

Progetto in itinere per il corso di **Distributed System and Big Data**

Federico Cucci
Gianluca Misenti

Matricola: 1000013454
Matricola: 055000412

Relazione progetto : DOCAUTH

Un sistema di verifica dell'autenticità dei documenti basato su microservizi.

PREMESSA

Lo scopo di questo progetto è quello di permettere la verifica dell'autenticità di un documento quando questo viene trasferito utilizzando canali non sicuri.

Se Alice, dopo aver prodotto un documento, lo invia a Bob attraverso una mail, o tramite mezzi fisici, come chiavette usb affidate a terzi o altre tipologie di supporti, non potrà essere sicura che il file inviato sia esattamente una copia dell'originale, dato che la copia elettronica di un documento può essere modificata/falsificata. Allo stesso modo Bob potrebbe voler verificare che il documento ricevuto non sia stato alterato in alcun modo.

Questa categoria di problema è attualmente risolta attraverso l'uso di tecnologie di cifratura asimmetrica, utilizzando, ad esempio, una firma digitale e, di conseguenza, una chiave pubblica e una privata.

In questo progetto mostriamo una semplice applicazione alternativa per la verifica dell'autenticità dei documenti, che non pretende ovviamente di sostituirsi ai metodi esistenti, ma che offre lo spunto di testare e sperimentare le tecniche di programmazione basate su microservizi apprese nel corso delle lezioni.

Workflow

Un utente si registra al servizio **Docauth** utilizzando un nome utente univoco e una password.

Questa operazione crea un'identità digitale all'interno del sistema, e ne abilita le operazioni di caricamento di un documento in archivio o di verifica di un documento precedentemente caricato.

Un'identità digitale per il sistema è tanto un Publisher, in grado di inserire documenti originali che poi possano essere verificati, quanto un Verifier, ovvero un utente interessato alla verifica che il documento che ha in mano, ottenuto tramite mezzi non sicuri, sia una copia conforme all'originale.

La registrazione di una nuova identità digitale comporta l'accredito di 5 token di benvenuto.

Il token è la moneta utilizzata all'interno di **Docauth**. Il servizio di upload di un documento costa 1 token. Il servizio di verifica di un documento non ha alcun costo.

Se un utente ha speso tutti i suoi token di benvenuto dovrà acquistarne altri.

Immaginiamo ora che l'utente appena registrato desideri effettuare l'upload di un documento originale per permettere che un altro utente registrato possa verificarne l'autenticità.

Tale utente utilizzerà il servizio di upload del documento al prezzo di un token, fornendo una serie di metadati (titolo, descrizione ecc) **e indicando l'username dell'utente cui è destinato il documento stesso.**

Il sistema non memorizza il documento in archivio, ma solo i metadati (compreso un timestamp) e l'hash (MD5) calcolato sul documento appena caricato.

Da questo momento il Verifier indicato in fase di upload, ha la possibilità di utilizzare il servizio di verifica del documento che ha in mano, facendo a sua volta un upload di quest'ultimo sul sistema.

Il sistema calcolerà l'hash (MD5) del documento inviato dal Verifier, e lo confronterà con quelli memorizzati. Se esiste un match verrà restituito un messaggio attestante l'autenticità del documento e i metadati forniti dal Publisher in fase di caricamento.

Se non c'è un match verrà restituito un messaggio attestante la non conformità del documento inserito.

Attenzione: Il Verifier ha a disposizione un certo intervallo di tempo (per debug 3 minuti) per verificare il documento a lui destinato. Se il documento caricato dal Publisher non viene verificato dal legittimo Verifier entro il tempo stabilito, verrà eliminato dagli archivi, e il token restituito al Publisher.

Qualsiasi utente Verifier non espressamente indicato dal Publisher in fase di caricamento del documento, che desideri verificare la sua copia utilizzando il servizio di verifica, può farlo senza spendere alcun token, ma non cambierà lo stato di “non verificato” del documento in quanto questa è una prerogativa dell’utente indicato dal Publisher.

Specifiche progettuali

Docauth è un sistema basato su tre microservizi.

