PROGETTO DI MELI FRANCESCO 1000050455 E MICIELI VINCENZO 1000050456

IDEA SVILUPPATA

L'applicazione prevede agli utenti di sottoscriversi ad eventi meteorologici nelle città di interesse. L'utente può sottoscriversi ad un insieme di città, fornendo per ognuno di essi le seguenti serie di constraint: temperatura massima, temperatura minima, presenza di pioggia e presenza di neve. Qualora si verifichino le condizioni specificate, il sistema invia una mail all'utente (si è usato per fare ciò il servizio esterno Mailtrap). A tale scopo, ad intervalli regolari recupera le informazioni meteorologiche tramite rest API da un servizio esterno (si è usato OpenWeatherMap), le filtra e le elabora sulla base delle condizioni sottomesse dagli utenti.

DESCRIZIONE MICROSERVIZI E LORO INTERAZIONI

I microservizi implementati sono:

1) **Database service**: Il microservizio utilizza un database SQLite attraverso SQLAlchemy per interagire con i dati. I principali modelli ORM includono User per gli utenti, City per le città e CitySubscription per le preferenze di sottoscrizione alle città. Tra le funzionalità principali, il microservizio fornisce endpoint per ottenere informazioni su tutte le città presenti nel database (/cities), aggiungere nuovi utenti (/user), ottenere informazioni su un utente specifico e le relative sottoscrizioni (/user/<username>), gestire le sottoscrizioni alle città (/user/subscribe_to_city e /user/modify_to_city), e gestire l'autenticazione degli utenti (/user/login). Inoltre, espone un endpoint /metrics per fornire metriche di Prometheus. E' implementata una metrica di tipo Gauge (utenti_registrati) che rappresenta il numero totale di utenti registrati nel sistema.
2) **Notifier service**: Questo microservizio Flask in Python utilizza l'estensione Flask-Mail e il modulo APScheduler per notificare gli utenti in base alle loro preferenze meteo. Il servizio è progettato per eseguire il controllo delle preferenze ogni 60 minuti e inviare notifiche via email. La configurazione di Flask-Mail include dettagli come il server SMTP, la porta, l'uso di TLS/SSL, nome utente, password e indirizzo email del mittente. In questo caso, l'applicazione è configurata per utilizzare MailTrap come server di posta. Il microservizio recupera gli utenti e le loro preferenze dal servizio di gestione degli utenti e dal servizio di database, rispettivamente. Confronta poi queste preferenze con i dati sulla città e invia notifiche via email agli utenti con il resoconto delle preferenze soddisfatte e non soddisfatte. L'invio di email avviene attraverso MailTrap, con Flask-Mail che costruisce e invia messaggi di notifica. La funzione send_notification_email prende l'indirizzo email del destinatario, il nome della città, le preferenze soddisfatte e non soddisfatte, e invia un'email con il resoconto delle preferenze.
3) **Scraper service:** Questo microservizio Flask fornisce dati meteorologici utilizzando l'API di OpenWeatherMap. L'endpoint principale (/weather/<city>) accetta richieste GET per ottenere dati meteo per una città specifica. La funzione get_weather incrementa la metrica numero_richieste e richiama la funzione fetch_weather_data per ottenere i dati meteorologici grezzi da OpenWeatherMap. La funzione fetch_weather_data effettua una richiesta all'API di OpenWeatherMap utilizzando la chiave API specificata (OPENWEATHERMAP_API_KEY). Il microservizio espone un endpoint /metrics che restituisce il valore corrente della metrica numero_richieste. Questa metrica di tipo Gauge tiene traccia del numero totale di richieste.
4) **User management service:** Questo microservizio gestisce le preferenze meteorologiche degli utenti. Il microservizio è integrato con Flask-JWT-Extended per la gestione dei token JWT, consentendo l'autenticazione degli utenti. La variabile DATABASE_SERVICE_URL contiene l'URL del servizio di database, utilizzato per interagire con il servizio di gestione del database. L'endpoint /user/preferences è protetto da JWT e consente agli utenti autenticati di aggiornare le proprie preferenze meteorologiche. La funzione add_weather_preferences verifica l'autenticazione mediante token JWT, recupera l'identità dell'utente, e invia una richiesta al servizio di database per aggiornare le preferenze. L'endpoint /user/subscribe_to_city consente agli utenti autenticati di sottoscriversi a una nuova città o di aggiornare le preferenze di una città esistente. Analogamente a /user/preferences, verifica l'autenticazione mediante token JWT, recupera l'identità dell'utente e invia richieste al servizio di database per gestire le sottoscrizioni alle città. L'endpoint /user permette di aggiungere un nuovo utente. L'endpoint /user/login consente agli utenti di effettuare il login. Invia una richiesta al servizio di database per verificare le credenziali e restituisce un token JWT in caso di successo. L'endpoint /users restituisce la lista degli utenti e le relative sottoscrizioni alle città.

Infine, si descrive di seguito il servizio **SLA Manager**. Il microservizio si integra con diverse librerie e servizi. Le librerie principali includono prometheus_client per la gestione delle metriche Prometheus, confluent_kafka per la produzione di dati su Kafka e sqlite3 per l'interazione con un database SQLite. Viene creato un produttore Kafka per inviare dati a un broker Kafka. La funzione send_to_kafka è responsabile di inviare i dati serializzati in formato JSON a un topic specifico su Kafka. Il microservizio gestisce un database SQLite denominato 'sla_manager_data.db' contenente una tabella

'sla_metrics' che conserva diverse metriche con i relativi valori. Sono definite diverse metriche di tipo Gauge all'interno di un dizionario denominato sla_metrics, ognuna rappresentante un aspetto specifico del sistema da monitorare (utenti registrati, richieste, utilizzo della CPU, utilizzo della memoria, metriche SLA, ecc.).
Il microservizio recupera e aggiorna le metriche da diverse fonti, inclusi servizi come database-service e scraper-service, oltre a metriche di CPU e memoria. Le metriche ottenute sono confrontate con i valori desiderati e i risultati vengono registrati in un apposito dizionario denominato sla_manager_results. Viene implementato un sistema di gestione per la rimozione temporanea di metriche, consentendo di riaggiungerle successivamente. Sono definite API per aggiungere/rimuovere metriche, ottenere dati dal database, ottenere le metriche del sistema, gestire SLA tramite API e ottenere il conteggio delle violazioni.

