



TrasparenzAI - Piattaforma per l'analisi e la consultazione della trasparenza amministrativa delle Pubbliche Amministrazioni

Release 1.0.0

CNR - ANAC

09 apr 2025

Contents

1 Panoramica della soluzione	2
2 Architettura della soluzione	3
2.1 Scansione dei siti delle PA	3
3 Componenti principali	5
3.1 Public Sites Service	6
3.2 Config Service	9
3.3 Result Service	10
3.4 Result Aggregator Service	13
3.5 Task Scheduler Service	14
3.6 Rule Service	16
3.7 Conductor service	17
4 Risultati ottenuti	21
5 Installazione e configurazione	22
5.1 Autenticazione	22
5.2 Autorizzazione	23
5.3 Config service	23
5.4 Public Sites Service	25
5.5 Result Service	26
5.6 Result Aggregator Service	27
5.7 Task Scheduler Service	27
5.8 Risorse hardware consigliate	29
6 Appendice	30
6.1 Autori	30

Piattaforma per l'analisi e la consultazione della trasparenza amministrativa delle Pubbliche Amministrazioni

La trasparenza amministrativa rappresenta un pilastro fondamentale per il buon funzionamento delle istituzioni pubbliche, garantendo ai cittadini, agli operatori economici e agli organismi di controllo la possibilità di accedere in modo semplice e immediato alle informazioni riguardanti l'attività delle Pubbliche Amministrazioni.

Il Decreto Legislativo 33/2013 ha istituito l'obbligo per le amministrazioni di pubblicare una serie di dati, documenti e informazioni nella sezione **Amministrazione Trasparente**, con l'obiettivo di prevenire fenomeni di corruzione e di favorire una maggiore accountability.

Tuttavia, l'applicazione di questa normativa si è rivelata nel tempo complessa e disomogenea, generando difficoltà sia per le amministrazioni stesse, che devono gestire la pubblicazione e l'aggiornamento dei dati, sia per gli utenti finali, che incontrano ostacoli nell'individuazione e nella consultazione delle informazioni.

A fronte di queste criticità, la piattaforma *TrasprenzAI* nasce con lo scopo di semplificare e rendere più efficiente l'accesso e il monitoraggio delle informazioni pubblicate nella sezione Amministrazione Trasparente dei siti delle amministrazioni pubbliche.

La piattaforma è disponibile all'indirizzo:

- <https://www.trasprenzai.it>

CHAPTER 1

Panoramica della soluzione

La piattaforma *TransparenzAI* nasce con lo scopo di semplificare e rendere più efficiente l'accesso e il monitoraggio delle informazioni pubblicate nella sezione Amministrazione Trasparente dei siti delle amministrazioni pubbliche.

Il problema principale riscontrato è la grande eterogeneità delle modalità di pubblicazione. Ogni amministrazione, pur rispettando formalmente gli obblighi normativi, adotta soluzioni tecniche differenti per l'organizzazione e la struttura della sezione, con variazioni nella denominazione delle sotto-sezioni, nei percorsi di accesso e nella tipologia dei file pubblicati.

Questa frammentazione complica la consultazione delle informazioni e rende estremamente difficoltosa la verifica della loro presenza e conformità. Se per un cittadino alla ricerca di un determinato documento può risultare frustrante dover navigare in siti web strutturati in modo diverso, per ANAC diventa ancora più complesso ed oneroso monitorare il rispetto degli obblighi di pubblicazione su scala nazionale.

La piattaforma risponde a queste necessità introducendo un sistema automatizzato in grado di verificare in modo sistematico la presenza e la struttura delle sezioni Amministrazione Trasparente, fornendo un quadro aggiornato della situazione a livello nazionale.

Questo strumento non si limita a rilevare le irregolarità, ma offre anche un supporto alle amministrazioni per adeguarsi agli standard richiesti, migliorando così il livello complessivo della trasparenza amministrativa.

L'assenza di un meccanismo centralizzato che permetta di uniformare la pubblicazione delle informazioni rappresenta oggi un ostacolo alla piena efficacia della normativa vigente, e la piattaforma si pone l'obiettivo di colmare questa lacuna attraverso l'adozione di strumenti tecnologici avanzati.

L'architettura della piattaforma realizzata è composta da molti componenti integrati tra di loro, qui puoi trovare la documentazione dell'architettura generale, dei singoli componenti e dei risultati di validazione ottenuti.

Se sei interessato al codice sorgente della piattaforma lo puoi trovare su github:

- <https://github.com/cnr-anac>

CHAPTER 2

Architettura della soluzione

La piattaforma è composta da molti componenti, integrati in una architettura a microservizi, comunicanti tramite il paradigma REST. Ogni microservizio è realizzato con framework opensource e rilasciato a sua volta come software opensource. Anche tutti i componenti infrastrutturali utilizzati nella piattaforma sono opensource, azzerando i costi di licenza e permettendo una totale riusabilità di questa piattaforma per scopi uguali o analoghi a quelli per cui è stata pensata.

È previsto un accesso differenziato alle funzionalità della piattaforma, tramite un sistema di autenticazione e autorizzazione basato sul protocollo OAuth2 e dei ruoli predefiniti inseriti nel token JWT.

Sia l'interfaccia WEB, realizzata dal componente *UI Service*, che ogni API REST, è integrata quindi con un IDP OAuth2. Il servizio in staging realizzato per ANAC utilizza [Keycloak](#) come IDP OAuth2.

Il grafico in Fig. 2.1 sottostante riassume i componenti principali del sistema.

Rispetto a una tipica architettura a microservizi è stato scelto di non introdurre al momento un API Gateway e un Service Discovery, in quanto la sua adozione potrebbe dipendere dalle politiche di deploy della soluzione. In particolare qualora si decida di usare Kubernetes potrebbe essere indicato utilizzare i meccanismi di API Gateway e Service Discovery disponibili nel coordinatore dei container.

2.1 Scansione dei siti delle PA

La fase di scansione dei siti delle PA viene coordinata dal servizio *Conductor Service*. Questo servizio è basato sul componente opensource [Conductor](#) realizzato da Netflix.

La definizione dei workflow e dei task necessari per compiere tutte le operazioni di analisi, verifica e salvataggio dei risultati delle scansioni dei siti delle PA è definita tramite alcuni file JSON. Questo garantisce una facile configurabilità e adattabilità di questa soluzione a evoluzioni di questa piattaforma oppure a problematiche di crawling e analisi di natura diversa da quella di questo progetto.

L'avvio della fase di scansione di tutti i siti delle PA viene avviata dal microservizio *Task Scheduler Service*, il quale invoca via REST il *Conductor Service* con una cadenza configurabile (per esempio 3 o 4 volte la settimana).

La lista dei siti delle PA è prelevata dal microservizio *Public Site Service*, mentre i risultati delle validazioni sono inseriti sia nel microservizio *Result Service* che nel servizio *Result Aggregator Service*. Inoltre le screenshot delle



Figure2.1: TrasprenzAI - Overview Diagram

pagine HTML ritenute problematiche dal sistema vengono archiviate in uno storage *S3 like*, in particolare nel servizio fornito in staging a ANAC viene utilizzato il prodotto Opensource **Minio**.

Le regole applicate per la verifica della corrispondenza del sito con la legge sulla trasparenza sono definite tramite il servizio **Rule Service**. Quest'ultimo è stato realizzato in modo da essere un servizio generico di applicazioni di regole di parsing e configurabile tramite file JSON.

Per quanto riguarda la parte di crawling e di rendering delle pagine HTML da analizzare la soluzione prevede l'utilizzo di un proprio crawler per prelevare lo streaming hml delle pagine da analizzare e l'adozione di un **Selenium Hub** per distribuire su più istanze di Google Chrome il rendering delle pagine HTML che contengono codice javascript da interpretare.

CHAPTER 3

Componenti principali

Il sistema TrasparenzAI è di tipo modulare ed è composta sia da componenti sviluppati ad-hoc per il progetto che da software opensource già disponibili.

