Integration testing con Postman e Newman

20 Gennaio 2023 - 8 min. read

CI/CD | DevOps | Integration Testing | Newman | Postman

Introduzione

Bentornato sul nostro blog Proud2beCloud!

Oggi andremo ad esplorare come implementare test d'integrazione per le tue API direttamente all'interno di pipeline utilizzando Postman e Newman.

Parleremo di Newman, Pipeline, Codebuild, Testing, Scripting, condivisibilità della soluzione e molto altro.

Cominciamo!

Perché dovrei implementare integration test?

Avere la possibilità di rilasciare API in sicurezza in produzione ed avere la certezza che il comportamento atteso sia rispettato è cruciale. **Solo con unit test non possiamo avere questo tipo di certezza.**

Ecco un esempio: immagina di avere un'applicazione unit testata perfettamente. Dato un input noto, il codice funziona perfettamente. Davanti al codice applicativo abbiamo un utente che chiama un API e invia dati all'applicazione.

Gli Unit test non sono abbastanza; vogliamo essere sicuri che anche la chiamata sia valida, testato header, body prima dell'applicativo, timeout, status code e vari tipi di dati e metadati relativi alla chiamata in sè e non solo alla logica.

In applicazioni reali, col passare del tempo i comportamenti utente e le applicazioni cambiano, moduli connessi tra di loro è probabile che vengano modificati o che venga

modificato come i dati sono passati tra di essi.

Potremmo addirittura avere applicazioni terze parti integrate con la nostra soluzione e non avremmo la possibilità di effettuare unit test.

Altre volte (anche se questo non dovrebbe capitare!), gli sviluppatori rilasciano applicazioni in velocità senza test completi ed esaustivi. Ecco dove i test d'integrazione vengono in nostro aiuto.

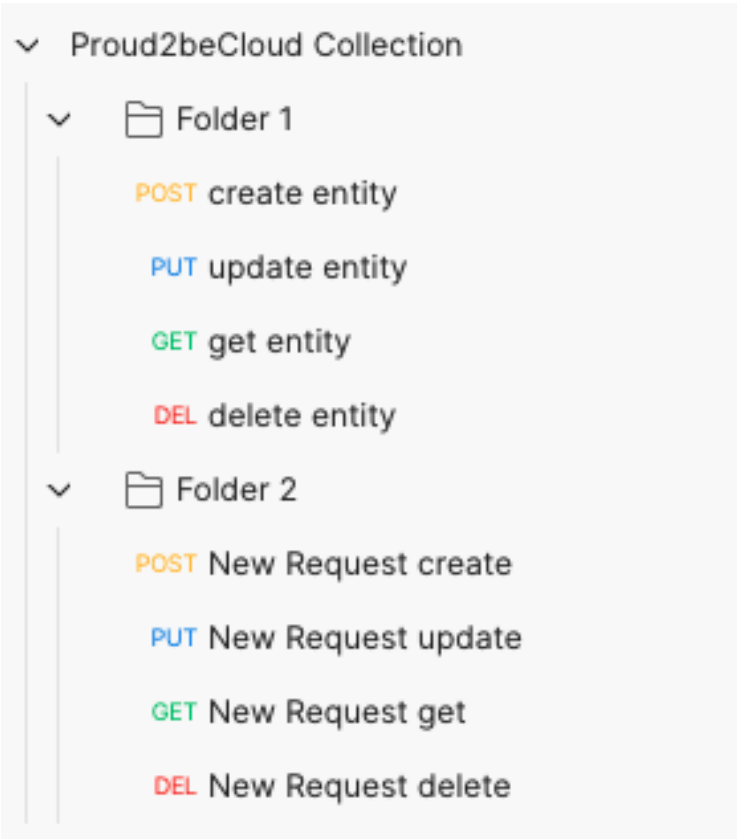
Queste appena citate sono alcune delle motivazioni principali per cui è necessario utilizzare anche i test di integrazione.

Nel nostro caso, parlando di API Gateway e Lambda, vogliamo anche testare le rotte, i decoratori e tutto ciò che si frappone tra il client e il codice applicativo.

Diamo un'occhiata a Postman

Best practices:

- Separare API in collection in base al progetto
- Separare API in cartelle in base al tipo di entità o alla categoria
- Scrivere API in ordine logico: POST - GET - PUT - DELETE
- Usare variabili, ove possibile, aiuta nella generazione di dati random e quindi più realistici e aiuta enormemente la creazione di una collection dinamica, anche su ambienti diversi.