- **USERMANAGER:** fornisce un set di API REST based che permette di gestire le identità digitali, dall'inserimento alla modifica alla cancellazione.
- **TOKENMANAGER:** si occupa della gestione dei token di ciascun utente. Dopo il bonus di benvenuto di 5 token, l'utente ha la possibilità di incrementare il numero dei propri token procedendo "all'acquisto" di nuovi token. Sono presenti anche metodi di servizio utilizzati dal sistema per l'incremento o il decremento dei token, metodi per visualizzare i token di un particolare utente e tutto ciò che in generale può essere utile a una corretta gestione del sistema dei token e per permettere un'analisi di debug semplificata.
- **DOCUMENTMANAGER:** il centro nevralgico del sistema Docauth. Gestisce i metadati caricati dai Publisher, fornendo un set di API REST based in grado di permettere l'upload o la verifica dei documenti, la modifica e la cancellazione degli stessi, la gestione temporizzata delle scadenze tramite un apposito cronjob, e ulteriori metodi di servizio utili per il debug.

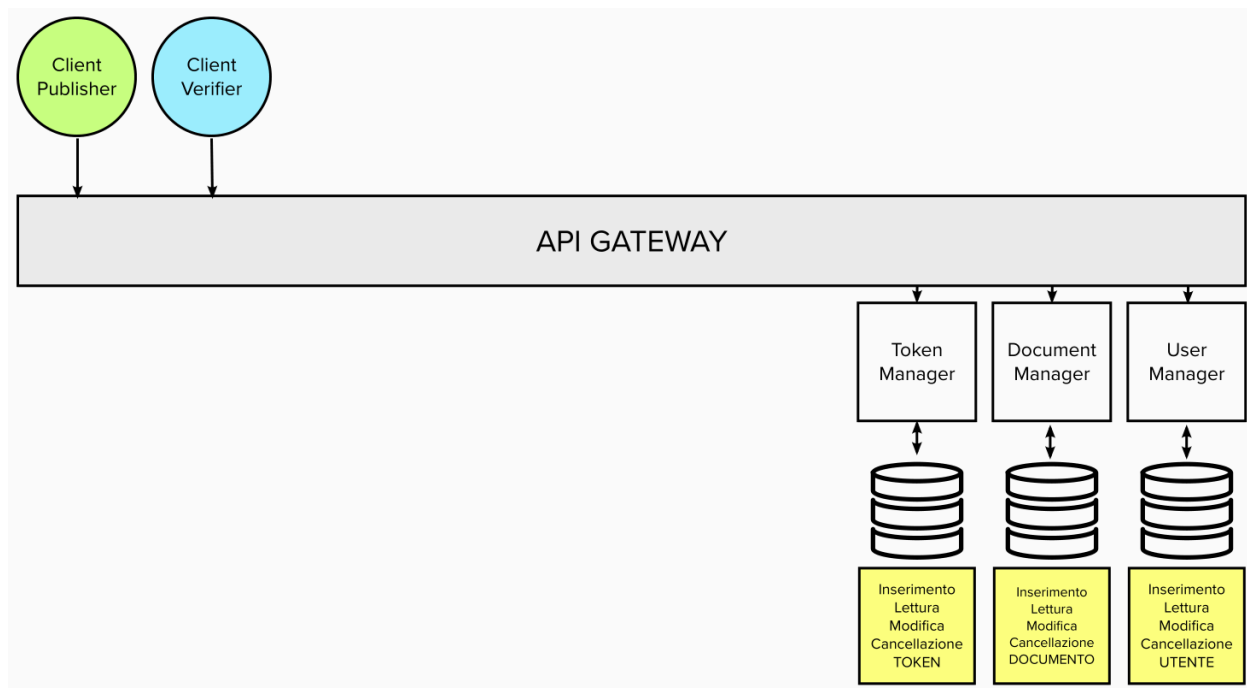
Ciascuno dei servizi elencati comunica con **un proprio database MySQL**, anch'esso ospitato all'interno di un proprio servizio isolato, per permettere e promuovere il decoupling e facilitare la scalabilità.

Un **api-gateway** basato su una istanza di nginx configurata come reverse proxy permette la trasparenza dei servizi verso l'utente, effettuando il dispatch delle richieste ai rispettivi microservizi sulla base dei path configurati in fase di **deploy su kubernetes**.

