# 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):

Immagine che contiene schermata, testo, software, Software multimediale

Descrizione generata automaticamente

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)"

Immagine che contiene testo, schermata, Carattere, software

Descrizione generata automaticamente}

1. **Registrazione utente:**

$url = "http://localhost:5001/user"

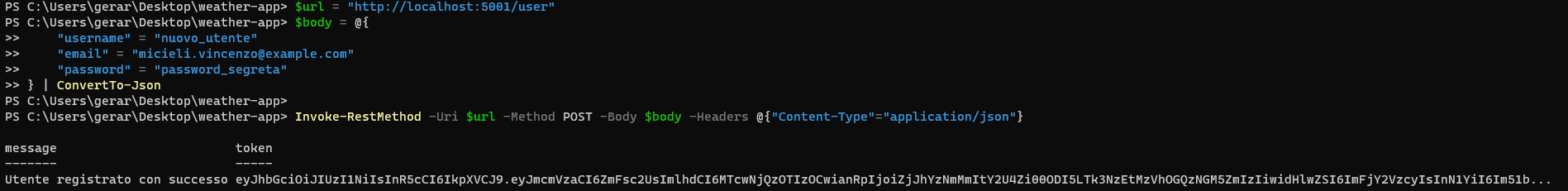
$body = @{

"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"}

Gli utenti vengono registrati all’interno di questo database:

Immagine che contiene schermata, software, Software multimediale, testo

Descrizione generata automaticamente

1. **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

1. **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

Immagine che contiene testo, schermata, software

Descrizione generata automaticamenteInvoke-RestMethod -Uri $url -Method POST -Body $body -Headers $headers

Immagine che contiene testo, schermata, Carattere, software

Descrizione generata automaticamenteUna 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.

1. **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

1. **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)"

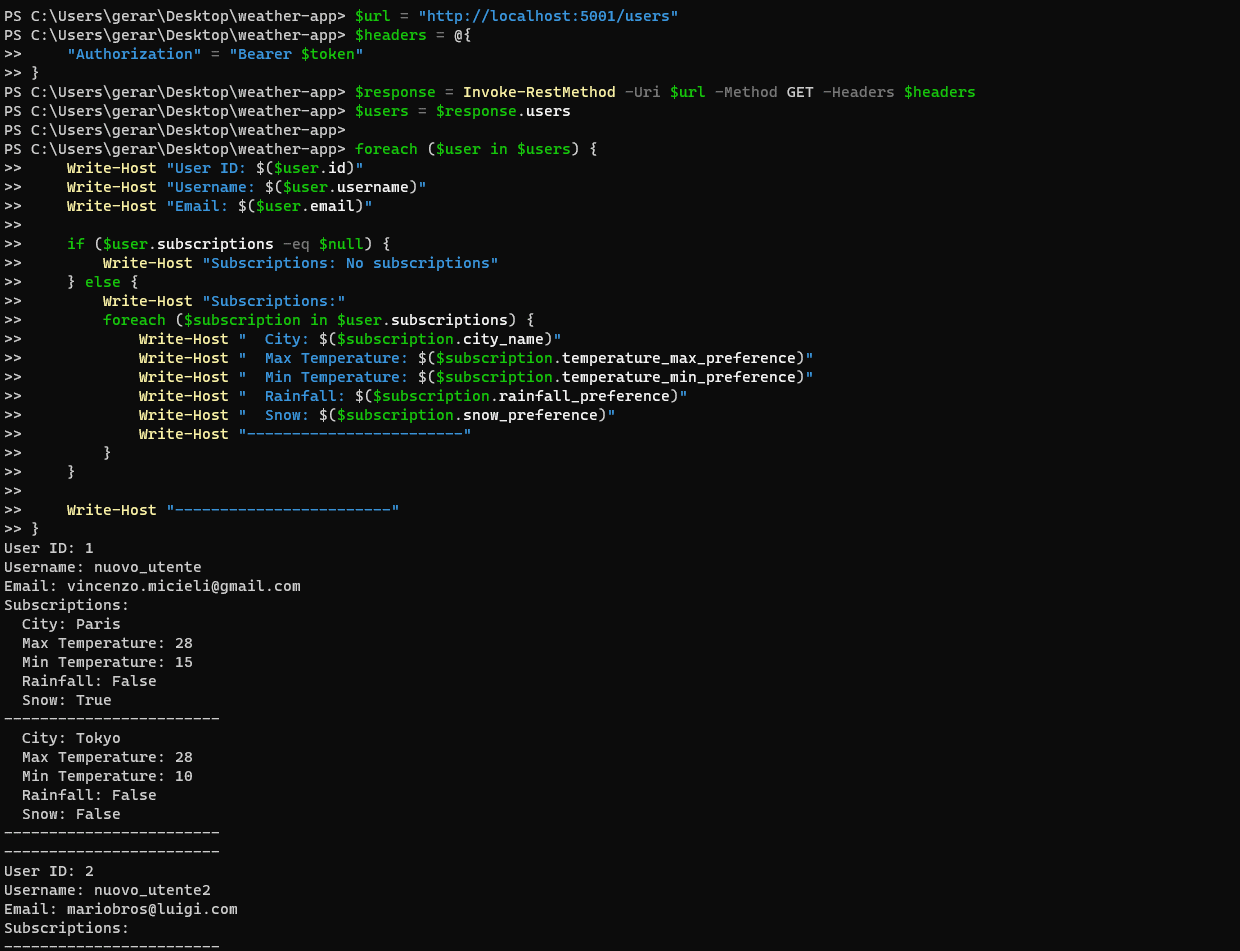
Write-Host " Snow: $($subscription.snow\_preference)"