I componenti sviluppati per il progetto sono:

- <https://github.com/cnr-anac/public-sites-service>
- <https://github.com/cnr-anac/rule-service>
- <https://github.com/cnr-anac/result-service>
- <https://github.com/cnr-anac/result-aggregator-service>
- <https://github.com/cnr-anac/task-scheduler-service>
- <https://github.com/cnr-anac/conductor>
- <https://github.com/cnr-anac/workflow-definition>
- <https://github.com/cnr-anac/ui-service>

Alcuni di questi componenti sono stati sviluppati in un ottica di possibile riuso da parte della comunità opensource italiana, alcuni così come sono, altri come esempio per progetti di crawling e analisi di siti di web similari. I componenti stati sviluppati principalmente in Java (con [Spring Boot](#)) e Python ([FastAPI](#) e [Uvicorn](#)) per la parte backend e typescript ([Angular](#)) per la parte frontend.

I software opensource utilizzati sono:

- Keycloak
- Postgresql
- Minio
- Selenium Grid
- Selenium Node Chrome
- Conductor
- Traefik

Del software *Conductor* è stato effettuato un fork per introdurre l'autenticazione come client OAuth2 nei task che interagiscono con le API REST della piattaforma.

3.1 Public Sites Service

Public Sites Service è il componente che si occupa di gestire le informazioni principali relative agli enti pubblici italiani ed in particolare i siti istituzionali.

Public Sites Service mantiene nel proprio datastore locale le informazioni degli enti che possono essere inserite/aggiornate tramite gli OpenData di IndicePA, oppure inserite tramite appositi servizi endpoint REST.

L'idea è quella di avere una fonte facile da consultare e estendibile delle informazioni delle organizzazioni pubbliche da analizzare. In particolare sono trattate automaticamente le info utili derivanti IndicePA ma è possibile inserire altri enti da sottoporre a analisi, inserendoli via REST in questo servizio oppure integrando altre fonti esterne sincronizzate automaticamente.

Public Sites Service fornisce alcuni servizi REST utilizzabili in produzione per:

- mostrare la lista degli enti presenti negli OpenData di IndicePA
- inserire ed aggiornare all'interno del servizio le informazioni degli Enti tramite gli OpenData di IndicePA
- geolocalizzare gli Enti italiani tramite il servizio Nominatim di OpenStreetMap
- visualizzare i dati di un Ente
- mostrare la lista paginata degli Enti presenti nel servizio, con possibilità di filtrarli per codiceCategoria, codiceFiscaleEnte, codiceIpa, denominazioneEnte
- inserire, aggiornare e cancellare le informazioni degli Enti all'interno del servizio (direttamente senza passare da IndicePA)

Il servizio sincronizza e rende disponibili via REST anche le informazioni dei comuni italiani, prelevando ogni notte il CSV dal sito dell'ISTAT dei comuni e aggiornando questo info dentro il servizio stesso. Le info dei comuni servono anche per effettuare una geolocalizzazione più precisa degli enti, che su IndicePA sono classificati solamente tramite il codice catastale del comune.

L'aggiornamento dei dati locali al servizio Public Sites Service tramite IndicePA avviene ogni mattina alle 6:30. L'aggiornamento dei dati locali al servizio Public Sites Service tramite il CSV di ISTAT avviene ogni mattina alle 6:40.

Il codice sorgente di questo componente è disponibile su GitHub:

- <https://github.com/cnr-anac/public-sites-service>

3.1.1 OpenAPI e Swagger UI

Una volta avviato il servizio i servizi REST sono documentati tramite OpenAPI e consultabili all'indirizzo /swagger-ui/index.html.

L'OpenAPI del servizio di staging è disponibile all'indirizzo <https://dica33.ba.cnr.it/public-sites-service/swagger-ui/index.html>.

The screenshot shows the TrasparenzAI API documentation interface. At the top, there's a navigation bar with the Swagger logo, the URL '/public-sites-service/v3/api-docs', and a green 'Explore' button. Below the header, the title 'Public sites Service' is displayed with a version '0.1.0' and a 'GAS 3.0' badge. A sub-header '(public-sites-service/v3/api-docs)' follows. A brief description states: 'Public Sites Service si occupa di gestire le informazioni principali relative agli enti pubblici italiani ed in particolare i siti istituzionali'. On the left, there's a 'Servers' dropdown set to 'ipublic-sites-service - Public Sites Service URL' and an 'Authorize' button with a lock icon. The main content area is divided into sections: 'Geo Controller' (with 13 API endpoints listed), 'Admin Controller' (with 10 API endpoints listed, one of which is highlighted in red), and 'Company Controller' (with 8 API endpoints listed, two of which are highlighted in red). Each endpoint entry includes the method (e.g., POST, GET), the path, and a brief description.

Geo Controller Visualizzazione e gestione delle informazioni geografiche

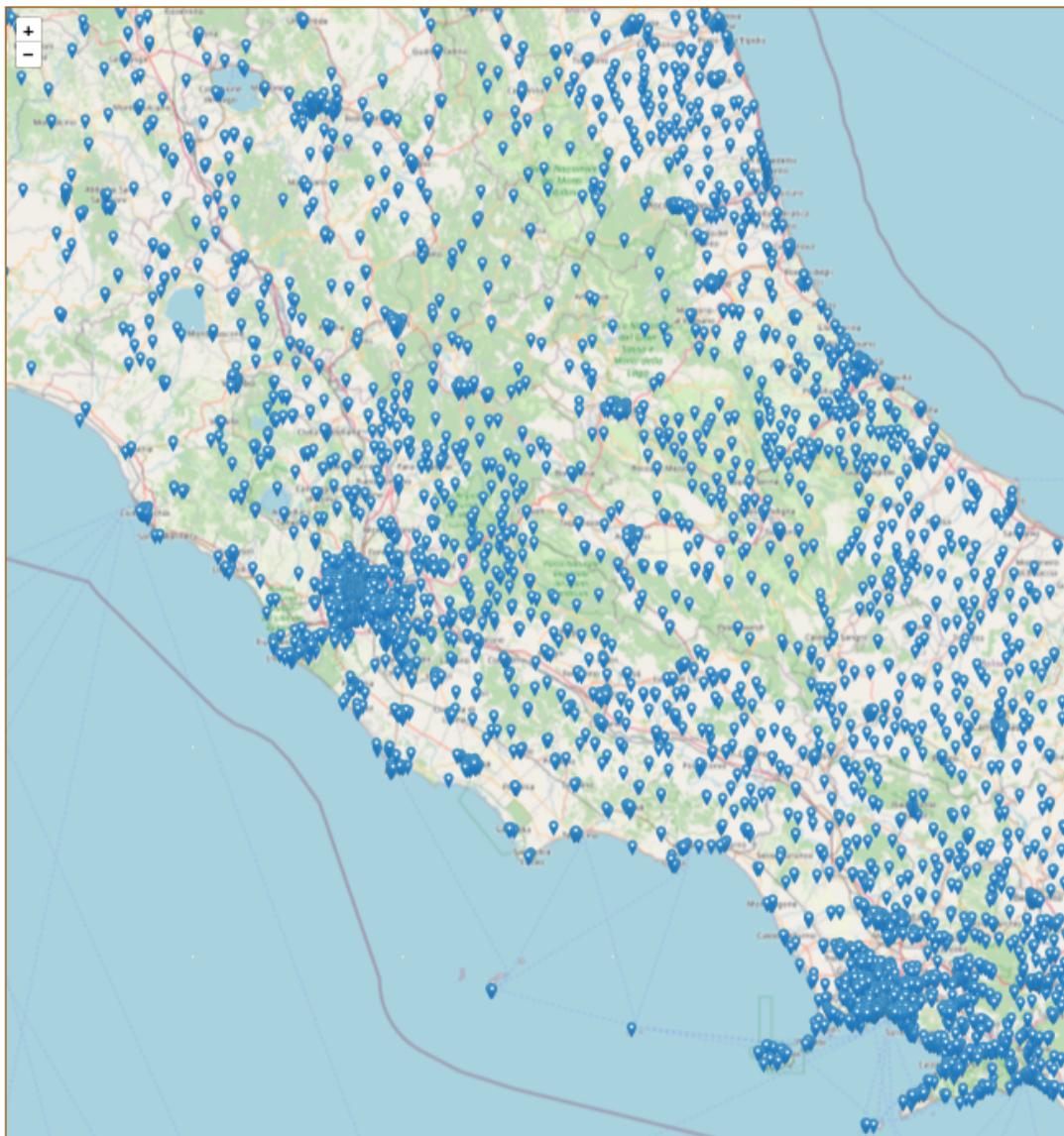
Admin Controller Metodi di supporto per visualizzazione e gestione dei comuni italiani

Company Controller Gestione delle informazioni degli Enti

3.1.2 Mappa delle PA Italiane

Il servizio contiene anche una mappa geografica delle PA italiane realizzata tramite leaflet.

Mappa Pubbliche Amministrazioni Italiane



3.1.3 Sicurezza

Gli endpoint REST di questo servizio sono protetti tramite autenticazione OAuth con Bearer Token. E' necessario configurare l'idp da utilizzare per validare i token OAuth tramite le due proprietà mostrate nell'esempio seguente:

```
- spring.security.oauth2.resource-server.jwt.issuer-uri=https://dica33.ba.cnr.it/keycloak/  
- spring.security.oauth2.resource-server.jwt.jwk-set-uri=https://dica33.ba.cnr.it/  
- keycloak/realm/trasparenzai/protocol/openid-connect/certs
```

Per l'accesso in HTTP GET all'API è sufficiente essere autenticati, per gli endpoint accessibili con PUT/POST/DELETE è necessario oltre che essere autenticati che il token OAuth contenga un role ADMIN o SUPERUSER.

3.2 Config Service

Config Service è il componente che si occupa di archiviare e distribuire alcune informazioni di configurazione dei servizi che compongono lo stack del progetto TrasparenzAI.

Config Service mantiene nel proprio datastore locale le configurazioni che sono fornite agli altri microservizi. Le configurazioni possono essere inserite/aggiorinate tramite gli appositi endpoint REST presenti in questo servizio.

Le configurazioni disponibili sono fornite sia sotto forma di endpoint REST con le relative CRUD, che nel formato utilizzato da *Spring Cloud Config* attraverso il path **/config**, inserendo il nome del servizio e il profilo richiesto nell'url, come per esempio:

```
$ http GET :8888/config/task-scheduler/default  
  
{  
  "label": null,  
  "name": "task-scheduler",  
  "profiles": [  
    "default"  
  ],  
  "propertySources": [  
    {  
      "name": "task-scheduler-default",  
      "source": {  
        "tasks.fake.cron.expression": "0 46 15 * * ?",  
        "test.property1": "testme"  
      }  
    }  
  ],  
  "state": null,  
  "version": null  
}
```

I microservizi Spring che vogliono utilizzare questo servizio di configurazione centralizzato possono farlo specificando nella propria configurazione tre parametri tipo:

```
spring.config.import=optional:configserver:http://@localhost:8888/config  
spring.cloud.config.username=config-service-user  
spring.cloud.config.password=PASSWORD_DA_IMPOSTARE_E_CONDIVIDERE_CON_I_CLIENT
```

Dove naturalmente va impostato il corretto URL a cui risponde questo servizio.

Il codice sorgente di questo componente è disponibile su GitHub:

- <https://github.com/cnr-anac/config-service>

3.2.1 OpenAPI e Swagger UI

Una volta avviato il servizio i servizi REST sono documentati tramite OpenAPI e consultabili all'indirizzo /swagger-ui/index.html.

L'OpenAPI del servizio di staging è disponibile all'indirizzo <https://dica33.ba.cnr.it/config-service/swagger-ui/index.html>.

3.2.2 Sicurezza

L'accesso in lettura alla configurazione di tipo *Spring Cloud Config* disponibile al path /config è protetto con autenticazione di tipo *Basic Auth*, i microservizi che vogliono utilizzare questo path per ottenere la configurazione devono utilizzare l'utente e la password specificati tramite i parametri *spring.security.user.name* e *spring.security.user.password* , i quali possono essere specificati nel *docker-compose.yml* come nell'esempio seguente:

- `spring.security.user.name=config-service-user`
 - `spring.security.user.password=PASSWORD_DA_IMPOSTARE_E_CONDIVIDERE_CON_I_CLIENT`

Invece gli endpoint REST di questo servizio disponibili al path /properties sono protetti tramite autenticazione OAuth con Bearer Token. È necessario configurare l'IDP da utilizzare per validare i token OAuth tramite le due proprietà impostabili nel *docker-compose.yml* come nell'esempio seguente:

- `spring.security.oauth2.resourceserver.jwt.issuer-uri=https://dica33.ba.cnr.it/keycloak/realms/trasparenzai`
 - `spring.security.oauth2.resourceserver.jwt.jwk-set-uri=https://dica33.ba.cnr.it/keycloak/realms/trasparenzai/protocol/openid-connect/certs`

Per l'accesso in HTTP GET all'API è sufficiente essere autenticati, per gli endpoint accessibili con PUT/POST/DELETE è necessario oltre che essere autenticati che il token OAuth contenga un role ADMIN o SUPERUSER.

3.3 Result Service

Result Service è il componente che si occupa di gestire i risultati delle verifiche sulla corrispondenza dei siti degli enti pubblici italiani in relazione al decreto legge 33/2013 sulla transparenza.

Result Service fornisce alcuni servizi REST utilizzabili in produzione per:

- inserire, aggiornare e cancellare all'interno del servizio le informazioni di una verifica effettuata su un sito web di una PA
- visualizzare i dati di una verifica su un sito web
- mostrare la lista delle verifiche effettuate
- esportare in CSV i risultati delle validazioni presenti

Il codice sorgente di questo componente è disponibile su GitHub:

- <https://github.com/cnr-anac/result-service>

The screenshot shows the Swagger UI interface for the config-service API. At the top, it displays the title "OpenAPI definition" with a "v0" version indicator and a "OAS 3.0" badge. Below this, there's a "Servers" section with a dropdown menu set to "http://edica33.ha.cnr.it/config-service - Generated server url". The main content area is organized into sections for different controllers:

- profile-controller**:
 - GET /profile
 - GET /profile/properties
- property-entity-controller**:
 - GET /properties
 - POST /properties
 - GET /properties/{id}
 - PUT /properties/{id}
 - DELETE /properties/{id}
 - PATCH /properties/{id}
- encryption-controller**:
 - POST /config/encrypt
 - POST /config/encrypt/{name}/{profiles}
 - POST /config/decrypt
 - POST /config/decrypt/{name}/{profiles}
 - GET /config/key
 - GET /config/key/{name}/{profiles}
 - GET /config/encrypt/status
- environment-controller**:
 - GET /config/{name}/{profiles}/{label}
 - GET /config/{name}/{profiles}
 - GET /config/{name}-{profiles}.yaml
 - GET /config/{name}-{profiles}.yml
 - GET /config/{name}-{profiles}.properties
 - GET /config/{name}-{profiles}.json
 - GET /config/{label}/{name}-{profiles}.yaml
 - GET /config/{label}/{name}-{profiles}.yml
 - GET /config/{label}/{name}-{profiles}.properties
 - GET /config/{label}/{name}-{profiles}.json

3.3.1 OpenAPI e Swagger UI

Una volta avviato il servizio i servizi REST sono documentati tramite OpenAPI e consultabili all'indirizzo /swagger-ui/index.html.

The screenshot shows the Swagger UI interface for the Transparency Results Service. At the top, there's a navigation bar with the 'Swagger' logo and a 'Explore' button. Below the title, it says 'Transparency Results Service 0.2.0 OAS 3.0'. A note below the title states: 'Transparency Results Service si occupa di gestire i risultati delle verifiche di conformità sulla legge della trasparenza del decreto legge 33/2013 per i siti degli enti pubblici italiani.' On the left, there's a 'Servers' dropdown set to 'result-service - Transparency Results Service URL' and an 'Authorize' button. The main content area is titled 'Result Controller' and lists various API endpoints:

- GET /v1/results** Visualizzazione dei risultati di validazione presenti nel sistema, filtrabili utilizzando alcuni parametri.
- PUT /v1/results** Creazione di un risultato di validazione.
- POST /v1/results** Aggiornamento dei dati di un risultato di validazione.
- PUT /v1/results/bulk** Creazione di più di un risultato di validazione.
- GET /v1/results/{id}** Visualizzazione delle informazioni di un risultato di validazione.
- DELETE /v1/results/{id}** Eliminazione di un risultato di validazione. (This endpoint is highlighted with a red border)
- GET /v1/results/lastRunAsCsv** Visualizzazione dei risultati dell'ultima validazione registrata nel sistema.
- GET /v1/results/lastResult** Visualizzazione delle informazioni del ultimo risultato memorizzato nel sistema.
- GET /v1/results/csv** Visualizzazione dei risultati di validazione presenti nel sistema.
- GET /v1/results/countAndGroupByWorkflowIdAndStatus** visualizzazione delle informazioni presenti nel sistema raggruppate per flusso e stato della regola applicata.
- GET /v1/results/all** Visualizzazione dei risultati di validazione presenti nel sistema, filtrabili utilizzando alcuni parametri.
- DELETE /v1/results/byWorkflow/{id}** Eliminazione dei risultati di validazione associati a un workflow id. (This endpoint is highlighted with a red border)

L'OpenAPI del servizio di staging è disponibile all'indirizzo <https://dica33.ba.cnr.it/result-service/swagger-ui/index.html>.

3.3.2 Sicurezza

Gli endpoint REST di questo servizio sono protetti tramite autenticazione OAuth con Bearer Token. E' necessario configurare l'idp da utilizzare per validare i token OAuth tramite le due proprietà mostrare nell'esempio seguente:

```
- spring.security.oauth2.resourceserver.jwt.issuer-uri=https://dica33.ba.cnr.it/keycloak/  
- realms/trasparenzai  
- spring.security.oauth2.resourceserver.jwt.jwk-set-uri=https://dica33.ba.cnr.it/  
- keycloak/realms/trasparenzai/protocol/openid-connect/certs
```

Per l'accesso in HTTP GET all'API è sufficiente essere autenticati, per gli endpoint accessibili con PUT/POST/DELETE è necessario oltre che essere autenticati che il token OAuth contenga un role ADMIN o SUPERUSER.

3.4 Result Aggregator Service

Result Aggregator Service è il componente che si occupa di gestire i risultati delle verifiche sulla corrispondenza, aggregando i risultati di validazione con altre informazioni sugli enti pubblici prelevate da altri servizi.

Result Aggregator Service fornisce alcuni servizi REST utilizzabili in produzione per:

- inserire, aggiornare e cancellare all'interno del servizio le informazioni di una verifica effettuata su un sito web di una PA ed dei dati geografici degli enti pubblici
- esportare in geoJson i risultati delle validazioni presenti arricchiti con la geolocalizzazione degli enti

Il codice sorgente di questo componente è disponibile su GitHub:

- <https://github.com/cnr-anac/result-aggregator-service>

3.4.1 OpenAPI e Swagger UI

Una volta avviato il servizio i servizi REST sono documentati tramite OpenAPI e consultabili all'indirizzo /swagger-ui/index.html.

The screenshot shows the Swagger UI interface for the Transparency Results Service. At the top, there's a navigation bar with the 'Swagger' logo and a 'Explore' button. Below the title, there's a brief description of the service: 'Transparency Results Service si occupa di gestire i risultati delle verifiche di conformità sulla legge della trasparenza del decreto legge 33/2013 per i siti degli enti pubblici italiani.' The main content area is titled 'Result Controller' and contains a list of API endpoints:

- GET /v1/results** Visualizzazione dei risultati di validazione presenti nel sistema, filtrabili utilizzando alcuni parametri.
- PUT /v1/results** Creazione di un risultato di validazione.
- POST /v1/results** Aggiornamento dei dati di un risultato di validazione.
- PUT /v1/results/bulk** Creazione di più di un risultato di validazione.
- GET /v1/results/{id}** Visualizzazione delle informazioni di un risultato di validazione.
- DELETE /v1/results/{id}** Eliminazione di un risultato di validazione.
- GET /v1/results/lastRunAsCsv** Visualizzazione dei risultati dell'ultima validazione registrata nel sistema.
- GET /v1/results/lastResult** Visualizzazione delle informazioni del ultimo risultato memorizzato nel sistema.
- GET /v1/results/csv** Visualizzazione dei risultati di validazione presenti nel sistema.
- GET /v1/results/countAndGroupByWorkflowIdAndStatus** visualizzazione delle informazioni presenti nel sistema raggruppate per flusso e stato della regola applicata.
- GET /v1/results/all** Visualizzazione dei risultati di validazione presenti nel sistema, filtrabili utilizzando alcuni parametri.
- DELETE /v1/results/byWorkflow/{id}** Eliminazione dei risultati di validazione associati a un workflow id.

L'OpenAPI del servizio di staging è disponibile all'indirizzo <https://dica33.ba.cnr.it/result-aggregator-service/swagger-ui/index.html>.

3.4.2 Dipendenze e configurazione

Questo servizio ha due dipendenze dagli altri componenti per funzionare:

- il [Result Service](#) da cui leggere le info sulle verifiche
- il [Public Site Service](#) da cui prelevare le info geografiche delle PA

L'indirizzo di entrambi questi servizi è da configurabile nel file `application.properties` oppure tramite variabili d'ambiente se avviato tramite Docker.

3.4.3 Sicurezza

Gli endpoint REST di questo servizio sono protetti tramite autenticazione OAuth con Bearer Token. E' necessario configurare l'idp da utilizzare per validare i token OAuth tramite le due proprietà mostrare nell'esempio seguente:

```
- spring.security.oauth2.resourceserver.jwt.issuer-uri=https://dica33.ba.cnr.it/keycloak/  
  ↵realms/trasparenzai  
- spring.security.oauth2.resourceserver.jwt.jwk-set-uri=https://dica33.ba.cnr.it/  
  ↵keycloak/realms/trasparenzai/protocol/openid-connect/certs
```

Per l'accesso in HTTP GET all'API è sufficiente essere autenticati, per gli endpoint accessibili con PUT/POST/DELETE è necessario oltre che essere autenticati che il token OAuth contenga un role ADMIN o SUPERUSER.

Inoltre questo servizio interagisce con il `_result_service_` e il `_public_site_service_` per prelevare i risultati da aggregare.

Per configurare il client REST che accede a questi due servizi è necessario configurare questi parametri nel `docker-compose.yml`, in particolare verificare `client-id`, `client-secret` e `issuer-uri`.

Esempio di configurazione dell'environment nel docker-compose.yml:

```
# Generare un Service Account Oidc con questo client-id, oppure cambiare questo valore  
- spring.security.oauth2.client.registration.oidc.client-id=result-aggregator  
# Client Secret da generare nel Identity Provider e impostare qui  
- spring.security.oauth2.client.registration.oidc.client-secret=client_secret_da_generare  
# URL dell'issuer OIDC da impostare  
- spring.security.oauth2.client.provider.oidc.issuer-uri=https://dica33.ba.cnr.it/  
  ↵keycloak/realms/trasparenzai  
# - spring.security.oauth2.client.registration.oidc.authorization-grant-type=client_  
  ↵credentials #DEFAULT  
# - spring.security.oauth2.client.registration.oidc.scope=openid #DEFAULT  
# - spring.security.oauth2.client.registration.oidc.provider=oidc #DEFAULT
```

3.5 Task Scheduler Service

Task Scheduler Service è il componente che si occupa di avviare alcuni processi eseguiti a intervalli fissi, come per esempio l'**avvio delle scansioni** dei siti del PA per la verifica della corrispondenza dei requisiti e la **cancellazione dei risultati di scansione più vecchi**.

Nell'utilizzo tramite `docker-compose.yml` ricordarsi di impostare nel `.env` la corretta variabile d'ambiente che specifica l'url del config-service da utilizzare e la password per l'autenticazione Basic Auth con il `config-service`:

environment:

- confighost=\${CONFIG_HOST}
- spring.security.oauth2.client.registration.oidc.client-secret=\${OIDC_CLIENT_SECRET}

Le informazioni di configurazione dei cron relativi ai workflow possono essere visualizzate all'url [/tasks/workflowCronConfig](#).

Il codice sorgente di questo componente è disponibile su GitHub:

- <https://github.com/cnr-anac/task-scheduler-service>

3.5.1 OpenAPI e Swagger UI

Una volta avviato il servizio i servizi REST sono documentati tramite OpenAPI e consultabili all'indirizzo /swagger-ui/index.html.

