognome": "${workflow.input.cognome}",
              "dataNascita": "${workflow.input.dataNascita}",
              "sesso": "${workflow.input.sesso}",
              "codiceFiscale": "${workflow.input.codiceFiscale}",
              "medicoBase": "${workflow.input.medicoBase}"
            }
          }
              },
                "type": "HTTP"
                  }],
      [{
        "name": "richiestaCollocamento",
        "taskReferenceName": "richiestaCollocamentoPar",
        "inputParameters": {
              "http_request":{
            "uri":  "http://172.17.0.1:9090/users/add",
              "method": "POST",
            "contentType": "application/json",
            "body":{
              "nome": "${workflow.input.nome}",
              "cognome": "${workflow.input.cognome}",
              "cfis": "${workflow.input.codiceFiscale}",
              "data_nascita": "${workflow.input.dataNascita}",
              "lavoro_attuale": "${workflow.input.lavoroAtt}",
              "lavoro_precedente": "${workflow.input.lavoroPrec}"
            }
          }
              },
          "type": "HTTP"
        }]
          ]},
          {
```

```
148             "name": "joint",
149             "taskReferenceName": "joint",
150             "type": "JOIN",
151             "joinOn" :[
152               "richiestaPBPar",
153               "richiestaAslPar",
154         "richiestaCollocamentoPar"
155       ]
156       }]
157   }
158 }],
159   "outputParameters": {
160     "infoU": "${richiestaPB.output.response.body}",
161     "saluteU": "${richiestaAsl.output.response.body}",
162   "lavoroU": "${richiestaCollocamento.output.response.body}",
163     "infoAll": "${richiestaPBPar.output.response.body}",
164     "saluteAll": "${richiestaAslPar.output.response.body}",
165   "lavoroAll": "${richiestaCollocamentoPar.output.response.body}",
166
167 },
168 "schemaVersion": 2 }
```

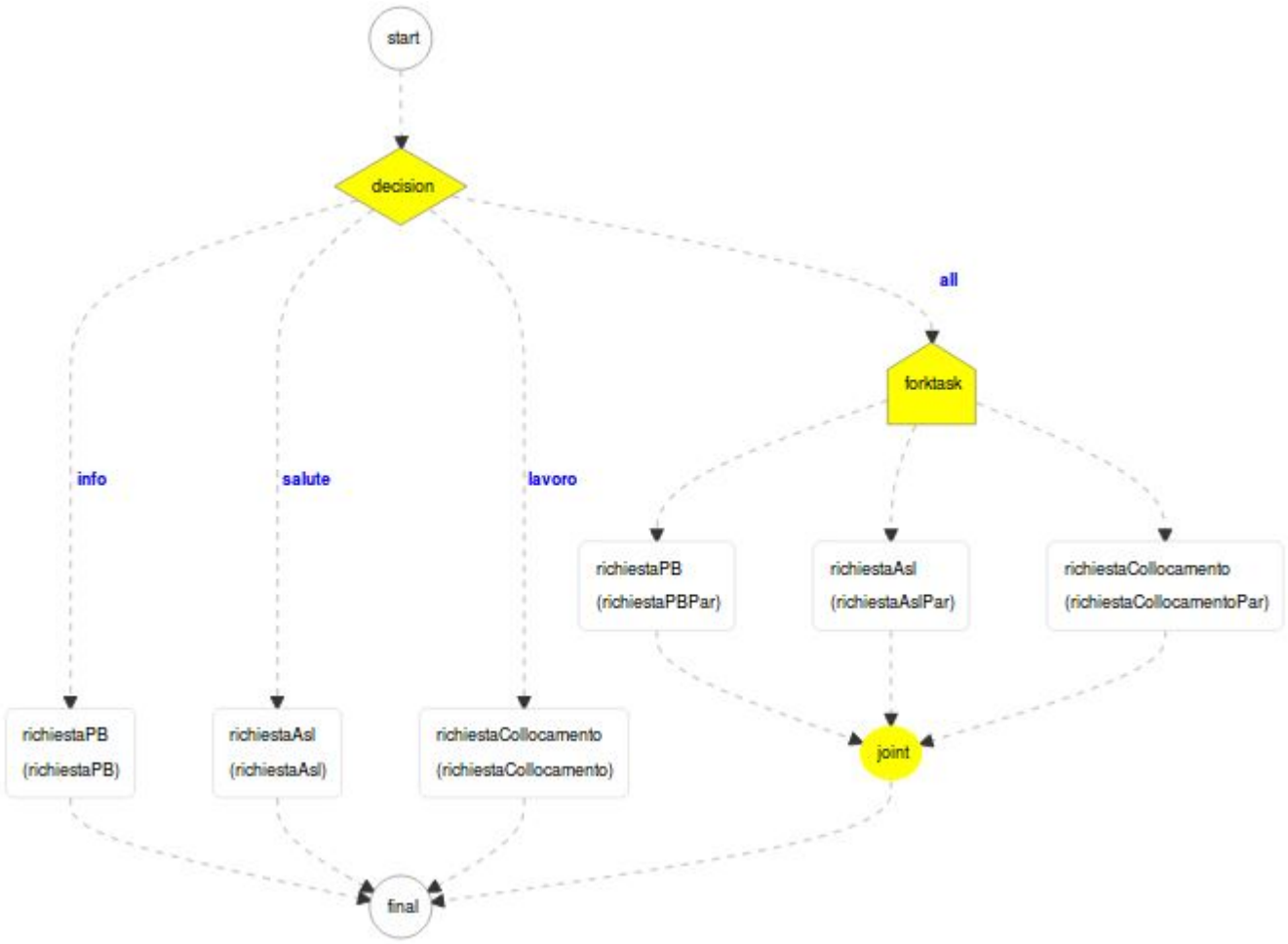


Figura 4.4: WorkflowPost

4.2.3 Esempi di esecuzione

In tale sezione sono riportati alcuni esempi di utilizzo dei servizi , task e workflow creati. In particolare di seguito sono riportati i passi generici per poter effettuare una richiesta :

- Collegarsi all'interfaccia Swagger localhost:8080
- Tra le varie liste di operazioni disponibili selezionare il menù Workflow Management
- Selezionare "Start a new workflow with StartWorkflowRequest, which allows task to be executed in a domain"

4.2.3.1 Richiesta informazioni Salute

Esempio di richiesta al servizio Asl:

```
{
    "name": "workflowGET",
    "version": 1,
```

```
    "input": {
    "scelta": "salute",
        "cf": "GRVMHL94"
                },
    "taskToDomain": {}
}
```

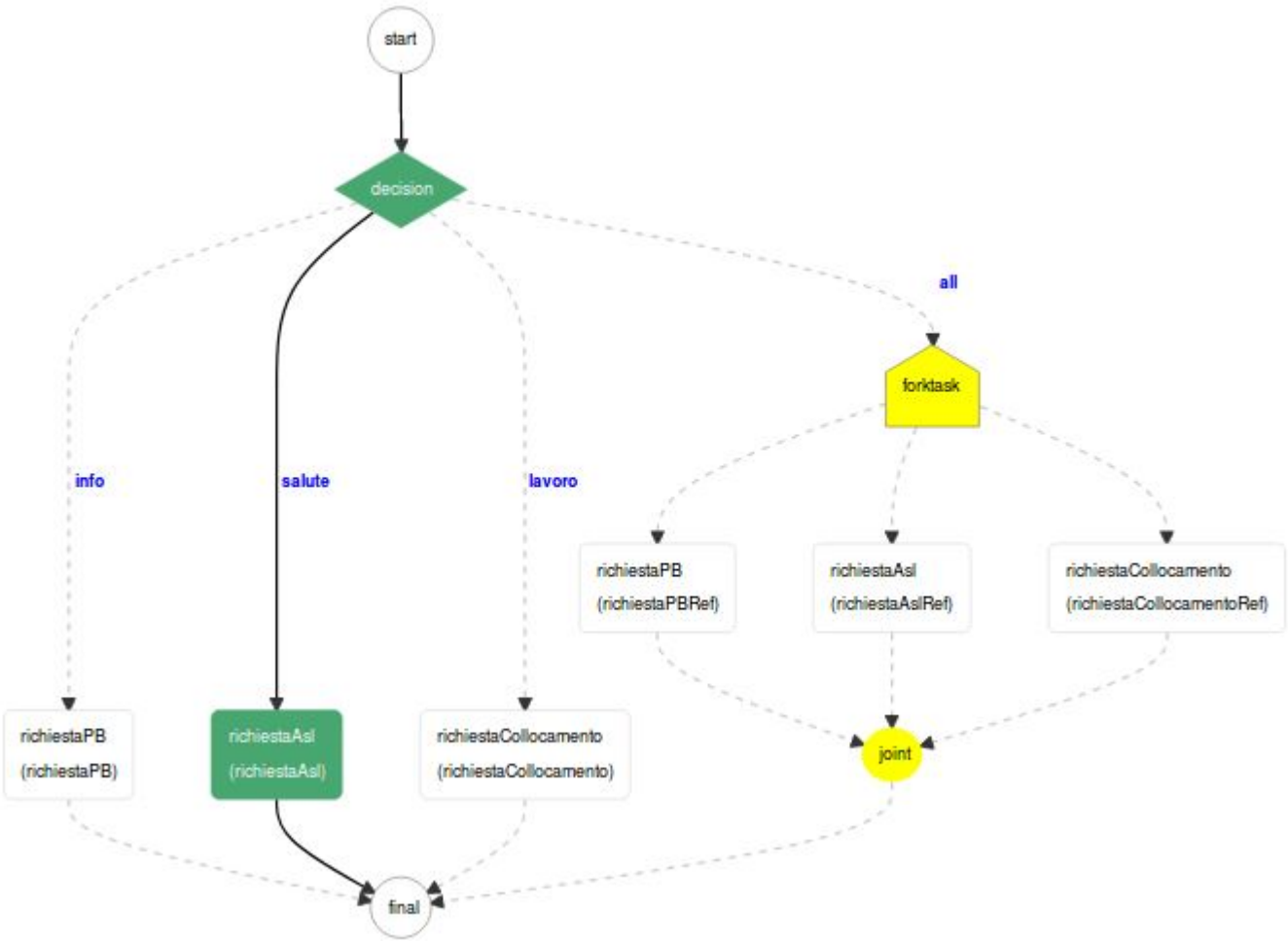


Figura 4.5: Workflow richiesta Asl

Figura 4.6: Input/Output richiesta asl

4.2.3.2 Richiesta ai servizi Asl, Pagine Bianche e Collocamento:

```
{
    "name": "workflowGET",
    "version": 1,
    "input": {
    "scelta": "all",
        "cf": "GRVMHL94"
                },
    "taskToDomain": {}
}
```

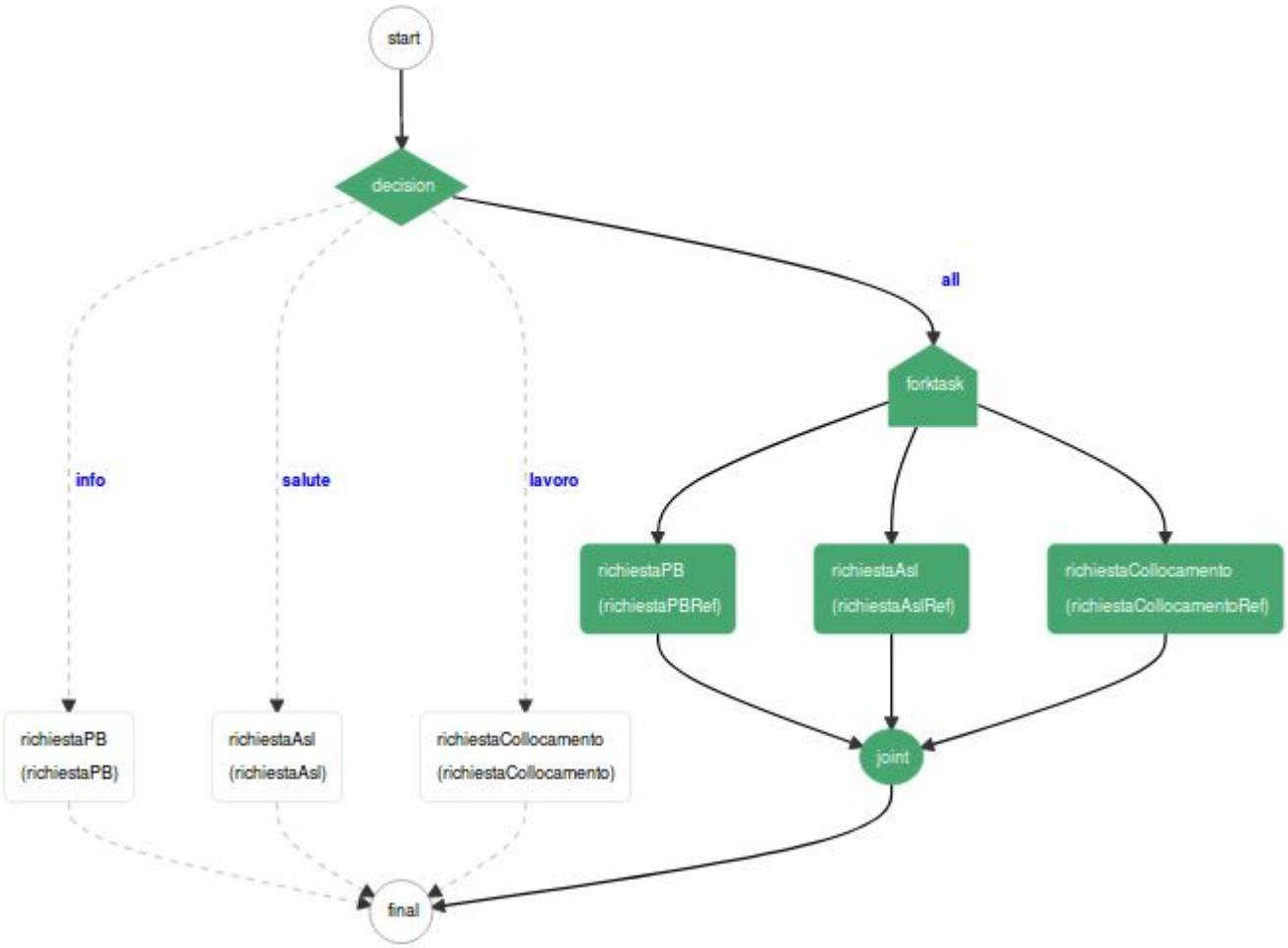


Figura 4.7: Workflow richiesta Asl

Figura 4.8: Input/Output richiesta asl, pagine bianche e collocamento

4.2.3.3 Richiesta di memorizzazione nuovo utente al servizio Asl:

```
{
    "name": "workflowPOST",
   "version": 1,
    "input": {
```

```
    "scelta": "salute",
      "codiceFiscale": "GLLANN91",
      "nome": "Anna",
      "cognome": "Galli",
      "dataNascita": "23/09/1991",