Write-Host "------------------------"

}

}

Write-Host "------------------------"

}

1. **Metriche SLA**

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

**Immagine che contiene testo, schermata, Carattere, Software multimediale

Descrizione generata automaticamente**

1. **Immagine che contiene testo, schermata, software, Software multimediale

   Descrizione generata automaticamenteRimozione 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

Infatti, adesso avremo:

Immagine che contiene testo, schermata

Descrizione generata automaticamente

1. **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

1. **Prelevamento delle violazioni dal database**

Immagine che contiene schermata, testo, software, Software multimediale

Descrizione generata automaticamenteLe 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:  
 Immagine che contiene testo, schermata, software

Descrizione generata automaticamente

1. **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

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamenteNotiamo infatti adesso che le metriche desiderate sono state aggiornate:  
Immagine che contiene testo, schermata, design

Descrizione generata automaticamente

1. **Stato SLA**

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

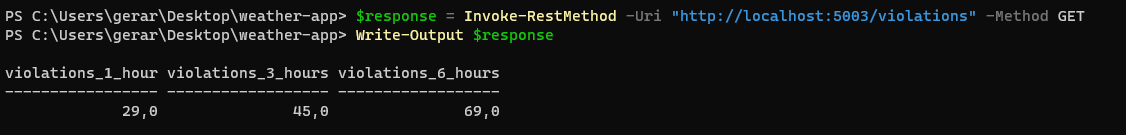
**Immagine che contiene testo, schermata, software

Descrizione generata automaticamente**Write-Output $response

1. **Numero violazioni SLA 1h, 3h e 6h**

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

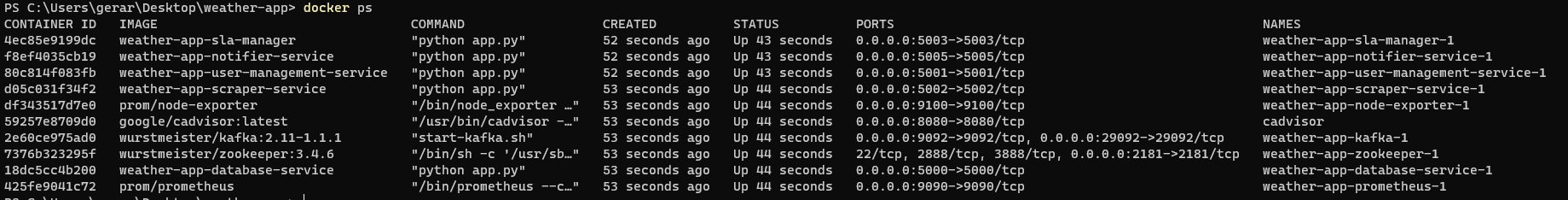
Write-Output $response

****

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

1. **Kafka**

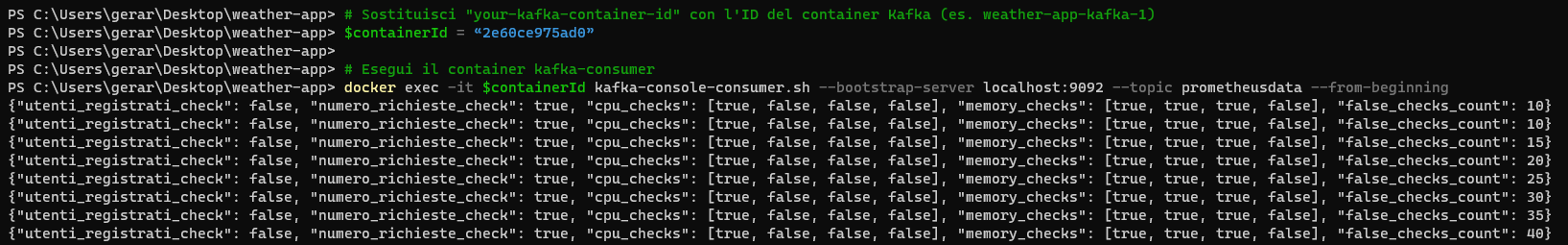
Attaverso il comando docker ps si preleva l’id di kafka:

 docker ps per trova id di kafka

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

1. **Probabilità violazioni nei prossimi X minuti**

# Definisci l'URL dell'endpoint

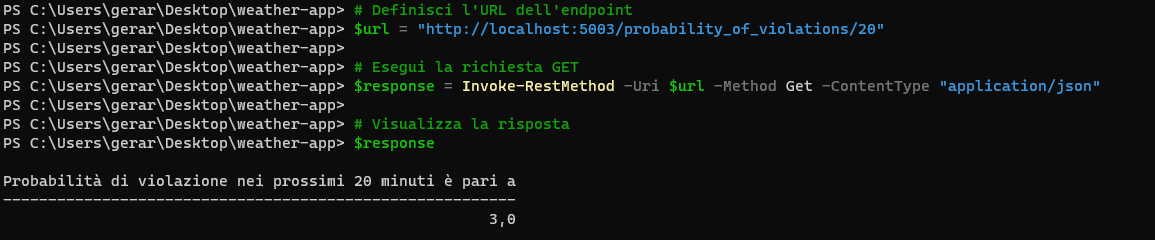
$url = "http://localhost:5003/probability\_of\_violations/inserisci X minuti"

# 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: