MINISTERO DELLA CULTURA

ICDP – DL

PNRR - M1 C3 “Cultura e Turismo 4.0”
Investimento 1.1 - “Digital Strategy and Platforms for Cultural Heritage”
Sub-investimento 1.1.10 - “Piattaforme di accesso integrato alla Digital Library”

CONTRATTO ESECUTIVO DELL’ACCORDO QUADRO CONSIP ID 2102
PER L’AFFIDAMENTO DI
“SERVIZI APPLICATIVI DI DATA MANAGEMENT E SERVIZI DI PMO
PER LE PUBBLICHE AMMINISTRAZIONI” - ID 2102
LOTTO 1 - “SERVIZI APPLICATIVI DI DATA MANAGEMENT PER LE PAC”

DPaC - Piattaforma per la gestione della campagna di digitalizzazione: documentazione tecnica"

Cliente:	Ministero della Cultura
Progetto:	Digital Strategy and Platforms for Cultural Heritage
Responsabile Progetto:	
Redatto da:	
Verificato da:	
Data:	08/11/2023
Versione:	1.0
Nome documento:	

VERSIONI

Ver.	Motivo	Data	Redatto da	Approvato da
1.0	Prima emissione	08/11/2023		
1.1	Correzione formato	15/11/2023		

Sommario

1 DPaC

Nell'ambito del sub-investimento M1C3 1.1/4 - "Infrastruttura digitale per il patrimonio culturale" è stata prevista la realizzazione di una piattaforma (DPaC) da mettere a disposizione degli istituti coinvolti nel piano nazionale di digitalizzazione del patrimonio culturale a supporto dell'acquisizione, della gestione della qualità e della valorizzazione dei dati che saranno prodotti dagli stessi.

La piattaforma DPaC rappresenta un'estensione funzionale all'ISPC: essa cioè ha il compito di raccogliere i prodotti che verranno realizzati nei vari cantieri durante la campagna di digitalizzazione e consegnarli a ISPC tramite un'apposita procedura di *ingestion*.

1.1 Scopo e contenuto di questo manuale

Lo scopo di questo manuale è quello di contenere tutte le informazioni utili e necessarie ad organizzare, progettare, pianificare, eseguire e gestire il complesso processo di manutenzione.
I contenuti del manuale sono strettamente dipendenti dal suo scopo.

2 Architettura

La piattaforma DPaC è basata su un'architettura ibrida composta da micro-servizi orchestrati attraverso il cluster OpenShift di RedHat e virtual machines.

- Lo IAM **WSO2** si occupa dell'autenticazione dell'utente tramite SPID oppure LDAP del MiC;
- Si utilizza un API Manager **RH3Scale** per l'autorizzazione della sessione utente;
- L'antivirus risiede su una virtual machine (VM) dedicata e si occupa della scansione dei pacchetti che arrivano in piattaforma tramite protocollo ICAP.
- Il Quality Assurance tramite ML prevede una serie di VMs che si distribuiranno il carico di lavoro, ognuna specializzata e addestrata con una rete neurale che risponde ad una definita tipologia di immagini prodotte dal cantiere;
- L'area dati è composta da:
    a. Un **RDBMS Oracle** che si occuperà di gestire tutti i metadati della piattaforma e il repository del BPMN Camunda
    b. Un **Object Storage S3** che si occuperà di memorizzare tutti i pacchetti di contenuto (immagini più file a contorno) dei vari cantieri organizzati con logica Multi tenant
    c. Un'istanza **MySql** per i dati del modulo descrittivo
- Sia il FrontEnd che il BackEnd della piattaforma sono composti da un insieme di micro-servizi orchestrati da **OpenShift**. Autonomi ed indipendenti tra di loro, possono scalare orizzontalmente all'occorrenza, ovvero quando aumentano le richieste ai servizi da parte degli utenti.
  Tutti i micro-servizi accedono ad una base di dati comune e possono comunicare tra di loro attraverso API.
- Il Message Broker rappresentato da **RabbitMQ** è un middleware che utilizza un protocollo di rete AMQP (Advanced Message Queuing Protocol) adatto per lo scambio di messaggi. Gestisce l'invio e la ricezione dei messaggi dai vari client detti producer (produttori di messaggi) o consumer (destinatari dei messaggi).
  Viene utilizzato sia per la comunicazione asincrona tra il modulo di caricamento dei pacchetti attraverso Axway, sia per la comunicazione con le VMs per il ML.
- Ed infine il **Redis** utilizzato dal front end per il salvataggio dei dati di sessione.
  Redis è un DBMS NoSQL composto da un valore abbinato ad una chiave univoca che permette di recuperarlo velocemente.
  Redis memorizza i dati in memoria salvandoli in maniera persistente solo quando necessario.

3 Micro-servizi

Di seguito verranno dettagliate le funzionalità di ogni singolo micro-servizio che compone la piattaforma DPaC.

3.1 DLWEB (Front-End DPaC)

Il micro-servizio "DLWEB" è costituito da un web server Nginx ed ospita la Single Page Application DPaC scritta in Angular.
Rappresenta l'interfaccia dove l'utente accede per poter svolgere l'intero processo di lavorazione a partire dalla configurazione base del cantiere fino all'*ingestion* di un pacchetto di contenuto passando per la pianificazione ed il collaudo delle risorse digitali, mostrando in tempo reale lo stato di avanzamento dei vari processi.

3.1.1 Configurazione

La configurazione del servizio utilizza le variabili di ambiente affinché possa essere eseguita indipendentemente dall'ambiente di utilizzo (dev, preprod, prod).
Di seguito un esempio di impostazione:

Puntamenti ai vari micro-servizi e corrispondenze:

- SERVICE_END_POINT_PLAN: micdl-plan-server
- SERVICES_END_POINT: micdl-server
- SERVICES_END_POINT_REPORT: micdl-report-server
- SERVICES_END_POINT_COLLAUDO: micdl-collaudo
- SERVICES_END_POINT_UPLOAD: micdl-upload
- DOCUMENTARY_END_POINT: micdl-documentale-server
- SERVICE_CAMUNDA: camunda-app
- NOTIFICHE_END_POINT: micdl-nofiche-server
- STORAGE_END_POINT: relativo allo storage s3 delle immagini
- METAFAD_END_POINT: relativo al micro-servizio metafad
- POWERBI_END_POINT: relativo al micro-servizio (MICDL-Connector) che espone le informazioni per il caricamento dei report

Ticket e Issue:

- ticketStatus: contiene gli id degli stati assunti da un ticket o da una issue

Cantiere:

- cantiereStatusDescriptionList: contiene i possibili stati di un cantiere in formato stringa
- cantiereStatusList: contiene gli id dei possibili stati di un cantiere

Ruoli:
- roles: contiene gli id dei diversi ruoli sulla piattaforma

Template:
- templateTypesList: elenco degli enum di tipologia dei template
- templatePM: elenco degli enum di tipologia dei template visibili dal PM
- templateBM: elenco degli enum di tipologia dei template visibili dal BM
- templatesSorting: elenco ordinato dei template
- istanzaTemplatesStatuses: elenco degli id corrispondenti allo stato che un'istanza template può assumere
- templatesFacsimile: elenco dei documenti scaricabili in facsimile

Camunda:
- camundaAutoTask: elenco di task restituite da camunda che vengono completate tramite azioni automatiche nel flusso della piattaforma
- camundaApprove: elenco di task di camunda che richiedono approvazione/rigetto
- templateCamundaTask: associazioni tra i task di camunda e i template
- camundaLotti: elenco dei task di camunda afferenti i lotti
- camundaWorkflow: elenco del workflow di camunda

Lotti:
- lottiTypes: elenco tipologia lotti
- esitiRisorsa: stati che una risorsa può assumere
- statiPacchetto: id stati che può assumere un pacchetto

Storage S3 documentale:
- s3DocStatus: stati che un documento pubblicato può assumere

Powerbi:
- powerbiReports: contiene il workspace e tutti i report id disponibili

3.1.2 Dipendenze/Librerie utilizzate

Librerie utilizzate:
- Angular 14: framework di sviluppo
- Syncfusion: richiamata per il componente gantt, i grafici (torta e istogramma), date e time picker
- powerbi-client-angular: integrazione dei report powerbi per consentire l'apertura da piattaforma
- Bootstrap v5

- Transloco: libreria di traduzione
- @ng-bootstrap/ng-bootstrap: libreria di utilità per componentistica Angular con stile Bootstrap
- @ng-select/ng-select: componente all-in-one select, multiselect e autocomplete
- angular-oauth2-oidc: libreria di supporto per OAuth 2 e OpenId Connect
- ngx-toastr: generazione dei toasters informativi
- papaparse: generazione dei files csv

3.1.3 Deploy

Per effettuare il deploy della piattaforma Angular è necessario aver installato OpenShift CLI, successivamente seguire i seguenti step:

STEP 1: Build
All'interno del progetto eseguire il comando *ng build -c <ambiente in cui si vuole fare il deploy>*
Gli ambienti sono:

- *dev* per l'ambiente di sviluppo
- *preprod* per l'ambiente di collaudo
- *prod* per l'ambiente di produzione

STEP 2: Login e selezione ambiente di progetto
Attraverso il cmd prompt accedere alla location del file oc ed eseguire il comando di login (disponibile dalla tendina delle informazioni personali di OpenShift)
oc login […]
Una volta effettuato il login, selezionare il progetto dell'ambiente desiderato tramite *oc project <nome del progetto>*

- *micdlocp-dev* per l'ambiente di sviluppo
- *micdlocp-preprod* per l'ambiente di collaudo
- *micdlocp-prod* per l'ambiente di produzione

STEP 3: Accesso al pod e rimozione file precedenti
Individuare all'interno della topologia dell'ambiente il pod dlweb, eseguire poi il comando:
oc rsh <id del pod dlweb>
Una volta effettuato l'accesso eseguire:
bash
E successivamente *cd /usr/share/nginx/dlweb*

A questo punto, prima di effettuare la copia della build in locale sul pod, possiamo rimuovere la precedente versione del codice, inserendola in una cartella nominata con la data del giorno (yyyy-mm-dd) all'interno delle cartelle di backup (BK o BU) altrimenti eliminandoli direttamente.

Per facilitare il processo si consiglia l'utilizzo di MC Midnight Commander, richiamabile direttamente nel pod con il comando appunto *mc.* Se non dovesse essere installato si potrebbe usare il comando: *apt-get update && apt-get install mc*

Una volta terminato e, eventualmente, chiuso MC, per uscire dal pod digitare due volte *exit*

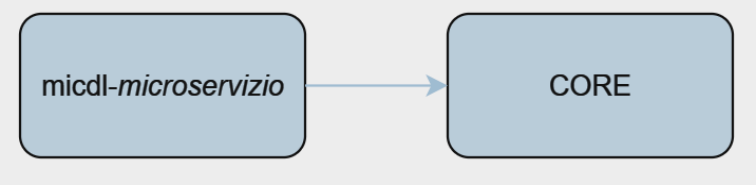
STEP 4: Rilascio

Una volta rimossi o spostati in una cartella di backup i file precedenti e la build di Angular è completa, è possibile effettuare la copia della build in locale sul pod:
oc rsync <path della build nella cartella dist del progetto> <id del pod>: /usr/share/nginx/dlweb

3.2 MICDL-CORE

Micro-servizio che gestisce la sessione dell'utente autenticato.

3.2.1 Funzionalità



Tutti i micro-servizi che interagiscono con il micro-servizio "MICDL-CORE" utilizzano il Jason Web Token affinché possano essere riconosciuti.
Le funzionalità di questo micro-servizio sono esposte dalle API nel paragrafo successivo. Ogni micro-servizio può, attraverso il micro-servizio "MICDL-CORE", avere informazioni sull'utente connesso: cantiere in sessione e ruolo dell'utente afferente al cantiere.

3.2.2 API

Comandi	Query	Eventi
/session/setCantiereId		Viene fatto il set del cantiere in sessione
/session/setRuoloCorrenteUtente		Viene fatto il set del ruolo
/session/getUserInfo		Restituisce le informazioni dell'utente connesso

3.2.3 Dipendenze/Librerie utilizzate

- spring-boot-starter
- spring-boot-starter-test
- spring-boot-starter-oauth2-resource-server

- Lombok
- spring-boot-starter-data-jpa
- spring-session-core
- spring-boot-starter-data-rest

3.2.4 Deploy

Partendo dal CORE si va ad eseguire la build dell'applicazione attraverso il comando:
maven clean, install, -DSkipTests

3.3 MICDL-SERVER

Questo micro-servizio gestisce le informazioni e i servizi relativi agli utenti, cantieri, ticket e ODA.

3.3.1 Funzionalità

Utente e utente cantiere:

- Il micro-servizio gestisce le informazioni delle anagrafiche degli utenti e le associazioni con i cantieri con i rispettivi ruoli.

Cantiere:

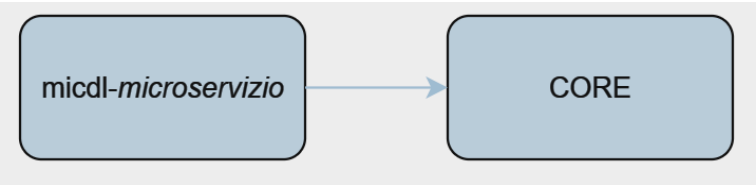
- MICDL-SERVER, inoltre, si occupa della gestione dei cantieri visibili agli utenti, dello stato dei cantieri *(aperto, attivo, chiuso, disattivo)* e delle relative funzionalità.
- Attraverso MICDL-SERVER andiamo a settare i moduli cantiere e la pianificazione.

Ticket e Segnalazioni:

- MICDL-SERVER gestisce tutto ciò che concerne i ticket tra (ROP e PM) e le issues di *casi aperti.*
- Gestiste la tipologia (*amministrativo* o *tecnico*) e, nel caso delle segnalazioni, la tipologia di segnalazione
- Coordina gli invii e lo stato *letto* e *non letto*

ODA:

- Tutte le funzionalità dell'ODA sono gestite all'interno di questo micro-servizio.
- Creazione, approvazione, cancellazione, modifica di un ODA e del suo stato
- Creazione statica dell'id ODA in base al Cluster di riferimento e all'accordo quadro



Come tutti i micro-servizi, il **micro-servizio "MICDL-SERVER"** passa attraverso il **micro-servizio "MICDL-CORE"** per ottenere le informazioni del cantiere in sessione e attraverso la **getUserInfo** che restituisce le informazioni dell'utente connesso. Inoltre è il **micro-servizio "MICDL-SERVER"** stesso a mettere direttamente in sessione il cantiere e l'utente.

3.3.2 API

Comandi	Tipo Metodo	Query	Eventi
/utenteCantieres/search/getRuoloAuthenticated? projection=RuoloProj	GET	//	Restituisce un set di Ruolo
/moduloCantieres/search/findByCantiereIdAndModuloId? cantiereId={cantiereId}&moduloId={moduloId}	GET	cantiereId moduloId	Restituisce un oggetto in base all'id del cantiere e del modulo
/utentes/search/findByCodiceFiscaleIgnoreCase? CodiceFiscale={codiceFiscale}	GET	codiceFiscale	Restituisce un oggetto Utente in base al codice fiscale in input
/gestioneTickets/search/listRootTicket	GET	cantiereId-->Param statoTicketId--> Param ruolo-->Param ruoloDest-->Param dataAggiornamento -->Param	Restituisce una Page di GestioneTicket in base ai parametri di input
/ruoloes/search/getAllRuoliWithoutRop	GET	//	Restituisce una lista di Ruoli senza il ROP
/utentiCantieri/aggiungiUtenteCantiereRuolo? idUtenteCantiere=idUtenteCantiere &ruolo=ruolo	PUT	idUtenteCantiere -->RequestParam ruolo-->RequestParam	Setta sulla UtenteCantiereRuolo l'fk UtenteCantiere e l'fk Ruolo
/utentiCantieri/saveUtenteCantiere/{idUtenteCantiere}	PATCH	idUtenteCantiere -->PathVariable utenteCantiere -->RequestBody	Salva un nuovo oggetto UtenteCantiere
/cantieres/search/searchByNomeOrCod? pageparam&filter={filtro}	GET	filter-->Param Pageable p	Restituisce una Page di cantieri che possono essere filtrati per nome
/cantieres/search/getCantieriByRole? idRuolo={idRuolo} &filter={filtro}	GET	idRuolo-->Param filter-->Param Pageable p	Restituisce una page di cantieri in base al ruolo, che possono essere filtrati per nome
/notifiche/countNotificheNonLette	GET	//	Restituisce la differenza tra le notifiche totali del cantiere e quelle lette
/gestioneTickets/search/getCasiApertiByCantiere		//	Restituisce la count dei casi aperti del cantiere preso dalla sessione
/gestioneTickets/search/countAllTicketsNonLettiByCantiere	GET	//	Restituisce la count dei tickets non letti del cantiere preso dalla sessione
/clusters/search/getClusterByIdRuolo? idRuolo={idRuolo}&idUtente={idUtente}	GET	idRuolo idUtente codCluster pageable	Restituisce una Page di Cluster in base ai parametri di input