RELAZIONI

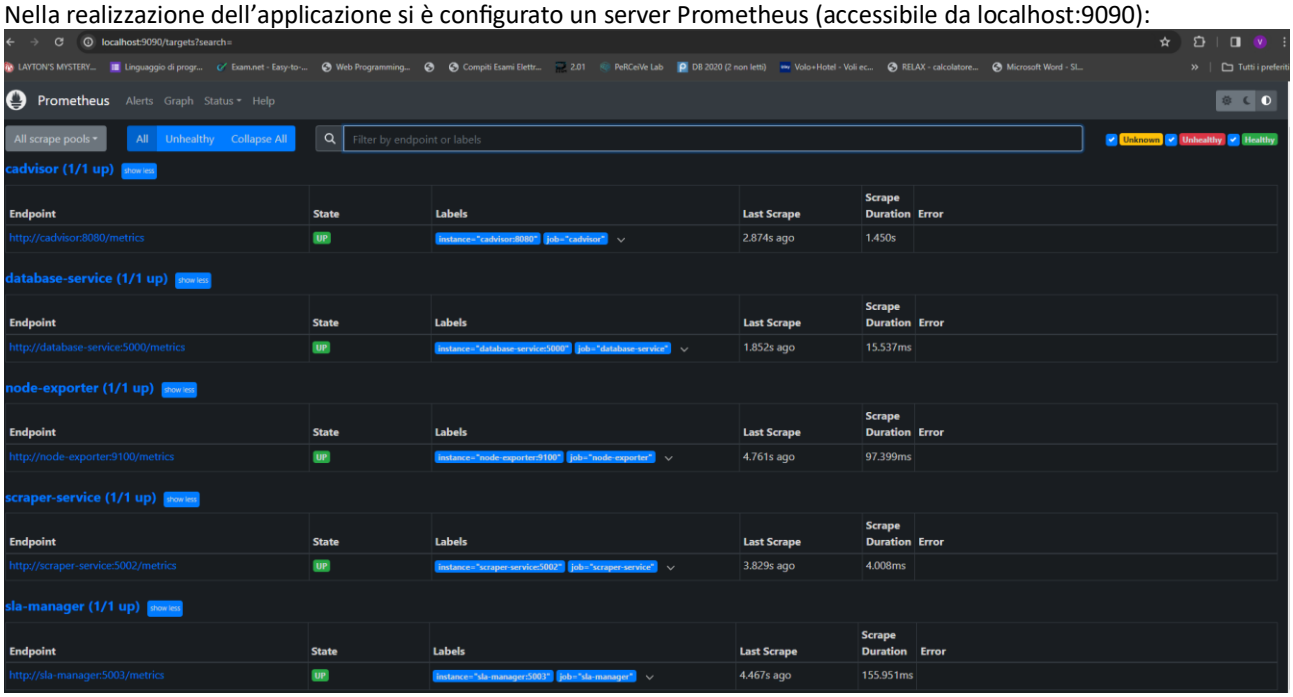
Il diagramma seguente spiega le relazioni tra i microservizi interni ed esterni:

Dipende da	**Database service**	**Notifier service**	**Scraper service**	**User management service**	**OpenWeatherMap**	**MailTrap**
Database service						
Notifier service						
Scraper service						
User management service						
SLA Manager						

Verde: si, Rosso: no

Come si evince dallo schema, nella strutturazione del progetto si è cercato di isolare il più possibile le funzionalità dei microservizi. Il sistema è stato strutturato in modo tale che esso non dipenda fortemente da un microservizio in particolare, ma dipenda dalla cooperazione tra tutti i microservizi. Sono comunque presenti relazioni di dipendenza tra loro. Questo è dovuto al fatto che, per fare in modo che il sistema funzioni correttamente, sono necessarie le dipendenze indicate nella tabella.

Nella realizzazione dell'applicazione si è configurato un server Prometheus (accessibile da localhost:9090):



Si è voluto monitorare:

1) Performance del nodo sulla quale è deployata l'app (node exporter);
2) Performance dei container (cAdvisor)
3) Numero di utenti registrati e numero di richieste all'API OpenWeatherMap (database service e scraper service)
4) Numero totali di violazioni in 1h, 3h, 6. Valori desiderati e attuali di utenti registrati, numero richieste all'API OpenWeatherMap, cpu_usage e memory_usage. Numero di violazioni (true/false) di quest'ultimi (sla manager). Inoltre, viene inviato ad un topic kafka "PrometheusData" un messaggio contenente i valori calcolati.

BUILD & DEPLOY

Di seguito viene elencato l'ordine di esecuzione per il funzionamento di ogni funzionalità dell'applicativo.