La tecnologia utilizzata per la realizzazione del sistema è basata sul linguaggio Java, attraverso l'uso del framework Spring.

Ognuna delle applicazioni elencate è ospitata all'interno di un container docker, lasciando a kubernetes il compito di gestire e organizzare il deploy dei container, le repliche e gli eventuali rolling updates che abilitano il continuous delivery e la continuous integration.

Forniamo nella prossima pagina un primo grafico che semplifica la comprensione della sola struttura del sistema, evitando in questa fase di inserire il flusso dei messaggi scambiati e le attività coinvolte nei vari scenari di utilizzo.



Specifiche implementative API

Elenco delle API REST per ogni microservizio:

USERMANAGER:

POST /usersapi/insertUser/

Inserisce un nuovo utente all'interno del sistema e fornisce un pin di 4 cifre per utilizzare i servizi legati all'utente (cambio password ecc). L'utente deve avere una user name univoca. La post prevede l'invio nel body di un messaggio Json di questo tipo:

```
{"userName":"userName","password":"password"}
```

GET /usersapi/getAllUsers/

Restituisce l'elenco di tutti gli utenti registrati nel sistema.

POST /usersapi/updateUserPin/

Rigenera il pin di un particolare utente. La post prevede l'invio nel body di un messaggio Json di questo tipo:

```
{"userName":"userName","password":"password"}
```

POST /usersapi/updateUserPassword/{userName}/{pin}

Permette la modifica della password dell'utente. Prevede l'invio nel body della nuova password, passata come parametro.

DELETE /usersapi/deleteUser/{userName}/{pin}

Elimina un utente dal sistema.

TOKENMANAGER:

POST /tokensapi/addTokensToUser/{amount}

Permette di caricare un quantitativo di token stabilito ed associarlo a un utente identificato da nome utente e password. La post prevede l'invio nel body di un messaggio Json di questo tipo:

```
{"userName":"userName","password":"password"}
```

POST /tokensapi/getUserTokens/

Permette di elencare la quantità di token associata a un utente identificato da nome utente e password. La post prevede l'invio nel body di un messaggio Json di questo tipo:

```
{"userName":"userName","password":"password"}
```

GET /tokensapi/getAllTokens/

Permette di elencare tutti i token contenuti in archivio. Utilizzata per facilitare il debug.

POST /tokensapi/decreaseUserToken/

Decrementa di uno il quantitativo di token di un dato utente. Utilizzata dai servizi che prevedono il consumo di token, prevede l'invio nel body di un messaggio Json di questo tipo:

```
{"userName":"userName","password":"password"}
```

POST /tokensapi/increaseUserToken/

Incrementa di uno il quantitativo di token di un dato utente. Utilizzata dai servizi che prevedono il consumo di token, prevede l'invio nel body di un messaggio Json di questo tipo:

```
{"userName":"userName","password":"password"}
```

DOCUMENTMANAGER:

POST /documentsapi/uploadDocument/

Permette a un Publisher di fare l'upload di un documento. Tale documento viene analizzato dal sistema, ne viene estratto il codice hash usando l'algoritmo MD5 e viene infine caricato in archivio assieme a tutti i metadati forniti dal Publisher e a quelli generati autonomamente (timestamp, hash ecc). Prevede l'invio nel body di un messaggio Json di questo tipo:

```
{“userName”:”userName”,”password”:”password”,  
  “documentTitle”:”documentTitle”,”description”:”description”,  
  “receiverName”:”receiverName”,”file”:”multipart-file”}
```

POST /documentsapi/verifyDocument/

Permette a un utente di verificare che la propria copia di un dato documento sia originale, in base all'hash calcolato dopo l'upload e alla verifica che questo sia stato depositato in archivio da un publisher. Prevede l'invio nel body di un messaggio Json di questo tipo:

```
{“userName”:”userName”,”password”:”password”,”file”:”multipart-file”}
```

GET /documentsapi/getAllDocuments/

Restituisce le informazioni di tutti i documenti in archivio.

GET /documentsapi/getDocumentByTitle/{documentTitle}

Restituisce le informazioni del documento selezionato dal titolo.