The screenshot shows the Swagger UI interface for the Task Scheduler Service. At the top, it displays the title "Task Scheduler Service OpenAPI" with a version of "0.1.0" and "OAS 3.0". Below the title, there's a navigation bar with links for "/task-scheduler-service/v3/api-docs" and "Explore". A "Servers" dropdown is set to "http://dica33.ba.cnr.it/task-scheduler-service - Generated server url". The main area shows the API endpoints for the "task-info-controller". The endpoints listed are:

- POST /tasks/startWorkflow
- GET /tasks/workflowIdsToPreserveFromConfig
- GET /tasks/workflowCronConfig
- GET /tasks/expiredWorkflows
- GET /tasks/completedWorkflows
- DELETE /tasks/deleteWorkflow
- DELETE /tasks/deleteExpiredWorkflows

Each endpoint row has a lock icon and a dropdown arrow. The "DELETE /tasks/deleteWorkflow" and "DELETE /tasks/deleteExpiredWorkflows" rows are highlighted with a red background.

L'OpenAPI del servizio di staging è disponibile all'indirizzo <https://dica33.ba.cnr.it/task-scheduler-service/swagger-ui/index.html>.

3.5.2 Dipendenze e configurazione

Questo servizio ha quattro dipendenze per funzionare:

- il **Config Service** da cui prelevare i parametri per l'avvio dei nuovi flussi e la configurazione con le policy di cancellazione dei vecchi risultati
- il **Conductor Service** per avviare nuovi flussi di scansioni dei siti, per prelevare la lista dei flussi terminati e per cancellare dal Conductor i flussi più vecchi
- il **Result Service** per cancellare i risultati di validazione più vecchi

- il **Result Aggregator Service** per cancellare i risultati di validazione aggregati più vecchi

L'indirizzo di questi servizi è da configurabile nel file application.properties oppure tramite variabili d'ambiente se avviato tramite Docker.

3.6 Rule Service

Rule Service è il componente che definisce ed implementa le regole relative al D.Lgs. n. 33-2013 sulla trasparenza nella PA.

Fornisce l'albero delle regole definito in [application.yaml](#) oppure all'interno del *Config Service*, che quindi può essere modificato o ampliato sia attraverso variabili di ambiente o della JVM prima di avviare il servizio, oppure aggiornando la configurazione per poi successivamente invocare l'endpoint [dell'actuator](#) per recepire le modifiche.

È possibile interagire con il servizio attraverso degli endpoint REST che permettono la consultazione dell'albero delle regole in formato **json**, e la verifica di una o più regole su un contenuto **html**

3.6.1 Docker

Il servizio è dotato del plugin **jib** che permette di effettuare la **build** tramite [gradle](#) e fornisce l'immagine per l'esecuzione tramite **docker**

```
./gradlew jibDockerBuild  
docker run -p 8080:8080 -ti rule-service:{version}
```

La documentazione relativa ai servizi REST è consultabile [qui](#) ed è possibile interagire con i servizi attraverso **Swagger** alla seguente [URL](#), inoltre è possibile visualizzare l'albero delle regole in formato **json**

Oppure verificare la regola **root** ad esempio attraverso una **cURL** con un **html** di esempio:

```
curl -X POST http://localhost:8080/v1/rules -H 'Content-type:application/json' --data  
→'PGh0bWw+CiAgICA8aGVhZD4KICAgICAgICA8dG10bGU+R2VuZXJpY2EgQW1taW5pc3RyYXppb25lPC90aXRsZT4KICAgIDwvaGVh  
→' | jq .
```

In alternativa scaricare il contenuto del Sito istituzionale di una Pubblica Amministrazione

```
curl "https://www.anticorruzione.it" | base64 > base64.html  
curl -X POST http://localhost:8080/v1/rules -H 'Content-type:application/json' --data  
→@base64.html | jq .
```

La risposta **json** del servizio:

```
{  
  "url": "https://www.anticorruzione.it/amministrazione-trasparente",  
  "ruleName": "amministrazione-trasparente",  
  "term": "Amministrazione Trasparente",  
  "content": "Amministrazione Trasparente",  
  "where": "text",  
  "leaf": false,  
  "status": 200,  
  "score": 4.3884144  
}
```

3.7 Conductor service

Il coordinatore è basato su [Netflix - Conductor](#) una piattaforma gratuita e open source per l'orchestrazione dei microservizi, attraverso flussi di lavoro che definiscono le interazioni tra servizi, il progetto principale è stato [forkato](#) per permettere e gestire l'autenticazione e l'autorizzazione sull'esecuzione dei flussi e sul passaggio della stessa ai microservizi invocati dal flusso.

Nel progetto [workflow-definition](#) sono presenti le definizioni in formato *json* dei flussi necessari al completamento degli obiettivi del progetto.

3.7.1 Flusso principale - Amministrazione Trasparente

Importante: Il flusso [principale](#) necessita dei seguenti parametri di input per la sua corretta invocazione come mostrato in Tabella 3.1

Table3.1: Parametri di Input per il flusso principale

Nome	Descrizione	Valore consigliato/default	Può essere Vuoto?
page_size	Dimensione della pagina per il recupero delle PA	2000	No
parent_workflow_id	Identificativo del flusso, viene valorizzato con UUID generato	vuoto	Si
codice_categoria	Se valorizzato filtra le PA che fanno parte della categoria	vuoto	Si
codice_ipa	Se valorizzato individua la singola PA	vuoto	Si
crawling_mode	Modalità base di esecuzione del crawler può assumere i valori (httpStream/htmlSource)	httpStream	Si
crawler_save_object	Booleano indica se salvare sempre la pagina HTML	false	No
crawler_save_scre	Booleano indica se salvare sempre lo screenshot della pagina	false	No
rule_name	Nome della regola	amministrazione-trasparente	No
root_rule	Nome della regola di base dell'albero	amministrazione-trasparente	No
execute_child	Booleano indica se controllare le regole figlie	true	No
id_ipa_from	Identificativo numerico della PA da cui partire	0	No
connection_timeout	Timeout in millisecondi della connessione	60000	No
read_timeout	Timeout in millisecondi della lettura	60000	No
connection_timeout_max	Timeout in millisecondi della connessione	120000	No
read_timeout_max	Timeout in millisecondi della lettura	120000	No
crawler_child_type	Modalità di esecuzione dei flussi figli (SUB/START)_WORKFLOW	START_WORKFLOW	No
rule_base_url	URL di base del microservizio delle Regole	URL	No
public_company_b	URL di base del microservizio delle PA	URL	No
result_aggregator_url	URL di base del microservizio Aggregato	URL	No
result_base_url	URL di base del microservizio dei Risultati	URL	No
crawler_uri	URL di base del microservizio Crawler	URL	No

3.7.2 Dettagli del flusso principale

Il primo **TASK** del flusso si occupa di invocare l'aggiornamento della configurazione del microservizio delle regole, dopo aver valorizzato la variabile necessaria al controllo delle pagine elaborate, il flusso invoca il **microservizio delle PA** descritto in *Public Sites Service* e recupera le informazioni necessarie.

Il blocco recuperato contenente le informazioni di n PA viene parcellizzato in base al parametro fornito in input **page_size** e diviso per 10 , utilizzando infine il **TASK FORK/JOIN** vengono eseguiti in parallelo 10 istanze del flusso **Rule** valorizzando il parametro in input **companies**.

All'uscita del **TASK delle PA**, se il flusso è stato eseguito non per una singola PA, allora vengono rielaborati i risultati con i codici **400** e **407** con i timeout massimi ed eseguiti i flussi **Crawler Result Failed**.

Infine viene eseguito il **TASK** per elaborare la Mappa geolocalizzata dei risultati.