      "indirizzo": "via Napoli",
       "email": "anna@liber.it",
       "lavoroPrec": "disoccupata",
       "lavoroAtt":  "impiegata",
       "medicoBase": "Giovanni Marsico",
        "sesso": "femmina",
        "numFisso": "0825884384",
        "cellulare": "3425678456"
   },
    "taskToDomain": {}
   }
```

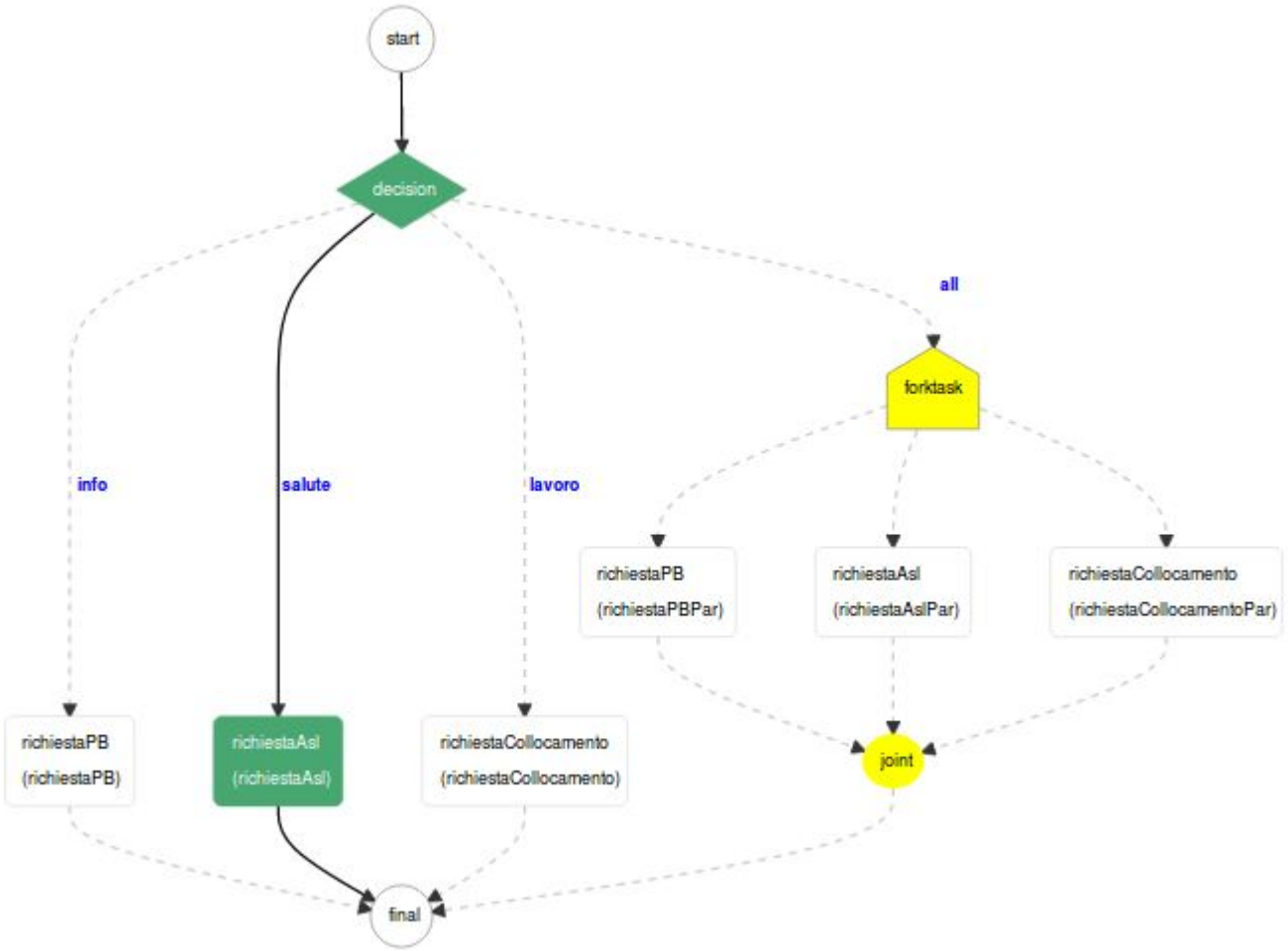


Figura 4.9: Workflow richiesta salvataggio utente Asl

Figura 4.10: Input/Output richiesta asl

4.2.3.4 Richiesta di memorizzazione nuovo utente ai servizi Asl, Pagine Bianche e Collocamento :

```
{
    "name": "workflowPOST",
   "version": 1,
    "input": {
    "scelta": "all",
      "codiceFiscale": "GLLANN91",
      "nome": "Anna",
      "cognome": "Galli",
      "dataNascita": "23/09/1991",
      "indirizzo": "via Napoli",
       "email": "anna@liber.it",
       "lavoroPrec": "disoccupata",
       "lavoroAtt":  "impiegata",
       "medicoBase": "Giovanni Marsico",
        "sesso": "femmina",
        "numFisso": "0825884384",
        "cellulare": "3425678456"
   },
    "taskToDomain": {}
   }
```

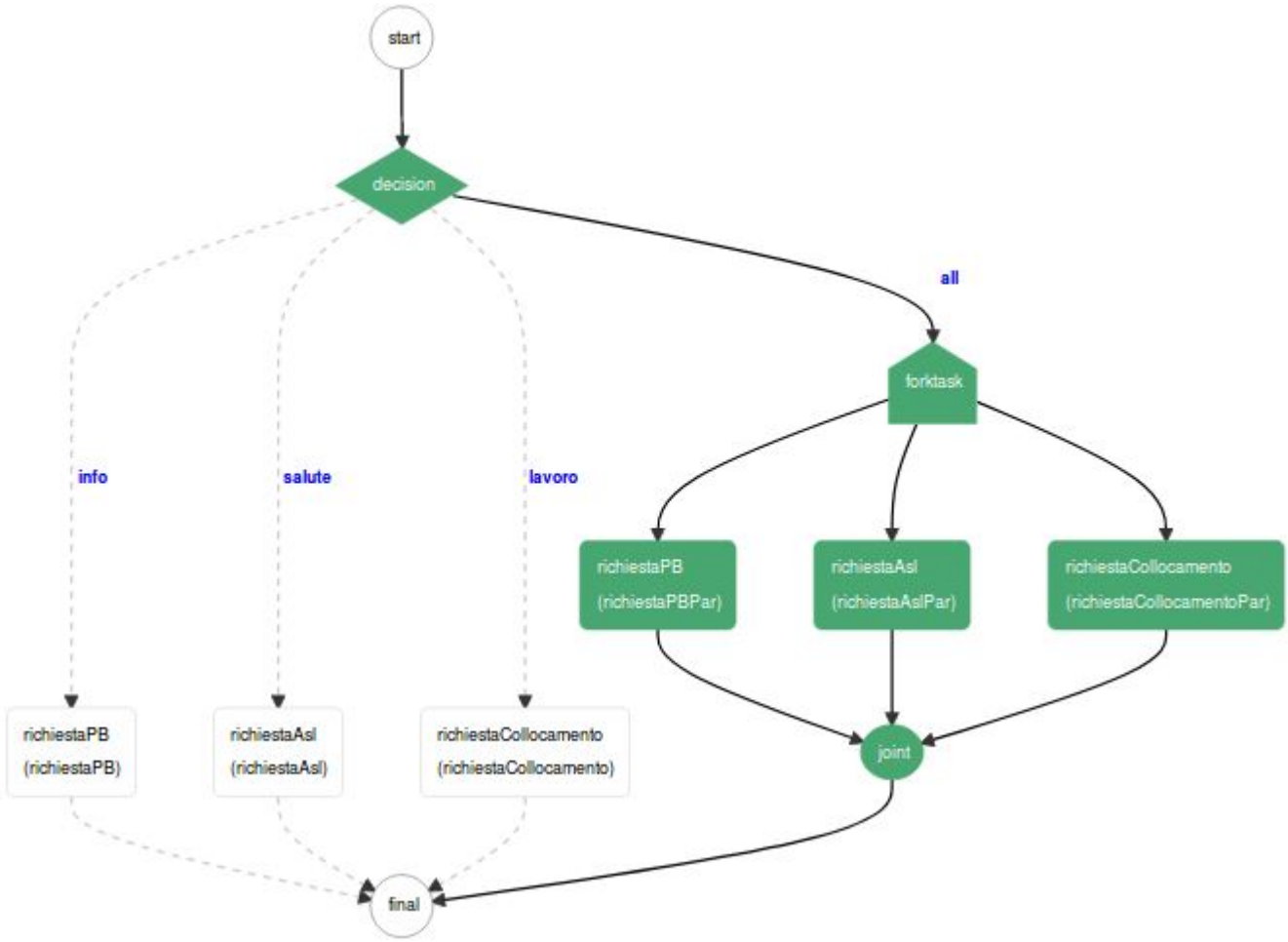


Figura 4.11: Workflow richiesta salvataggio utente Asl, Pagine Bianche e Collocamento

Workflow Input


```
{
   "scelta": "all",
   "codiceFiscale": "GLLANN94",
   "nome": "Anna",
   "cognome": "Galli",
   "dataNascita": "23/09/1991",
   "indirizzo": "via Napoli",
   "email": "anna@liber.it",
   "lavoroPrec": "disoccupata",
   "lavoroAtt": "impiegata",
   "medicoBase": "Giovanni Marsico",
```


Workflow Output


```
{
   "infoU": null,
   "saluteU": null,
   "lavoroU": null,
   "infoAll": "true",
   "saluteAll": "true",
   "lavoroAll": "true"
}
```


Figura 4.12: Input/Output richiesta asl, pagine bianche e collocamento