PUT /documentsapi/updateDocumentDescription/{Title}

Modifica la descrizione del documento indicato. Prevede l'invio nel body di un messaggio Json di questo tipo:

```
{“newDocumentDescription”:”newDocumentDescription”,  
  “userName”:”userName”,”password”:”password”}
```

DELETE /documentsapi/deleteDocumentByTitle/{documentTitle}

Elimina un documento dalla lista dei documenti

(Scopo debug)

DELETE /documentsapi/deleteDocumentsByPublisherName/{userName}

Elimina tutti i documenti di un particolare Publisher dalla lista dei documenti

(Scopo debug)

Specifiche Database:

Il sistema utilizza un database MySQL per ciascuno dei microservizi. Avremo quindi un database dedicato al servizio USERMANAGER, uno dedicato al servizio TOKENMANAGER e uno dedicato al servizio DOCUMENTMANAGER.

I database sono essi stessi dei microservizi.

Di seguito l'elenco dei campi delle tabelle di ciascun archivio:

mysqluserdb:

ID	: id del record, auto incrementato all'interno del database.
userName	: nome utente che deve essere unico nell'archivio.
password	: password identificativa.
pin	: pin di 4 cifre utilizzato per non esporre la password.

mysqltokendb:

ID	: id del record.
userOwnerId	: id dell'utente proprietario del token.
amount	: quantità di token associata all'utente.

mysqldocumentdb:

ID	: id del record.
documentTitle	: titolo del documento. Deve essere unico.
description	: descrizione del documento.
hashCode	: codice hash (MD5). Deve essere unico.
timeStamp	: istante di caricamento.
publisherName	: user name del Publisher.
receiverName	: user name del Verifier.
isVerified	: false fino a che il Verifier indicato non effettua la verifica.
status	: validità del documento, legato al pattern SAGA.

Comunicazione tra i microservizi e pattern SAGA:

Nel corso del funzionamento del nostro sistema, è spesso necessario effettuare comunicazioni tra metodi che si trovano fisicamente su microservizi diversi. Per esempio, all'atto della registrazione di un nuovo utente, è necessario effettuare una chiamata al metodo "addTokensToUser" ospitato nel servizio TOKENMANAGER.

Si è optato per l'utilizzo di un sistema basato sulla classe RestTemplate disponibile nella versione 5.3.15 dello SPRING framework. ([documentazione](#))

Comunicazioni di questo tipo, anche se di tipo sincrono, si prestano alla possibilità tutt'altro che remota di provocare inconsistenze tra le varie parti del sistema, in particolare riguardo gli aspetti di sincronizzazione dei dati nel database.

Ad esempio, potrebbe verificarsi il caso in cui l'eliminazione di un utente dall'archivio di USERMANAGER non comporti (a causa ad esempio di problemi momentanei di rete) la conseguente eliminazione dei token record contenuti nell'archivio di TOKENMANAGER.

Uno dei pattern principali per la gestione della consistenza tra diversi microservizi è SAGA.

Abbiamo implementato questo algoritmo in un caso specifico, ovvero in quello relativo all'upload dei documenti nel database di DOCUMENTMANAGER.

Il flusso della SAGA è il seguente:

- 1] Un publisher effettua l'upload di un documento nel database.
- 2] Il sistema verifica l'esistenza del publisher e crea immediatamente il record del documento.
- 3] Il sistema salva quindi il record nel database ma imposta il campo status a false per indicare lo stato "pendente" dello stesso.
- 4] Il sistema invoca il metodo documentCreatedEvent, che ha lo scopo di comunicare con il TOKENMANAGER avvisandolo del fatto che è stato appena creato un nuovo documento. Il sistema invia attraverso questo metodo l'id del documento appena creato e l'id dell'utente che ha caricato il documento.
- 5] Il TOKENMANAGER ricevuto il messaggio, e notificato quindi della creazione del documento di cui ha ricevuto l'id, da parte del Publisher di cui ha ricevuto l'id, controlla nel suo archivio che quel particolare utente abbia token sufficienti per completare l'operazione. Nel caso che i token non fossero sufficienti verrebbe restituito un messaggio di errore al DOCUMENTMANAGER, altrimenti verrebbe restituito il semplice id del documento.
- 6] Il DOCUMENTMANAGER riceve un messaggio di errore (i token non erano sufficienti) quindi **ELIMINA (rollback)** il documento precedentemente caricato in archivio nello stato pendente. Oppure il DOCUMENTMANAGER riceve l'id del documento appena creato e **IMPOSTA A TRUE** il campo status di questo documento, confermando il completamento con successo dell'operazione (**commit**).