/clusters/search/getAllClusterByIdLottoGeografico? idLottoGeografico={idLottoGeografico}	GET	idLottoGeografico	Restituisce una lista di Cluster per lotto geografico
/clusters/search/getLottoGeograficoByClusterCode? clusterCode={clusterCode}	GET	clusterCode	Restituisce la descrizione del lotto geografico in base al clusterCode
/clusters/search/getProceduraGaraByClusterSelezionato? clusterCode={clusterCode}	GET	clusterCode	Restituisce la procedura di gara in base al clusterCode
/clusters/search/getOperatoreEconomicoByClusterCode? clusterCode={clusterCode}	GET	clusterCode	Restituisce l'operatore economico in base al clusterCode
/clusters/search/getImpOperativoComplessivoInApprovazione? clusterCode=${clusterCode}	GET	clusterCode	Restituisce la somma di tutti gli importi operativi in approvazione in base al clusterCode
/clusters/search/getImpOpzionaleComplessivoInApprovazione? idLottoGeografico={idLottoGeografico}	GET	idLottoGeografico	Restituisce la somma di tutti gli importi opzionali in approvazione in base al lotto geografico
/clusters/search/getImpBudgetCluster? clusterCode={clusterCode}	GET	clusterCode	Restituisce il budget in base al cluster code
/lottoGeograficoes/search/getImpBudgetBorsellino? idLottoGeografico={idLottoGeografico}	GET	idLottoGeografico	Restituisce il budget in base al lotto geografico
/clusters/search/getImpOperativoComplessivoApprovato? clusterCode={clusterCode}	GET	clusterCode	Restituisce la somma di tutti gli importi operativi approvati in base al clusterCode
/clusters/search/getImpOpzionaleComplessivoApprovato? idLottoGeografico={idLottoGeografico}	GET	idLottoGeografico	Restituisce la somma di tutti gli importi opzionali approvati in base al lotto geografico
/oda/getBudgetResiduoCluster?clusterCode={clusterCode}	GET	clusterCode	Restituisce la differenza tra il budget iniziale, il budget ordinato e il budget impegnato in base al cluster code
/oda/getBudgetResiduoBorsellinoLotto? clusterCode={clusterCode}&idLottoGeografico={idLottoGeografico}	GET	idLottoGeografico	Restituisce la differenza tra il budget iniziale, il budget ordinato e il budget impegnato in base al lotto geografico
/clusters/search/getDescNote?codCodiceOda={codCodiceOda}	GET	codCodiceOda	Restituisce le possibili note associate ad un Istanza Oda
/oda/getBudgetResiduoClusterInApprovazione? clusterCode={clusterCode}&codCodiceOda={codCodiceOda}	GET	clusterCode codCodiceOda	Restituisce la differenza tra il budget residuo e l'importo operativo complessivo

/oda/getBudgetResiduoBorsellinoLottoInApprovazione? clusterCode={clusterCode}& idLottoGeografico={idLottoGeografico}& codCodiceOda={codCodiceOda}	GET	clusterCode idLottoGeografico codCodiceOda	Restituisce la differenza tra il budget residuo di borsellino lotto e l'importo opzionale complessivo
/oda/getPrestazioniCollegate?codCodiceOda={codCodiceOda}	GET	codCodiceOda	Restituisce una Page di Prestazioni in base al codCodiceOda
/prestazioni/search/getPrestazione? codPrestazione={codPrestazione}	GET	codPrestazione	Restituisce un oggetto Prestazione in base al suo codice
/prestazioni/search/getCodAndDenomPrestazione? descAccordoQuadro={descAccordoQuadro}	GET	codDescDenomPrestazione descAccordoQuadro pageable	Restituisce una Page di Prestazione in base ai parametri di input
/oda/deleteByPrestazione? idPrestazioneOda={idPrestazioneOda}	DELETE	idPrestazioneOda	Cancella la prestazione in base al suo id
/oda/approvaOda?codCodiceOda={codCodiceOda}	PUT	codCodiceOda	Cambia lo stato dell'istanza oda in input in approvato
/oda/revisionaOda?codCodiceOda={codCodiceOda}	PUT	codCodiceOda	Cambia lo stato dell'istanza oda in input in da revisionare
/oda/insertNote?codCodiceOda={codCodiceOda}	PUT	codCodiceOda	Inserisce le note sull'istanza oda in input
/oda/updateNumQuantitaPrestazione? idPrestazioneOda={idPrestazioneOda}& numQuantitaPrestazione={numQuantitaPrestazione}	PUT	idPrestazioneOda numQuantitaPrestazione	Aggiorna il numQuantitaPrestazione dell'oda in input
/oda/updateNumRisorseDigitaliStimate? idPrestazioneOda={idPrestazioneOda}& numRisorseDigitaliStimate={numRisorseDigitaliStimate}	PUT	idPrestazioneOda numRisorseDigitaliStimate	Aggiorna il numRisorseDigitaliStimate dell'oda in input
/gestioneTickets/search/findByParentId? parentId={rootTicketId}& projection=GestioneTicketListPrj	GET	parentId	Restituisce una lista di Ticket in base al parentId
/gestioneTickets/search/listRootTicket? cantiereId={cantiereId}&projection=GestioneTicketListPrj& sort=dataAggiornamento,{order}&ruoloMittente={ruoliMittente}& ruoloDest={ruoliDestinatari}&{pageParam}	GET	cantiereId statoTicketId ruoloMittente ruoloDest dataAggiornamento pageable	Restituisce una Page di ticket in base ai parametri di input
/utenteCantieres/search/searchByCantiereAndCF? projection=UtenteCantierePrj& cantiereId={cantiereId}	GET	cantiereId cf flagCancellato pageable	Restituisce una Page di UtenteCantiere in base ai parametri di input

3.3.3 Metriche

/actuator/metrics:

- Endpoint base di spring-boot offre una serie di metriche sull'applicazione.

```
{
        "names": [
                "application.ready.time",
                "application.started.time",
                … ,
                ]
}
```

/actuator/info:

- Endpoint base di spring-boot fornisce informazioni personalizzate sull'applicazione.
- Attualmente il servizio non restituisce alcuna informazione

{}

/actuator/health:

- Restituisce un codice di stato HTTP 200 OK se l'applicazione è sana.

```
{
        "status": "UP",
        "groups": [
                "liveness",
                "readiness"
                ]
}
```

- Il servizio restituisce 503 qualora non fosse disponibile.

3.3.4 Configurazioni

La variabile d'ambiente comune per tutti i micro-servizi nella configurazione è la seguente:

spring.profiles.active=local

Per quanto riguarda il micro-servizio "MICDL-SERVER" invece:

- JDBC_URL: Contiene l'URL di connessione al database
- DATABASE_USER: Contiene il nome utente per l'autenticazione al db
- DATABASE_PASSWORD: Contiene la password associata all'utente del db per la connessione

- ISSUER_URI: Contiene l'URI dell'emittente del token di autenticazione
- SHOW_SQL: Settata a false per disabilitare la visualizzazione delle query
- REST_TEMPLATE_HOST: Contiene l'URL dei micro-servizi con i quali il modulo di collaudo comunica
- FISCAL_CODE_KEY: Contiene l'identificatore utilizzato per accedere ai dati del codice fiscale in un contesto specifico dell'applicazione CODE_GUI_KEY: Contiene l'identificatore utilizzato per accedere ai dati dell'interfaccia grafica dell'applicazione
- USER_MAIL: Contiene l'indirizzo mail per accedere ai dati dell'utente
- BASE_PATH_REPORT: Contiene il percorso di base utilizzato per la generazione di documenti
- BASE_PATH_UPLOAD: Contiene il percorso di base utilizzato per l'upload di documenti
- BASE_PATH_DOCUMENTALE: Contiene il percorso di base utilizzato per la generazione di documenti
- REST_TEMPLATE_HOST_UPLOAD: Specifica la radice del path del micro-servizio di upload
- REST_TEMPLATE_HOST_REPORT: Specifica la radice del path del micro-servizio di report-server
- REST_TEMPLATE_HOST_DOCUMENTALE: Specifica la radice del path del micro-servizio di documentale
- TICKET_TIPO_AMMINISTRATIVO_ID: configurazione per il tipo di ticket amministrativo. Se la variabile è definita, il valore di questa configurazione sarà uguale a quello. Altrimenti, il valore predefinito sarà 2
- TICKET_TIPO_TECNICO_ID: configurazione per il tipo di ticket tecnico. Se la variabile è definita, il valore di questa configurazione sarà uguale a quello. Altrimenti, il valore predefinito sarà 1
- TICKET_STATO_APERTO_ID: configurazione per lo stato di apertura di un ticket. Se la variabile è definita, il valore di questa configurazione sarà uguale a quello. Altrimenti, il valore predefinito sarà 1
- TICKET_STATO_INLAVORAZIONE_ID: configurazione per lo stato di lavorazione di un ticket. Se la variabile è definita, il valore di questa configurazione sarà uguale a quello. Altrimenti, il valore predefinito sarà 2
- RUOLO_ROP_ACRONIMO: configurazione per il ruolo con acronimo ROP. Se la variabile è definita, il valore di questa configurazione sarà uguale a quello. Altrimenti, il valore predefinito sarà ROP
- RUOLO_PM_ACRONIMO: configurazione per il ruolo con acronimo PM. Se la variabile è definita, il valore di questa configurazione sarà uguale a quello. Altrimenti, il valore predefinito sarà PM
- RUOLO_PO_ACRONIMO: configurazione per il ruolo con acronimo PO. Se la variabile è definita, il valore di questa configurazione sarà uguale a quello. Altrimenti, il valore predefinito sarà PO
- RUOLO_BM_ACRONIMO: configurazione per il ruolo con acronimo BM. Se la variabile è definita, il valore di questa configurazione sarà uguale a quello. Altrimenti, il valore predefinito sarà BM
- MICDL_LOG: Configura il percorso dei file di log
- MICDL_CORE_LOG: Configura il percorso dei file di log per il core

3.3.5 Dipendenze/Librerie utilizzate

- spring-boot-starter-actuator
- spring-boot-starter-data-jpa
- spring-boot-starter-data-rest
- spring-session-data-redis
- spring-boot-starter-data-redis
- spring-session-core
- micdl-core
- H2
- Ojdbc8
- Lombok
- spring-boot-starter-test
- springdoc-openapi-ui
- springdoc-openapi-data-rest

3.3.6 Deploy

Partendo dal CORE si va ad eseguire la build dell'applicazione attraverso il comando:
maven clean, install, -DSkipTests

Con lo stesso comando, a seguire, si possono eseguire le builds di tutti gli altri micro-servizi; non esiste un ordine predefinito da seguire, l'importante è fare la build del CORE prima di tutti gli altri.

Dopodiché si procede con il caricamento sulla piattaforma di OpenShift:

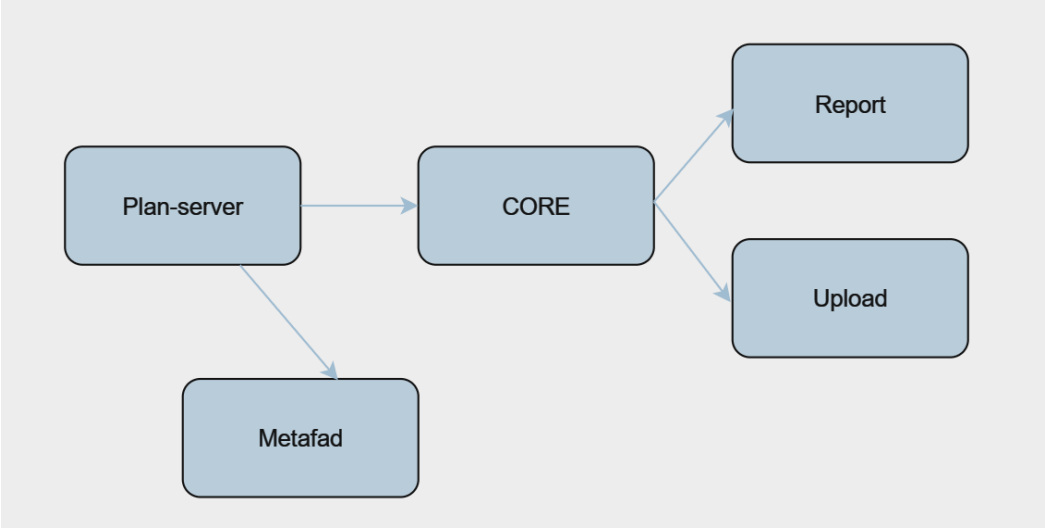
- Per farlo selezionare il pod di interesse e nella sezione *Actions* selezionare *Edit <nome microservizio>*
- Nell'apposita sezione caricare il file .jar generato dalla build, che si trova nella cartella target all'interno della cartella del micro-servizio che si intende rilasciare
- Una volta selezionato premere *Save* e sarà in automatico eseguito il deploy del file jar.

Prima di procedere al rilascio del micro-servizio attendere che sia terminato il caricamento del file e che il pod si sia riavviato.

3.4 MICDL-PLAN-SERVER

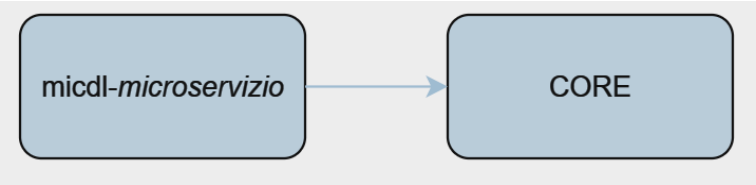
Micro-servizio che gestisce la pianificazione del lavoro di digitalizzazione da parte del cantiere. Consente al project manager (PM) di pianificare l'intero ciclo di vita del cantiere, programmare il numero di lotti (prototipo, digitalizzazione, descrizione), il numero di pacchetti per ogni lotto e il numero di risorse digitali e schede descrittive che verranno prodotte.

3.4.1 Funzionalità



- Nel momento in cui si procede con la pubblicazione del documento generato dal Gantt, il **micro-servizio "MICDL-PLAN-SERVER"**, attraverso il **micro-servizio "MICDL-CORE"**, comunica con il micro-servizio di Report per la creazione delle istanze template relative ai lotti pianificati.
- Al rifiuto di un task di validazione il **micro-servizio "MICDL-PLAN-SERVER"** comunica con il micro-servizio di Report per modificare lo stato dei template da "*pubblicato*" a "*da aggiornare*"
- Il **micro-servizio "MICDL-PLAN-SERVER"** comunica con il micro-servizio di Upload al rifiuto dei pacchetti lotto
- Alla creazione o eliminazione del task di *recupero del digitale pregresso*, **micro-servizio "MICDL-PLAN-SERVER"** comunica con il micro-servizio di Report per la creazione dell'istanza template relativa al *template di recupero del digitale pregresso*
- All'eliminazione di un lotto durante la fase di pianificazione/ripianificazione è necessario eliminarne anche le dipendenze, i pacchetti e i relativi template. **micro-servizio "MICDL-PLAN-SERVER"** comunica quindi con i micro-servizi di Report e di Upload
- Il **micro-servizio "MICDL-PLAN-SERVER"** comunica direttamente con il micro-servizio di metafad per inviare le informazioni di cambio stato dei lotti descrittivi e l'avvio di un cantiere

Il micro-servizio “MICDL-PLAN-SERVER” comunica direttamente con il micro-servizio di Camunda, questo per consentire l’attivazione di due chronjob per l’attivazione programmata dei lotti (che passano in stato in lavorazione) e del task, ove presente, di recupero del digitale pregresso.



Come tutti i micro-servizi, micro-servizio “MICDL-PLAN-SERVER” passa attraverso il CORE per le informazioni del cantiere in sessione e per la getUserInfo che restituisce le informazioni dell’utente connesso.

3.4.2 API

Comandi
/lotti/forceChronUpdateLotti
/task/forceChronAttivazioneRecuperoDigitale
/task/saveGantt
/lotti/pubblicaGantt
/task/tasksHome

/metafad/invioCantiere
/pianificazioni/pianificazioneStandard/attivazioneCantiere?tipoPianificazione={pianificazioneSelezionata}
/lotti/creaLottoZero
/pianificaziones/search/findByIdTenant?idTenant={cantiereId}
/lottoes/search/getLottoZeroByCantiere
/lotti/creazioneDigitalePregresso
/lotti/deleteDigitalePregresso
/lotti/getOverviewLotti
/lotti/creazione/tasks?lottoPrototipo={lotti.prototipazione}&lottoDigitalizzazione={lotti.digitalizzazione}&lottoDescrittivo={lotti.descrizione}&flagRilavor
/lottoes/search/getLottiByCantiere?projection=lottoProj &lavorazione={}&idTipoMateriale={}&idScheda={}&idTipoLotto={}&flagRecupero={} &flagRilavo &skipLottoZero=true &idFondo={}

/lotti/updateStatoLotto/{idLotto}
/lotti/validazioneLotti
/lottoes/search/getAllLottiByCantiere?projection=lottoProj
/lotti/eliminaLotto/{lottoId}
/lottoes/search/tuttiLottiChiusi
/lotti/checkAllLottiDescClosed
/lotti/chiudiLotto/{idLotto}
/lotti/checkCanLottoProceed/{idLotto}

/lotti/getLottiGrouped
/pianificazioni/rifiutaStep/{step}

3.4.3 Metriche

/actuator/metrics:

- Endpoint base di spring-boot offre una serie di metriche sull'applicazione.

```
{
        "names": [
                "application.ready.time",
                "application.started.time",
                … ,
        ]
}
```

/actuator/info:

- Endpoint base di spring-boot fornisce informazioni personalizzate sull'applicazione.
- Attualmente il servizio non restituisce alcuna informazione

```
{}
```

/actuator/health:

- Restituisce un codice di stato HTTP 200 OK se l'applicazione è sana.

```
{
        "status": "UP",
        "groups": [
                "liveness",
                "readiness"
                ]
}
```

- Il servizio restituisce 503 qualora non fosse disponibile

3.4.4 Configurazioni

La variabile d'ambiente comune per tutti i micro-servizi nella configurazione è la seguente:

spring.profiles.active=local

Per quanto riguarda il micro-servizio "MICDL-PLAN-SERVER" invece:

- JDBC_URL: Contiene l'URL di connessione al database
- DATABASE_USER: Contiene il nome utente per l'autenticazione al db
- DATABASE_PASSWORD: Contiene la password associata all'utente del db per la connessione
- ISSUER_URI: Contiene l'URI dell'emittente del token di autenticazione
- SHOW_SQL: Settata a false per disabilitare la visualizzazione delle query
- REST_TEMPLATE_HOST: Contiene l'URL dei micro-servizi con i quali il modulo di collaudo comunica
- FISCAL_CODE_KEY: Contiene l'identificatore utilizzato per accedere ai dati del codice fiscale in un contesto specifico dell'applicazione
- CODE_GUI_KEY: Contiene l'identificatore utilizzato per accedere ai dati dell'interfaccia grafica dell'applicazione
- USER_MAIL: Contiene l'indirizzo mail per accedere ai dati dell'utente
- BASE_PATH_REPORT: Contiene il percorso di base utilizzato per la generazione di documenti
- BASE_PATH_UPLOAD: Contiene il percorso di base utilizzato per l'upload di documenti
- BASE_PATH_DOCUMENTALE: Contiene il percorso di base utilizzato per la generazione di documenti
- REST_TEMPLATE_HOST_UPLOAD: Specifica la radice del path del micro-servizio di upload
- REST_TEMPLATE_HOST_REPORT: Specifica la radice del path del micro-servizio di report-server
- REST_TEMPLATE_HOST_DOCUMENTALE: Specifica la radice del path del micro-servizio di documentale
- CAMUNDA_ENDPOINT: Endpoint di Camunda
- METAFAD_ENDPOINT_CANTIERE: Endpoint di Metafad con il contesto specifico di cantiere
- METAFAD_ENDPOINT_LOTTO: Endpoint di Metafad con il contesto specifico di cantiere

3.4.5 Dipendenze/Librerie utilizzate

- Spring Boot Starter Actuator
- Spring Boot Starter Data JPA
- Spring Boot Starter Data REST
- Spring Session Data Redis
- Spring Boot Starter Data Redis
- Spring Session Core

- Micdl Core
- H2 Database,
- Oracle JDBC Driver (ojdbc8)
- Lombok,Spring Boot Starter Test
- Springdoc OpenAPI UI
- Springdoc OpenAPI Data REST

3.4.6 Deploy

Partendo dal CORE si va ad eseguire la build dell'applicazione attraverso il comando: maven clean, install, -DSkipTests

Con lo stesso comando, a seguire, si possono eseguire le builds di tutti gli altri micro-servizi; non esiste un ordine predefinito da seguire, l'importante è fare la build del CORE prima di tutti gli altri.

Dopodiché si procede con il caricamento sulla piattaforma di OpenShift:

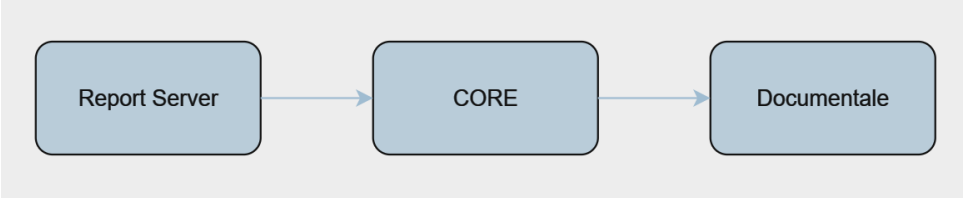
- Per farlo selezionare il pod di interesse e nella sezione *Actions* selezionare *Edit <nome micro-servizio>*
- Nell'apposita sezione caricare il file .jar generato dalla build, che si trova nella cartella target all'interno della cartella del micro-servizio che si intende rilasciare
- Una volta selezionato premere *Save* e sarà in automatico eseguito il deploy del file jar.

Prima di procedere al rilascio del micro-servizio attendere che sia terminato il caricamento del file e che il pod si sia riavviato.

3.5 MICDL-REPORT-SERVER

Micro-servizio che gestisce le funzioni di reportistica dell'applicazione.
Espone le API che gestiscono il download e l'upload dei file, tutti i file.jrxml e i relativi compilati .jasper. Gestisce, inoltre, tutte le informazioni in input dai template e quelle che devono essere mostrate.

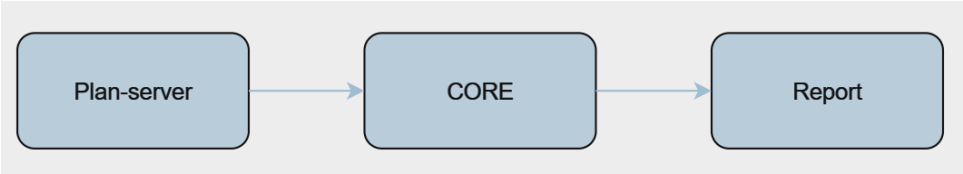
3.5.1 Funzionalità



Per le funzionalità di pubblica template il micro-servizio "MICDL-REPORT-SERVER" comunica con il micro-servizio "MICDL-DOCUMENTALE-SERVER" attraverso il CORE; il micro-servizio "MICDL-REPORT-SERVER" genera il pdf e attraverso il documentale il file appena generato viene caricato su s3.

Report gestisce inoltre la pubblicazione delle segnalazioni di *Casi aperti:*

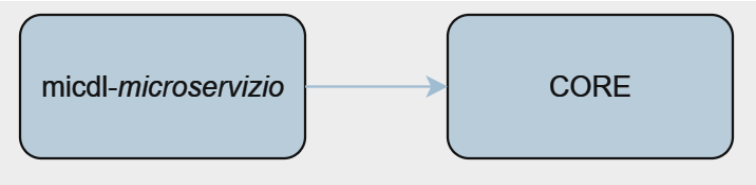
- Quando viene pubblicata una segnalazione (diversa da generica), il micro-servizio "MICDL-REPORT-SERVER genera il file e comunica, attraverso il CORE, con il micro-servizio "MICDL-DOCUMENTALE-SERVER per la pubblicazione del file della conversazione sul S3 documentale.



Il micro-servizio "MICDL-REPORT-SERVER" attraverso il CORE riceve informazioni dal micro-servizio "MICDL-PLAN-SERVER":

- Al rifiuto di un task di validazione il micro-servizio "MICDL-PLAN-SERVER" comunica con il micro-servizio "MICDL-REPORT-SERVER" per modificare lo stato dei template (da *pubblicato* a *da aggiornare*)
- Alla creazione o eliminazione del task di *recupero del digitale pregresso*, il micro-servizio "MICDL-PLAN-SERVER" comunica con il micro-servizio "MICDL-REPORT-SERVER" per la creazione dell'istanza template relativa al *template di recupero del digitale pregresso*
- All'eliminazione di un lotto durante la fase di pianificazione/ripianificazione è necessario eliminarne anche le dipendenze, i pacchetti e i relativi template. il micro-servizio "MICDL-PLAN-SERVER" comunica quindi con il micro-servizio "MICDL-REPORT-SERVER".

- Nel momento in cui si procede con la pubblicazione del documento generato dal gantt, il micro-servizio "MICDL-PLAN-SERVER" attraverso il CORE comunica con il micro-servizio di Report per la creazione delle istanze template relative ai lotti pianificati.



Come tutti i micro-servizi, il micro-servizio "MICDL-REPORT-SERVER" passa attraverso il CORE per ottenere le informazioni del cantiere in sessione e attraverso la getUserInfo che restituisce le informazioni dell'utente connesso.

Una funzionalità rilevante del micro-servizio "MICDL-REPORT-SERVER" è la gestione della *data di inizio lavori*, un campo aggiuntivo popolato nel template di verbale inizio lavori e successivamente validato attraverso il set del flag1 a 'S' (vedi la API /immagine/sign?idIstanzaTemplate={istanzaTemplateId})

3.5.2 API

Comandi
/reports/creaIstanzeTemplate
/capitoliManuali/getCapitoliManualiFiltered
/checkListLottoPrototipoes/search/findByIstanzaTemplateId?idIstanzaTemplate={istanzaTemplateId} &projection=checkListLottoPrototipoProj
/contrTipoMateriales/search/getContrTipoMaterialePerCantiereETipo?idIstanzaTemplate={istanzaTemplateId}&projection=contrTipoMaterialeProj
/datiAggiuntiviTemplates/search/findByIstanzaTemplateId?idIstanzaTemplate={istanzaTemplateId}&projection=datiAggiuntiviTemplateProj

/datiAggiuntiviTemplates/search/getDataInizioLavori
/datiAggiuntiviTemplates/search/getFlagInizioLavori
/immagine/save/{datiAggiuntiviTemplateId}
/dispAccessoCantieres/search/findByIstanzaTemplateId?istanzaTemplateId={istanzaTemplateId}&projection=dispAccessoCantiereProj
/fondoes/search/getFondoByCantiere?projection=fondoProj
/fondoDigitalizzaziones/search/findByIstanzaTemplateId?istanzaTemplateId={istanzaTemplateId}&projection=fondoDigitalizzazioneProj
/istanzaTemplates/search/findByTemplateDescBreveNomeTemplateAndCantiere?breveNomeTemplate={nomeTemplate}&projection=istanzaTemplate
/istanzaTemplates/search/findByTemplateCHKDescBreveNomeTemplateAndCantiereAndLotto?breveNomeTemplate={nomeTemplate}&lottoId={lottoId

/istanzaTemplates/search/findByTemplateCHKDigBreveNomeTemplateAndCantiereAndLotto?breveNomeTemplate={nomeTemplate}&lottoId={lottoId}&
/istanzaTemplates/search/findByTemplateCHKPrtBreveNomeTemplateAndCantiereAndLotto?breveNomeTemplate=${nomeTemplate}&lottoId=${lottoI Proj
istanzaTemplates/search/getAllIstanzaTemplateByIdTenant?projection=istanzaTemplateProj
/reports/pubblicaTemplate/{istanzaTemplateId}
/immagine/sign?idIstanzaTemplate={istanzaTemplateId}
/lottoDescrittivoes/search/findByCantiereList?projection=lottoDescrittivoProj
/lottoDescrittivoes/search/findByIstanzaTemplateId?idIstanzaTemplate={idIstanzaTemplate}&projection=lottoDescrittivoProj
/lottoDigitalizzazioni/search/findByCantiereList?projection=lottoDigitalizzazioneProj
/lottoDigitalizzazioni/search/findByIstanzaTemplateId?idIstanzaTemplate={idIstanzaTemplate}&projection=lottoDigitalizzazioneProj
/lottoDigitalizzazione/edit/{idDigitalizzazione}?lottoPrototipo={idPrototipo}&flagOCR={flagOcr}

/prototipoCantieres/search/findByIstanzaTemplateId?idIstanzaTemplate={istanzaTemplateId}
/prototipoCantieres/search/findByCantiereAndPrototipoTemplateList?projection=prototipoCantiereProj
/reports/pubblicaGantt/creaIstanzeLottiEPrototipiCantieri
/recuperoDigitalizzatoes/search/findByIstanzaTemplateId?idIstanzaTemplate={istanzaTemplateId}&projection=recuperoDigitalizzatoProj
/specialistaCantieres/search/findByIstanzaTemplateId?istanzaTemplateId={istanzaTemplateId}&projection=specialistaCantiereProj&sort=id
/strumentoCantieres/search/findByIstanzaTemplateId?istanzaTemplateId={istanzaTemplateId}&projection=strumentoCantiereProj&sort=id
/suppArchivCantieres/search/findByIstanzaTemplateId?istanzaTemplateId={istanzaTemplateId}
/reports/{templateFile}/format/pdf?I_cantiere_id={cantiereId}&I_istanzaTemplate_id={istanzaTemplateId}
/reports/pubblicaTicket?idIssue={idTicket}

3.5.3 Metriche

/actuator/metrics:

- Endpoint base di spring-boot offre una serie di metriche sull'applicazione.


```
{
    "names": [
            "application.ready.time",
            "application.started.time",
            … ,
            ]
}
```


/actuator/info:

- Endpoint base di spring-boot fornisce informazioni personalizzate sull'applicazione.
- Attualmente il servizio non restituisce alcuna informazione

/actuator/health:

- Restituisce un codice di stato HTTP 200 OK se l'applicazione è sana.


```
{
        "status": "UP",
        "groups": [
                "liveness",
                "readiness"
        ]
}
```


- Il servizio restituisce 503 qualora non fosse disponibile

3.5.4 Configurazioni

La variabile d'ambiente comune per tutti i micro-servizi nella configurazione è la seguente:
spring.profiles.active=local

Per quanto riguarda il micro-servizio "MICDL-REPORT-SERVER" invece:

- JDBC_URL: Contiene l'URL di connessione al database
- DATABASE_USER: Contiene il nome utente per l'autenticazione al db
- DATABASE_PASSWORD: Contiene la password associata all'utente del db per la connessione
- ISSUER_URI: Contiene l'URI dell'emittente del token di autenticazione
- SHOW_SQL: Settata a false per disabilitare la visualizzazione delle query
- SERVLET_CONTEXT_PATH: Contiene il percorso del contesto del servlet nell'ambiente web, per specificare la radice dell'applicazione
- REST_TEMPLATE_HOST: Contiene l'URL dei micro-servizi con i quali il modulo di collaudo comunica

- FISCAL_CODE_KEY: Contiene l'identificatore utilizzato per accedere ai dati del codice fiscale in un contesto specifico dell'applicazione
- CODE_GUI_KEY: Contiene l'identificatore utilizzato per accedere ai dati dell'interfaccia grafica dell'applicazione
- USER_MAIL: Contiene l'indirizzo mail per accedere ai dati dell'utente
- BASE_PATH_REPORT: Contiene il percorso di base utilizzato per la generazione di documenti
- BASE_PATH_UPLOAD: Contiene il percorso di base utilizzato per l'upload di documenti
- BASE_PATH_DOCUMENTALE: Contiene il percorso di base utilizzato per la generazione di documenti
- REST_TEMPLATE_HOST_UPLOAD: Specifica la radice del path del micro-servizio di upload
- REST_TEMPLATE_HOST_REPORT: Specifica la radice del path del micro-servizio di report-server
- REST_TEMPLATE_HOST_DOCUMENTALE: Specifica la radice del path del micro-servizio di documentale

3.5.5 Dipendenze/Librerie utilizzate

- org.springframework.boot:spring-boot-starter-actuator
- org.springframework.boot:spring-boot-starter-data-rest
- org.springframework.session:spring-session-core
- org.springframework.session:spring-session-data-redis
- org.springframework.boot:spring-boot-starter-data-redis
- it.dstech.micdlcore:micdl-core:0.0.1-SNAPSHOT
- org.springframework.boot:spring-boot-starter-data-jpa
- org.springframework.boot:spring-boot-starter-data-rest
- com.h2database:h2 com.oracle.database.jdbc:ojdbc8
- org.projectlombok:lombok
- org.springframework.boot:spring-boot-starter-test
- net.sf.jasperreports:jasperreports:6.16.0, com.lowagie:itext
- net.sf.jasperreports:jasperreports-fonts:6.16.0
- net.sf.jasperreports:jasperreports-functions:6.16.0
- org.springframework.boot:spring-boot-starter-data-jpa
- org.springdoc:springdoc-openapi-ui:1.6.9
- org.springdoc:springdoc-openapi-data-rest:1.6.9

3.5.6 Deploy

Partendo dal CORE si va ad eseguire la build dell'applicazione attraverso il comando: maven clean, install, -DSkipTests

Con lo stesso comando, a seguire, si possono eseguire le builds di tutti gli altri micro-servizi; non esiste un ordine predefinito da seguire, l'importante è fare la build del CORE prima di tutti gli altri.

Dopodiché si procede con il caricamento sulla piattaforma di OpenShift:

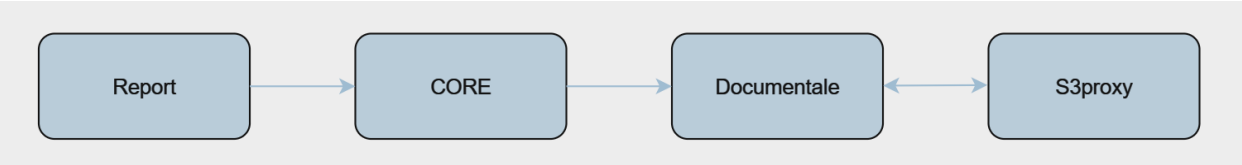
- Per farlo selezionare il pod di interesse e nella sezione *Actions* selezionare *Edit <nome micro-servizio>*
- Nell'apposita sezione caricare il file .jar generato dalla build, che si trova nella cartella target all'interno della cartella del micro-servizio che si intende rilasciare
- Una volta selezionato premere *Save* e sarà in automatico eseguito il deploy del file jar.

Prima di procedere al rilascio del micro-servizio attendere che sia terminato il caricamento del file e che il pod si sia riavviato.

3.6 MICDL-DOCUMENTALE-SERVER

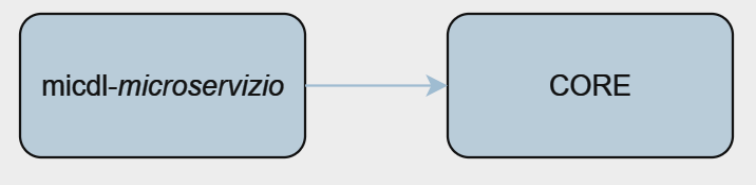
Il micro-servizio espone le API per le funzionalità di download/upload di file sullo storage S3 documentale, espone inoltre le API per la modifica e gestione dello stato dei documenti caricati.

3.6.1 Funzionalità



La funzionalità principale del micro-servizio è quella di upload dei documenti sullo storage S3 documentale. Nel momento in cui viene richiamata la pubblicaTemplate sul micro-servizio "MICDL-REPORTER-SERVER", il documentale riceve in input *file, tipo documento, istanzaTemplateId* e se esiste *idLotto* attraverso il CORE; effettua così una chiamata al micro-servizio "S3ProxyFE" che effettua il caricamento del file.

Altra funzionalità rilevante è quella del download del singolo file, della cartella di file ed il download massimo di più cartelle di più cantieri in contemporanea che, una volta gestite le informazioni da voler scaricare, esegue una richiesta dei file attraverso il micro-servizio "S3ProxyFE".



Come tutti i micro-servizi, il micro-servizio "MICDL-DOCUMENTALE-SERVER" passa attraverso il CORE per ottenere le informazioni del cantiere in sessione e per la getUserInfo che restituisce le informazioni dell'utente connesso.

3.6.2 API

Comandi
/documentale/upload?tipoDocumento=${tipoDocumento}&istanzaTemplateId=${istanzaTemplate}

/documentale/multipleS3Download
/documentale/changeStatoDocumento?istanzaTemplateId=${istanzaTemplate}&statoDocumento=${status}
/documentoes/search/documenti?projection=documentoProj
/storiaFiles/search/findStoriaFileByDocumentoId?documentoId=${document.id}&projection=StoriaFilePrj
/fileDocumentoes/search/findByDoc?idDocumento=${documentId}&projection=fileDocumentoProj
/documentale/massiveDownload
/documentale/download/${fileId}
/documentale/upload/${documentoId}
/storiaDocumentoes/search/getLastStatoDocumentoByIstanzaTemplate/?idIstanzaTemplate=${idIstanzaTemplate}&projection=storiaDocumentoProj

3.6.3 Metriche

/actuator/metrics:

- Endpoint base di spring-boot offre una serie di metriche sull'applicazione.

```
{
        "names": [
                "application.ready.time",
                "application.started.time",
                … ,
                ]
}
```

/actuator/info:

- Endpoint base di spring-boot fornisce informazioni personalizzate sull'applicazione.
- Attualmente il servizio non restituisce alcuna informazione

/actuator/health:

- Restituisce un codice di stato HTTP 200 OK se l'applicazione è sana.

```
{
        "status": "UP",
        "groups": [
                "liveness",
                "readiness"
                ]
}
```

- Il servizio restituisce 503 qualora non fosse disponibile

3.6.4 Configurazioni

La variabile d'ambiente comune per tutti i micro-servizi nella configurazione è la seguente:
spring.profiles.active=local
Per quanto riguarda il micro-servizio "MICDL-DOCUMENTALE-SERVER invece:

- JDBC_URL: Contiene l'URL di connessione al database
- DATABASE_USER: Contiene il nome utente per l'autenticazione al db
- DATABASE_PASSWORD: Contiene la password associata all'utente del db per la connessione
- ISSUER_URI: Contiene l'URI dell'emittente del token di autenticazione
- SHOW_SQL: Settata a false per disabilitare la visualizzazione delle query

- SERVLET_CONTEXT_PATH: Contiene il percorso del contesto del servlet nell'ambiente web, per specificare la radice dell'applicazione
- REST_TEMPLATE_HOST: Contiene l'URL dei micro-servizi con i quali il modulo di collaudo comunica
- FISCAL_CODE_KEY: Contiene l'identificatore utilizzato per accedere ai dati del codice fiscale in un contesto specifico dell'applicazione
- CODE_GUI_KEY: Contiene l'identificatore utilizzato per accedere ai dati dell'interfaccia grafica dell'applicazione
- USER_MAIL: Contiene l'indirizzo mail per accedere ai dati dell'utente
- BASE_PATH_REPORT: Contiene il percorso di base utilizzato per la generazione di documenti
- BASE_PATH_UPLOAD: Contiene il percorso di base utilizzato per l'upload di documenti
- BASE_PATH_DOCUMENTALE: Contiene il percorso di base utilizzato per la generazione di documenti
- REST_TEMPLATE_HOST_UPLOAD: Specifica la radice del path del micro-servizio di upload
- REST_TEMPLATE_HOST_REPORT: Specifica la radice del path del micro-servizio di report-server
- REST_TEMPLATE_HOST_DOCUMENTALE: Specifica la radice del path del micro-servizio di documentale
- S3_ENDPOINT_UPLOAD: URL dell'archiviazione S3 utilizzato per l'upload di file
- S3_ENDPOINT_DOWNLOAD: URL dell'archiviazione S3 utilizzato per il download di file
- MICDL_LOG: Configura il percorso dei file di log
- MICDL_CORE_LOG: Configura il percorso dei file di log per il core

3.6.5 Dipendenze/Librerie utilizzate

- spring-boot-starter-actuator
- spring-boot-starter-data-jpa
- spring-boot-starter-data-rest
- spring-session-data-redis
- spring-boot-starter-data-redis
- spring-session-core
- micdl-core
- H2
- Ojdbc8
- Lombok
- spring-boot-starter-test
- springdoc-openapi-ui, springdoc-openapi-data-rest

3.6.6 Deploy

Partendo dal CORE si va ad eseguire la build dell'applicazione attraverso il comando: maven clean, install, -DSkipTests

Con lo stesso comando, a seguire, si possono eseguire le builds di tutti gli altri micro-servizi; non esiste un ordine predefinito da seguire, l'importante è fare la build del CORE prima di tutti gli altri.

Dopodiché si procede con il caricamento sulla piattaforma di OpenShift:

- Per farlo selezionare il pod di interesse e nella sezione *Actions* selezionare *Edit <nome micro-servizio>*
- Nell'apposita sezione caricare il file .jar generato dalla build, che si trova nella cartella target all'interno della cartella del micro-servizio che si intende rilasciare
- Una volta selezionato premere *Save* e sarà in automatico eseguito il deploy del file jar.

Prima di procedere al rilascio del micro-servizio attendere che sia terminato il caricamento del file e che il pod si sia riavviato.

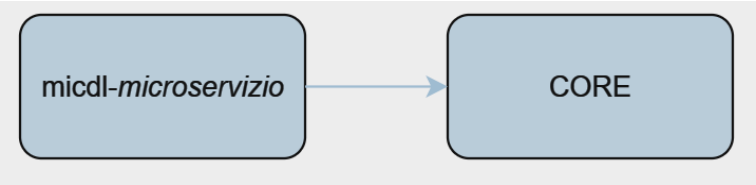
3.7 MICDL-UPLOAD

Micro-servizio che contiene le API per la pagina del modulo di collaudo, espone i dati per popolare i grafici (torta e istogramma) e le funzionalità di valida/scarta pacchetti.

3.7.1 Funzionalità

Il micro-servizio "MICDL-UPLOAD" riceve informazioni dal micro-servizio "MICDL-PLAN-SERVER" attraverso il core:

- Al rifiuto dei pacchetti lotto
- All'eliminazione di un lotto durante la fase di pianificazione/ripianificazione, della quale è necessario eliminarne anche le dipendenze e i pacchetti.



Come tutti i micro-servizi, il micro-servizio "MICDL-UPLOAD" passa attraverso il CORE per le informazioni del cantiere in sessione e per la getUserInfo che restituisce le informazioni dell'utente connesso.

3.7.2 API

Comandi	Tipo Metodo	Query
/collaudo/accettaPacchettiLotto/{idLotto}	PUT	IdLotto--> PathVariable
/collaudo/tortaPianif/{idLotto}	GET	IdLotto--> PathVariable
/collaudo/scartati/ {idLotto}	GET	IdLotto--> PathVariable
/collaudo/istogramma/ {idLotto}	GET	IdLotto--> PathVariable

/collaudo/pacchettiAnalizzatiAcquisite/ {idLotto}	GET	IdLotto--> Path-Variable desc--> Request-Param stato--> Request-Param pageable
/collaudo/validaPacchetto/{idLotto}	PUT	IdPacchettoLotto--> PathVariable
/collaudo/scartaPacchetto/{idLotto}	PUT	IdPacchettoLotto--> PathVariable
/collaudo/risorseDigitaliAnalizzatiAcquisite/{idLotto}?{pageParam}&esito={esito}&idPacchettoLotto={idPacchettoLotto}	GET	IdLotto--> PathVariable IdPacchettoLotto--> RequestParam desc--> RequestParam esito--> RequestParam pageable

3.7.3 Metriche

/actuator/metrics:

- Endpoint base di spring-boot offre una serie di metriche sull'applicazione.

```
{
        "names": [
                "application.ready.time",
                "application.started.time",
                … ,
                ]
}
```

/actuator/info:

- Endpoint base di spring-boot fornisce informazioni personalizzate sull'applicazione.
- Attualmente il servizio non restituisce alcuna informazione

/actuator/health:

- Restituisce un codice di stato HTTP 200 OK se l'applicazione è sana.

```
{
        "status": "UP",
        "groups": [
                "liveness",
                "readiness"
        ]
}
```

- Il servizio restituisce 503 qualora non fosse disponibile

3.7.4 Configurazioni

La variabile d'ambiente comune per tutti i micro-servizi nella configurazione è la seguente:
spring.profiles.active=local

Per quanto riguarda il micro-servizio "MICDL-UPLOAD" invece:

- JDBC_URL: Contiene l'URL di connessione al database
- DATABASE_USER: Contiene il nome utente per l'autenticazione al db
- DATABASE_PASSWORD: Contiene la password associata all'utente del db per la connessione
- ISSUER_URI: Contiene l'URI dell'emittente del token di autenticazione
- SHOW_SQL: Settata a false per disabilitare la visualizzazione delle query
- SERVLET_CONTEXT_PATH: Contiene il percorso del contesto del servlet nell'ambiente web, per specificare la radice dell'applicazione
- REST_TEMPLATE_HOST: Contiene l'URL dei micro-servizi con i quali il modulo di collaudo comunica
- FISCAL_CODE_KEY: Contiene l'identificatore utilizzato per accedere ai dati del codice fiscale in un contesto specifico dell'applicazione
- CODE_GUI_KEY: Contiene l'identificatore utilizzato per accedere ai dati dell'interfaccia grafica dell'applicazione
- USER_MAIL: Contiene l'indirizzo mail per accedere ai dati dell'utente
- BASE_PATH_REPORT: Contiene il percorso di base utilizzato per la generazione di documenti
- BASE_PATH_UPLOAD: Contiene il percorso di base utilizzato per l'upload di documenti
- BASE_PATH_DOCUMENTALE: Contiene il percorso di base utilizzato per la generazione di documenti
- REST_TEMPLATE_HOST_UPLOAD: Specifica la radice del path del micro-servizio di upload
- REST_TEMPLATE_HOST_REPORT: Specifica la radice del path del micro-servizio di report-server

- REST_TEMPLATE_HOST_DOCUMENTALE: Specifica la radice del path del micro-servizio di documentale

3.7.5 Dipendenze/Librerie utilizzate

- org.springframework.boot:spring-boot-starter-actuator
- org.springframework.boot:spring-boot-starter-data-jpa
- org.springframework.boot:spring-boot-starter-data-redis
- org.springframework.session:spring-session-data-redis
- org.springframework.session:spring-session-core
- org.springframework.boot:spring-boot-starter-data-rest
- it.dstech.micdlcore:micdl-core
- com.h2database:h2
- com.oracle.database.jdbc:ojdbc8
- org.projectlombok:lombok
- org.springframework.boot:spring-boot-starter-test
- org.springdoc:springdoc-openapi-ui
- org.springdoc:springdoc-openapi-data-rest

3.7.6 Deploy

Partendo dal CORE si va ad eseguire la build dell'applicazione attraverso il comando: maven clean, install, -DSkipTests

Con lo stesso comando, a seguire, si possono eseguire le builds di tutti gli altri micro-servizi; non esiste un ordine predefinito da seguire, l'importante è fare la build del CORE prima di tutti gli altri.

Dopodiché si procede con il caricamento sulla piattaforma di OpenShift:

- Per farlo selezionare il pod di interesse e nella sezione *Actions* selezionare *Edit <nome micro-servizio>*
- Nell'apposita sezione caricare il file .jar generato dalla build, che si trova nella cartella target all'interno della cartella del micro-servizio che si intende rilasciare
- Una volta selezionato premere *Save* e sarà in automatico eseguito il deploy del file jar.

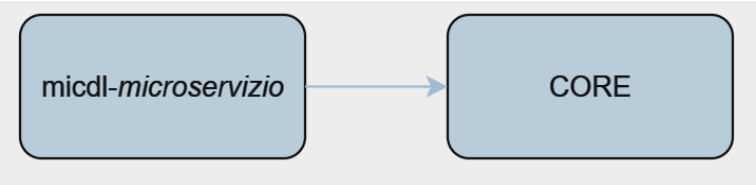
Prima di procedere al rilascio del micro-servizio attendere che sia terminato il caricamento del file e che il pod si sia riavviato.

3.8 MICDL-COLLAUDO

Micro-servizio che contiene le API che restituiscono le statistiche dei lotti per il modulo di collaudo.

3.8.1 Funzionalità

Il micro-servizio “MICDL-COLLAUDO” contiene le API di tipo GET delle statistiche dei lotti.



Come tutti i micro-servizi, il micro-servizio “MICDL-COLLAUDO” passa attraverso il CORE per le informazioni del cantiere in sessione e per la getUserInfo che restituisce le informazioni dell’utente connesso.

3.8.2 API

Comandi	Tipo metodo	Query	Eventi
/descrizione/statisticheLotto/{IdLotto}	GET	IdLotto --> PathVariable	
/descrizione/statistiche/{IdLotto}	GET	IdLotto --> PathVariable	
/descrizione/schedePubblicate/{IdLotto}?filtroNomeScheda={filtroNomeScheda}&filtroData={filtroData}&filtroTipologia={filtroTipologia}report=&{pageParam}	GET	IdLotto --> PathVariable report --> RequestParam filtroNomeScheda--> RequestParam filtroData--> RequestParam filtroTipologia--> RequestParam pageable	Restituisce tutte le schede pubblicate appartenenti ad un certo lotto

3.8.3 Metriche

/actuator/metrics:

- Endpoint base di spring-boot offre una serie di metriche sull'applicazione.

```
{
        "names": [
                "application.ready.time",
                "application.started.time",
                … ,
        ]
}
```

/actuator/info:

- Endpoint base di spring-boot fornisce informazioni personalizzate sull'applicazione.
- Attualmente il servizio non restituisce alcuna informazione

/actuator/health:

- Restituisce un codice di stato HTTP 200 OK se l'applicazione è sana.

```
{
        "status": "UP",
        "groups": [
                "liveness",
                "readiness"
        ]
}
```

- Il servizio restituisce 503 qualora non fosse disponibile

3.8.4 Configurazioni

La variabile d'ambiente comune per tutti i micro-servizi nella configurazione è la seguente:
spring.profiles.active=local

Per quanto riguarda il micro-servizio di collaudo invece:

- JDBC_URL: Contiene l'URL di connessione al database
- DATABASE_USER: Contiene il nome utente per l'autenticazione al db
- DATABASE_PASSWORD: Contiene la password associata all'utente del db per la connessione
- ISSUER_URI: Contiene l'URI dell'emittente del token di autenticazione
- SHOW_SQL: Settata a false per disabilitare la visualizzazione delle query

- SERVLET_CONTEXT_PATH: Contiene il percorso del contesto del servlet nell'ambiente web, per specificare la radice dell'applicazione
- REST_TEMPLATE_HOST: Contiene l'URL dei micro-servizi con i quali il modulo di collaudo comunica
- FISCAL_CODE_KEY: Contiene l'identificatore utilizzato per accedere ai dati del codice fiscale in un contesto specifico dell'applicazione
- CODE_GUI_KEY: Contiene l'identificatore utilizzato per accedere ai dati dell'interfaccia grafica dell'applicazione
- USER_MAIL: Contiene l'indirizzo mail per accedere ai dati dell'utente
- BASE_PATH_REPORT: Contiene il percorso di base utilizzato per la generazione di documenti
- BASE_PATH_UPLOAD: Contiene il percorso di base utilizzato per l'upload di documenti
- BASE_PATH_DOCUMENTALE: Contiene il percorso di base utilizzato per la generazione di documenti
- REST_TEMPLATE_HOST_UPLOAD: Specifica la radice del path del micro-servizio di upload
- REST_TEMPLATE_HOST_REPORT: Specifica la radice del path del micro-servizio di report-server
- REST_TEMPLATE_HOST_DOCUMENTALE: Specifica la radice del path del micro-servizio di documentale

3.8.5 Dipendenze/Librerie utilizzate

- spring-boot-starter-actuator
- spring-boot-starter-data-jpa
- spring-boot-starter-data-rest
- spring-session-data-redis
- spring-boot-starter-data-redis
- spring-session-core
- it.dstech.micdlcore:micdl-core
- H2
- com.oracle.database.jdbc:ojdbc8
- org.projectlombok:lombok
- org.springframework.boot:spring-boot-starter-test
- org.springdoc:springdoc-openapi-ui
- org.springdoc:springdoc-openapi-data-rest

3.8.6 Deploy

Partendo dal CORE si va ad eseguire la build dell'applicazione attraverso il comando: *maven clean, install, -DSkipTests*

Con lo stesso comando, a seguire, si possono eseguire le builds di tutti gli altri micro-servizi; non esiste un ordine predefinito da seguire, l'importante è fare la build del CORE prima di tutti gli altri.

Dopodiché si procede con il caricamento sulla piattaforma di OpenShift:

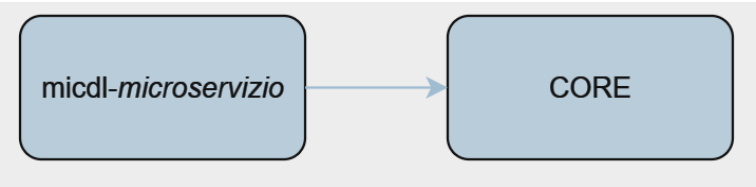
- Per farlo selezionare il pod di interesse e nella sezione *Actions* selezionare *Edit <nome micro-servizio>*
- Nell'apposita sezione caricare il file .jar generato dalla build, che si trova nella cartella target all'interno della cartella del micro-servizio che si intende rilasciare
- Una volta selezionato premere *Save* e sarà in automatico eseguito il deploy del file jar.

Prima di procedere al rilascio del micro-servizio attendere che sia terminato il caricamento del file e che il pod si sia riavviato.

3.9 MICDL-NOTIFICHE-SERVER

Micro-servizio che si occupa della restituzione delle notifiche inerenti alle attività della piattaforma.

3.9.1 Funzionalità



Come tutti i micro-servizi, il micro-servizio "MICDL-NOTIFICHE-SERVER" passa attraverso il CORE per le informazioni del cantiere in sessione e per la getUserInfo che restituisce le informazioni dell'utente connesso.

3.9.2 API

Comandi	Tipo Metodo	Query	Eventi
/notifiche/countNotificheNonLette	GET	//	Restituisce il conteggio delle notifiche non lette

3.9.3 Metriche

/actuator/metrics:
Endpoint base di spring-boot offre una serie di metriche sull'applicazione.

```
{
    "names": [
        "application.ready.time",
        "application.started.time",
        … ,
    ]
}
```

/actuator/info:

- Endpoint base di spring-boot fornisce informazioni personalizzate sull'applicazione.
- Attualmente il servizio non restituisce alcuna informazione

/actuator/health:

- Restituisce un codice di stato HTTP 200 OK se l'applicazione è sana.

```
{
        "status": "UP",
        "groups": [
                "liveness",
                "readiness"
        ]
}
```

- Il servizio restituisce 503 qualora non fosse disponibile

3.9.4 Configurazioni

La variabile d'ambiente comune per tutti i micro-servizi nella configurazione è la seguente:
spring.profiles.active=local

Per quanto riguarda il micro-servizio di notifiche invece:

- JDBC_URL: Contiene l'URL di connessione al database
- DATABASE_USER: Contiene il nome utente per l'autenticazione al db
- DATABASE_PASSWORD: Contiene la password associata all'utente del db per la connessione
- ISSUER_URI: Contiene l'URI dell'emittente del token di autenticazione
- SHOW_SQL: Settata a false per disabilitare la visualizzazione delle query
- SERVLET_CONTEXT_PATH: Contiene il percorso del contesto del servlet nell'ambiente web, per specificare la radice dell'applicazione
- REST_TEMPLATE_HOST: Contiene l'URL dei micro-servizi con i quali il modulo di collaudo comunica
- FISCAL_CODE_KEY: Contiene l'identificatore utilizzato per accedere ai dati del codice fiscale in un contesto specifico dell'applicazione
- CODE_GUI_KEY: Contiene l'identificatore utilizzato per accedere ai dati dell'interfaccia grafica dell'applicazione
- USER_MAIL: Contiene l'indirizzo mail per accedere ai dati dell'utente
- BASE_PATH_REPORT: Contiene il percorso di base utilizzato per la generazione di documenti
- BASE_PATH_UPLOAD: Contiene il percorso di base utilizzato per l'upload di documenti
- BASE_PATH_DOCUMENTALE: Contiene il percorso di base utilizzato per la generazione di documenti
- REST_TEMPLATE_HOST_UPLOAD: Specifica la radice del path del micro-servizio di upload
- REST_TEMPLATE_HOST_REPORT: Specifica la radice del path del micro-servizio di report-server

- REST_TEMPLATE_HOST_DOCUMENTALE: Specifica la radice del path del micro-servizio di documentale

3.9.5 Dipendenze/Librerie utilizzate

- spring-boot-starter-actuator
- spring-boot-starter-data-rest
- spring-boot-starter-data-jpa
- spring-boot-starter-oauth2-resource-server
- spring-session-core
- spring-session-data-redis
- spring-boot-starter-data-redis
- Ojdbc8
- lombok
- spring-boot-starter-test
- springdoc-openapi-ui
- springdoc-openapi-data-rest
- micdl-core

3.9.6 Deploy

Partendo dal CORE si va ad eseguire la build dell'applicazione attraverso il comando: maven clean, install, -DSkipTests

Con lo stesso comando, a seguire, si possono eseguire le builds di tutti gli altri micro-servizi; non esiste un ordine predefinito da seguire, l'importante è fare la build del CORE prima di tutti gli altri.