1) Visualizzazione delle città:

```
$result = Invoke-RestMethod -Uri "http://localhost:5000/cities" -Method Get
$result.cities | ForEach-Object {
   "City: $($_.name), Rainfall: $($_.rainfall), Snow: $($_.snow), Max Temp: $($_.temperature_max), Min Temp: $($_.temperature_min)"
}
```

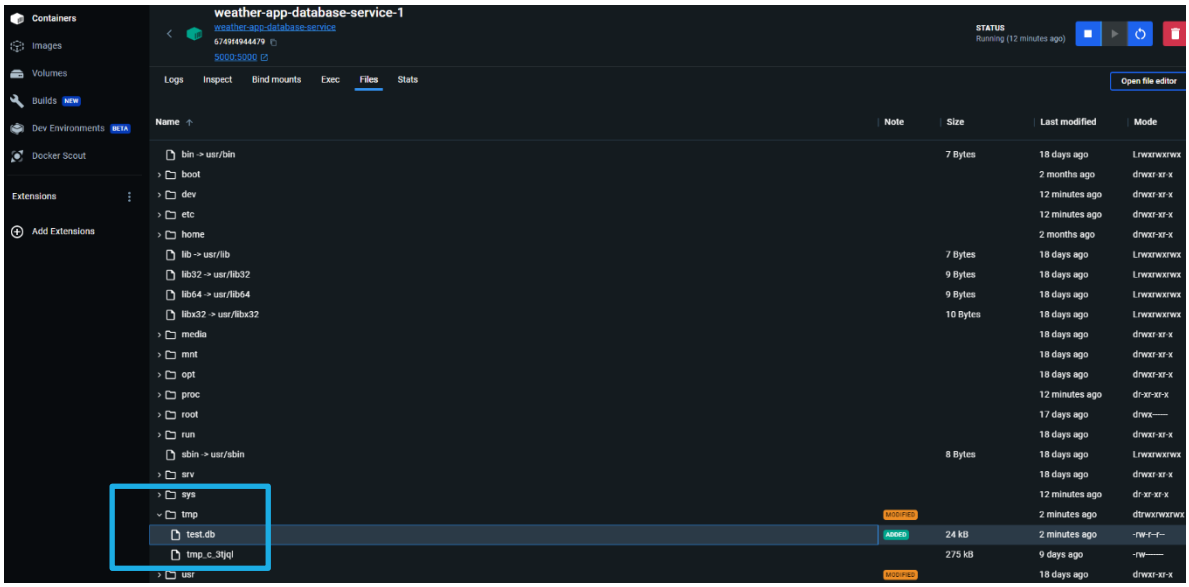
```
PS C:\Users\gerar\Desktop\weather-app> $url = "http://localhost:5000/cities"
PS C:\Users\gerar\Desktop\weather-app> Invoke-RestMethod -Uri $url -Method GET

cities
------
{@{id=1; name=Rome,IT; rainfall=False; snow=False; temperature_max=16,28000000000003; temperature_min=9,970000000000...


PS C:\Users\gerar\Desktop\weather-app> $result = Invoke-RestMethod -Uri "http://localhost:5000/cities" -Method Get
PS C:\Users\gerar\Desktop\weather-app> $result.cities | ForEach-Object {
>>     "City: $($_.name), Rainfall: $($_.rainfall), Snow: $($_.snow), Max Temp: $($_.temperature_max), Min Temp: $($_.temperature_min)"
>> }
City: Rome,IT, Rainfall: False, Snow: False, Max Temp: 16.28000000000003, Min Temp: 9.970000000000027
City: Lucca, Rainfall: False, Snow: False, Max Temp: 11.870000000000005, Min Temp: 8.100000000000023
City: Paris, Rainfall: False, Snow: False, Max Temp: 6.230000000000018, Min Temp: 3.2800000000000296
City: London, Rainfall: False, Snow: False, Max Temp: 8.330000000000041, Min Temp: 5.430000000000007
City: New York, Rainfall: True, Snow: False, Max Temp: 4.980000000000018, Min Temp: 3.090000000000032
City: Tokyo, Rainfall: False, Snow: False, Max Temp: 8.79000000000002, Min Temp: 4.810000000000002
PS C:\Users\gerar\Desktop\weather-app>
```

2) Registrazione utente:

```
$url = "http://localhost:5001/user"
$body = @{
   "username" = "nuovo_utente"
   "email" = "micieli.vincenzo@example.com"
   "password" = "password_segreta"
} | ConvertTo-Json

Invoke-RestMethod -Uri $url -Method POST -Body $body -Headers @{"Content-Type"="application/json"}
```

```
PS C:\Users\gerar\Desktop\weather-app> $url = "http://localhost:5001/user"
PS C:\Users\gerar\Desktop\weather-app> $body = @{
>>     "username" = "nuovo_utente"
>>     "email" = "micieli.vincenzo@example.com"
>>     "password" = "password_segreta"
>> } | ConvertTo-Json
PS C:\Users\gerar\Desktop\weather-app>
PS C:\Users\gerar\Desktop\weather-app> Invoke-RestMethod -Uri $url -Method POST -Body $body -Headers @{"Content-Type"="application/json"}

message                      token
-------                      -----
Utente registrato con successo eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTcwNjQzOTIzOCwianRpIjoiZjJhYzNmMmItY2U4Zi00ODI5LTk3NzEtMzVhOGQzNGM5ZmIzIiwidHlwZSI6ImFjY2VzcyIsInN1YiI6Im51b...
```

Gli utenti vengono registrati all'interno di questo database:



3) Login di un utente

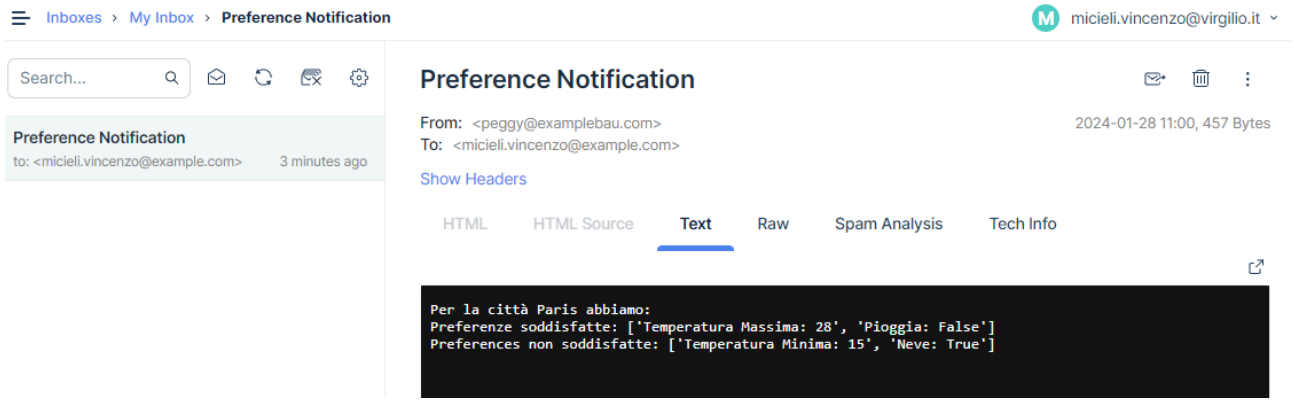
```
$url = "http://localhost:5001/user/login"
$body = @{
   "username" = "nuovo_utente"
   "password" = "password_segreta"
} | ConvertTo-Json

$response = Invoke-RestMethod -Uri $url -Method POST -Body $body -Headers @{"Content-Type"="application/json"}
$token = $response.token
```

4) Sottoscrizione ad una città

```
$url = "http://localhost:5001/user/subscribe_to_city"
$headers = @{
   "Authorization" = "Bearer $token"
   "Content-Type" = "application/json"
}
$body = @{
   "city_name" = "Paris"
   "temperature_max_preference" = 28
   "temperature_min_preference" = 15
   "rainfall_preference" = $false
   "snow_preference" = $true
} | ConvertTo-Json

Invoke-RestMethod -Uri $url -Method POST -Body $body -Headers $headers
```

```
PS C:\Users\gerar\Desktop\weather-app> $url = "http://localhost:5001/user/subscribe_to_city"
PS C:\Users\gerar\Desktop\weather-app> $headers = @{
>>     "Authorization" = "Bearer $token"
>>     "Content-Type" = "application/json"
>> }
PS C:\Users\gerar\Desktop\weather-app> $body = @{
>>     "city_name" = "Paris"
>>     "temperature_max_preference" = 28
>>     "temperature_min_preference" = 15
>>     "rainfall_preference" = $false
>>     "snow_preference" = $true
>> } | ConvertTo-Json
PS C:\Users\gerar\Desktop\weather-app>
PS C:\Users\gerar\Desktop\weather-app> Invoke-RestMethod -Uri $url -Method POST -Body $body -Headers $headers

message
-------
Sottoscrizione alla città effettuata con successo!
```

Una volta che l'utente è sottoscritto possiamo vedere se il servizio MailTrap funziona correttamente:



N.B. Per questa prova è stato settato l'IntervallTrigger ad 1 minuto.

5) Modifica preferenze ad una città

```
$url = "http://localhost:5001/user/preferences"
$headers = @{
   "Authorization" = "Bearer $token"
   "Content-Type" = "application/json"
}
$body = @{
   "city_name" = "Paris"
   "temperature_max_preference" = 15
   "temperature_min_preference" = 18
   "rainfall_preference" = $true
   "snow_preference" = $false
} | ConvertTo-Json

Invoke-RestMethod -Uri $url -Method POST -Body $body -Headers $headers
```

6) Lista utenti

Si è provveduto alla registrazione di un altro utente senza inserimento di preferenza e introdurne una nuova per il primo utente. Per cui si avrà:

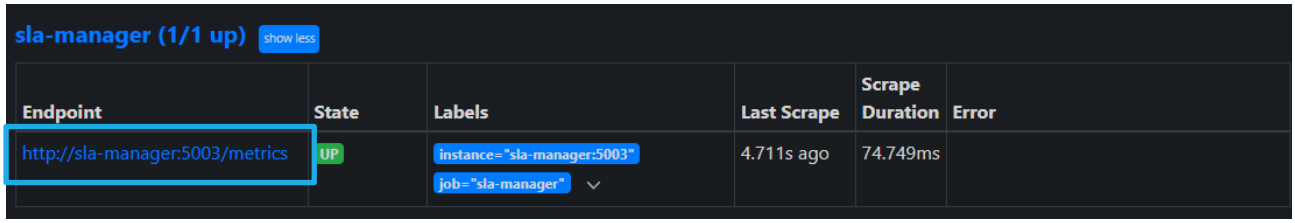
```
$url = "http://localhost:5001/users"
$headers = @{
   "Authorization" = "Bearer $token"
}
$response = Invoke-RestMethod -Uri $url -Method GET -Headers $headers
$users = $response.users

foreach ($user in $users) {
   Write-Host "User ID: $($user.id)"
   Write-Host "Username: $($user.username)"
   Write-Host "Email: $($user.email)"

   if ($user.subscriptions -eq $null) {
      Write-Host "Subscriptions: No subscriptions"
   } else {
      Write-Host "Subscriptions:"
      foreach ($subscription in $user.subscriptions) {
         Write-Host "  City: $($subscription.city_name)"
         Write-Host "  Max Temperature: $($subscription.temperature_max_preference)"
         Write-Host "  Min Temperature: $($subscription.temperature_min_preference)"
         Write-Host "  Rainfall: $($subscription.rainfall_preference)"
         Write-Host "  Snow: $($subscription.snow_preference)"
         Write-Host "------------------------"
      }
```

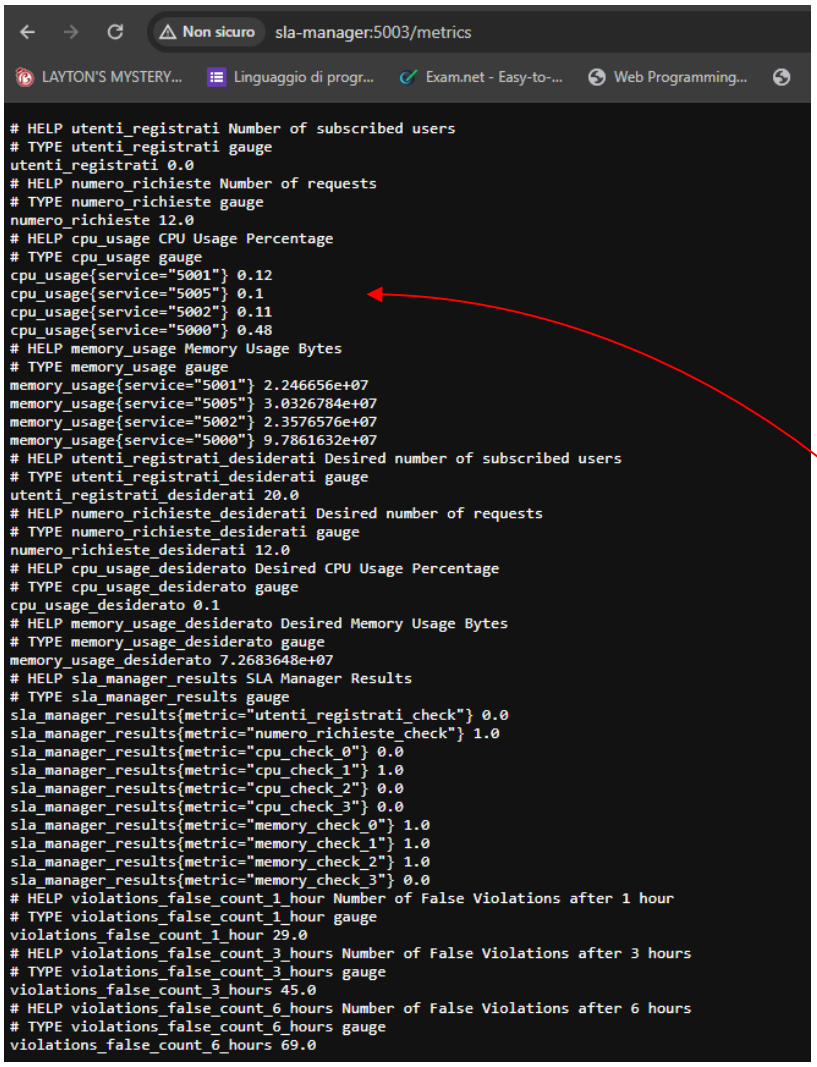
```
    }

    Write-Host "------------------------"
}
```

```
PS C:\Users\gerar\Desktop\weather-app> $url = "http://localhost:5001/users"
PS C:\Users\gerar\Desktop\weather-app> $headers = @{
>>     "Authorization" = "Bearer $token"
>> }
PS C:\Users\gerar\Desktop\weather-app> $response = Invoke-RestMethod -Uri $url -Method GET -Headers $headers
PS C:\Users\gerar\Desktop\weather-app> $users = $response.users
PS C:\Users\gerar\Desktop\weather-app>
PS C:\Users\gerar\Desktop\weather-app> foreach ($user in $users) {
>>     Write-Host "User ID: $($user.id)"
>>     Write-Host "Username: $($user.username)"
>>     Write-Host "Email: $($user.email)"
>>
>>     if ($user.subscriptions -eq $null) {
>>         Write-Host "Subscriptions: No subscriptions"
>>     } else {
>>         Write-Host "Subscriptions:"
>>         foreach ($subscription in $user.subscriptions) {
>>             Write-Host "  City: $($subscription.city_name)"
>>             Write-Host "  Max Temperature: $($subscription.temperature_max_preference)"
>>             Write-Host "  Min Temperature: $($subscription.temperature_min_preference)"
>>             Write-Host "  Rainfall: $($subscription.rainfall_preference)"
>>             Write-Host "  Snow: $($subscription.snow_preference)"
>>             Write-Host "------------------------"
>>         }
>>     }
>>
>>     Write-Host "------------------------"
>> }
User ID: 1
Username: nuovo_utente
Email: vincenzo.micieli@gmail.com
Subscriptions:
  City: Paris
  Max Temperature: 28
  Min Temperature: 15
  Rainfall: False
  Snow: True
------------------------
  City: Tokyo
  Max Temperature: 28
  Min Temperature: 10
  Rainfall: False
  Snow: False
------------------------
------------------------
User ID: 2
Username: nuovo_utente2
Email: mariobros@luigi.com
Subscriptions:
------------------------
```

7) Metriche SLA

Collegandosi al localhost:9090 (Prometheus) abbiamo deciso di osservare le seguenti metriche:





8) Rimozione metrica SLA

Supponiamo di voler rimuovere la metrica 'cpu_usage' (questo è solo un esempio, modifica con la metrica che desideri rimuovere)
Invoke-RestMethod -Uri "http://localhost:5003/remove_metric/cpu_usage" -Method GET

```
PS C:\Users\gerar\Desktop\weather-app> # Supponiamo di voler rimuovere la metrica 'cpu_usage' (questo è solo un esempio, modifica con la metrica che desideri rimuovere)
PS C:\Users\gerar\Desktop\weather-app> Invoke-RestMethod -Uri "http://localhost:5003/remove_metric/cpu_usage" -Method GET
Metrica cpu_usage rimossa con successo.
```