Deploy

Il sistema può essere eseguito su minikube.

Le immagini compilate sono su Docker Hub. I file di configurazione per il deploy permettono di scaricarli automaticamente. Non è quindi necessaria una fase di build preliminare, a meno di una Vostra necessità di verifica.

Ricordarsi il bind del nome dell'applicazione (docauth.dev.loc) sull'indirizzo ip di minikube, da aggiungere du /etc/hosts

```
kubectl apply -f Deploy_K8s/
```

```
# Get all users
```

```
curl docauth.dev.loc/usersapi/getAllUsers/
```

```
# Create a user
```

```
curl docauth.dev.loc/usersapi/insertUser/ -X POST -H "Content-Type: application/json" -d '{"userName":"Pippo","password":"Saracinesca"}'
```

Monitoring

Si è scelto di implementare alcune funzioni di monitoring nella sezione dell'applicativo di maggiore importanza, ovvero sul servizio di DOCUMENTMANAGER.

In particolare si è analizzato il metodo uploadDocument, fornendo meccanismi in grado di generare tre diverse timeseries rappresentanti nell'ordine:

- Il tempo che trascorre dall'inizio della request al suo completamento (duration).
 - **Tali valori sono stati conservati nel file uploadsDurationMetrics.csv**
- Il numero di uploads effettuati nell'unità di tempo. Per calcolare questa quantità si è scelto di calcolare il tempo che intercorre tra cinque chiamate (in secondi) e di dividere le cinque chiamate per questo intervallo.
 - **Tali valori sono stati conservati nel file uploadsRequestRateMetrics.csv**
- Il numero di errori commessi durante gli upload nell'unità di tempo. Per calcolare questa quantità si è scelto di calcolare il tempo che intercorre tra cinque errori (in secondi) e di dividere le cinque chiamate per questo intervallo.
 - **Tali valori sono stati conservati nel file uploadsErrorCountMetrics.csv**

DISCLAIMER:

I valori raccolti, nonostante l'impegno profuso da entrambi nel tentativo di effettuare il più alto volume di traffico possibile, sono estremamente pochi.

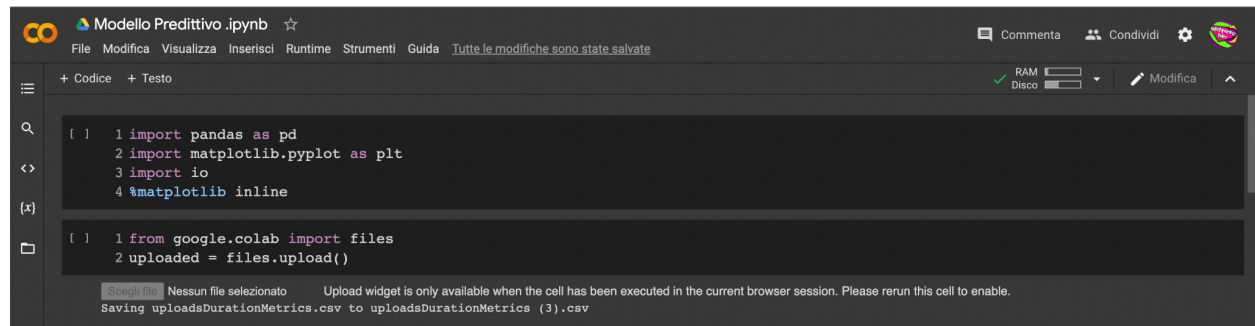
183 campioni per uploadsDurationMetrics, 37 campioni per uploadsRequestRateMetrics (un campione per 5 di uploadsDurationMetrics), 101 campioni per uploadsErrorCountMetrics sono un numero insignificante a livello statistico, sebbene siano stati raccolti in un arco temporale che va dalle 7:15 circa del mattino fino alle 22:00 del giorno 30/01/2021.

I risultati di una predizione con un numero così esiguo di campioni non può che essere un mero esercizio senza riscontri realistici.

Detto questo, abbiamo proceduto con il trattamento dei dati utilizzando l'ambiente COLAB offerto da google per la generazione di notebook in linguaggio Python.

L'ambiente fornisce la possibilità di sfruttare una scheda video dedicata per l'addestramento di modelli di machine learning.

Per prima cosa si è provveduto a importare i dati relativi alla duration:

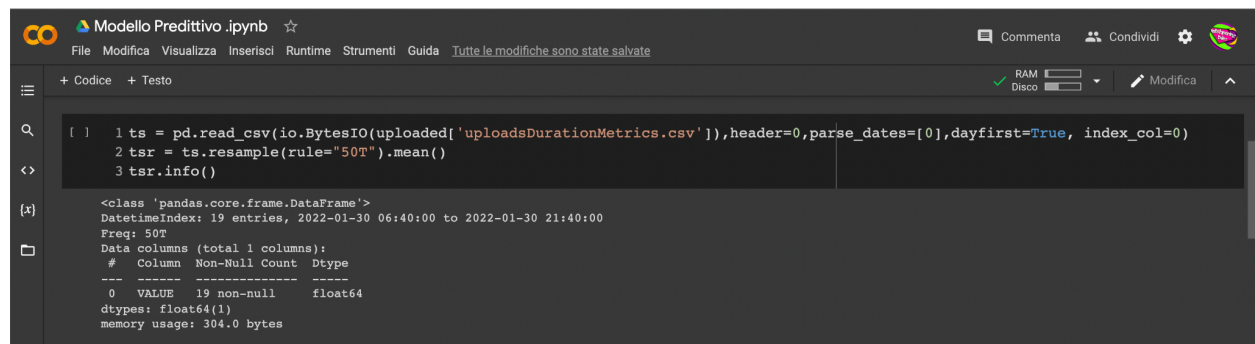


```
[ ] 1 import pandas as pd
    2 import matplotlib.pyplot as plt
    3 import io
    4 %matplotlib inline

[ ] 1 from google.colab import files
    2 uploaded = files.upload()
```

Scegli file Nessun file selezionato Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving uploadsDurationMetrics.csv to uploadsDurationMetrics (3).csv

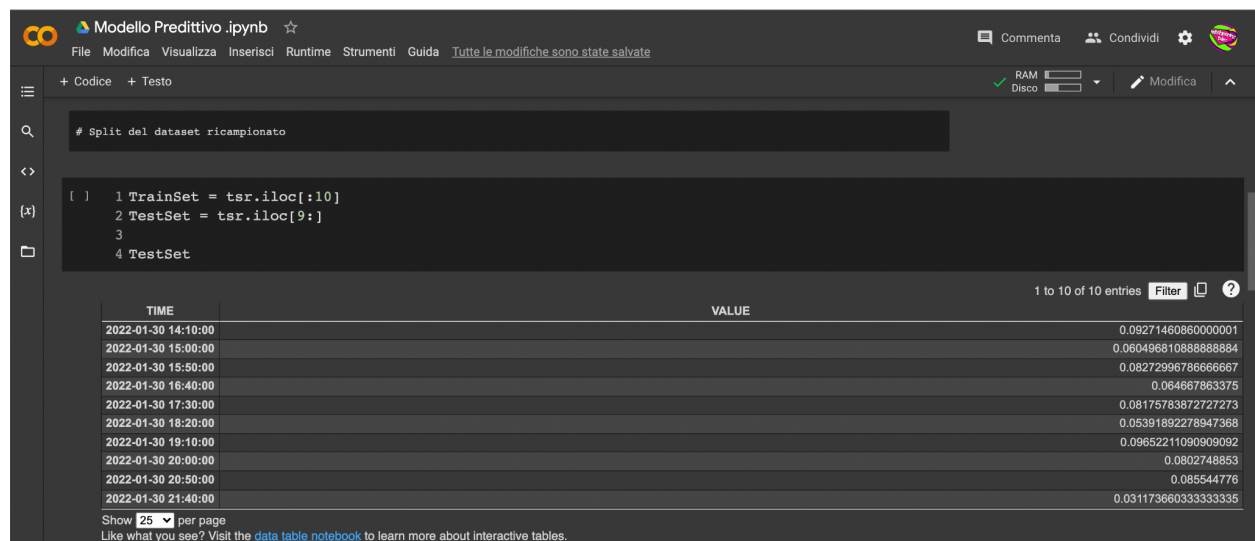
Successivamente abbiamo effettuato un resample dei dati per fornire una frequenza di campionamento costante:



```
[ ] 1 ts = pd.read_csv(io.BytesIO(uploaded['uploadsDurationMetrics.csv']), header=0, parse_dates=[0], dayfirst=True, index_col=0)
    2 tsr = ts.resample(rule="50T").mean()
    3 tsr.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 19 entries, 2022-01-30 06:40:00 to 2022-01-30 21:40:00
Freq: 50T
Data columns (total 1 columns):
#   Column  Non-Null Count  Dtype  
---  -
0   VALUE    19 non-null         float64
dtypes: float64(1)
memory usage: 304.0 bytes
```

