

Cronologia delle revisioni

Versione	Data	Autore	Descrizione delle Modifiche
1.0.0	2025-09-02	Antonio Musarra	Prima release
1.0.1	2025-09-06	Antonio Musarra	Minor fix e miglioramenti
1.0.2	2025-09-07	Antonio Musarra	Minor fix, aggiunta di note e nuovo capitolo

[Cronologia delle revisioni](#)

[Prerequisiti](#)

[Configurazione della struttura del progetto](#)

[Configurazione delle variabili d'ambiente di Postman](#)

[Configurazione dei certificati client in Postman CLI](#)

[Come Postman CLI seleziona il certificato client](#)

[Creazione della collection Postman](#)

[Esecuzione della collection con Postman CLI](#)

[Verifica tramite exit code](#)

[Integrazione di Postman CLI in pipeline CI/CD](#)

[Esempio di utilizzo in GitHub Actions](#)

[Quando l'errore è il nostro migliore amico](#)

[Vantaggi dell'integrazione](#)

[Conclusioni](#)

[Risorse utili](#)

[Postman](#)

[GitHub](#)

[Articoli e Guide](#)

[Wikipedia](#)

[Altre Risorse](#)

L'obiettivo di questa guida è mostrare come testare endpoint REST protetti da mTLS ([Mutual TLS](#)), come quelli del progetto [quarkus-mtls-auth](#), attraverso l'utilizzo della [Postman CLI](#).

Vedremo come sfruttare gli ambienti per una configurazione flessibile e come gestire più certificati client in modo centralizzato tramite un file di configurazione, una pratica essenziale quando si lavora con diversi servizi.

La documentazione ufficiale di Postman CLI non fornisce informazioni dettagliate su come configurare mTLS, ne quanto meno esempi pratici. Seguendo questa guida, **ti assicuro che sarà un gioco da ragazzi!**

Segui la guida fino in fondo!

Alla fine troverai un bonus: il link al repository GitHub che contiene l'intero progetto pronto all'uso, con tutte le configurazioni, gli script e le collection Postman illustrate in questo articolo.

Prerequisiti

Per seguire questa guida, assicuriamoci di avere:

- [Postman CLI](#) installato sull'ambiente di esecuzione. Versione minima consigliata: 1.18.0, poiché alcune funzionalità potrebbero non essere disponibili nelle versioni precedenti;
- un progetto con endpoint protetti da mTLS, come per esempio [quarkus-mtls-auth](#);
- certificati client e CA ([Certificate Authority](#)) necessari per l'autenticazione mTLS.

Il login su Postman non è necessario per eseguire collection locali come in questo esempio, ma è richiesto nel caso in cui si voglia sincronizzare collection, ambienti o cronologia con il cloud di Postman.

Il progetto [quarkus-mtls-auth](#) offre un'implementazione di API REST protette dal meccanismo di sicurezza mTLS, che utilizzeremo per dimostrare la configurazione di Postman CLI. Quanto descritto in questa guida è applicabile a qualsiasi altro progetto e/o scenario simile.

Per quanto riguarda i certificati client e CA, esploreremo in seguito come generarli e configurarli correttamente in Postman CLI. La versione CLI, a differenza della versione Desktop, attualmente supporta solo certificati in formato PEM ([Privacy Enhanced Mail](#)).

Non ci sono restrizioni particolari riguardo il sistema operativo utilizzato, puoi seguire questa guida su Windows, macOS o Linux senza problemi. Per garantire la massima compatibilità, consiglio di utilizzare un terminale con supporto per i comandi Unix/Linux.

Per seguire in modo fluido questa guida, assicurati di avere familiarità con i concetti di base di mTLS e Postman e avere una certa dimestichezza con la linea di comando e il formato JSON.

Configurazione della struttura del progetto

Come primo passo, organizziamo un ambiente di lavoro che sia agevole e ben strutturato per Postman CLI. Inizieremo creando una cartella dedicata al nostro progetto e all'interno di essa, andremo a creare tutto il necessario per la configurazione.

```
1  # Creiamo la struttura delle cartelle
2  # La directory di lavoro (o root directory) sarà postman-mtls
3  mkdir postman-mtls
4
5  # Creiamo la cartella per i certificati client
6  mkdir -p postman-mtls/certificates/client
7
8  # Creiamo la cartella per i certificati CA
9  mkdir -p postman-mtls/certificates/ca
10
11 # Creiamo la cartella per la collection postman
12 mkdir postman-mtls/collections
13
14 # Creiamo la cartella per i file di configurazione (environments)
15 mkdir -p postman-mtls/config/environments
16
17 # Creiamo la cartella per i file di configurazione (certificates)
18 mkdir -p postman-mtls/config/certificates
```

Console 1 - Creazione della struttura delle cartelle per il progetto Postman mTLS

Una volta eseguiti i comandi precedenti, la struttura delle cartelle dovrebbe apparire come segue.

```
1  postman-mtls/
2  |─ certificates/
3  |   |─ client/
4  |   |   └─ ca/
5  |─ collections/
6  |─ config/
7  |   |─ environments/
8  |   └─ certificates/
```

Console 2 - Verifica della struttura delle cartelle (uso del comando `tree`)

Configurazione delle variabili d'ambiente di Postman

Usare le [variabili d'ambiente in Postman](#) è fondamentale per gestire configurazioni diverse senza modificare manualmente le richieste. Le variabili possono essere utilizzate per memorizzare valori come URL, token di accesso e certificati.

Per il nostro esempio, creeremo due ambienti: uno per lo sviluppo locale e un altro per l'ambiente di test. Vediamo quali sono le variabili che andremo a definire:

- `base_url`: l'URL di base del servizio. Questa variabile avrà un valore differente per ogni ambiente;
- `api_connection_info_path`: il path dell'API per le informazioni di connessione;
- `api_connection_user_identity`: il path dell'API per l'identità dell'utente;
- `env_name`: il nome dell'ambiente. È utile per identificare l'ambiente attivo durante l'esecuzione delle richieste e dei test.

La tabella a seguire riassume le variabili per ciascun ambiente.

Variabile	Ambiente local	Ambiente test	Descrizione
<code>env_name</code>	local	test	Nome identificativo dell'ambiente
<code>base_url</code>	https://localhost:8443	https://blog.quarkus.dontesta.it:8443	URL di base del servizio
<code>api_connection_info_path</code>	/api/v1/connection-info/info	/api/v1/connection-info/info	Path API per informazioni di connessione
<code>api_connection_user_identity</code>	/api/v1/connection-info/user-identity	/api/v1/connection-info/user-identity	Path API per identità utente

Tabella 1 - Variabili per gli ambienti

Queste variabili possono essere definite e create dall'interfaccia di Postman o tramite file di configurazione JSON. Noi faremo a meno dell'interfaccia grafica e utilizzeremo file di configurazione JSON per definire i nostri ambienti.

Andremo a creare due file JSON, uno per l'ambiente di sviluppo locale e uno per l'ambiente di test. Questi file conterranno le variabili necessarie per ciascun ambiente. Chiameremo i file `local.json` e `test.json` che andranno posizionati nella cartella `postman-mtls/config/environments`.

```
1 # Creiamo i file di configurazione per gli ambienti
2 touch postman-mtls/config/environments/local.json
3 touch postman-mtls/config/environments/test.json
```

Console 3 - Creazione dei file di configurazione per gli ambienti.

Il contenuto del file `local.json` sarà il seguente.

```
1  {
2    "id": "quarkus-mtls-local-environment",
3    "name": "Quarkus mTLS local environment",
4    "values": [
5      {
6        "key": "env_name",
7        "value": "local",
8        "description": "Name of the environment",
9        "type": "default",
10       "enabled": true
11     },
12     {
13       "key": "base_url",
14       "value": "https://localhost:8443",
15       "description": "Base URL for the local environment",
16       "type": "default",
17       "enabled": true
18     },
19     {
20       "key": "api_connection_info_path",
21       "value": "/api/v1/connection-info/info",
22       "description": "Path for API connection info",
23       "type": "default",
24       "enabled": true
25     },
26     {
27       "key": "api_connection_user_identity",
28       "value": "/api/v1/connection-info/user-identity",
29       "description": "Path for API user identity",
30       "type": "default",
31       "enabled": true
32     }
33   ]
34 }
```

Config. 1 - Configurazione variabili per l'ambiente locale.

Per l'ambiente di test il contenuto del file `test.json` sarà il seguente.

```

1  {
2    "id": "quarkus-mtls-test-environment",
3    "name": "Quarkus mTLS test environment",
4    "values": [
5      {
6        "key": "env_name",
7        "value": "test",
8        "description": "Name of the environment",
9        "type": "default",
10       "enabled": true
11     },
12     {
13       "key": "base_url",
14       "value": "https://blog.quarkus.dontesta.it:8443",
15       "description": "Base URL for the test environment",
16       "type": "default",
17       "enabled": true
18     },
19     {
20       "key": "api_connection_info_path",
21       "value": "/api/v1/connection-info/info",
22       "description": "Path for API connection info",
23       "type": "default",
24       "enabled": true
25     },
26     {
27       "key": "api_connection_user_identity",
28       "value": "/api/v1/connection-info/user-identity",
29       "description": "Path for API user identity",
30       "type": "default",
31       "enabled": true
32     }
33   ]
34 }

```

Config. 2 - Configurazione variabili per l'ambiente di test.

Importante: come `base_url` per l'ambiente di test abbiamo indicato l'FQDN `blog.quarkus.dontesta.it`, dobbiamo quindi assicurarci che questo sia risolvibile. In questo caso è più che sufficiente aggiungere una voce nel file `/etc/hosts` del nostro ambiente di sviluppo.

Una volta creati i file di configurazione per gli ambienti, possiamo procedere con la configurazione dei certificati client che saranno usati dalla CLI di Postman per eseguire le richieste HTTPS verso i servizi.

Configurazione dei certificati client in Postman CLI

La configurazione dei certificati client in Postman CLI è un passaggio cruciale per l'autenticazione mTLS. A differenza della versione Desktop di Postman, che consente di [configurare i certificati tramite l'interfaccia grafica](#), la CLI richiede l'uso di file di configurazione JSON.

Per gestire più certificati client in modo centralizzato, creeremo un file di configurazione adatto allo scopo. Questo file conterrà le informazioni necessarie per ogni certificato, inclusi i percorsi dei file PEM per il certificato e la chiave privata, compresa la relativa passphrase.

Chiameremo questo file di configurazione `client-certificates.json` e lo posizioneremo nella cartella `postman-mtls/config/certificates`.

A seguire l'esempio di configurazione per i certificati client. In questo caso, abbiamo configurato due certificati, sia per l'ambiente locale che per l'ambiente di test. Questa configurazione è solo un esempio e può essere adattata in base alle esigenze specifiche.

```

1  [
2    {
3      "name": "local-certificate-1",
4      "matches": ["https://localhost:8443/api/v1/connection-info/info"],
5      "key": { "src": "certificates/client/client_with_device_id_and_roles_key.pem" },
6      "cert": { "src": "certificates/client/client_with_device_id_and_roles_cert.pem"
7    },
8      "passphrase": "pZAXwq+l1BRlC+3X"
9    },
10   {
11     "name": "local-certificate-2",
12     "matches": ["https://localhost:8443/api/v1/connection-info/user-identity"],
13     "key": { "src": "certificates/client/client_with_device_id_and_roles_1_key.pem"
14   },
15     "cert": { "src": "certificates/client/client_with_device_id_and_roles_1_cert.pem"
16   },
17     "passphrase": "z93F18nhQR8shWTq"
18   },
19   {
20     "name": "test-certificate-1",
21     "matches": ["https://blog.quarkus.dontesta.it:8443/api/v1/connection-info/user-identity"],
22     "key": { "src": "certificates/client/client_new_device_id_key.pem" },
23     "cert": { "src": "certificates/client/client_new_device_id_cert.pem" },
24     "passphrase": "410nJ9nvreqleFnV"
25   },
26   {
27     "name": "test-certificate-2",
28     "matches": ["https://blog.quarkus.dontesta.it:8443/api/v1/connection-info/info"],
29     "key": { "src": "certificates/client/client_with_device_id_and_roles_1_key.pem"
30   },
31     "cert": { "src": "certificates/client/client_with_device_id_and_roles_1_cert.pem"
32   },
33     "passphrase": "z93F18nhQR8shWTq"
34   }
35 ]