Infatti, adesso avremo:

```
# HELP utenti_registrati Number of subscribed users
# TYPE utenti_registrati gauge
utenti_registrati 0.0
# HELP numero_richieste Number of requests
# TYPE numero_richieste gauge
numero_richieste 12.0
# HELP memory_usage Memory Usage Bytes
# TYPE memory_usage gauge
memory_usage{service="5001"} 2.246656e+07
memory_usage{service="5005"} 3.0326784e+07
memory_usage{service="5002"} 2.3662592e+07
memory_usage{service="5000"} 9.7882112e+07
```

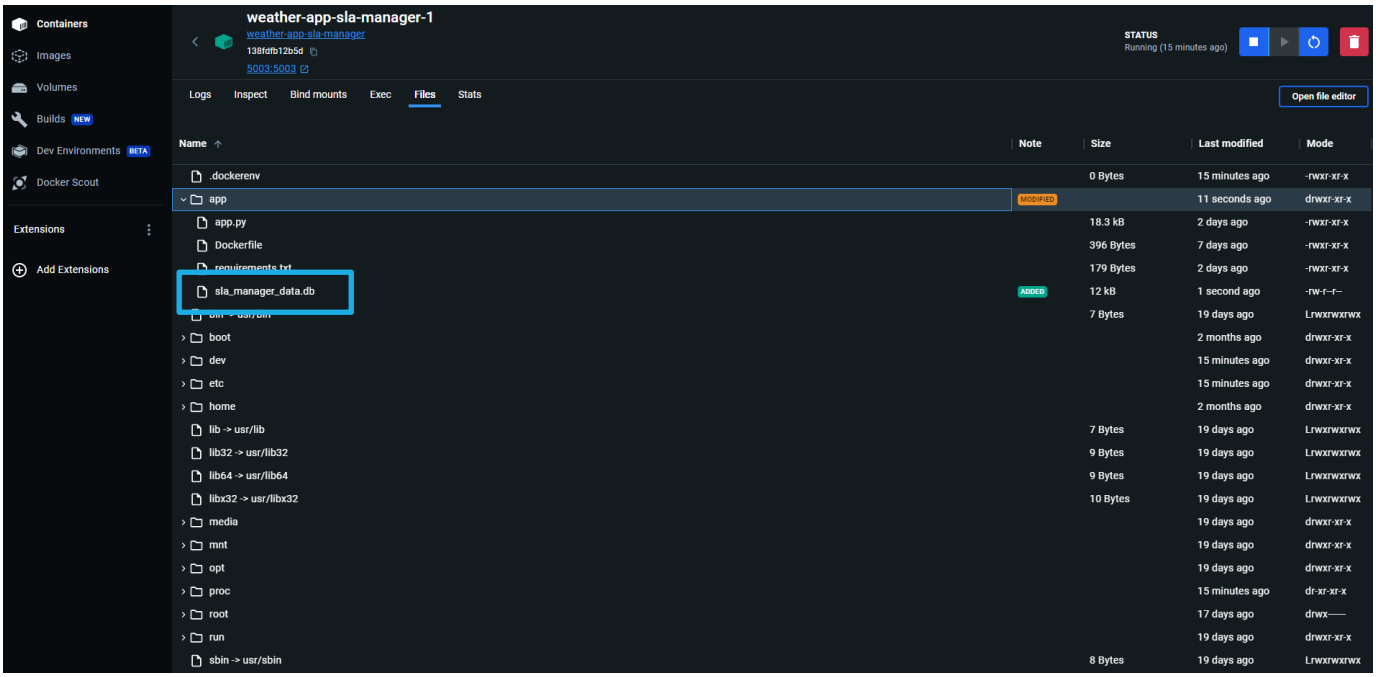
9) Reintroduzione metrica SLA

Supponiamo di voler reintrodurre la metrica 'cpu_usage' (questo è solo un esempio, modifica con la metrica che desideri reintrodurre)
Invoke-RestMethod -Uri "http://localhost:5003/add_metric/cpu_usage" -Method GET

```
PS C:\Users\gerar\Desktop\weather-app> # Supponiamo di voler reintrodurre la metrica 'cpu_usage' (questo è solo un esempio, modifica con la metrica che desideri reintrodurre)
PS C:\Users\gerar\Desktop\weather-app> Invoke-RestMethod -Uri "http://localhost:5003/add_metric/cpu_usage" -Method GET
Metrica cpu_usage riaggiunta con successo.
```

10) Prelevamento delle violazioni dal database

Le metriche violate vengono memorizzate all'interno di questo database:



Quindi con:

```
# Esegui la richiesta HTTP alla tua applicazione Flask
$response = Invoke-RestMethod -Uri 'http://localhost:5003/get_data_from_db' -Method Get

# Visualizza la risposta
$response
```

Si procede all'estrazione dei dati contenuti nel database:

```
PS C:\Users\gerar\Desktop\weather-app> # Esegui la richiesta HTTP alla tua applicazione Flask
PS C:\Users\gerar\Desktop\weather-app> $response = Invoke-RestMethod -Uri 'http://localhost:5003/get_data_from_db' -Method Get
PS C:\Users\gerar\Desktop\weather-app>
PS C:\Users\gerar\Desktop\weather-app> # Visualizza la risposta
PS C:\Users\gerar\Desktop\weather-app> $response

metric_name             metric_value
-----------             ------------
utenti_registrati_check          0,0
numero_richieste_check           1,0
cpu_check_0                      0,0
cpu_check_1                      0,0
cpu_check_2                      0,0
cpu_check_3                      0,0
memory_check_0                   1,0
memory_check_1                   1,0
memory_check_2                   1,0
memory_check_3                   0,0
```

11) Aggiornamento metriche SLA

```
$slaData = @{
        'utenti_registrati_desiderati' = 50
        'numero_richieste_desiderati' = 15
        'cpu_usage_desiderato' = 0.15
        'memory_usage_desiderato' = 8e+07
}

$response = Invoke-RestMethod -Uri "http://localhost:5003/sla" -Method PUT -Body ($slaData | ConvertTo-Json) -ContentType "application/json"
Write-Output $response
```

```
PS C:\Users\gerar\Desktop\weather-app> $slaData = @{
>>     'utenti_registrati_desiderati' = 50
>>     'numero_richieste_desiderati' = 15
>>     'cpu_usage_desiderato' = 0.15
>>     'memory_usage_desiderato' = 8e+07
>> }
PS C:\Users\gerar\Desktop\weather-app>
PS C:\Users\gerar\Desktop\weather-app> $response = Invoke-RestMethod -Uri "http://localhost:5003/sla" -Method PUT -Body ($slaData | ConvertTo-Json) -ContentType "application/json"
PS C:\Users\gerar\Desktop\weather-app> Write-Output $response

message
-------
SLA updated successfully
```

Notiamo infatti adesso che le metriche desiderate sono state aggiornate:

```
# HELP utenti_registrati Number of subscribed users
# TYPE utenti_registrati gauge
utenti_registrati 0.0
# HELP numero_richieste Number of requests
# TYPE numero_richieste gauge
numero_richieste 24.0
# HELP memory_usage Memory Usage Bytes
# TYPE memory_usage gauge
memory_usage{service="5001"} 2.246656e+07
memory_usage{service="5005"} 3.2919552e+07
memory_usage{service="5002"} 2.7357184e+07
memory_usage{service="5000"} 1.00315136e+08
# HELP utenti_registrati_desiderati Desired number of subscribed users
# TYPE utenti_registrati_desiderati gauge
utenti_registrati_desiderati 50.0
# HELP numero_richieste_desiderati Desired number of requests
# TYPE numero_richieste_desiderati gauge
numero_richieste_desiderati 15.0
# HELP cpu_usage_desiderato Desired CPU Usage Percentage
# TYPE cpu_usage_desiderato gauge
cpu_usage_desiderato 0.15
# HELP memory_usage_desiderato Desired Memory Usage Bytes
# TYPE memory_usage_desiderato gauge
memory_usage_desiderato 8e+07
# HELP sla_manager_results SLA Manager Results
# TYPE sla_manager_results gauge
sla_manager_results{metric="utenti_registrati_check"} 0.0
sla_manager_results{metric="numero_richieste_check"} 1.0
sla_manager_results{metric="cpu_check_0"} 0.0
sla_manager_results{metric="cpu_check_1"} 1.0
sla_manager_results{metric="cpu_check_2"} 0.0
sla_manager_results{metric="cpu_check_3"} 0.0
sla_manager_results{metric="memory_check_0"} 1.0
sla_manager_results{metric="memory_check_1"} 1.0
sla_manager_results{metric="memory_check_2"} 1.0
sla_manager_results{metric="memory_check_3"} 0.0
# HELP violations_false_count_1_hour Number of False Violations after 1 hour
# TYPE violations_false_count_1_hour gauge
violations_false_count_1_hour 29.0
# HELP violations_false_count_3_hours Number of False Violations after 3 hours
# TYPE violations_false_count_3_hours gauge
violations_false_count_3_hours 45.0
# HELP violations_false_count_6_hours Number of False Violations after 6 hours
# TYPE violations_false_count_6_hours gauge
violations_false_count_6_hours 69.0
# HELP cpu_usage CPU Usage Percentage
# TYPE cpu_usage gauge
cpu_usage{service="5001"} 0.17
cpu_usage{service="5005"} 0.13
cpu_usage{service="5002"} 0.3
cpu_usage{service="5000"} 1.63
```

12) Stato SLA

$response = Invoke-RestMethod -Uri "http://localhost:5003/sla" -Method GET
Write-Output $response

```
PS C:\Users\gerar\Desktop\weather-app> $response = Invoke-RestMethod -Uri "http://localhost:5003/sla" -Method GET
PS C:\Users\gerar\Desktop\weather-app> Write-Output $response


cpu_usage_5000_3                : 1,86
cpu_usage_5001_0                : 0,17
cpu_usage_5002_2                : 0,32
cpu_usage_5005_1                : 0,13
cpu_usage_desiderato_0          : 0,15
memory_usage_5000_3             : 105222144,0
memory_usage_5001_0             : 24416256,0
memory_usage_5002_2             : 27357184,0
memory_usage_5005_1             : 32919552,0
memory_usage_desiderato_0       : 80000000,0
numero_richieste_0              : 24,0
numero_richieste_desiderati_0   : 15,0
sla_manager_results_0           : 0,0
utenti_registrati_0             : 0,0
utenti_registrati_desiderati_0  : 50,0
violations_false_count_1_hour_0 : 29,0
violations_false_count_3_hours_0 : 45,0
violations_false_count_6_hours_0 : 69,0
```

13) **Numero violazioni SLA 1h, 3h e 6h**

$response = Invoke-RestMethod -Uri "http://localhost:5003/violations" -Method GET
Write-Output $response

```
PS C:\Users\gerar\Desktop\weather-app> $response = Invoke-RestMethod -Uri "http://localhost:5003/violations" -Method GET
PS C:\Users\gerar\Desktop\weather-app> Write-Output $response

violations_1_hour violations_3_hours violations_6_hours
----------------- ----------------- -----------------
             29,0              45,0              69,0
```

Per testare questi valori 1h, 3h e 6h sono stati diminuiti a tempi ragionevoli per provarne il funzionamento.