Dopodiché si procede con il caricamento sulla piattaforma di OpenShift:

- Per farlo selezionare il pod di interesse e nella sezione *Actions* selezionare *Edit <nome micro-servizio>*
- Nell'apposita sezione caricare il file .jar generato dalla build, che si trova nella cartella target all'interno della cartella del micro-servizio che si intende rilasciare
- Una volta selezionato premere *Save* e sarà in automatico eseguito il deploy del file jar.

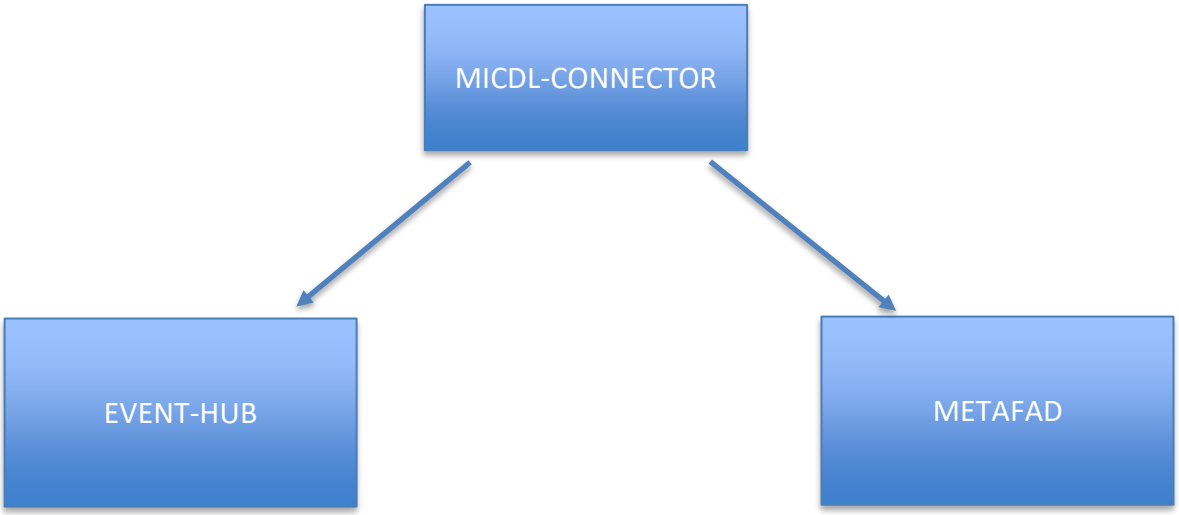
Prima di procedere al rilascio del micro-servizio attendere che sia terminato il caricamento del file e che il pod si sia riavviato.

3.10 MICDL-CONNECTOR

Micro-servizio che orchestra e fa da raccordo per i seguenti applicativi:

- EVENT-HUB (colloquio con ISPC)
- METAFAD

3.10.1 Funzionalità



Il micro-servizio "MICDL-CONNECTOR" viene richiamato mediante esposizione di API REST per diversi applicativi:

- EVENT-HUB
  - *Ingestion* dei pacchetti processati a parità di Cantiere.
- METAFAD
  - Pubblicazione e "spubblicazione" di una scheda e relativo salvataggio dell'operazione.

3.10.2 API

Comandi	Tipo Metodo	Query	Eventi
/azure-access-token/{workspaceId}/{reportId} -> getAzureAccessToken	GET	workspaceId -> PathVariable reportId -> PathVariable	
/{queue}/{queueType} ->pushMessageRabbitMQ	PUT	jwt -> RequestHeader (JWT) queue -> PathVariable queueType -> PathVariable message ->RequestBody	Invio di un messaggio generico su una coda ottenuta mediante la definizione del nome della coda e il tipo della stessa.
/setSchedaPubblicata -> saveMetaFadMessage	POST	jwt -> RequestHeader (JWT) metaFadMessage -> RequestBody bindingResult -> BindingResult (campi non validi)	L'oggetto bindingResult contiene eventuali errori di validazione sul body ricevuto (METAFAD). A seconda dell'azione fornita nel body, si procede ad inserire o eliminare l'entità nel database nella tabella DL_OGGETTO_PACCHETTO.
/getLottoElaboratoML/{worksiteId}/{batchId} -> retrieveProcessedBatchMl	GET	worksiteId -> PathVariable batchId -> PathVariable	
/recognition	POST (Multipart)	Body: xmlFile (multipart) mqMessage (String)	Controllo sulla presenza all'interno del messaggio mqMessage della presenza dei campi obbligatori, il messaggio viene depositato per controllo METS sulla coda **mlquorum** e salvato nella tabella **T_RABBIT_MQ_ML_LOG.** Si aggiorna lo stato a 'Controllo METS' del relativo record della tabella **DL_PACCHETTO_LOTTO**
/receive-event	POST	Il body è composto dal messaggio standard di event-hub	Check uuid del messaggio ricevuto se è presente nella tabella **DL_EVENT_HUB_MESSAGE**, se il riscontro positivo si aggiorna lo stato del record associato nella tabella DL PACCHETTO Lotto, il messaggio viene depositato slla coda mlquorum

/ingestion	POST	Body composto da cantiereId pacchettoId	Check di esistenza coppia: cantiereId, pacchettoId nella tabelle opportune. Popolamento del messaggio per event-hub e chiamata ad esso. Salvataggio nella tabella DL_EVENT_HUB_INGESTION del messaggio inviato. Modifica dello stato del record associato nella tabella DL_PACCHETTO_LOTTO al valore 'Trasferito a ISPC'

3.10.3 Metriche

/actuator/health

- Restituisce un codice di stato HTTP 200 OK se l'applicazione è sana.

```
{
  "status": "UP"
}
```

- Il servizio restituisce 503 qualora non fosse disponibile

```
{
  "status": "DOWN"
}
```

3.10.4 Configurazioni

Sono disponibili su OCP per i relativi progetti (dev,preprod,prod) i secret contenenti le variabili di ambiente utilizzate dal micro-servizio.
Di seguito le variabili d'ambiente impostate:

1. Credenziali Azure Active Directory
   - A_AUTH_MODE
   - A_CLIENT_ID
   - A_CLIENT_SECRET
   - A_INSTANCE
   - A_SCOPE_BASE
   - A_TENANT_ID

2. Puntamenti al bucket S3 Cineca

- AWS_ACCESS_KEY_ID
  - Chiave di accesso del bucket S3
- AWS_REGION
- AWS_SECRET_ACCESS_KEY
  - Secret key del bucket S3
- BUCKET_NAME
  - Nome del bucket da richiamare
- OWNER_TO_GRANT
  - Id owner ISPC

3. Event-Hub

- DPAC_SOURCE
  - Valore predefinito fornito da ISPC => ESTRAIVMETS
- DPAC_TOPIC
  - Valore predefinito fornito da ISPC =>EV_ESITO_CHECK_ECOMIC
- EVENT_HUB_URL_CONSUMER
  - Endpoint fornito da ISPC relative al consumer
- EVENT_HUB_URL_INGESTION
  - Endpoint fornito da ISPC relativa alla procedura di ingestion del pacchetto
- EVENT_HUB_URL_PRODUCER
  - Endpoint fornito da ISPC relative al producer
- HAS_TO_SKIP
  - Variabile utilizzata come un workaround per l'ambiente di PRODUZIONE, impostato a false
- INGESTION_JWT
  - Json Web Token fornito statico da ISPC.

4. Credenziali database e RabbitMQ

- JDBC_PWD
  - Password di accesso al database relativa all'utente evidenziato nella JDBC_USER
- JDBC_URL
  - Connessione al database
- JDBC_USER
  - Utente da utilizzare per accedere al database
- RABBIT_MQ_HOST
  - Host da utilizzare per la coda RABBIT_MQ
- RABBIT_MQ_PORT

  - Porta da utilizzare per la coda RABBIT_MQ
- RABBIT_MQ_PWD
  - Password da utilizzare per la coda RABBIT_MQ
- RABBIT_MQ_USER
  - Utente da utilizzare per la coda RABBIT_MQ

5. Endpoint bucket S3 e API getUserInfo

- S3_USER_INFO
  - Endpoint riguardante la getUserInfo utilizzata per il controllo degli accessi ai cantieri rispettivamente all'utente loggato in DPAC

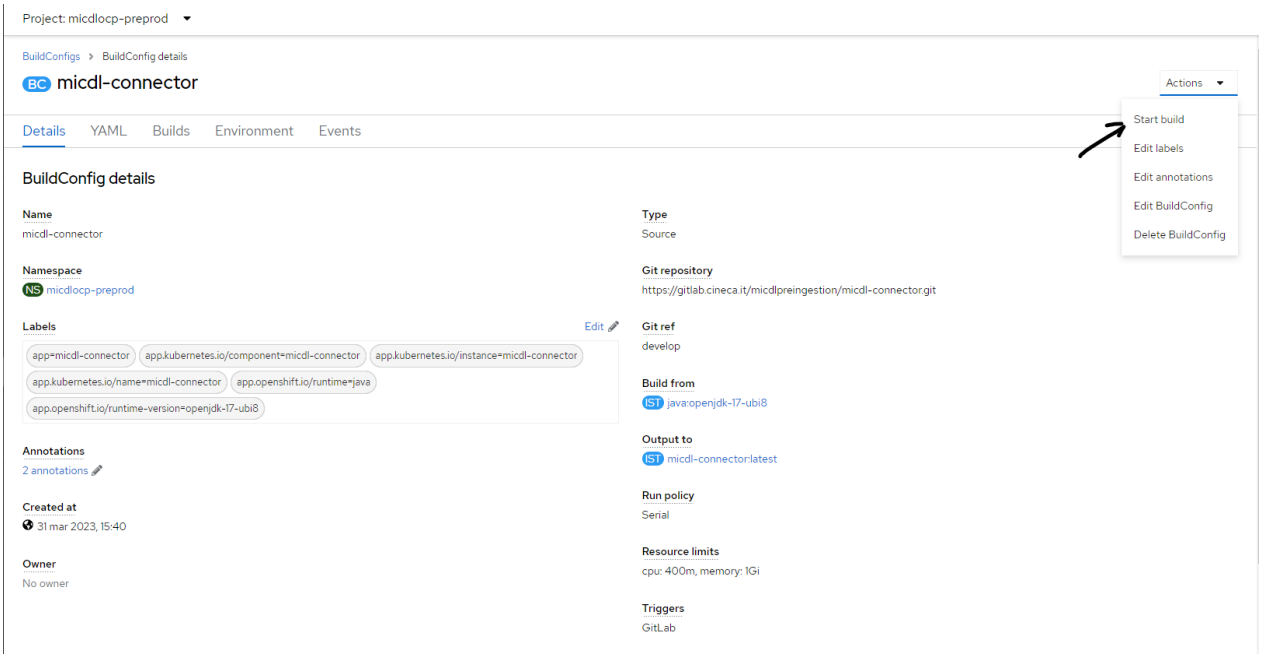
3.10.5 Dipendenze/Librerie utilizzate

- Lombok
- MapStruct
- Easy Random
- Spring Boot 3 (3.1.3)
- Amazon AWS SDK
- Oracle

3.10.6 Deploy

Per effettuare il deploy del micro-servizio, i passi sono i seguenti:

- Build del micro-servizio effettuato manualmente da OCP selezionando l'opzione "Start Build" presente nella sezione "Actions"



- In alternativa, per ogni ambiente distinto, risulta disponibile un trigger gitlab che al push di modifiche al codice sorgente procede autonomamente alla build del micro-servizio.
- Terminata la fase della build, accedere alla sezione "Deployments", selezionare il nome del deploy, e verificare che il pods sia effettivamente disponibile e in esecuzione.

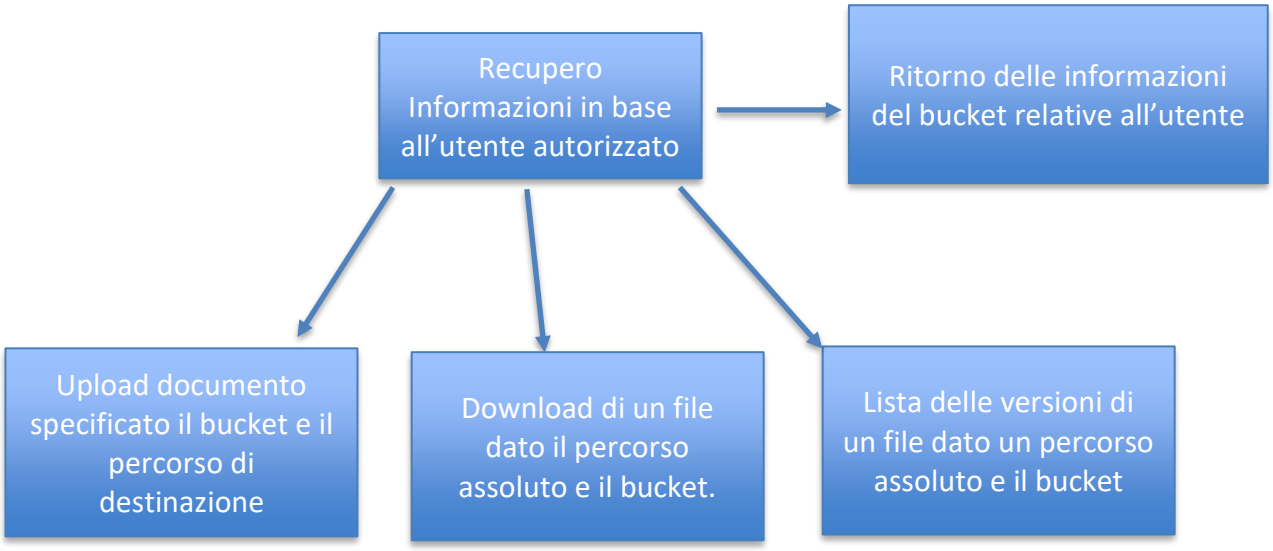
Qualora si renda necessaria qualche modifica alle variabili d'ambiente del micro-servizio, è necessario accedere solamente alla sezione "Deployments" selezionare il deploy di interesse, scalarlo a zero, e reimpostarlo nuovamente a "1", in modo da rendere disponibili le nuove modifiche alle variabili interessate.

3.11 S3ProxyFE

Micro-servizio che si occupa di indirizzare le richieste relative alle operazioni di lettura/scrittura sul bucket s3 previo recupero dati dell'utenza che ha provveduto ad effettuare l'accesso alla piattaforma.

3.11.1 Funzionalità

Lo scopo del micro-servizio "S3ProxyFE" è quello di fornire l'accesso alle operazioni di lettura/scrittura del bucket S3.
Nel dettaglio, espone diversi endpoint, nel seguente ordine:



3.11.2 API

Comandi	Tipo Metodo	Query	Eventi
/s3fe/{bucketName} -> getUserInfoBucket	GET	bucketName -> PathVariable token -> RequestHeader (JWT) tokenAuth -> RequestHeader(X-Auth-Token)	Tracciamento nella tabella DL_LOG_S3_ACCESS delle informazioni relative all'utenza loggata.

/s3fe/{bucketName}/** -> Upload File	PUT	token -> RequestHeader (JWT) tokenAuth -> RequestHeader(X-Auth-Token) file -> MultipartFile bucketName -> PathVariable	Tracciamento nella tabella DL_LOG_S3_ACCESS delle informazioni relative all'utenza loggata per l'operazione richiesta.
/s3fe/{bucketName}/** -> Download File o DownloadFilePerVersionId se impostato il parametro versionId	GET	token -> RequestHeader (JWT) tokenAuth -> RequestHeader(X-Auth-Token) bucketName -> PathVariable versionId ->RequestParam	Tracciamento nella tabella DL_LOG_S3_ACCESS delle informazioni relative all'utenza loggata per l'operazione richiesta.
/s3fe/{bucketName}/** -> versionList	POST	token -> RequestHeader (JWT) tokenAuth -> RequestHeader(X-Auth-Token) bucketName -> PathVariable	Tracciamento nella tabella DL_LOG_S3_ACCESS delle informazioni relative all'utenza loggata per l'operazione richiesta.

3.11.3 Metriche

/actuator/health

- Restituisce un codice di stato HTTP 200 OK se l'applicazione è sana.

```
{
  "status": "UP"
}
```

- Il servizio restituisce 503 qualora non fosse disponibile

```
{
  "status": "DOWN"
}
```

3.11.4 Configurazioni

Di seguito le variabili d'ambiente memorizzate all'interno della sezione Secrets:

- RTI_BUCKET
  - Variabile contenente il nome del bucket condiviso
- S3_ACCESS_KEY
  - Chiave di accesso del bucket S3
- S3_USER_INFO
  - Endpoint riguardante la getUserInfo utilizzata per il controllo degli accessi ai cantieri rispettivamente all'utente loggato in DPAC
- S3_ENDPOINT_URL:
  - Endpoint S3
- RTI_S3_ACCESS_KEY

    - Chiave di accesso del bucket condiviso
- JDBC_URL
    - Connessione al database
- JDBC_USER:
    - Utente da utilizzare per accedere al database
- S3_SECRET_KEY:
    - Secret key S3
- S3_SIGNIN_REGION:
- RTI_S3_SECRET_KEY:
    - Secret key bucket condiviso
- JDBC_PWD:
    - Password di accesso al database relativa all'utente evidenziato nella JDBC_USER
- DPAC_ROOT_FOLDER

3.11.5 Dipendenze/Librerie utilizzate

- Lombok
- MapStruct
- Easy Random
- Spring Boot 3 (3.1.3)
- Amazon AWS SDK
- Oracle

3.11.6 Deploy

Per effettuare il deploy del micro-servizio, i passi sono i seguenti:

- Build del micro-servizio effettuato manualmente da OCP selezionando l'opzione "Start Build" presente nella sezione "Actions"
- In alternativa, per ogni ambiente distinto, risulta disponibile un trigger gitlab che al push di modifiche al codice sorgente procede autonomamente alla build del micro-servizio.
- Terminata la fase della build, accedere alla sezione "Deployments", selezionare il nome del deploy, e verificare che il pods sia effettivamente disponibile e in esecuzione.

Qualora si renda necessaria qualche modifica alle variabili d'ambiente del micro-servizio, è necessario accedere solamente alla sezione "Deployments" selezionare il deploy di interesse, scalarlo a zero, e re-impostarlo nuovamente a "1", in modo da rendere disponibili le nuove modifiche alle variabili interessate.

3.12 MICDL-WFM

Micro-servizio Spring Boot che incapsula il prodotto Camunda per la gestione di processi (BPMN) offrendo API per la gestione dei workflow di dominio, le relative notifiche e le date effettive di inizio e fine di ciascun task; il micro-servizio offre inoltre un'interfaccia web per la configurazione e la gestione dello stato della piattaforma Camunda.

Riferimenti tecnici: Java 17, Spring Boot 2.7.16, Oracle DB, Camunda 7.18, build con Maven, properties configurate su variabili d'ambiente, deployato con immagine Docker basata su immagine eclipse-temurin:17-jdk su cluster OpenShift, con deployment configurato da leggere le variabili d'ambiente da config maps e secret del namespace di riferimento.

Dipendenze con altri servizi: Backoffice per gestione utente (validità e ruolo), Connector per la verifica ML.

3.12.1 Funzionalità

Il micro-servizio permette la gestione di processi, definiti a monte tramite file BPMN ed inseriti tra le risorse, che rispecchiano i flussi funzionali delle attività offerte dalla piattaforma. I file BPMN che rappresentano la definizione del processo sono salvati nel database e versionati, mentre le istanze dei processi rappresentano il flusso attivo, dove ogni attività da svolgere è un task. Le API offerte sono un *layer* sopra le API interne di Camunda e permettono di personalizzare la gestione dei processi, gestire le autorizzazioni di ruolo per poter operare su una determinata attività di un processo, interfacciarsi alle notifiche della piattaforma e aggiornare le date di inizio e fine di un processo.
Il riferimento tra istanza di processo e cantiere (ed eventuale lotto) avviene tramite le variabili di processo offerte da Camunda, una mappa dove è possibile inserire variabili custom: ogni ricerca o attivazione di un processo si basa su tali variabili.
Per Camunda un'istanza di processo vive solo finché è attivo: una volta completato, l'istanza termina e per tanto tutto lo storico va ricercato nell'history delle istanze di processo.
Le operazioni principali offerte sono:

- consultazione dei processi attivi o conclusi dato l'id di un cantiere
- l'avvio di un processo per un dato cantiere
- il completamento di un task di un dato processo (per id cantiere ed eventualmente anche per id lotto)
- la sospensione o la ripresa di un processo per un dato cantiere.

Queste attività sono tutte frutto di operazioni lato utente, quindi richiedono la ricezione della coppia di header Authentication e X-Auth-Token, che vengono usati per consultare il backoffice (servizio attivo all'interno del cluster) per verificare la validità dell'utente e che il ruolo ricoperto in quel cantiere sia valido per l'operazione richiesta.
Altre operazioni offerte per la piattaforma sono:

- avvio di un processo per un dato cantiere e lotto da parte di un processo automatico.

Infine, vi sono operazioni amministrative:

- pulizia di tutte le attività di un cantiere (utile in fase di test e collaudo).

3.12.2 API

Comandi	Metodo	Query	Eventi
/api/engine/processes	GET	RequestParam: (Long) cantiereId	Ricerca di tutti i processi attivi o completati per un dato cantiere, parametro che viene usato in fase di ricerca tra le variabili del processo.
/api/engine/processes/start	POST	RequestParam: (String) workflow RequestParam: (Long) cantiereId Opzionali: RequestParam: (String) business-Key RequestParam: (Long) lottoId	Avvia il workflow richiesto per un determinato cantiere (ed eventuale lotto), verificandone prima la validità della richiesta (se non è in corso o è stato già concluso quel workflow per quel cantiere/lotto).
/api/engine/processes/{proces-sId}/tasks/{taskId}/complete	POST	PathVariable: (String) processId PathVariable: (String) taskId RequestParam: (Long) cantiereId Opzionali: RequestParam: (String) esito	Completa il dato task per il dato processo, verificandone la validità della richiesta; alcuni task, se comportano una variazione di flusso in base all'esito dell'operazione, richiedono un parametro "esito" con valore OK o KO.
/api/engine/processes/suspend	PUT	RequestParam: (Long) cantiereId	Sospende tutti i processi attivi per il dato cantiere.
/api/engine/processes/resume	PUT	RequestParam: (Long) cantiereId	Riprende tutti i processi sospesi per il dato cantiere.
/internal/engine/processes/start	POST	RequestParam: (String) workflow RequestParam: (Long) cantiereId Opzionali: RequestParam: (String) business-Key RequestParam: (Long) lottoId	Dedicato solamente a richieste provenienti da processi automatici (es. avvio di processi schedulati per un lotto), avvia il workflow richiesto per un determinato cantiere (ed eventuale lotto), verificandone prima la validità della richiesta (se non è in corso o è stato già concluso quel workflow per quel cantiere/lotto).
/admin/history/clear	DELETE	RequestParam: (Long) cantiereId	Cancella tutti i processi attivi o pregressi dato un cantiere.

3.12.3 Metriche

/actuator/metrics:
Endpoint di Spring Boot Actuator che offre una serie di metriche sull'applicazione, richiamabili concatenando actuator/metrics/{metric_name}

/actuator/info:
Endpoint di Spring Boot Actuator che fornisce informazioni di base sull'applicazione.

/actuator/health

- Restituisce un codice di stato HTTP 200 OK se l'applicazione è sana.

```
{
        "status": "UP",
        "groups": [
                "liveness",
                "readiness"
        ]
}
```

Gli elementi presenti in groups possono essere utilizzati per info su liveness e readiness, utili per le sonde di OpenShift:

/actuator/health/liveness

- Restituisce un codice di stato HTTP 200 OK se l'applicazione è sana.

```
{
        "status":"UP"
}
```

/actuator/health/readiness

- Restituisce un codice di stato HTTP 200 OK se l'applicazione è sana.

```
{
        "status":"UP"
}
```

3.12.4 Configurazioni

Esiste un profilo dev per attività di debug per fare partire il servizio puntando l'ambiente di sviluppo, seguendo le configurazioni riportate negli application-dev.yml.

Il servizio, deployato nel cluster OpenShift, non sfrutta alcun profilo e parte seguendo le configurazioni riportate nel file application.yml. Qui, le variazioni in base all'ambiente vengono prese da variabili di ambiente, che sono iniettate via configurazione del Deployment in base a ConfigMaps e Secret.
I file di build dell'immagine Docker sono riportati nella cartella /docker e sono operative: la BuildConfig su OpenShift legge questi file e crea l'immagine Docker sulla base delle istruzioni ivi riportate.
I file presenti nella cartella /config sono solo di riferimento e riportano principalmente le configurazioni di ConfigMaps e Secret dei vari ambienti, ma non sono operative quindi per eventuali modifiche si devono modificare le relative voci sul cluster OpenShift.

Vengono riportate le variabili di ambiente attive

- Da ConfigMaps:
  DB_CONNECTION_TIMEOUT: tempo di timeout per la connessione al DB
  DB_MIN_IDLE: tempo di idle ammesso per la connessione al DB
  OAUTH2_JWT_ISSUER_URI: URI issuer JWT token OAuth2
  VERIFICA_ML_URI: URI servizio verifica ML (servizio MICDL-Connector)
  DB_MAX_LIFETIME: lifetime per connessione DB
  DB_MAX_POOL_SIZE: size pool connessioni a DB
  CLAIM_FISCAL_CODE: claim nel JWT per indicare codice fiscale
  CMND_ADM_USER_NAME: nome utente Camunda
  CLAIM_USER: claim nel JWT per indicare user
  BACKOFFICE_URI: URI backoffice (Micdl-Server) per verifica ruolo utente
  DB_IDLE_TIMEOUT: timeout idle connessione DB

- Da Secret:
  CMND_ADM_USER_ID: username utente Camunda
  CMND_ADM_USER_PWD: password utente Camunda
  DB_PWD: password DB
  DB_URL: connection URL DB
  DB_USERNAME: user DB

3.12.5 Dipendenze/Librerie utilizzate

Spring-boot: 2.7.16
- spring-boot-starter-web
- spring-boot-starter-data-jpa
- ojdbc8
- camunda-bpm-spring-boot-starter-webapp
- camunda-bpm-spring-boot-starter-rest

- camunda-engine-rest-openapi
- springdoc-openapi-ui
- jaxb-impl
- spring-boot-devtools
- lombok
- spring-boot-starter-test
- spring-boot-starter-oauth2-resource-server
- spring-boot-starter-actuator

Servizi all'interno del cluster:
- micdl-server
- MICDL-Connector

Servizi esterni:
- issuer OAuth2

3.12.6 Tecnologie

Linguaggio: Java 17
Build e dipendenze: Maven
Spring Boot: 2.7.16
Camunda: 7.18
DB: Oracle

3.12.7 Deploy

La build in ambiente di sviluppo parte in automatico alla ricezione del commit sul ramo develop. Il webhook è configurato con un secret per l'utenza Gitlab con cui recuperare il codice.
Le build in ambiente di collaudo (ramo preprod) e produzione (ramo prod) avvengono solo manualmente, ma hanno anche in quei namespace dei secret con l'utenza Gitlab per recuperare il codice.

3.12.8 Camunda Web Console

Camunda offre un'interfaccia web dove è possibile controllare lo stato dei processi, le definizioni disponibili, gli utenti che possono accedere all'interfaccia, l'attività dei gruppi di utenti (sia lato interfaccia che lato utenti che operano nelle istanze del BPMN).

La URL è raggiungibile dalla url base dell'istanza concatenando /camunda, si verrà riportati alla schermata di login: user e password per accedere sono configurati nei secret del *namespace* di riferimento nel cluster.

3.13 PYTHON-RECOGNITION

La procedura Python riceve in input dalla coda RabbitMQ i riferimenti dei pacchetti di immagini caricati in ambiente S3, crea le miniature delle stesse e inserisce i relativi metadati all'interno di un'apposita tabella del Database. Il servizio effettua successivamente una chiamata API al micro-servizio "MICDL-Connector" a conclusione dell'elaborazione del pacchetto.

3.13.1 Funzionalità

Il servizio si attiva al ricevimento di un messaggio all'interno della coda RabbitMQ e richiama a cascata le seguenti funzioni:
main.py
lettura_S3.py
db_pacchetto_id.py
db_update.py
thumbnail_creation.py
api_finale.py
Tali funzioni, come verrà descritto nei paragrafi successivi, inseriscono in una tabella apposita i metadati delle immagini contenute nel pacchetto ricevuto in input e creano le miniature delle immagini stesse su S3.
La procedura a fine elaborazione del pacchetto effettua una chiamata POST al servizio "MICDL-Connector".
Per la scrittura del log è stata predisposta la funzione log_writer.py.

3.13.2 API

Comandi	Metodo	Query	Eventi
Api_finale.py	POST	response = requests.request("POST", cf.url_api+'/recognition', data={"message": message}, files={"xmlFile": (file_xml_decode)})	Invio dei riferimenti del pacchetto di immagini di cui si è conclusa l'elaborazione.

3.13.3 Configurazioni

Elenco dei file di configurazione del servizio Python Recognition.

Requirements.txt:
Elenco delle librerie necessarie per l'esecuzione della procedura Python.

log_config.py
File contenente i parametri necessari per la scrittura del log e il riferimento per percorso di salvataggio dei file di log.

config.py
Nel file di config.py sono richiamate le seguenti variabili di ambiente (di alcune si riporta una breve descrizione perché non immediatamente deducibile dal nome assegnato):

FILE di LOG

PATH_LOG = percorso di salvataggio file di log

RABBITMQ

RABBIT_PK_USER
RABBIT_PK_PSW
RABBIT_PK_ADRESS
RABBIT_PK_PORT
RABBIT_QUEQUE_NAME

S3

AWS_SERVICE_NAME
AWS_ENDPOINT_URL
AWS_ACCESS_KEY_ID
AWS_SECRET_ACCESS_KEY
AWS_REGION_NAME

Thumbnail

THM_MAX_WIDTH = larghezza massima per la creazione della thumbnail
THM_MAX_HEIGHT = altezza massima per la creazione della thumbnail
SUB_PATH_LCL = parametro per la creazione di una directory all'interno del pacchetto di cui è stato effettuato il download
SUB_PATH_S3 = parametro per la creazione di una directory all'interno del pacchetto in S3, in cui vengono caricate le miniature
THM_FRM_1, THM_FRM_2 = formati immagini per i quali viene creata la thumbnail

ORACLE

CX_DB_ADDRESS
CX_DB_PORT
CX_DB_SERVICE_NAME
CX_DB_USER
CX_DB_PSW

ORACLE Table List

TAB_ANAG_CNT
TAB_ANAG_LOTTO
TAB_ANAG_PK
TAB_ANAG_FRM
TAB_OGG_PK
TAB_LOG

API SERVIZI

API_ADDRESS = url utilizzata nella chiamata POST a fine elaborazione del pacchetto

3.13.4 Scripts

Invocazione metodo	Evento scatenato
main.py	Funzione che riceve in input il messaggio della coda RabbitMQ e richiama la funzione read_s3() che sulla base dei parametri presenti nel messaggio della coda effettua la lettura degli oggetti presenti in S3. Il messaggio contiene indicazione del Bucket e della Directory (CANTIERE, LOTTO, PACCHETTO) in cui sono presenti i file da processare. Il primo step eseguito dalla funzione è la creazione della directory di log (se non già presente).
lettura_S3.py	La funzione riceve in input il messaggio della coda RabbitMQ e sulla base di questo effettua le seguenti operazioni: - lettura degli oggetti presenti nel pacchetto - inserimento o aggiornamento dei metadati degli oggetti presenti nel path indicato nel messaggio (Bucket, Cantiere, Lotto, Pacchetto) nella tabella Oracle DL_PACCHETTO_OGGETTO mediante il richiamo alla funzione db_update.py - Download degli oggetti di tipo immagine jpg, jpeg e creazione relativa miniatura con nome analogo. Upload della miniatura su S3 mediate la funzione thumbnail_creation.py - chiamata api finale

db_pacchetto_id.py	La funzione legge dal DB ORACLE gli ID del pacchetto, lotto e cantiere e del tipo materiale. L'output della funzione è utilizzato da db_update.py e api_finale.py.
db_update.py	Funzione che scrive nel DB ORACLE (tabella DL_OGGETTO_PACCHETTO) i metadati degli oggetti presenti al path di S3 indicato nel messaggio della coda RabbitMQ. La funzione riceve in input anche gli ID del pacchetto, lotto, cantiere precedentemente letti sul DB. La funzione aggiorna la tabella DL_OGGETTO_PACCHETTO: se l'oggetto è già presente nella tabella il record relativo viene aggiornato altrimenti viene inserito un nuovo record.
thumbnail_creation.py	Funzione che crea la miniatura delle immagini jpg, jpeg e le carica su S3. I passi eseguiti dalla funzione nell'ordine sono: 1. creazione della miniatura e salvataggio di questa: la miniatura mantiene il nome dell'immagine originale 2. caricamento della miniatura in S3: al path su S3 dove presente l'immagine originale viene inserita una directory THUMBS che contiene la miniatura. 3. cancellazione dell'immagine originale (di cui era stato effettuato il download) e della miniatura
api_finale.py	Funzione che effettua una chiamata POST a conclusione dell'elaborazione del pacchetto indicato nel messaggio RabbitMQ ricevuto in input.
Log_writer.py.	Funzione che esegue la scrittura del log (dell'intero processo "PYTHON RECOGNITION") all'interno della tabella Oracle DL_PYTHON_RECOGNITION_LOG. La funzione viene richiamata in vari punti delle funzioni main(), read_s3(), thumbnail_creation(), api_end()

3.13.5 Deploy

La pubblicazione del micro-servizio può avvenire, per ogni ambiente distinto, attraverso un trigger Gitlab che al push di modifiche del codice sorgente procede autonomamente alla build del micro-servizio.
Tale pubblicazione avviene mediante l'utilizzo di dockerfile.

4 Virtual Machines

Descrizione dei servizi ospitati su VMs.

4.1 Quality Assurance – IQM

Il servizio IQM verifica che le immagini digitalizzate rispettino i requisiti imposti. Viene attivato dal micro-servizio "MICDL-Connector" subito dopo l'esito positivo della verifica METS. Ogni immagine elaborata viene sottoposta a più controlli e l'esito di ciascuno viene memorizzato in una tabella Oracle per poi essere visualizzato in ambito del collaudo.

4.1.1 Utilizzo

Per avviare IQM, va invocato il relativo file eseguibile.
Di seguito, la lista completa degli step necessari per eseguire IQM sulla macchina di produzione:

1. Assicurarsi di essere connessi alla VPN di Cineca[1]. Le metodologie di accesso sono molteplici, un possibile esempio, utilizzando openconnect, è il seguente:

   sudo openconnect -u <your_email> vpnconcentrator.cineca.it

2. Una volta assicurato l'accesso alla VPN, bisognerà procedere all'accesso alla macchina remota. Anche qui, le modalità di accesso sono molteplici, ad esempio usando ssh[2].

   ssh <your_user>@10.100.2.220

3. Una volta inserita la password verrà effettuato il login del tuo utente. A questo punto seguire il seguente percorso[3].

   cd /production/iqm/prod

4. Infine, per avviare l'esecuzione, semplicemente invocare il comando:

   ./main/main

[1] Per maggior informazioni, consultare il https://customerportal.cineca.it/.
[2] Nota: il nome utente e/o l'ip della macchina remota potrebbero essere soggetti a modifiche.
[3] Nella cartella deploy, è salvata una copia dei modelli/files di configurazione di default.

4.1.2 Flusso dati

Data l'architettura generale dell'infrastruttura, il software IQM comunica costantemente con gli altri moduli utilizzando due mezzi: una coda RabbitMQ e un OracleDB. La coda viene utilizzata per ricevere dati relativi all'esecuzione di un singolo pacchetto, mentre il database viene utilizzato per memorizzare i risultati ottenuti dalle singole esecuzioni. Inoltre, al fine di scaricare in locale i dati che andranno processati, è necessaria la comunicazione con un server S3. Per tal motivo, la comunicazione con le suddette risorse è necessaria per il funzionamento di IQM.

4.1.3 Logging

IQM si occupa di loggare in automatico tutti gli eventi che accadono durante il processamento dei dati. Data un'esecuzione, verranno creati molteplici file di log, all'interno della cartella di *logging*. Verrà creato in automatico un file per il *logging* relativo alla singola esecuzione del processo e, allo stesso tempo, molteplici file di log per i singoli pacchetti processati. Inoltre, informazioni aggiuntive verranno loggate direttamente nel database di produzione.

4.1.4 Configurazioni

Determinati aspetti di IQM sono configurabili attraverso dei semplici, human-readable file di configurazione, che sono memorizzati nella cartella config in dist.
Di seguito, la lista dei vari file di configurazione:

- config.json: contiene la definizione dei derivati e delle loro caratteristiche. Ciò comprende il nome della directory del derivato, l'estensione e il numero di dpi.
- config.yaml: contiene diversi tipi di informazione.
    1. I nomi dei vari file di configurazione .yaml, in base al loro scopo. Ciò è utile nel caso in cui si voglia cambiare un determinato file di configurazione senza modificare il file di configurazione originale.
    2. I parametri di logging.
    3. I parametri di connettività per la coda RabbitMQ.
    4. I parametri per la connettività al server S3.
- db_config.yaml: contiene tutte le specifiche relative alla connessione al server OracleDB e i riferimenti alle tabelle/colonne usate come schema.
- model_config.yaml: contiene le informazioni relative ai modelli di Machine Learning utilizzati da IQM.
- process_config.yaml: contiene dei parametri che configurano il funzionamento di alcuni algoritmi presenti in IQM.

4.1.5 Dipendenze/Librerie utilizzate

Le dipendenze si suddividono in due categorie: **di sistema** e **integrate**. Le prime sono dipendenze che devono essere installate sul sistema operativo che eseguirà il programma, mentre le seconde sono dipendenze incluse già nel programma stesso e che sono automaticamente presenti ogni volta che si copia l'eseguibile e tutti i file correlati.

4.1.5.1 Dipendenze di sistema

La lista di dipendenze che potrebbero non essere presenti sul sistema e che sono necessarie all'esecuzione del software:

- python = 3.10.12
- build-essential
- libc6
- zlib1g
- cmake
- git
- perl
- vim
- vim-common
- vim-runtime
- wget

4.1.5.2 Dipendenze integrate

La lista di dipendenze presenti internamente al software stesso:

- flake8 ≥ 6.0.0
- pytest ≥ 7.2.2
- black ≥ 23.1.0
- pre-commit ≥ 3.1.1
- isort ≥ 5.12.0
- pytest-cov ≥ 4.0.0
- pydantic ≥ 1.10.6 3
- sqlalchemy-stubs ≥ 0.4
- torch ≥ 2.0.0
- torchvision ≥ 0.15.1
- torchaudio ≥ 2.0.1
- lightning ≥ 2.0.0
- transformers ≥ 4.27.3

- ultralytics ≥ 8.0.58
- hydra-core ≥ 1.3.2
- onnx ≥ 1.13.1
- onnxruntime-gpu ≥ 1.15.0
- pytest-mock ≥ 3.10.0
- mypy ≥ 1.1.1
- datasets ≥ 2.11.0
- tensorboard ≥ 2.12.0
- sphinx ≥ 6.1.3
- sphinx-rtd-theme ≥ 1.2.0
- cryptography ≥ 40.0.1
- opencv-python ≥ 4.7.0.72
- alive-progress ≥ 3.1.3
- boto3 ≥ 1.28.38

4.2 Axway – Trasferimento pacchetti di contenuto

Il manuale di gestione applicativo del modulo di Axway è composto da due guide e sono allegate esternamente a questo documento:

- 1-DPAC_SecureTransport_Explotation Guide.pdf
- 2-MC-DPAC_SUMMARY.pdf

5 Message Brocker RabbitMQ

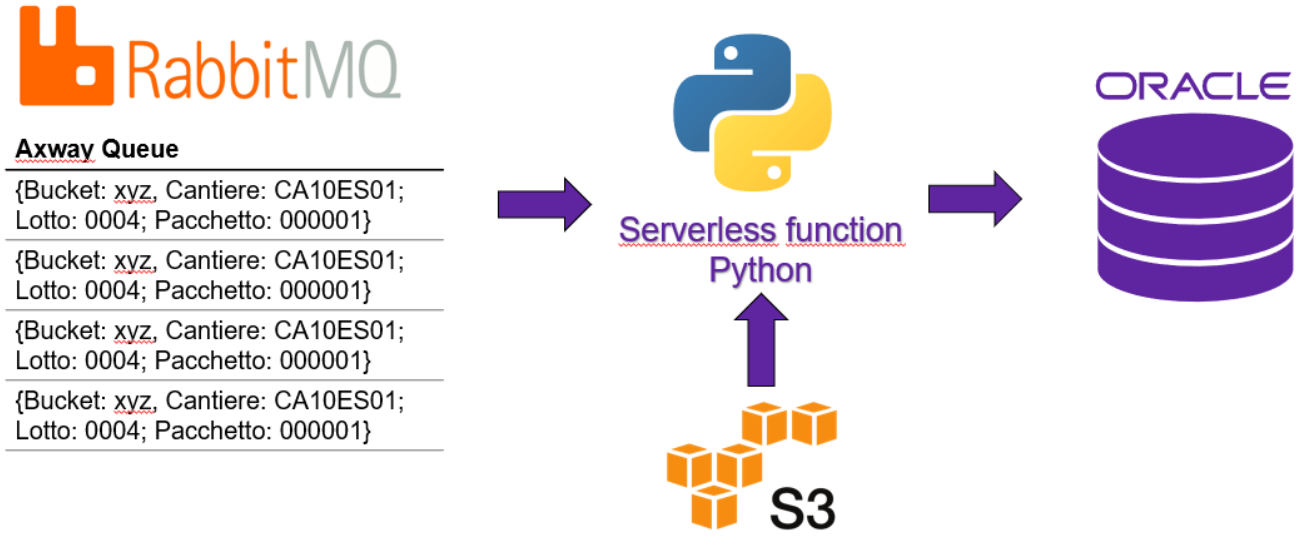
L'utilizzo di questo message brocker è utile affinché i servizi esterni in esecuzione sulle VMs possano comunicare con i micro-servizi che compongono la piattaforma DPaC.

Sono state definite due code di messaggi:

- *axwayqueue*
  utilizzata dal servizio Axway per informare l'arrivo di un pacchetto di contenuto
- *mlquorum*
  utilizzata dal servizio di Quality Assurance per essere informato della presenza di un pacchetto da poter elaborare

5.1 Coda "axwayqueue"

Gli attori coinvolti su questa coda sono il servizio di **Secure Transport di Axway** che ricopre il ruolo di produttore di messaggio e una o più repliche del servizio **PythonRecognition** che sono i consumatori dei messaggi.
Quando un pacchetto di contenuto arriva su DPAC viene decompresso, i file al suo interno vengono trasferiti sullo storage S3 e viene pubblicato un messaggio sulla coda "*axwayqueue*".
Il **micro-servizio "PythonRecognition"** in ascolto sulla coda, riceve il messaggio ed in base al suo contenuto ne esegue l'elaborazione.

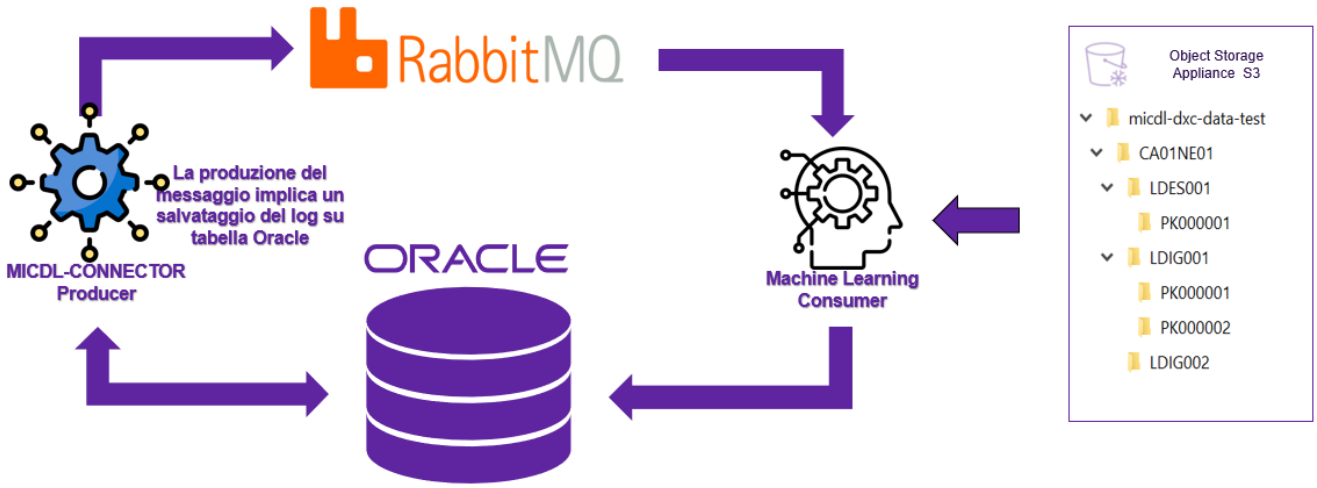


Formato del messaggio:
{Bucket: nome_bucket, Cantiere: codice_cantiere, Lotto: codice_lotto, Pacchetto: codice_pacchetto}

Esempio:
{Bucket: micdl-dxc-data-dev, Cantiere: MU04S204, Lotto: LDIG001, Pacchetto: PK0000007}

5.2 Coda "mlquorum"

Gli attori coinvolti sulla coda "*mlquorum*" sono il **micro-servizio "MICDL-Connector"** ed il **micro-servizio "Quality Assurance" (QA)** che risiede sulla VM.
Il primo ricopre il ruolo di produttore di messaggio mentre uno o più servizi di "QA" assumono i ruoli di consumatori.
Dopo che il pacchetto è stato acquisito viene sottoposto al controllo antivirus e poi sottoposto a "Verifica METS". In caso di esito positivo, il **micro-servizio "MICDL-Connector"** produce un messaggio sulla coda "*mlquorum*" dando indicazioni necessarie al **micro-servizio "QA"** di poter iniziare la sua elaborazione.



Formato del messaggio:
{"idMessaggio": identificativo_numerico, "codCantiere":codice_cantiere,"idCantiere":identificativo_cantiere,"idLotto":identificativo_lotto,"idTipologiaMateriale":identificativo_tipologia_materiale,"idPacchetto":identificativo_pacchetto,"bucketS3":nome_bucket,"path":path_completa_pacchetto}

Esempio:

{"idMessaggio": 334,
"codCantiere":"MU05CS02","idCantiere":102,"idLotto":225,"idTipologiaMateriale":11,"idPacchetto":1154
,"bucketS3":"micdl-dxc-data-test","path":"DPAC/MU05CS02/LDIG001/PK0000024/"}

6 Modello dati

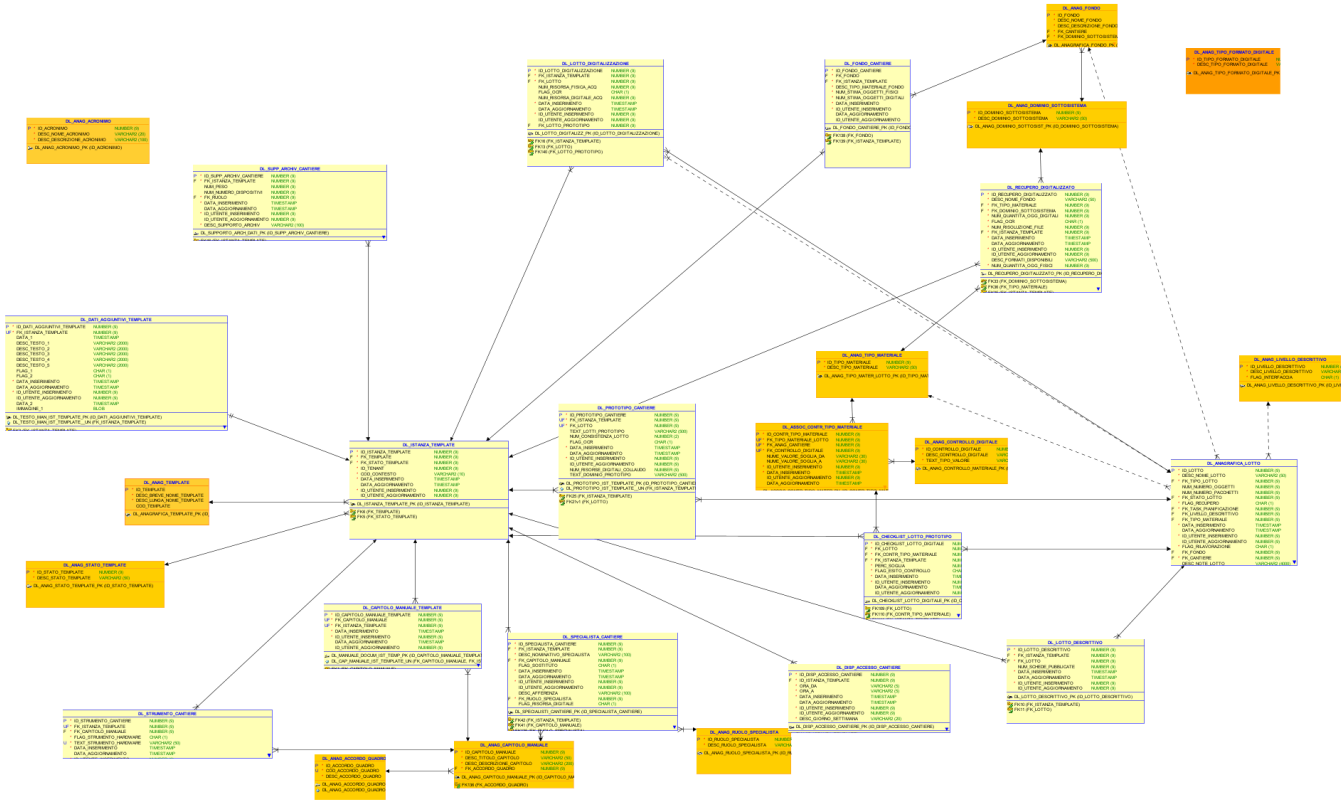
Il modello dati della sola piattaforma DPaC è stato suddiviso per aree tematiche per consentire ai micro-servizi di poter funzionare in maniera autonoma.

Le principali aree tematiche individuate sono le seguenti:

- Template (generatore report)
- Configurazione cantiere
- Collaudo
- Documentale
- Gantt (pianificazione)
- Notifiche
- ODA
- Upload

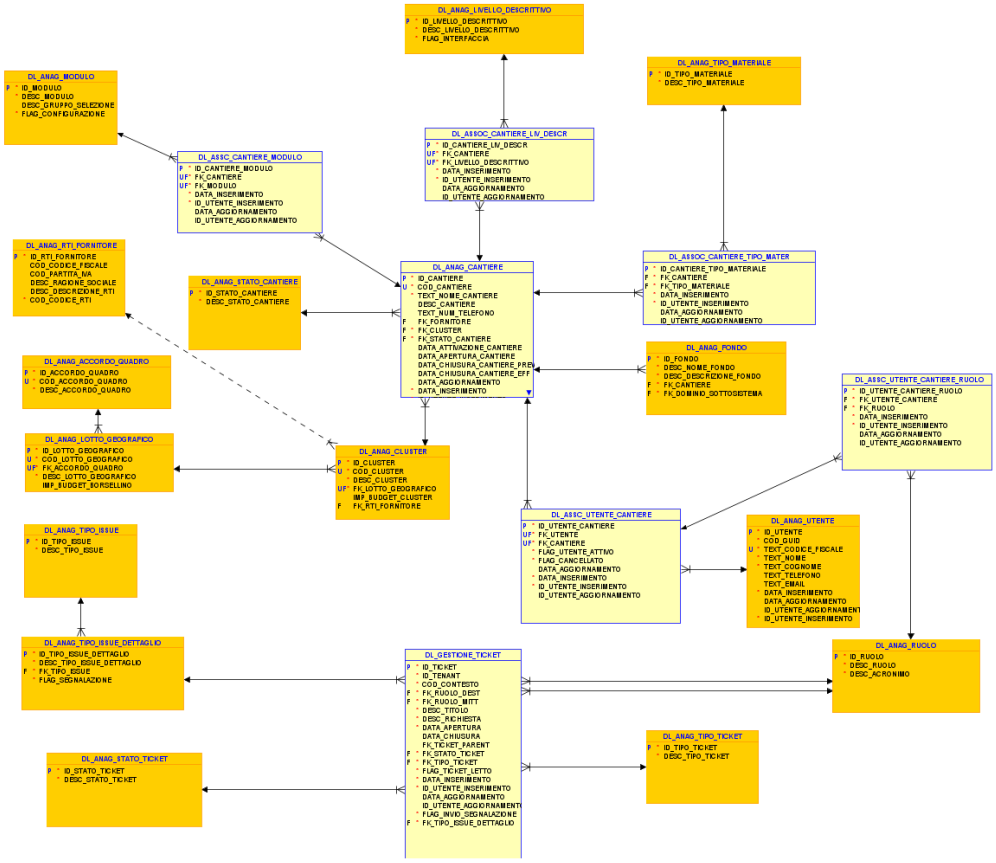
6.1 Template (generatore report)

Area DB dedicata al micro-servizio che si occupa della generazione dei reports.
I reports sono costituiti da una parte statica ed una parte dinamica che viene memorizzata sulle tabelle rappresentate nella figura sottostante.



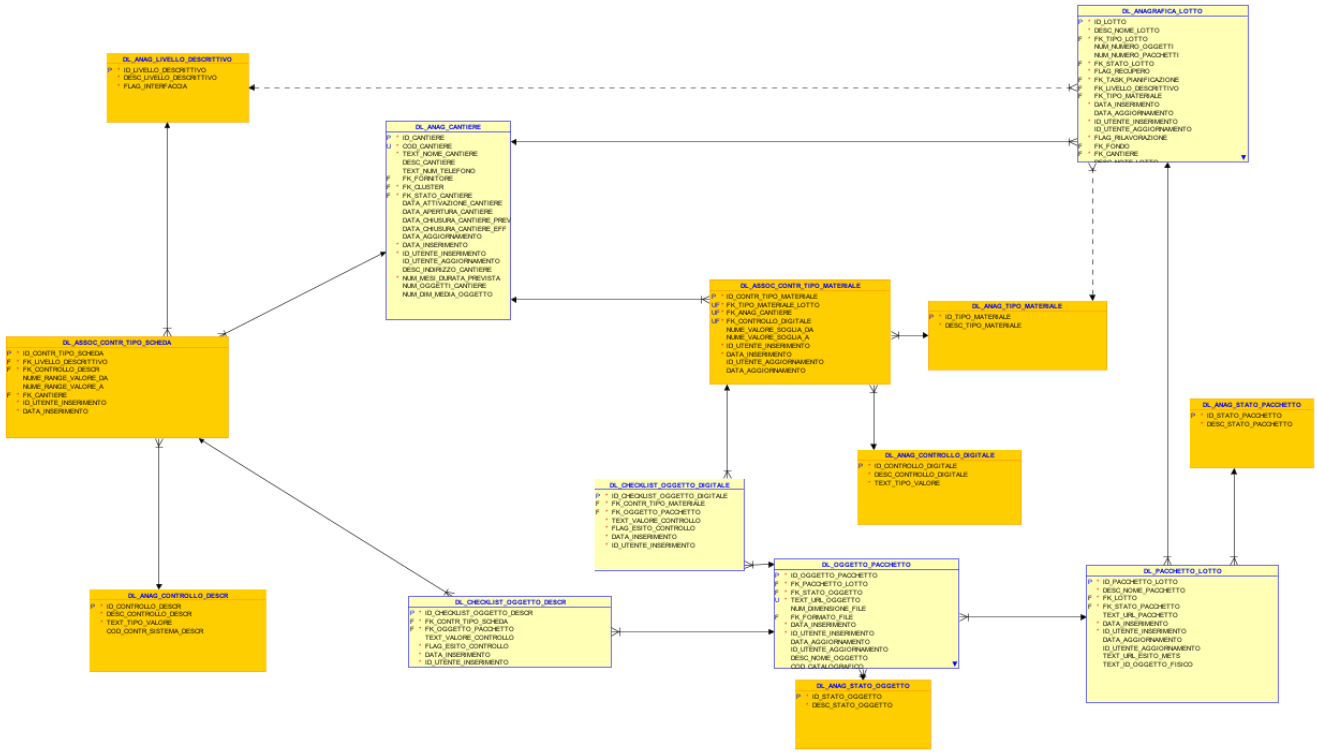
6.2 Configurazione Cantiere

Quest'area del database riguarda l'aspetto della configurazione del cantiere: anagrafica, correlazione con gli utenti, ruoli, moduli abilitati, accordo quadro, fondi e materiale.



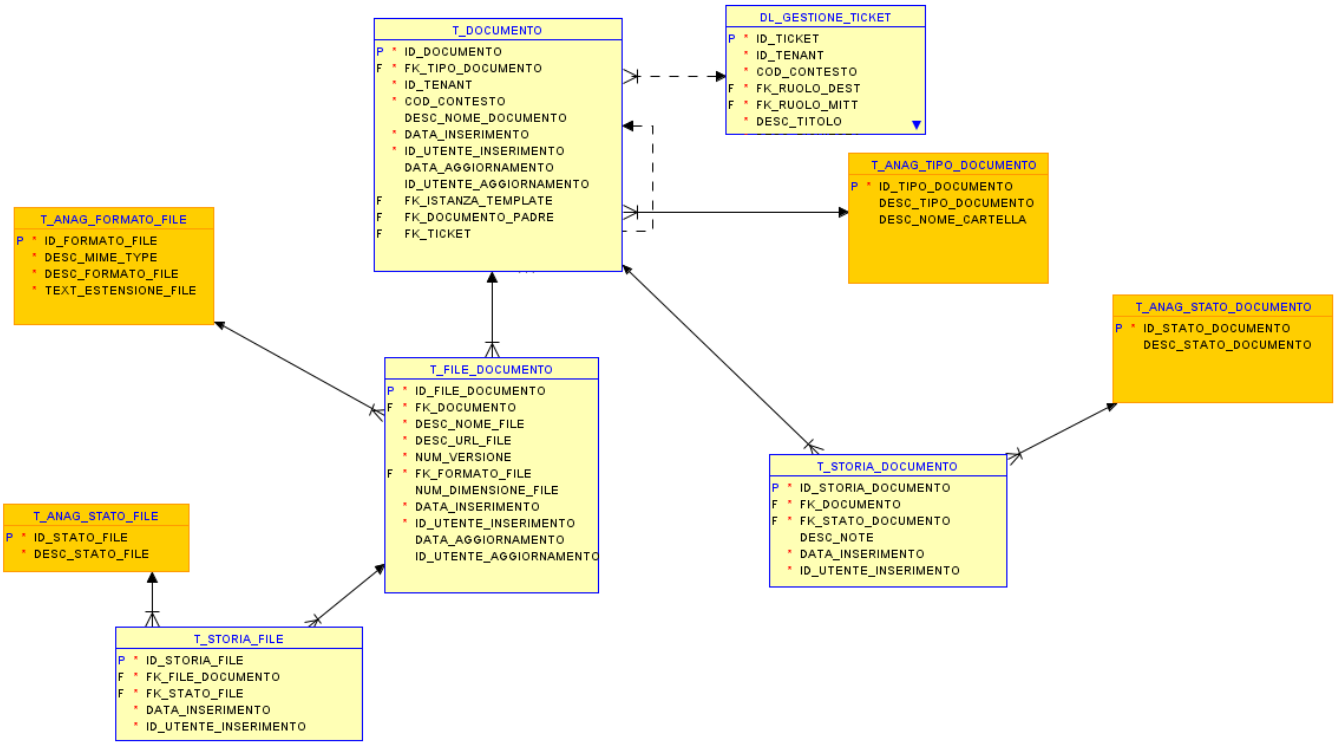
6.3 Collaudo

Area del database dedicata al servizio di collaudo, contenente le anagrafiche dei controlli configurati per gli oggetti digitali e i controlli post pubblicazione delle schede descrittive e i dati ottenuti attraverso l'esecuzione dei controlli eseguiti sui singoli oggetti digitali/schede descrittive.



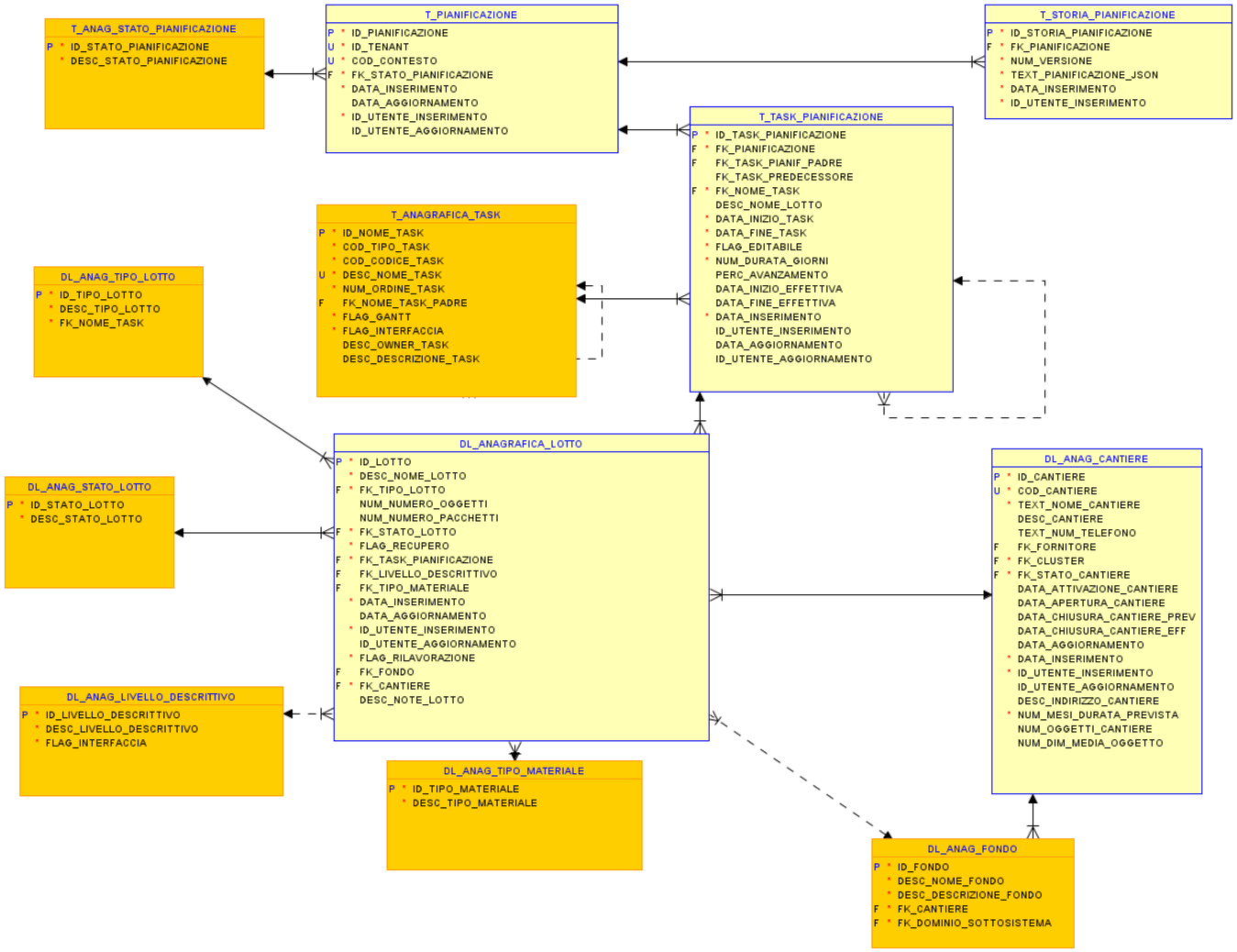
6.4 Documentale

Area del database dedicata al servizio di gestione documentale, che consente di tracciare il flusso dei documenti prodotti sulla piattaforma DPaC.
Per ogni documento prodotto sarà tracciato lo stato, il produttore del documento, gli utenti da cui è stato scaricato per la lettura, l'approvazione o il rifiuto, eventuali note aggiuntive.



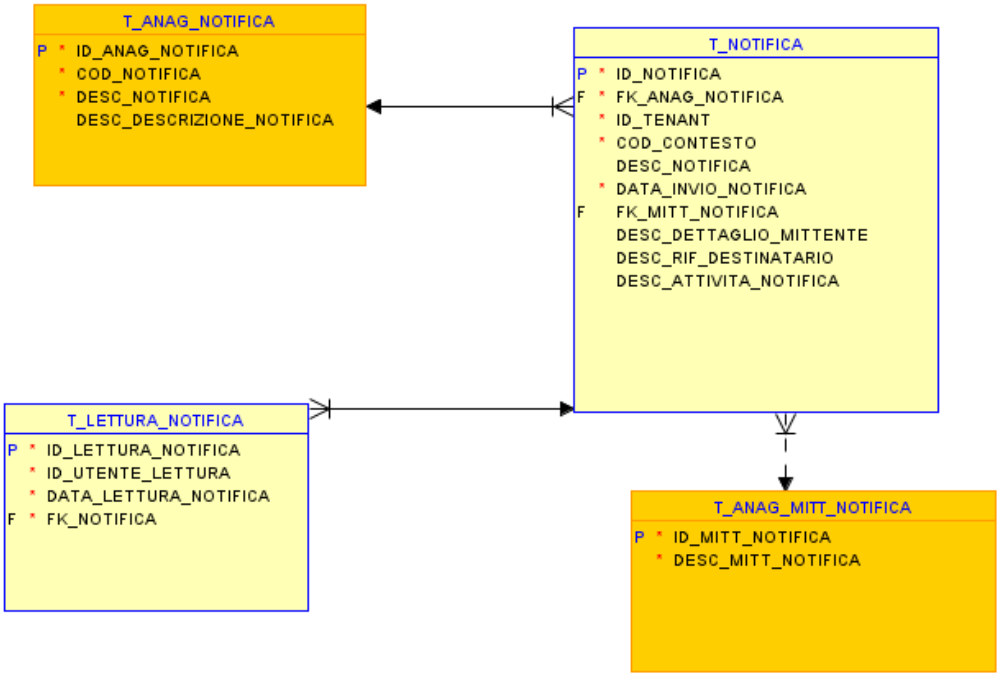
6.5 Gantt (Pianificazione)

Area del database dedicata al servizio di pianificazione delle diverse attività previste per ciascun cantiere oltre ai dati anagrafici di ciascun lotto di lavorazione.



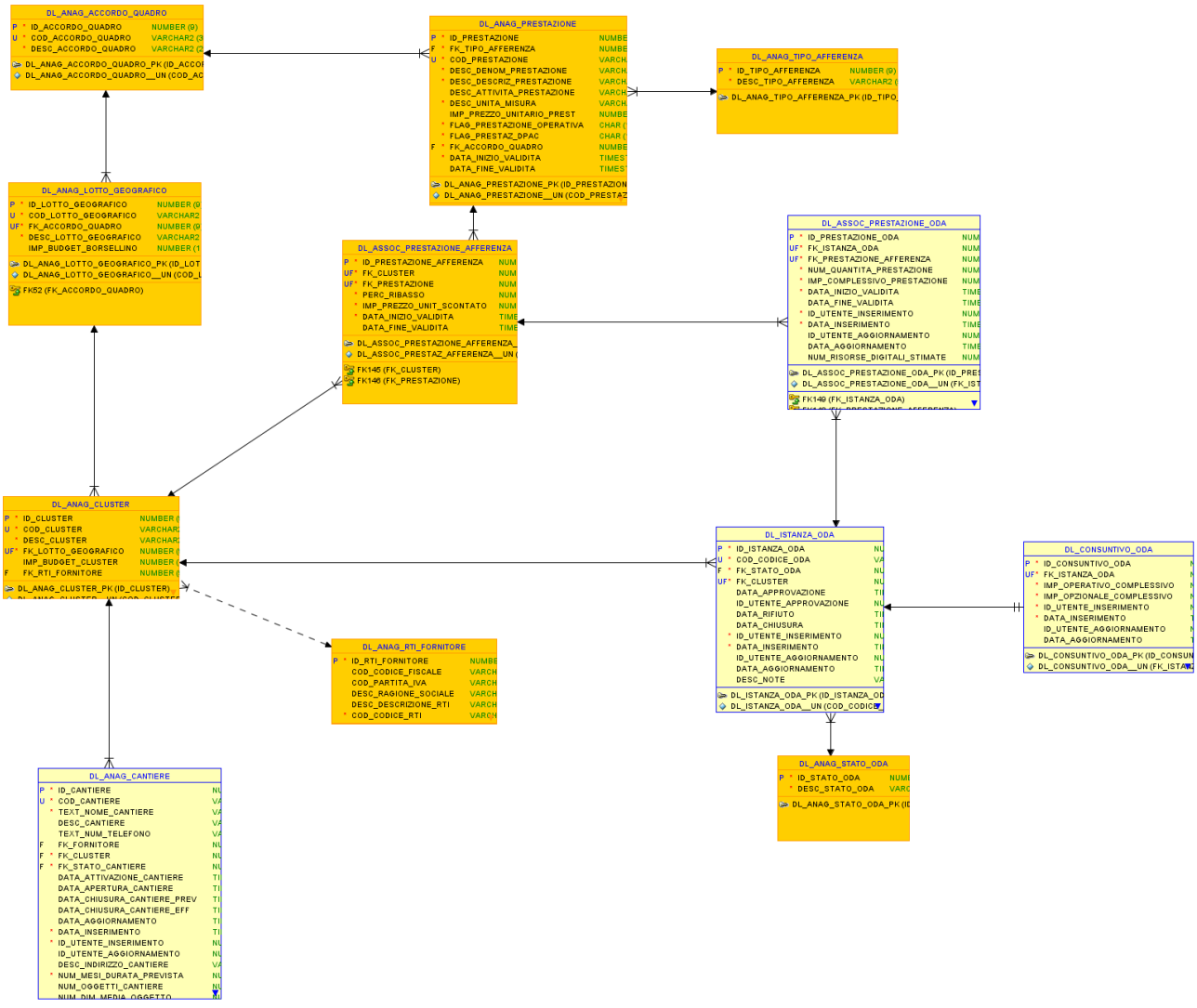
6.6 Notifiche

Area del database dedicata al servizio di gestione delle notifiche inviate dai processi BPMN verso la piattaforma DPaC.



6.7 ODA

Area del database dedicata al servizio di simulazione ODA, che consente la gestione degli ordini di acquisto grazie alla rendicontazione del budget residuo a livello di cluster e di borsellino; l'area comprende le anagrafiche necessarie, es. prestazioni, cluster, lotto geografico etc., e per ciascuna istanza ODA, i dati relativi alle singole prestazioni e l'importo complessivo opzionale e operativo.



6.8 Upload

Area del database dedicata al servizio di upload dei file prodotti attraverso la digitalizzazione, contente i dati anagrafici di supporto e la componente dati dove vengono tracciati i file prodotti.