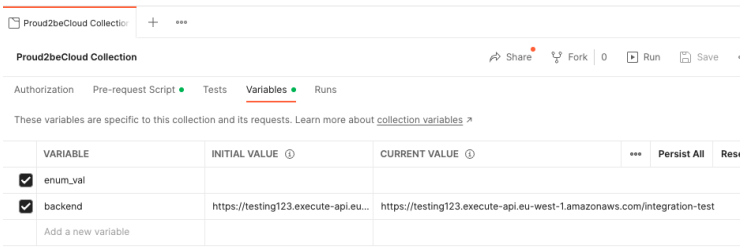
Variabili

In Postman abbiamo due tipi di variabili differenti:

1. Collection
2. Ambiente

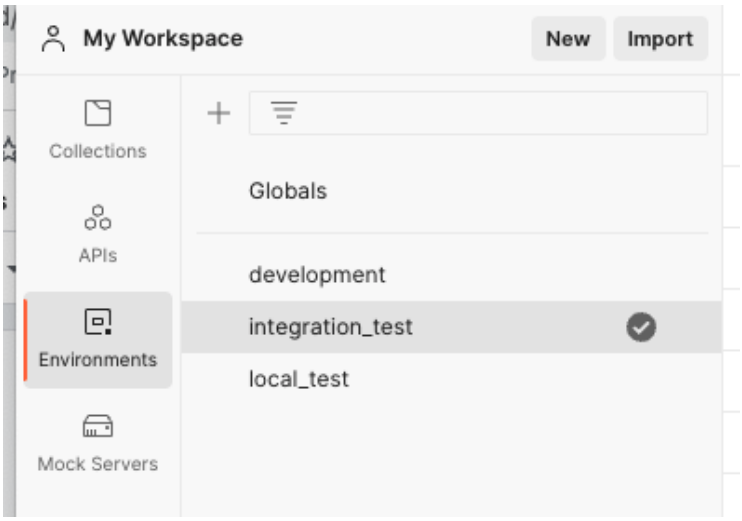
Collection:

Queste sono variabili utilizzate da tutta la collection, indipendentemente dall'ambiente in cui siamo. Dovrebbero essere utilizzate per parametri che sono indipendenti dall'ambiente come il nome dell'applicazione, variabili custom del business, etc.



Ambiente:

In Postman è possibile creare ambienti diversi. Possiamo definire variabili all'interno di questi ambienti come per la collection:



Se la stessa chiave di una variabile è presente sia come variabile di Collection, che di ambiente, verrà utilizzata quella di ambiente e quella di Collection sarà ignorata.

Best practice, secondo noi:

Creare almeno un ambiente locale, uno per integration test e uno per l'ambiente di sviluppo. Switchare tra ambienti semplificherà di molto i test diretti verso API Gateway senza dover mai modificare le singole chiamate.

Nota: Postman di default mette a disposizione già un set di variabili come {{randomEmail}}, {{randomInt}, {{$randomAlphaNumeric}} ed altre, molto utili per la randomizzazione del body in chiamate POST e PUT.

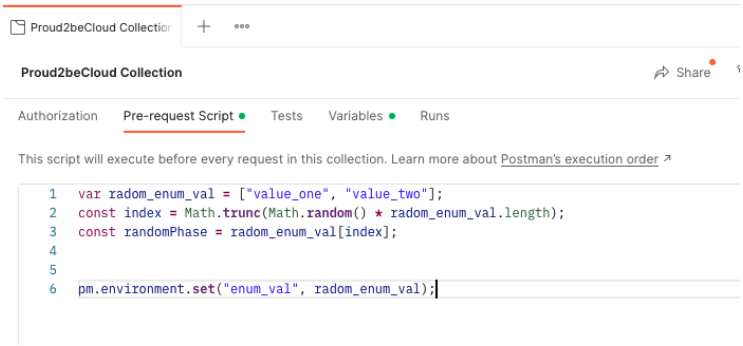
Pre-request script

Sia per la Colelction, che per una singola chiamata API, c'è la possibilità di usare la **pre-request script,** che è uno script js-like eseguito prima della chiamata API o dell'inizio dell'esecuzione della collection. Ecco alcuni esempi:

1. Setup variabili in maniera randomica riflettendo un ENUM tipo di dato

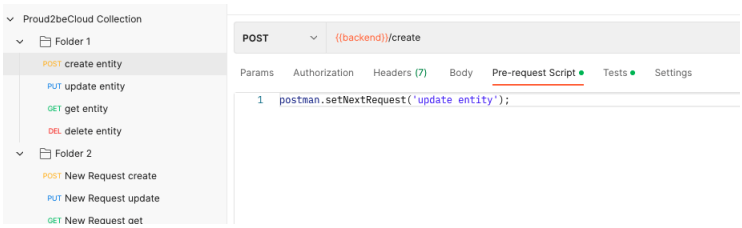
L'esecuzione di questo script andrà a settare nella variabile di collcection *enum_val* un valore random tra 'value_one' e 'value_two'

Molto utile per randomizzare variabili con valori custom, andrà scritto nel pre-request di collection:



2. Settare la prossima API eseguita. Postman di default eseguirà la collection nell'ordine in cui sono state inserite le chiamate API. Se vogliamo cambiare quest'ordine (eseguendo ad esempio le DELETE alla fine), possiamo farlo usando la pre-request della singola chiamata come si seguito:

```
postman.setNextRequest('update entity');
```



Per fermare la collection possiamo usare la stessa funzione settando come parametro *null:*

```
postman.setNextRequest(null).
```

É una buona pratica avere una colleciton strutturata come abbiamo detto in precedenza. La maggior parte delle volte però abbiamo entità che dipendono da altre e non possiamo quindi eliminare l'entità principale prima di aver eseguito test sull'altra.

Vogliamo quindi scrivere le nostre nextRequest secondo un ordine di questo tipo:

Entity1.POST → Entity1.GET → Entity1.PUT → Entity2.POST → Entity2.GET → Entity2.PUT → … → … → … → EntityN.Delete → EntityN-1.DELETE → … → Entity2.DELETE → Entity1.DELETE

Saremo così in grado di creare, modificare ed accedere ad oggetti finché ne abbiamo necessità e cancellarli in cascata solo alla fine.

Tests

Come suggerisce il titolo, per eseguire test d'integrazione, abbiamo bisogno di test.

In questo caso, abbiamo un tab dedicato *'test'* all'interno di ogni singola chiamata API e verrà eseguito dopo l'esecuzione della chiamata in modo da poter accedere anche alla risposta.

Useremo questa sezione per due motivi principali:

1. **Set up di variabili in base alla response**

USE CASE esempio: esecuzione di una POST, mi aspetto che venga creato un nuovo ID e voglio poterlo utilizzare in una seguente chiamata in GET. Posso farlo recuperando la risposta, parsandola e inserendo il valore all'interno di una variabile.

Nella altre API potrò quindi utilizzare la variabile che è stata valorizzata in maniera dinamica es {{id_entity}}

Come farlo:

- Creare una variabile con una chiave autoesplicativa e non valorizzarla
- Scriptare all'interno della chiamata API il recupero della risposta e il setup della variabile

```
var jsonData = JSON.parse(responseBody);
postman.setEnvironmentVariable("id_entity", jsonData.entityId);
```

Avremo quindi valorizzato la nostra variabile *id_entity* con l'id appena creato in maniera dinamica.

2. Testare la risposta

All'interno dello stesso tab possiamo invocare la funzione *.test* di postman e aggiungere un check sulla risposta, su qualsiasi parte di essa.

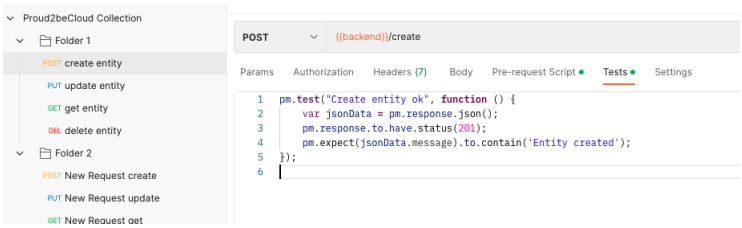
Il test sarà una valutazione True | False, il test quindi sarà OK o KO.

```
pm.test("Create entity ok", function () {
var jsonData = pm.response.json();
pm.response.to.have.status(201);
pm.expect(jsonData.message).to.contain('Entity created');
});
```

Questo script utilizza la response e controlla sia una parte del body che lo status code.