14) **Kafka**

Attaverso il comando docker ps si preleva l'id di kafka:

docker ps per trova id di kafka

```
PS C:\Users\gerar\Desktop\weather-app> docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED          STATUS          PORTS                                                  NAMES
4ec85e9199dc   weather-app-sla-manager              "python app.py"          52 seconds ago   Up 43 seconds   0.0.0.0:5003->5003/tcp                                 weather-app-sla-manager-1
f8ef4035cb19   weather-app-notifier-service         "python app.py"          52 seconds ago   Up 43 seconds   0.0.0.0:5005->5005/tcp                                 weather-app-notifier-service-1
80c814f083fb   weather-app-user-management-service  "python app.py"          52 seconds ago   Up 43 seconds   0.0.0.0:5001->5001/tcp                                 weather-app-user-management-service-1
d05c031f34f2   weather-app-scraper-service          "python app.py"          53 seconds ago   Up 44 seconds   0.0.0.0:5002->5002/tcp                                 weather-app-scraper-service-1
df343517d7e0   prom/node-exporter                   "/bin/node_exporter …"   53 seconds ago   Up 44 seconds   0.0.0.0:9100->9100/tcp                                 weather-app-node-exporter-1
[illegible]    google/cadvisor:latest               "/usr/bin/cadvisor -…"   53 seconds ago   Up 44 seconds   0.0.0.0:8080->8080/tcp                                 cadvisor
2e60ce975ad0   wurstmeister/kafka:2.11-1.1.1        "start-kafka.sh"         53 seconds ago   Up 44 seconds   0.0.0.0:9092->9092/tcp, 0.0.0.0:29092->29092/tcp       weather-app-kafka-1
[illegible]    wurstmeister/zookeeper:3.4.6         "/bin/sh -c '/usr/sb…"   53 seconds ago   Up 44 seconds   22/tcp, 2888/tcp, 3888/tcp, 0.0.0.0:2181->2181/tcp     weather-app-zookeeper-1
18dc5cc4b200   weather-app-database-service         "python app.py"          53 seconds ago   Up 44 seconds   0.0.0.0:5000->5000/tcp                                 weather-app-database-service-1
425fe9041c72   prom/prometheus                      "/bin/prometheus --c…"   53 seconds ago   Up 44 seconds   0.0.0.0:9090->9090/tcp                                 weather-app-prometheus-1
```

A questo punto eseguire:
Sostituisci "your-kafka-container-id" con l'ID del container Kafka
$containerId = "metti id"

Esegui il container kafka-consumer
docker exec -it $containerId kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic prometheusdata --from-beginning

```
PS C:\Users\gerar\Desktop\weather-app> # Sostituisci "your-kafka-container-id" con l'ID del container Kafka (es. weather-app-kafka-1)
PS C:\Users\gerar\Desktop\weather-app> $containerId = "2e60ce975ad0"
PS C:\Users\gerar\Desktop\weather-app>
PS C:\Users\gerar\Desktop\weather-app> # Esegui il container kafka-consumer
PS C:\Users\gerar\Desktop\weather-app> docker exec -it $containerId kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic prometheusdata --from-beginning
{"utenti_registrati_check": false, "numero_richieste_check": true, "cpu_checks": [true, false, false, false], "memory_checks": [true, true, true, false], "false_checks_count": 10}
{"utenti_registrati_check": false, "numero_richieste_check": true, "cpu_checks": [true, false, false, false], "memory_checks": [true, true, true, false], "false_checks_count": 10}
{"utenti_registrati_check": false, "numero_richieste_check": true, "cpu_checks": [true, false, false, false], "memory_checks": [true, true, true, false], "false_checks_count": 15}
{"utenti_registrati_check": false, "numero_richieste_check": true, "cpu_checks": [true, false, false, false], "memory_checks": [true, true, true, false], "false_checks_count": 20}
{"utenti_registrati_check": false, "numero_richieste_check": true, "cpu_checks": [true, false, false, false], "memory_checks": [true, true, true, false], "false_checks_count": 25}
{"utenti_registrati_check": false, "numero_richieste_check": true, "cpu_checks": [true, false, false, false], "memory_checks": [true, true, true, false], "false_checks_count": 30}
{"utenti_registrati_check": false, "numero_richieste_check": true, "cpu_checks": [true, false, false, false], "memory_checks": [true, true, true, false], "false_checks_count": 35}
{"utenti_registrati_check": false, "numero_richieste_check": true, "cpu_checks": [true, false, false, false], "memory_checks": [true, true, true, false], "false_checks_count": 40}
```

15) **Probabilità violazioni nei prossimi X minuti**

Definisci l'URL dell'endpoint
$url = "http://localhost:5003/probability_of_violations/inserisci X minuti"

Esegui la richiesta GET
$response = Invoke-RestMethod -Uri $url -Method Get -ContentType "application/json"

Visualizza la risposta
$response

Di seguito viene illustrata la probabilità di violazione nei prossimi 20 minuti:

```
PS C:\Users\gerar\Desktop\weather-app> # Definisci l'URL dell'endpoint
PS C:\Users\gerar\Desktop\weather-app> $url = "http://localhost:5003/probability_of_violations/20"
PS C:\Users\gerar\Desktop\weather-app>
PS C:\Users\gerar\Desktop\weather-app> # Esegui la richiesta GET
PS C:\Users\gerar\Desktop\weather-app> $response = Invoke-RestMethod -Uri $url -Method Get -ContentType "application/json"
PS C:\Users\gerar\Desktop\weather-app>
PS C:\Users\gerar\Desktop\weather-app> # Visualizza la risposta
PS C:\Users\gerar\Desktop\weather-app> $response

Probabilità di violazione nei prossimi 20 minuti è pari a
---------------------------------------------------------
                                                      3,0
```