In Fig. 3.1 l'immagine del flusso:

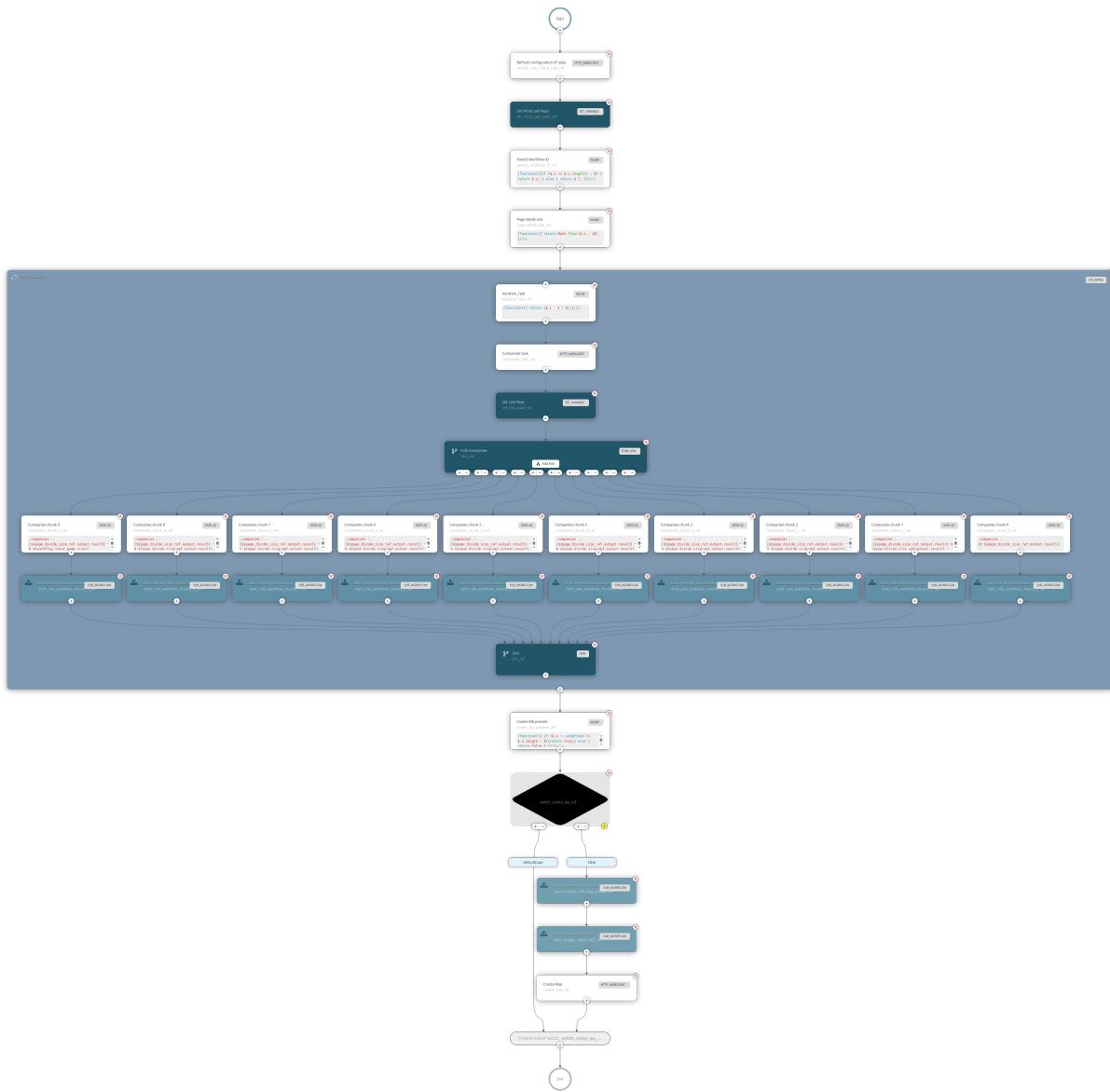
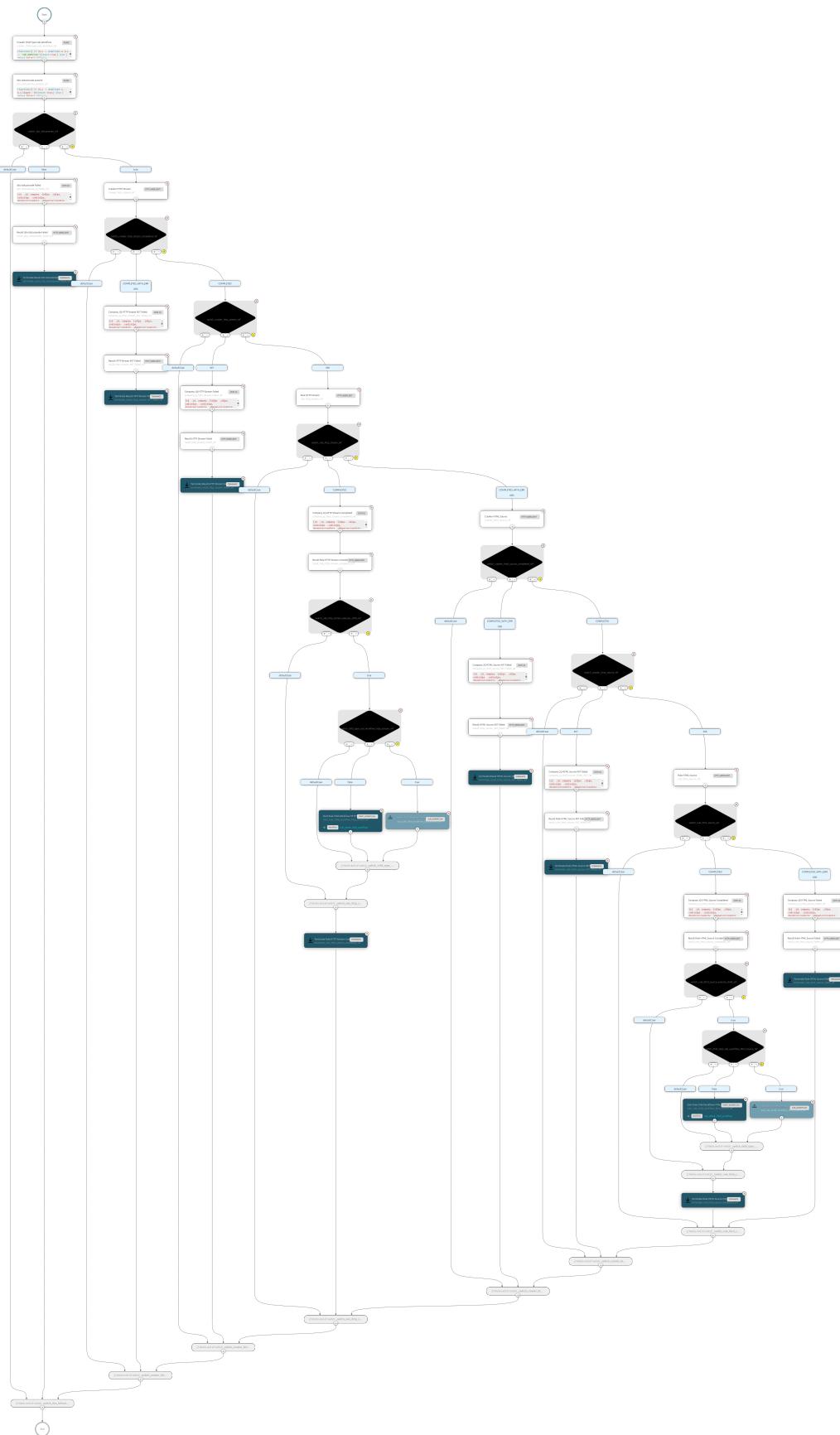


Figure3.1: Flusso principale - Amministrazione Trasparente

3.7.3 Flusso per singola Amministrazione

Il flusso [Rule Detail](#) come mostrato in [Fig. 3.2](#) viene eseguito per una singola PA passata come parametro in input ipa, controlla inizialmente la [presenza della URL istituzionale](#) e successivamente invoca il [crawler](#) il cui risultato viene passato al [microservizio delle regole](#) la cui risposta è utilizzata come input al Task dei risultati



CHAPTER 4

Risultati ottenuti

[TODO]

CHAPTER 5

Installazione e configurazione

Prima di cominciare assicurati di avere a disposizione sufficiente risorse di CPU, memoria RAM e spazio disco per l'utilizzo desiderato della piattaforma. Controlla la sezione [Risorse hardware consigliate](#) per verificare il dimensionamento necessario in produzione.

La modalità consigliata di installazione è tramite **Docker**, assicurati che su ogni server sui cui effettuare il deploy dei componenti dell'architettura sia installato sia [Docker](#), che [Docker Compose](#).