Se tutto ciò che viene testato all'interno della funzione .test() è corretto, il test verrà superato e risulterà un test 'verde'.

Una tab dedicata, come in screen, è presente in Postman e, come andremo a vedere, Newman fornirà inoltre un report su quante chiamate sono state eseguite, quanti test, quali sono passati, quali no e come mai.



Riassunto: postman collection completa

Spiegati i vari step e feature che vengono offerte dall'utilizzo di Postman, riassumiamo gli step per impostare una collection pronta per gli integration test:

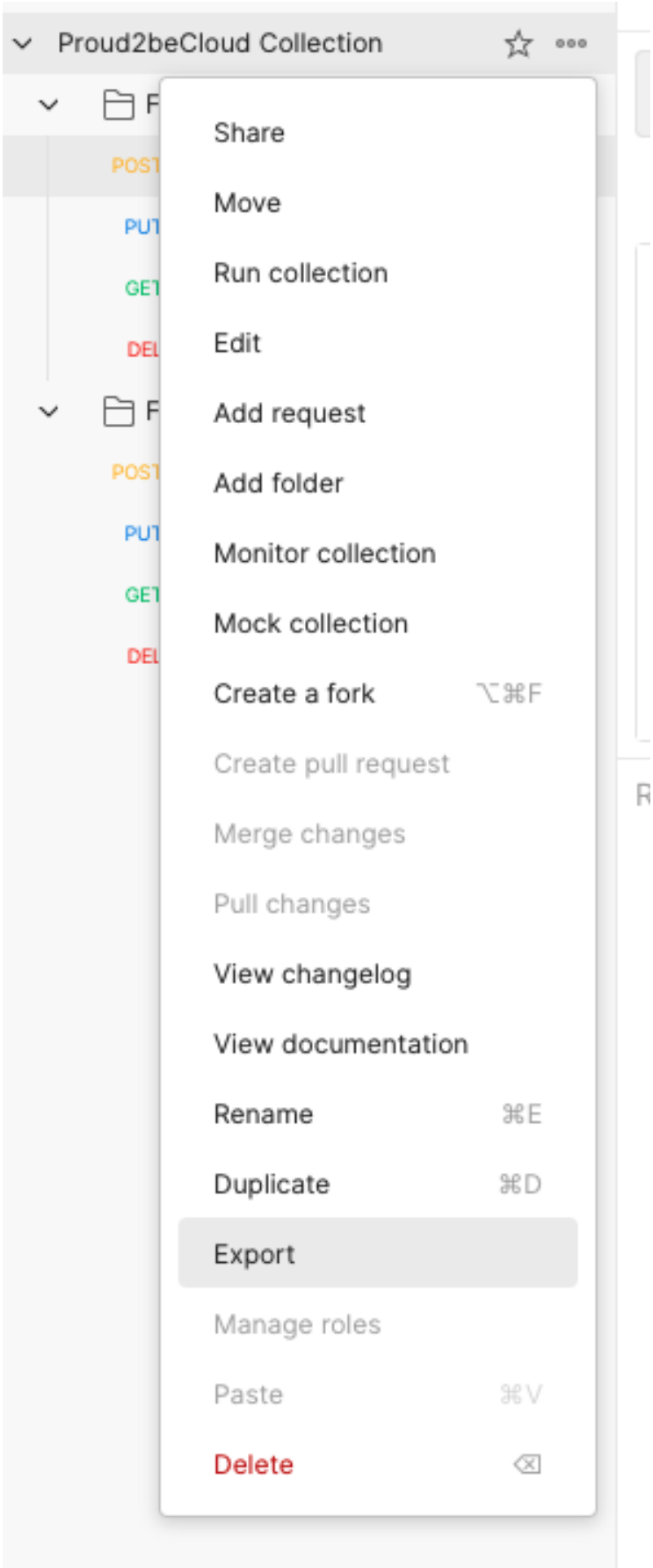
1. Creare un workspace
2. Creare una collection, e folder differenti
3. Creare ambienti, se necessario
4. Creare variabili sia di ambiente che di collection
5. Inserire Pre-request script di collection

6. Creare API in ordine logico (post, put, get, delete)
7. Inserire la chiamata .nextRequest in ogni Pre-request script per orchestrare l'esecuzione delle API
8. Impostare SetNextRequest(null) all'ultima chiamata API in modo da esser certi che la collection finisca
9. Eventualmente, settare variabili dinamiche all'interno delle singole API, es.: ID
10. Definire Test per le singole chiamate API

Collection in repository e Newman

Integrare la propria collection all'interno di Code Pipeline è piuttosto semplice:

1. Esportare la collection; verrà scaricato un file JSON che descrive tutte le chiamate API, tests e variabili di collection



2. Esportare, se presente, l'ambiente. Un JSON con le variabili di ambiente verrà scaricato

3. Inserire entrambi i file nella repository

4. Creare uno step di Code Build in CodePipeline. É importante installare newman, il buildspec potrebbe essere una cosa simile a:

```
version: 0.2
phases:
   install:
       runtime-versions:
           nodejs: 14
build:
   commands:
     - npm install -g newman
     - ./run_integration_test.sh
```

5. Creare un file bash *run_integration_test.sh* dove andremo ad eseguire la collection, in questo caso passando anche il file dell'ambiente 'integration_test':

```
#!/bin/bash

newman run postman_collection.json -e integration_test_env.json
```

Semplicemente così, Newman eseguirà la collection Postman con le variabili d'ambiente e seguendo dalla prima API l'ordine definito tramite le funzioni setNextRequest().

Ogni chiamata con un test definito sarà eseguita e valutata. Se anche solo un test fallisse lo step di build andrebbe in errore e la pipeline si fermerebbe. Sarà presente all'interno dei log il report sulle chiamate in modo da fare troubleshooting sulla/sulle chiamata/e in errore.

Note addizionali

Condivisibilità: tramite i workspace, è possibile creare workspace per cliente e condividerlo con il team (gratis fino a 3 membri) o condividere solo in view la collection.

Gli editor vedranno la collection aggiornata in tempo reale se modificata da altri membri del team e non dovranno importarsela ogni volta.

AWS Environment: per poter chiamare effettivamente le API è nesessario che il codice sia rilasciato prima dello step di integration test, questo non sarebbe possibile farlo in ambienti come development o Produzione perchè in caso di errore applicativo il rollback in pipeline non è automatico.

La soluzione da noi adottata è quella di creare un ambiente 'ombra'. Ogni pipeline che esegue gli integration test, staging & production ad esempio, rilascerà lo stesso pacchetto di codice applicativo su uno stack infrastrutturale dedicato per gli integration test: API

Gateway, Lambda, etc. Questo ambiente avrà così degli endpoint reali senza aver bisogno di una pipeline dedicata o di un setup dedicato lato codice.

Così possiamo sempre rilasciare il codice su questo ambiente, testarlo e, in caso di successo, rilasciarlo anche nell'ambiente reale con la sicurezza che si comporterà nello stesso modo (il pacchetto di codice è lo stesso).

Su questo ambiente di integration test verranno eseguiti spesso rilasci dalle pipeline che si sovrascriveranno a vicenda, rilasciando ad esempio produzione e in seguito UAT con nuovo codice.

Non è rilevante, l'unica cosa importante è che i test vengano eseguiti sul codice aggiornato.

Conclusioni

L'Integration testing è cruciale nello sviluppo di applicazioni web per garantire rilasci in produzione in sicurezza.

Per effetturali usiamo tool come Postman e Newman, configurabili e customizzabili per meglio adattarsi alle necessità di business. Questa soluzione è agnostica rispetto al linguaggio di sviluppo, essendo semplicemente un pacchetto npm che va ad eseguire chiamate tramite un file .json. Può essere integrato in progetti Python, Node, etc..

Integrare i test in un ambiente dedicato ed eseguirli direttamente all'interno di pipeline rende la tua soluzione di CI/CD più sicura e affidabile; mentenere il controllo sul comportamento del codice scritto, non è mai una brutta idea :)

Speriamo che questo articolo vi abbia lasciato buoni spunti per implementare una miglior strategia di testing. Lascia un commento per qualsiasi domanda o dubbio e raccontaci la tua esoperienza!

Rimani aggiornato sul mondo Cloud con i nostri prossimi articoli e, come sempre: **#Proud2beCloud**!

Alberto Casadei

DevOps Engineer @ beSharp. Sono pigro, questa è la realtà: il che mi permette di trovare sempre la soluzione tecnica più efficiente! Collezionista di carte, atleta e un sacco di altre cose nel tempo libero: “a ‘bit’ of everything” è il mio motto!