I pochi dati rimasti sono stati suddivisi in due insiemi: TrainSet e TestSet, cercando di inserire il maggior quantitativo di dati sul TrainSet.



```
# Split del dataset ricampionato

[ ] 1 TrainSet = tsr.iloc[:10]
    2 TestSet = tsr.iloc[9:]
    3
    4 TestSet
```

TIME	VALUE
2022-01-30 14:10:00	0.09271460860000001
2022-01-30 15:00:00	0.060496810888888884
2022-01-30 15:50:00	0.08272996786666667
2022-01-30 16:40:00	0.064667863375
2022-01-30 17:30:00	0.08175783872727273
2022-01-30 18:20:00	0.05391892278947368
2022-01-30 19:10:00	0.09652211090909092
2022-01-30 20:00:00	0.0802748853
2022-01-30 20:50:00	0.085544776
2022-01-30 21:40:00	0.031173660333333335

Show 25 per page
Like what you see? Visit the [data.table notebook](#) to learn more about interactive tables.

Si cerca adesso di eseguire un modello standard di apprendimento, Holtwinter per l'esattezza, cui si passano i dati contenuti nel nostro TrainSet e i parametri che più si adattano alla

stagionalità e al tipo di campioni ottenuti (che nel nostro caso sono purtroppo casuali e non mostrano trend, sia perché il numero dei dati è basso, ma anche per il tipo di applicazione)

Il modello fornisce una sua previsione basata sui dati in input. Abbiamo scelto di non superare i 10 campioni di previsione, già secondo noi, più che esagerati.

```
Modello Predittivo .ipynb
File Modifica Visualizza Inserisci Runtime Strumenti Guida Tutte le modifiche sono state salvate

+ Codice + Testo
Show 25 per page
Like what you see? Visit the data table notebook to learn more about interactive tables.

[ ] 1 from statsmodels.tsa.holtwinters import ExponentialSmoothing
    2 tsmodel = ExponentialSmoothing(TrainSet, trend="add", seasonal="add", seasonal_periods=5).fit()
    3

/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/holtwinters.py:712: ConvergenceWarning: Optimization failed to converge. Check mle_retvals.
ConvergenceWarning)

1 prediction = tsmodel.forecast(10)
2 prediction

2022-01-30 15:00:00    0.120069
2022-01-30 15:50:00    0.108955
2022-01-30 16:40:00    0.105292
2022-01-30 17:30:00    0.140855
2022-01-30 18:20:00    0.132702
2022-01-30 19:10:00    0.143180
2022-01-30 20:00:00    0.132066
2022-01-30 20:50:00    0.128402
2022-01-30 21:40:00    0.163965
2022-01-30 22:30:00    0.155812
Freq: 50T, dtype: float64
```

Infine abbiamo verificato attraverso il plotting dei dati quale correlazione ci fosse tra i valori di train, quelli di test e soprattutto quelli predetti, ottenendo, come immaginato, risultati poco vicini alla realtà anche se sorprendentemente correlati in una certa maniera:



La “curva” verde indica la previsione, mentre i dati di test da seguire sono in arancione.