La piattaforma è composta da diversi servizi e microservizi, esposti solitamente tramite interfacce REST via HTTP/HTTPS. L'esposizione dei servizi/microservizi pubblici tramite protocollo cifrato HTTPS è fortemente consigliata, è possibile utilizzare a questo scopo uno dei vari proxy http/https disponibili. Per la piattaforma fornita in staging è stato utilizzato [Traefik](#).

5.1 Autenticazione

La piattaforma necessita di un Identity Provider OAuth2 per l'autenticazione e autorizzazione nell'accesso ai componenti dell'architettura.

Nell'ambiente di staging è stato utilizzato Keycloak come Identity Provider ma un qualunque IDP compatibile OAuth2 può andare bene.

Per configurare i vari componenti è necessario procurarsi l'endpoint per ottenere il token jwt e l'endpoint contenente i certificati pubblici del IDP, per esempio:

```
- jwt.issuer-uri -> https://dica33.ba.cnr.it/keycloak/realm/trasparenza
- jwt.jwk-set-uri -> https://dica33.ba.cnr.it/keycloak/realm/trasparenza/protocol/
  openid-connect/certs
```

Sarà necessario impostare questi due parametri nei vari microservizi, come spiegato nel seguito.

Ci sono due tipologie di accesso ai servizi della piattaforma, quello degli utenti (le persone fisiche) e quello dei client (i vari componenti si autenticano se devono comunicare tra di loro).

Per quanto riguarda i client è necessario creare tre *Service Account* di tipo *OpenId Connect* e autenticazione di tipo **client_credentials**, i tre client_id devono essere:

- crawler
- result-aggregator
- task-scheduler

I valori dei rispettivi *client secret* dovrà essere impostato nei microservizi *crawler-service*, *result-aggregator-service* e *task-scheduler-service*.

A questi tre service account deve inoltre essere assegnato un *Service Account Role* di tipo **ROLE_SUPERUSER**.

È inoltre necessario creare un client, sempre di tipo *OpenId Connect*, per l'interfaccia Web Angular JS, il client si deve chiamare **angular-public** e deve avere impostato come **valid redirect url** il valore https://www.trasparenzai.it/*.

5.2 Autorizzazione

L'accesso all'interfaccia web è condizionato dalla presenza o meno di determinati ruoli nel token JWT fornito dal sistema di autenticazione. I ruoli attualmente previsti sono:

- ROLE_USER
- ROLE_ADMIN
- ROLE_SUPERUSER

Le funzionalità mostrate nell'interfaccia web cambiano in funzione del ruolo dell'utente, è quindi necessario attribuire nel Identity Provider OAuth2 il ruolo desiderato ai propri utenti.

5.3 Config service

Config Service è il componente che si occupa di archiviare e distribuire alcune informazioni di configurazione dei servizi che compongono lo stack del progetto TrasparenzAI.

Config Service mantiene nel proprio datastore locale le configurazioni che sono fornite agli altri microservizi.

Il codice sorgente è disponibile su github:

- <https://github.com/cnr-anac/config-service>

Nel repository github è compreso anche un script per la prima installazione del servizio first-setup.sh.

In particolare è necessario configurare la sezione della sicurezza.

5.3.1 Sicurezza

L'accesso in lettura alla configurazione di tipo Spring Cloud Config disponibile al path /config è protetto con autenticazione di tipo Basic Auth, l'utente e la password possono essere indicati nel docker-compose.yml come nell'esempio seguente:

```
- spring.security.user.name=config-service-user  
- spring.security.user.password=PASSWORD_DA_IMPOSTARE_E_CONDIVIDERE_CON_I_CLIENT
```

Invece gli endpoint REST di questo servizio disponibili al path /properties sono protetti tramite autenticazione OAuth con Bearer Token. È necessario configurare l'IDP da utilizzare per validare i token OAuth tramite le due proprietà impostabili nel docker-compose.yml come nell'esempio seguente:

- `spring.security.oauth2.resource-server.jwt.issuer-uri=https://dica33.ba.cnr.it/keycloak/realm/trasparenzai`
- `spring.security.oauth2.resource-server.jwk-set-uri=https://dica33.ba.cnr.it/keycloak/realm/trasparenzai/protocol/openid-connect/certs`

I valori dei parametri `jwt.issuer-uri` e `jwk-set-uri` sono quelli già descritti nella sezione [Autenticazione](#).

5.3.2 Configurazione di default

Il config-service viene fornito con una configurazione predefinita da personalizzare secondo le proprie esigenze.

In particolare sono presenti alcune URL degli altri microservizi che è necessario configurare secondo il proprio setup.

I dati di default possono essere modificati sia tramite l'API REST del servizio che tramite l'interfaccia web (il componente UI Service).

Nell'esempio seguente viene mostrata la configurazione predefinita modificabile direttamente tramite l'interfaccia web della piattaforma.

The screenshot shows the 'PARAMETRI' (Parameters) section of the configuration interface. It includes fields for 'Dimensione pagina' (2000), 'Nome della Regola' (AT_TO_BE_23-12-2024 - Amministrazione Traspar...), 'Modalità di esecuzione dei flussi figli' (START WORKFLOW), and 'URL del servizio delle Regole' (https://monitorai.ba.cnr.it/rule-service). There are also sections for 'Esegui regole figlie', 'Salva pagina sempre', 'Salva screenshot sempre', and various timeout settings (30000, 60000, 60000, 120000). At the bottom, there is a slider for 'Numero di controlli da conservare' set to 9, and a list of 'Controlli da conservare' with two entries: 'Controllo del 28/12/2024 21:00:17, stato COMPLETATO' and 'Controllo del 05/03/2025 19:00:00, stato COMPLETATO'. A 'Conferma' (Confirm) button is visible at the top right.

In particolare sono sicuramente da impostare:

- URL del servizio Risultati
- URL del servizio Crawler
- URL del servizio delle Regole
- URL del registro delle PA
- URL del servizio dei dati Aggregati

Nel caso si voglia modificare le tempistiche e la frequenza delle scansioni complete di tutte le PA è possibile utilizzare sempre l'interfaccia web.

Espressione per l'esecuzione di un nuovo controllo, la prossima esecuzione avverrà il 19/03/2025 alle ore 19, la successiva il 22/03/2025 alle ore 19

Ore Giorni Mesi

Ogni ora

Ogni 1 ora/e a partire dalle ore

Ora specifica (sceglierne una)

0 1 2 3 4 5 6 7 8 9 10 11
 12 13 14 15 16 17 18 19 20 21 22 23

Ogni ora tra le ore 0 e ora 0

 Conferma

5.4 Public Sites Service

Public Sites Service è il componente che si occupa di gestire le informazioni principali relative agli enti pubblici italiani ed in particolare i siti istituzionali.

Public Sites Service mantiene nel proprio datastore locale le informazioni degli enti che possono essere inserite/aggiornate tramite gli OpenData di IndicePA, oppure inserite tramite appositi servizi endpoint REST. Il servizio utilizza Nominatim di OpenStreetMap per la geolocalizzazione degli indirizzi degli enti pubblici.

Il codice sorgente è disponibile su github:

- <https://github.com/cnr-anac/public-sites-service>

Nel repository github è compreso anche un script per la prima installazione del servizio first-setup.sh.

In particolare è necessario configurare la sezione della sicurezza.

5.4.1 Sicurezza

Gli endpoint REST di questo servizio sono protetti tramite autenticazione OAuth con Bearer Token. È necessario configurare l'IDP da utilizzare per validare i token OAuth tramite le due proprietà impostabili nel docker-compose.yml come nell'esempio seguente:

```
- spring.security.oauth2.resourceserver.jwt.issuer-uri=https://dica33.ba.cnr.it/keycloak/  
  ↳realms/trasparenzai  
- spring.security.oauth2.resourceserver.jwt.jwk-set-uri=https://dica33.ba.cnr.it/  
  ↳keycloak/realms/trasparenzai/protocol/openid-connect/certs
```

I valori dei parametri *jwt.issuer-uri* e *jwk-set-uri* sono quelli già descritti nella sezione [Autenticazione](#).

5.4.2 Integrazione API Google Maps

Il servizio è già predisposto per l'integrazione con l'API di Google Maps per la geocalizzazione degli indirizzi degli enti pubblici. L'API Google Maps fornisce solitamente una migliore individuazione delle coordinate GPS degli indirizzi indicati nel IndicePA. L'API Google Maps è però a pagamento, con un freetier per un numero iniziale di ricerche, è necessario procurarsi una Google Maps Key per poter utilizzare questo servizio, la quale richiede di inserire una carta di credito per gli eventuali pagamento oltre il freetier.

L'utilizzo della API Google Maps può essere attivata nel public sites service impostando questo parametri nell'environment del docker-compose.yml:

```
- transparency.google.maps.enabled=true  
- transparency.google.maps.key=LA_CHIAVE_DA_PRELEVARE_DAI_SISTEMI_GOOGLE
```

5.5 Result Service

Result Service è il componente che si occupa di gestire i risultati delle verifiche sulla corrispondenza dei siti degli enti pubblici.

Result Service mantiene nel proprio datastore locale le informazioni relative ai risultati di validazione.

Il codice sorgente è disponibile su github:

- <https://github.com/cnr-anac/result-service>

Nel repository github è compreso anche un script per la prima installazione del servizio first-setup.sh.

In particolare è necessario configurare la sezione della sicurezza.

5.5.1 Sicurezza

Gli endpoint REST di questo servizio sono protetti tramite autenticazione OAuth con Bearer Token. È necessario configurare l'IDP da utilizzare per validare i token OAuth tramite le due proprietà impostabili nel docker-compose.yml come nell'esempio seguente:

```
- spring.security.oauth2.resourceserver.jwt.issuer-uri=https://dica33.ba.cnr.it/keycloak/  
  ↵realms/trasparenzai  
- spring.security.oauth2.resourceserver.jwt.jwk-set-uri=https://dica33.ba.cnr.it/  
  ↵keycloak/realms/trasparenzai/protocol/openid-connect/certs
```

I valori dei parametri *jwt.issuer-uri* e *jwk-set-uri* sono quelli già descritti nella sezione [Autenticazione](#).

5.6 Result Aggregator Service

Result Aggregator Service è il componente che si occupa di gestire i risultati delle verifiche sulla corrispondenza, aggregando i risultati di validazione con altre informazioni sugli enti pubblici prelevate da altri servizi.

Result Aggregator Service mantiene nel proprio datastore locale le informazioni relative ai risultati di validazione.

Il codice sorgente è disponibile su github:

- <https://github.com/cnr-anac/result-aggregator-service>

Nel repository github è compreso anche un script per la prima installazione del servizio first-setup.sh.

Questo servizio ha due dipendenze per funzionare:

- il *Result Service* da cui leggere le info sulle verifiche
- il *Public Sites Service* da cui prelevare le info geografiche delle PA

Attenzione: se il public-site-service o il result-service non sono avviati sullo stesso server tramite docker è necessario configurare l'url a cui rispondono, modificando nel .env le variabili d'ambiente *TRANSPARENCY_PUBLIC_SITE_URL* e *TRANSPARENCY_RESULT_SERVICE_URL*.

Per esempio nel .env:

```
TRANSPARENCY_PUBLIC_SITE_URL=https://dica33.ba.cnr.it/public-sites-service  
TRANSPARENCY_RESULT_SERVICE_URL=https://dica33.ba.cnr.it/result-service
```

Per configurare il client REST che accede a questi due servizi è necessario configurare nel .env il parametro **OIDC_CLIENT_SECRET**, impostando il valore generato quando si è creato il *Service Account result-aggregator*, vedi [Autenticazione](#).

Inoltre è necessario configurare la sezione della sicurezza.

5.6.1 Sicurezza

Gli endpoint REST di questo servizio sono protetti tramite autenticazione OAuth con Bearer Token. È necessario configurare l'IDP da utilizzare per validare i token OAuth tramite le due proprietà impostabili nel docker-compose.yml come nell'esempio seguente:

```
- spring.security.oauth2.resourceserver.jwt.issuer-uri=https://dica33.ba.cnr.it/keycloak/  
  ↵realms/trasparenzai  
- spring.security.oauth2.resourceserver.jwt.jwk-set-uri=https://dica33.ba.cnr.it/  
  ↵keycloak/realms/trasparenzai/protocol/openid-connect/certs
```

I valori dei parametri *jwt.issuer-uri* e *jwk-set-uri* sono quelli già descritti nella sezione [Autenticazione](#).

5.7 Task Scheduler Service

Task Scheduler Service è il componente che si occupa di avviare alcuni processi eseguiti a intervalli fissi, come per esempio l'**avvio delle scansioni** dei siti del PA per la verifica della corrispondenza dei requisiti e la **cancellazione dei risultati di scansione più vecchi**.

Nell'utilizzo tramite **docker-compose.yml** ricordarsi di impostare nel .env la corretta variabile d'ambiente che specifica l'url del config-service da utilizzare e la password per l'autenticazione Basic Auth con il *config-service*:

environment:

- confighost=\${CONFIG_HOST}
- spring.security.oauth2.client.registration.oidc.client-secret=\${OIDC_CLIENT_SECRET}

Vedi [Config service](#).

Il codice sorgente è disponibile su github:

- <https://github.com/cnr-anac/task-scheduler-service>

Nel repository github è compreso anche un script per la prima installazione del servizio first-setup.sh.

Questo servizio ha tre dipendenze per funzionare:

- il conductor-service tramite cui avviare i flussi di verifica e la cancellazione dei vecchi workflow
- il *Result Service* da cui cancellare i vecchi workflow
- il *Result Aggregator Service* da cui cancellare i vecchi workflow

La configurazione del **conductor-service** per avviare i nuovi flussi e cancellarli quelli vecchi viene letta automaticamente dal [Config service](#).

Attenzione: è invece importante impostare nel .env le URL dei servizi *result-service* e *result-aggregator-service* modificando le variabili d'ambiente *TRANSPARENCY_RESULT_SERVICE_URL* e *TRANSPARENCY_RESULT_AGGREGATOR_SERVICE_URL*:

```
# Configurazione indirizzi dei servizi dove cancellare i risultati del workflow scaduti
- transparency.clients.result-service.url=${TRANSPARENCY_RESULT_SERVICE_URL}
- transparency.clients.result-aggregator-service.url=${TRANSPARENCY_RESULT_AGGREGATOR_
SERVICE_URL}
```

Per configurare il client REST che accede a questi due servizi è necessario configurare nel .env il parametro **OIDC_CLIENT_SECRET**, impostando il valore generato quando si è creato il *Service Account task-scheduler*, vedi [Autenticazione](#).

```
# Client Secret da generare nel Identity Provider e impostare qui
- spring.security.oauth2.client.registration.oidc.client-secret=${OIDC_CLIENT_SECRET}
```

Inoltre è necessario configurare la sezione della sicurezza.

5.7.1 Sicurezza

Gli endpoint REST di questo servizio sono protetti tramite autenticazione OAuth con Bearer Token. È necessario configurare l'IDP da utilizzare per validare i token OAuth tramite le due proprietà impostabili nel docker-compose.yml come nell'esempio seguente:

```
- spring.security.oauth2.resourceserver.jwt.issuer-uri=https://dica33.ba.cnr.it/keycloak/
  realms/trasparenzai
- spring.security.oauth2.resourceserver.jwt.jwk-set-uri=https://dica33.ba.cnr.it/
  keycloak/realms/trasparenzai/protocol/openid-connect/certs
```

I valori dei parametri *jwt.issuer-uri* e *jwk-set-uri* sono quelli già descritti nella sezione [Autenticazione](#).

5.8 Risorse hardware consigliate

I test di funzionamento in produzione hanno evidenziato la necessità di suddividere su almeno 3 distinti virtual machine l'architettura del sistema.

In particolare è consigliato di mantenere separata la parte del crawler, dalla parte del coordinamento (conductor-service), dalla parte di gestione dei risultati e loro visualizzazione via Web.

[TODO].

CHAPTER 6

Appendice

6.1 Autori

Autore del codice: Ivan Duca <ivan.duca@cnr.it>

Autore del codice: Dario Elia <dario.eman@gmail.com>

Autore del codice: Claudia Greco <claudia.greco@cnr.it>

Autore del codice: Massimo Ianigro <massimo.ianigro@cnr.it>

Autore del codice: Cristian Lucchesi <cristian.lucchesi@cnr.it>

Autore del codice: Marco Spasiano <marco.spasiano@cnr.it>