```

```
29 |   }  
30 | ]
```

Config. 3 - Configurazione dei certificati client per mTLS.

Poiché il progetto su cui eseguiremo i test è [quarkus-mtls-auth](#), utilizzeremo certificati accettati dai servizi REST implementati in questo progetto. I certificati indicati sono stati generati appositamente per il testing tramite uno script specifico.

Per maggiori informazioni su come sono stati generati i certificati client, fare riferimento all'articolo [Implementazione di TLS Mutual Authentication \(mTLS\) con Quarkus](#) e in particolare al capitolo **Step 10 - Generazione di un set di certificati client per eseguire dei test di accesso**.

Come Postman CLI seleziona il certificato client

Quando si esegue una collection con Postman CLI e si specifica il file di configurazione dei certificati tramite l'opzione `--ssl-client-cert-list`, Postman seleziona automaticamente il certificato client da utilizzare per ogni richiesta HTTPS in base al campo `matches` presente nel file di configurazione.

Il campo `matches` è un array di URL o pattern che indicano a quali richieste il certificato deve essere associato. Durante l'esecuzione della collection, Postman confronta l'URL della richiesta con i valori definiti in `matches` per determinare quale certificato utilizzare.

- Se l'URL della richiesta corrisponde a uno dei valori definiti in `matches`, Postman utilizza il certificato associato a quella configurazione.
- Se nessun certificato corrisponde, la richiesta sarà inviata senza certificato client.

Questo meccanismo consente di gestire facilmente più certificati per diversi endpoint o ambienti, senza dover modificare manualmente la configurazione ad ogni esecuzione.

Esempio: se una richiesta viene inviata a `https://localhost:8443/api/v1/connection-info/info`, Postman CLI cercherà una configurazione con `matches` che includa esattamente quell'URL e userà il certificato specificato.

Per maggiori dettagli fare riferimento ad [Add and manage CA and client certificates in Postman](#)

Creazione della collection Postman

A questo punto possiamo creare la nostra collection Postman contenente le richieste necessarie per interagire con le API protette da mTLS, che in questo caso sono due: `/api/v1/connection-info/info` e `/api/v1/connection-info/user-identity`. La prima API restituisce informazioni sulla connessione, mentre la seconda fornisce dettagli sull'identità dell'utente.

Procediamo ora con la creazione del file JSON della collection, che salveremo nella cartella `postman-mtls/collections` con il nome `quarkus-mtls-collection.json`. Di seguito è riportato il contenuto del file.

```

1  {
2    "info": {
3      "name": "Quarkus mTLS",
4      "description": "Collection Postman per il test di servizi REST protetti da un
meccanismo di sicurezza di tipo mTLS.",
5      "schema":
"https://schema.getpostman.com/json/collection/v2.1.0/collection.json"
6    },
7    "item": [
8      {
9        "name": "connection-info/info",
10       "request": {
11         "method": "GET",
12         "header": [],
13         "url": {
14           "raw": "{{base_url}}{{api_connection_info_path}}",
15           "host": [
16             "{{base_url}}{{api_connection_info_path}}"
17           ]
18         },
19         "description": "La risposta del servizio conterrà un set
d'informazioni che riguardano la connessione instaurata."
20       },
21       "response": []
22     },
23     {
24       "name": "connection-info/user-identity",
25       "request": {
26         "method": "GET",
27         "header": [],
28         "url": {
29           "raw": "{{base_url}}{{api_connection_user_identity}}",
30           "host": [
31             "{{base_url}}{{api_connection_user_identity}}"
32           ]
33         },
34         "description": "La risposta del servizio conterrà un set
d'informazioni che riguardano l'identità del client che ha richiesto l'accesso alla
risorsa."

```

```

35         },
36         "response": [ ]
37     }
38 ]
39 }

```

Config. 4 - Creazione della collezione Postman per mTLS.

All'interno della collection abbiamo utilizzato le variabili definite nell'ambiente, come per esempio `base_url`, `api_connection_info_path` e `api_connection_user_identity`.

Vogliamo rendere la nostra collection ancora più "smart" per il testing automatizzato! Per farlo, aggiungeremo dei test alle richieste, così ogni ambiente avrà i suoi controlli dedicati. Immagina: ogni volta che lanci la collection, Postman CLI si trasforma in un vero e proprio "detective" delle API, pronto a scovare errori e confermare che tutto funzioni come previsto. La tabella qui sotto mostra quali test saranno eseguiti per ogni richiesta, a seconda dell'ambiente attivo. Pronti a scoprire chi supera l'esame?

Richiesta	Ambiente	Test eseguiti
connection-info/info	local	- Status code = 403
connection-info/info	test	- Status code = 200 - Content-Type = application/json;charset=UTF-8 - certCommonName = rd.quarkus.dontesta.it
connection-info/user-identity	local	- Status code = 200 - Content-Type = application/json;charset=UTF-8 - deviceId corretto
connection-info/user-identity	test	- Status code = 401

Tabella 2 - Test automatizzati per ogni richiesta e ambiente

Perché i test sono diversi tra gli ambienti?

Un'attenta osservazione della Tabella 2 rivela che i test per la stessa API cambiano a seconda dell'ambiente. Qualcuno potrebbe giustamente obiettare: "I test non dovrebbero essere identici?"

L'obiezione è assolutamente valida, ma in questo contesto la differenza è **intenzionale e strategica**. Non stiamo solo verificando che l'API "funzioni", ma stiamo validando **scenari di autorizzazione specifici** legati ai diversi certificati client usati in ogni ambiente.

Come descritto nella configurazione dei certificati (Config. 3), ogni ambiente associa certificati client differenti alle API. Di conseguenza:

- **Ambiente local1**: Per l'API `connection-info/info`, ci aspettiamo un `403 Forbidden`. Questo è un test di sicurezza "negativo": verifichiamo che un client autenticato ma **non autorizzato** venga correttamente respinto.
- **Ambiente test**: Per la stessa API, ci aspettiamo un `200 OK`. Questo è un test "positivo": verifichiamo che un client con un certificato diverso e **autorizzato** ottenga l'accesso.

In sintesi, sfruttiamo gli ambienti per simulare diversi profili di accesso e garantire che le regole di autorizzazione basate sui certificati siano implementate correttamente. Questo approccio va usato con le dovute considerazioni, ma in questo caso specifico è perfettamente giustificato.

Con riferimento alla tabella precedente, possiamo ora implementare i test all'interno della nostra collezione Postman. A seguire la nuova collection a cui daremo il nome `quarkus-mtls-collection-with-test.json` e posizioneremo sempre in `postman-mtls/collections`.

La sezione "event" di ogni richiesta conterrà gli script per l'esecuzione dei test. Gli script di test sono scritti in JavaScript e utilizzano l'API di test di Postman. Per maggiori approfondimenti fare riferimento alla documentazione ufficiale di Postman [Write scripts to test API response data in Postman](#).

```

1  {
2    "info": {
3      "name": "Quarkus mTLS",
4      "description": "Collection Postman per il test di servizi REST protetti da un
meccanismo di sicurezza di tipo mTLS.",
5      "schema":
"https://schema.getpostman.com/json/collection/v2.1.0/collection.json"
6    },
7    "item": [
8      {
9        "name": "connection-info/info",
10       "request": {
11         "method": "GET",
12         "header": [],
13         "url": {
14           "raw": "{{base_url}}{{api_connection_info_path}}",
15           "host": [
16             "{{base_url}}{{api_connection_info_path}}"
17           ]
18         },
19         "description": "La risposta del servizio conterrà un set
d'informazioni che riguardano la connessione instaurata."
20       },
21       "event": [
22         {
23           "listen": "test",
24           "script": {
25             "type": "text/javascript",
26             "exec": [
27               "console.log('Environment attivo:',
pm.environment.get('env_name'))";
28               "if (pm.environment.get('env_name') === 'local') {",
29               "    pm.test('Status code is 403', function () {",
30               "        pm.response.to.have.status(403);",
31               "    });",
32               "} else if (pm.environment.get('env_name') === 'test')
{",
33               "    pm.test('Status code is 200', function () {",
34               "        pm.response.to.have.status(200);",

```

```

35         });",
36         "    pm.test('Content-Type is application/json', function
37     () {",
38         "        pm.response.to.have.header('Content-Type',
39     'application/json;charset=UTF-8');",
40         "    });",
41         "    pm.test('certCommonName is rd.quarkus.dontesta.it',
42     function () {",
43         "        var jsonData = pm.response.json();",
44         "        pm.expect(jsonData.server.certCommonName).to.eql('rd.quarkus.dontesta.it');",
45         "    });",
46         "}"
47     ],
48     "response": []
49 },
50 {
51     "name": "connection-info/user-identity",
52     "request": {
53         "method": "GET",
54         "header": [],
55         "url": {
56             "raw": "{{base_url}}{{api_connection_user_identity}}",
57             "host": [
58                 "{{base_url}}{{api_connection_user_identity}}"
59             ]
60         },
61         "description": "La risposta del servizio conterrà un set
62     d'informazioni che riguardano l'identità del client che ha richiesto l'accesso alla
63     risorsa."
64     },
65     "event": [
66         {
67             "listen": "test",
68             "script": {
69                 "type": "text/javascript",
70                 "exec": [
71                     "console.log('Environment attivo:',
72     pm.environment.get('env_name'));",
73                     "if (pm.environment.get('env_name') === 'local') {",
74                     "    pm.test('Status code is 200', function () {",
75                     "        pm.response.to.have.status(200);",
76                     "    });",
77                     "    pm.test('Content-Type is application/json', function
78     () {",
79                     "        pm.response.to.have.header('Content-Type',
80     'application/json;charset=UTF-8');",
81                     "    });",
82                     "    pm.test('deviceId is valid', function () {",

```

```

78         "        var jsonData = pm.response.json();",
79         "
pm.expect(jsonData.attributes.deviceId).to.eql('MTc1NzAyNTE3OTk3MDAwODAwMCM1MjQwOGJkN
C1hZjQzLTQwZjYtODgxZi00YWNjOTQ4OTM3MjEjYW1lc2FycmEtbWFjYm9vaylwcm8ubG9jYWwzMmVlNjA0Y2
IyYzQ1Yjk0NDkwZDhmOWZiYTU4MTEwNjhkNDk1MmUwNjlkMzRmMzc0MjA5ODAzZmNmMjQ5ZWl0A==');",
80         "    });",
81         "} else if (pm.environment.get('env_name') === 'test')
{"",
82         "    pm.test('Status code is 401', function () {",
83         "        pm.response.to.have.status(401);",
84         "    });",
85         "}"
86     ]
87 }
88 }
89 ],
90 "response": [ ]
91 }
92 ]
93 }

```

Config. 5 - Aggiunta di test alla collection Postman

Ottimo lavoro! Ora che tutte le configurazioni sono pronte (ambienti, certificati e collection Postman sia con che senza test), siamo davvero a un passo dall'esecuzione pratica delle collection. Manca solo l'ultimo step: vedere tutto in azione!

Esecuzione della collection con Postman CLI

Arrivati a questo punto, la struttura del nostro progetto dovrebbe essere simile a quella indicata a seguire.

```

1 postman-mtls/
2 |— certificates
3 |   |— ca
4 |   |   |— ca_cert.pem
5 |   |— client
6 |       |— client_new_device_id_cert.pem
7 |       |— client_new_device_id_key.pem
8 |       |— client_new_device_id_key.pem.password
9 |       |— client_with_device_id_and_roles_1_cert.pem
10 |      |— client_with_device_id_and_roles_1_key.pem
11 |      |— client_with_device_id_and_roles_1_key.pem.password
12 |      |— client_with_device_id_and_roles_cert.pem
13 |      |— client_with_device_id_and_roles_key.pem
14 |      |— client_with_device_id_and_roles_key.pem.password
15 |— collections
16 |   |— quarkus-mtls-collection-with-test.json
17 |   |— quarkus-mtls-collection.json
18 |— config
19 |   |— certificates
20 |   |   |— client-certificates.json
21 |   |— environments
22 |       |— local.json
23 |       |— test.json

```

Console 4 - Struttura del progetto

Prima di eseguire le collection con Postman CLI, è importante assicurarsi che tutte le configurazioni siano corrette e che i certificati siano stati generati e posizionati nei percorsi appropriati.

Importante: prima di eseguire i test, assicurati che il progetto [quarkus-mtls-auth](#) sia avviato e che le API siano in ascolto sulla porta 8443 (come configurato negli ambienti local e test). Puoi trovare le istruzioni dettagliate nella sezione [Quickstart](#) del repository, dove viene spiegato come avviare il servizio tramite Docker o Podman. Ecco un esempio pratico:

```

1 # Scarica l'immagine aggiornata da Docker Hub
2 podman pull amusarra/quarkus-mtls-auth:latest
3
4 # Avvia il container sulla porta 8443
5 podman run -p 8443:8443 amusarra/quarkus-mtls-auth:latest

```

Per eseguire le collection con Postman CLI, si utilizza il comando `postman collection run` specificando il percorso della collection, il file di ambiente tramite l'opzione `--environment`, il file di configurazione dei certificati tramite l'opzione `--ssl-client-cert-list`.

Nel nostro caso specifichiamo anche l'opzione `--ssl-extra-ca-certs`. Questa opzione permette di specificare un file PEM contenente i certificati delle CA che Postman CLI utilizzerà per verificare il certificato del server. Questo è particolarmente utile quando si lavora con server che utilizzano certificati autofirmati o emessi da una CA privata evitando il classico errore: `unable to verify the first certificate`.

Il file `certificates/ca/ca_cert.pem` deve contenere il certificato della CA utilizzata per firmare il certificato del server. Può essere anche un bundle di certificati CA.

Ecco il comando base per eseguire una collection con Postman CLI usando environment e certificati.

```
1 # Comando base per l'esecuzione della collection
2 # Sostituisci i segnaposto con i valori appropriati
3 postman collection run collections/<nome_collection>.json \
4   --environment config/environments/<nome_ambiente>.json \
5   --ssl-client-cert-list
6   config/certificates/<nome_configurazione_certificati_client>.json \
7   --ssl-extra-ca-certs certificates/ca/<nome_certificato_ca>.pem
```

Console 4 - Esecuzione della collection con Postman CLI

A seguire una serie di comandi (completi) per eseguire le collection che abbiamo creato in precedenza e sui relativi ambienti: local e test.

```
1 # Esecuzione della collection senza test per l'ambiente local
2 postman collection run collections/quarkus-mtls-collection.json \
3   --environment config/environments/local.json \
4   --ssl-client-cert-list config/certificates/client-certificates.json \
5   --ssl-extra-ca-certs certificates/ca/ca_cert.pem
6
7 # Esecuzione della collection con test per l'ambiente local
8 postman collection run collections/quarkus-mtls-collection-with-test.json \
9   --environment config/environments/local.json \
10  --ssl-client-cert-list config/certificates/client-certificates.json \
11  --ssl-extra-ca-certs certificates/ca/ca_cert.pem
12
13 # Esecuzione della collection senza test per l'ambiente test
14 postman collection run collections/quarkus-mtls-collection.json \
15   --environment config/environments/test.json \
16   --ssl-client-cert-list config/certificates/client-certificates.json \
17   --ssl-extra-ca-certs certificates/ca/ca_cert.pem
18
19 # Esecuzione della collection con test per l'ambiente test
20 postman collection run collections/quarkus-mtls-collection-with-test.json \
21   --environment config/environments/test.json \
22   --ssl-client-cert-list config/certificates/client-certificates.json \
23   --ssl-extra-ca-certs certificates/ca/ca_cert.pem
```

Console 5 - Esempi di esecuzione della collection con e senza test per l'ambiente local e test

La figura mostra l'output della Postman CLI durante l'esecuzione della collection con test sull'ambiente di test. Si evidenziano i risultati dei test automatici sulle richieste, lo stato delle asserzioni, il dettaglio delle risposte HTTP ricevute e le statistiche di esecuzione (numero di richieste, test eseguiti, tempo medio di risposta, dati ricevuti, ecc.). Questo tipo di output consente di verificare rapidamente il successo dei test e la correttezza della configurazione mTLS.

```
Running your collection...
postman

Quarkus mTLS

→ connection-info/info

GET https://blog.quarkus.dontesta.it:8443/api/v1/connection-info/info [200 OK, 4.87 KB, 140 ms]
[
  'Environment attivo:', 'test'
]
✓ Status code is 200
✓ Content-Type is application/json
✓ certCommonName is rd.quarkus.dontesta.it

→ connection-info/user-identity

GET https://blog.quarkus.dontesta.it:8443/api/v1/connection-info/user-identity [401 Unauthorized, 180 B, 24 ms]
[
  'Environment attivo:', 'test'
]
✓ Status code is 401
```

	executed	failed
iterations	1	0
requests	2	0
test-scripts	2	0
prerequisite-scripts	0	0
assertions	4	0
total run duration: 199 ms		
total data received: 4.87 KB (approx)		
average response time: 82 ms [min: 24 ms, max: 140 ms, s.d.: 58 ms]		

Figura 1 - Esecuzione della collection con test per l'ambiente test

Nel caso in cui volessimo ottenere più informazioni in output come per esempio i dettagli delle richieste e delle risposte, possiamo utilizzare l'opzione `--verbose`.


```

Running your collection...
postman

Quarkus mTLS

→ connection-info/info

GET https://blog.quarkus.dontesta.it:8443/api/v1/connection-info/info
200 OK * 31 ms time * 268 B* 4.87 KB* size * 7* 2* headers * 0 cookies
[
  {
    "application/json * text * json * utf8 * 4.79 KB
    {
      "httpRequestHeaders": {
        "Accept": "*/*",
        "Cache-Control": "no-cache",
        "User-Agent": "PostmanRuntime/7.44.1",
        "Connection": "keep-alive",
        "Postman-Token": "c683384b-4463-473c-b12e-cca1a28d4eb3",
        "Host": "blog.quarkus.dontesta.it:8443",
        "Accept-Encoding": "gzip, deflate, br",
        "server": {
          "notAfter": "Sun Sep 07 00:16:21 CEST 2025",
          "keyAlgorithm": "RSA",
          "keySize": 2048,
          "certCommonName": "rd.quarkus.dontesta.it",
          "certIssuer": "CN=Dontesta CA,OU=IT Labs,O=Dontesta,L=Bronte,ST=Catania,C=IT",
          "subjectAlternativeNames": [
            [2, "admin.quarkus.dontesta.it"],
            [2, "blog.quarkus.dontesta.it"],
            [2, "localhost"]
          ],
          "certSubject": "CN=rd.quarkus.dontesta.it,OU=IT Labs,O=Dontesta,L=Bronte,ST=Catania,C=IT",
          "certSerialNumber": "581958667903693191343101950434363490497173665817",
          "certPEM": "MIID/jCCAuagAwIBAgIUZe/t4FUG9/ojdeqL38Zhq3WM0BkwDQYJKoZIhvcNAQELBQAwazELMAkGA1UEBhMCNVQxEDA0BgNVBAMGB0NhZGFuYWExDzANBgNVBACMBkYyb250ZTERMA8GA1UECgwIRG9udGVzdGEwYXQxwggEiMA0GCSqGSIb3DQEBQUAAQIDwAwggEKAoIBAQDLnmQigmVfsrSDiXLoGVS9+92aIWZAqt8pWu6rp6l137AV9XgAIsfvJAdHty/cG+kVeq4WFFq0HKk7uyB0SwP6P5MzmT1pF3WYFbul4kXYU2UitBoSbDoAFSJHqb/YrR8hJR3KX2LSANS6T2bCIZhj50zEUCUBeh8h3GnyKvQgkj2ngWx/Z3LCIAqZKx866ehP3TSI2qRAX7pKmI0KIvNmJsRzRm/BA4fV7BqRXCN8KFkK4oLDYWjcQagfeXp29JIU7/VCvASUNLiTpRr4XFndc4IMYu73pJELiFNG6rvCxY8NpQ9Wrr/ED7dsD8553f+QlKqoJWEsTKCchJmn307AgMBAAGjY4wYswSQYDVR0RBEIwQIIZYWRtaW4ucXVhcm1cy5kb250ZXN0YS5pdIIYYmxvZy5xdWYya3VzLmRvbnRlc3RhLmL0ggLsbn2NhbGhvc3QwHQYDVR0BBYEFMwZQvKv0vmfr5XV9KnaGakYyDWM8GA1UdIwQYMBaAFcJL01X++qptNNRFVp2oaPHkbSCjMA0GCSqGSIb3DQEBQwUAA4IBAQAPLJAQyikIXij8vwNdxtXV0oB0DDyFo57uyXkPjvKJa/7lr+lm0FmIKNiSyXkoleNtEH93PUy2s/LirllyaqLZ4obMCqf+JLyVsIhlaccvYeMoqkmcKkBEHJ5taSYxfajPL6CcMePc7mazwYteHv+CYQIa1D1Jq7FADBC1KHHS/bfuVlqT/ecXbkk0q50GZs40wNL2+/iMPNBKMKuI2o86Uih0P9JNSLJGCPQ8a5hFJTka8o1DZL+e31Tjr1v42iUjTXxSUsM9MojXsC
      }
    }
  ]
}
(showing 2 KB/4.79 KB)

prepare  wait   dns-lookup  tcp-handshake  ssl-handshake  transfer-start  download  process  total
112 ms   5 ms    1 ms        588 µs         12 ms          7 ms           3 ms      507 µs   143 ms

[
  'Environment attivo:', 'test'
]
✓ Status code is 200
✓ Content-Type is application/json
✓ certCommonName is rd.quarkus.dontesta.it

→ connection-info/user-identity

GET https://blog.quarkus.dontesta.it:8443/api/v1/connection-info/user-identity
401 Unauthorized * 22 ms time * 277 B* 180 B* size * 7* 2* headers * 0 cookies
[
  {
    "application/json * text * json * utf8 * 85 B
    {
      "statusCode": 401, "message": "Decoded roles do not match the expected pattern: Role="
    }
  ]
}

```

Figura 2 - Output dettagliato della collection con l'opzione `--verbose`

La Figura 2 mostra un esempio di output ottenuto con l'opzione `--verbose` abilitata. In questo caso vengono visualizzati ulteriori dettagli sulle richieste HTTP, sulle risposte ricevute, sulle intestazioni, sui tempi di handshake TLS e sulle statistiche di trasferimento dati. Questo livello di dettaglio è utile per il troubleshooting e per analizzare il comportamento della comunicazione mTLS tra client e server.

Le due immagini seguenti mostrano il caso in cui i test automatizzati sull'endpoint `/api/v1/connection-info/info` falliscono.

	executed	failed
iterations	1	0
requests	2	0
test-scripts	2	0
prerequisite-scripts	0	0
assertions	4	3
total run duration: 765 ms		
total data received: 85 B (approx)		
average response time: 295 ms [min: 26 ms, max: 565 ms, s.d.: 269 ms]		
average DNS lookup time: 24 ms [min: 3 ms, max: 45 ms, s.d.: 21 ms]		
average first byte time: 251 ms [min: 7 ms, max: 495 ms, s.d.: 243 ms]		

#	failure	detail
1.	AssertionError	Status code is 200 expected response to have status code 200 but got 403 at assertion:0 in test-script inside "connection-info/info"
2.	AssertionError	Content-Type is application/json expected response to have header with key 'Content-Type' at assertion:1 in test-script inside "connection-info/info"
3.	JSONError	certCommonName is rd.quarkus.dontesta.it No data, empty input at 1:1 ^ at assertion:2 in test-script inside "connection-info/info"

Figura 3 - Riepilogo esecuzione collection con test falliti

La **prima immagine** evidenzia il riepilogo dell'esecuzione della collection. È mostrato il numero totale di richieste, test-script e asserzioni eseguite, con il dettaglio degli errori. In questo esempio, 3 asserzioni sono fallite:

- Il test sullo status code atteso (200) ha fallito perché la risposta è stata 403 Forbidden.
- Il test sulla presenza dell'header `Content-Type` con valore `application/json` non è stato superato.
- Il test sul valore dell'attributo `certCommonName` nel JSON di risposta non è stato eseguito correttamente a causa di input vuoto (nessun body JSON restituito).

```

→ connection-info/info

GET https://blog.quarkus.dontesta.it:8443/api/v1/connection-info/info
403 Forbidden * 565 ms time * 268 B↑ 45 B↓ size * 7↑ 1↓ headers * 0 cookies
↓ text/plain * text * plain * utf8

prepare  wait  dns-lookup  tcp-handshake  ssl-handshake  transfer-start  download  process  total
125 ms   6 ms   45 ms      950 µs        13 ms         495 ms        2 ms      581 µs    691 ms

[ 'Environment attivo:', 'test'

1. Status code is 200
2. Content-Type is application/json
3. certCommonName is rd.quarkus.dontesta.it

```

Figura 4 - Dettaglio richiesta/risposta con test falliti

La **seconda immagine** mostra il dettaglio della richiesta e della risposta. Si vede la richiesta GET verso l'endpoint `/api/v1/connection-info/info` sull'ambiente di test, che restituisce HTTP 403 Forbidden. Sono riportati i tempi di preparazione, handshake TLS, download e processing.

È anche mostrato l'output degli script di test, con il dettaglio delle asserzioni fallite (status code, content-type, valore certCommonName), confermando che la risposta non soddisfa i criteri attesi dai test automatizzati.

Queste schermate aiutano a diagnosticare rapidamente i motivi del fallimento dei test e a individuare eventuali problemi di configurazione, autorizzazione o formato della risposta dell'API.

Verifica tramite exit code

Quando si esegue una collection con Postman CLI, il comando restituisce un **exit code** che permette di verificare rapidamente se l'esecuzione è andata a buon fine:

- **Exit code 0:** tutti i test sono passati con successo.
- **Exit code diverso da 0:** almeno un test è fallito oppure si è verificato un errore durante l'esecuzione.

Questo comportamento consente di integrare facilmente Postman CLI in pipeline di Continuous Integration/Continuous Delivery (CI/CD) o script di automazione, gestendo il flusso in base al risultato dei test.

Esempio di verifica in bash:

```
1 postman collection run collections/quarkus-mtls-collection-with-test.json \  
2   --environment config/environments/test.json \  
3   --ssl-client-cert-list config/certificates/client-certificates.json \  
4   --ssl-extra-ca-certs certificates/ca/ca_cert.pem  
5  
6 if [ $? -eq 0 ]; then  
7   echo "Tutti i test sono passati correttamente."  
8 else  
9   echo "Alcuni test sono falliti o si è verificato un errore."  
10 fi
```

Console 6 - Verifica del buon fine tramite exit code

Integrazione di Postman CLI in pipeline CI/CD

L'automazione dei test mTLS tramite Postman CLI può essere facilmente integrata in pipeline di CI/CD, come quelle basate su GitHub Actions, GitLab CI, Jenkins, Azure DevOps, ecc.

Esempio di utilizzo in GitHub Actions

Di seguito un esempio di step che esegue una collection Postman e verifica il risultato tramite l'exit code:

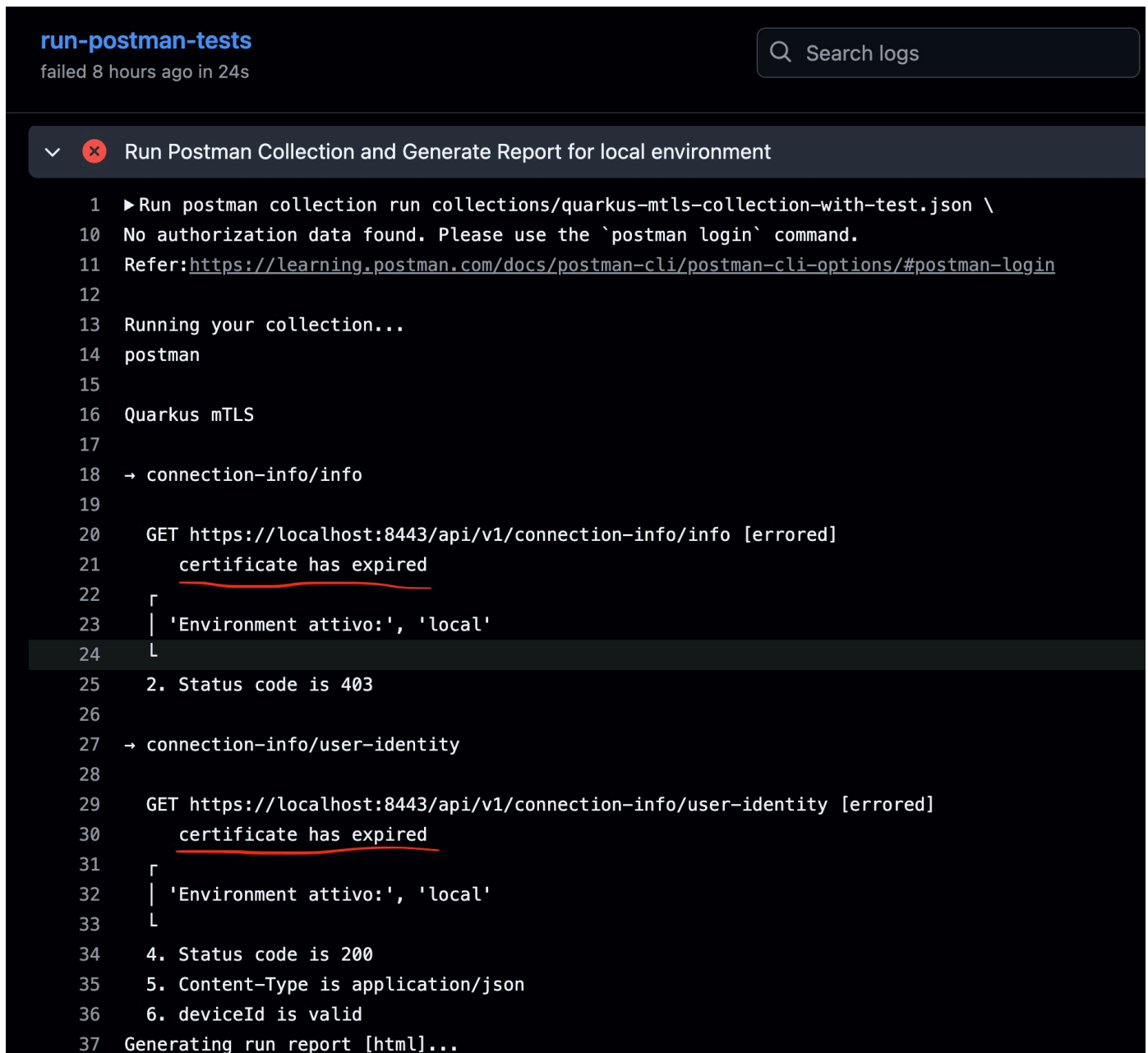
```
1  ...
2  - name: Install Postman CLI
3    run: |
4      curl -o- "https://dl-cli.pstmn.io/install/linux64.sh" | sh
5
6  - name: Run Postman Collection and Generate Report for local environment
7    run: |
8      postman collection run collections/quarkus-mtls-collection-with-test.json \
9      --environment config/environments/local.json \
10     --ssl-client-cert-list config/certificates/client-certificates.json \
11     --ssl-extra-ca-certs certificates/ca/ca_cert.pem \
12     --verbose \
13     --reporters html,cli \
14     --reporter-html-export postman_report_local.html
15  ...
```

GitHub Actions 1 - Esempio di step per installare ed eseguire Postman CLI

Se uno o più test falliscono, lo step viene marcato come failed e la pipeline si interrompe, garantendo che solo il codice che supera tutti i test venga distribuito.

Quando l'errore è il nostro migliore amico

Ecco il bello dell'automazione: quando la GitHub Action fallisce, come mostrato nell'immagine a seguire, non è solo una "X" rossa che ci fa storcere il naso. In realtà, quell'errore è stato fondamentale: ci ha subito segnalato che il certificato del servizio (REST API) era scaduto e che le richieste mTLS non potevano andare a buon fine. Meglio scoprirlo ora, in fase di test, che in produzione davanti agli utenti!



```
run-postman-tests
failed 8 hours ago in 24s

Run Postman Collection and Generate Report for local environment

1 ▶ Run postman collection run collections/quarkus-mtls-collection-with-test.json \
10 No authorization data found. Please use the `postman login` command.
11 Refer: https://learning.postman.com/docs/postman-cli/postman-cli-options/#postman-login
12
13 Running your collection...
14 postman
15
16 Quarkus mTLS
17
18 → connection-info/info
19
20 GET https://localhost:8443/api/v1/connection-info/info [errored]
21   certificate has expired
22   {
23     "Environment attivo": "local"
24   }
25   2. Status code is 403
26
27 → connection-info/user-identity
28
29 GET https://localhost:8443/api/v1/connection-info/user-identity [errored]
30   certificate has expired
31   {
32     "Environment attivo": "local"
33   }
34   4. Status code is 200
35   5. Content-Type is application/json
36   6. deviceId is valid
37   Generating run report [html]...
```

Figura 5 - GitHub Action fallita a causa di certificato scaduto

Insomma, il fallimento della pipeline è stato un vero alleato: ci ha permesso di intervenire tempestivamente, aggiornare i certificati e garantire la sicurezza e l'affidabilità del servizio. Ricorda: ogni errore ben evidenziato è un passo avanti verso un sistema più robusto!

Probabilmente senza questi test automatici non mi sarei mai accorto del certificato scaduto e avrei pubblicato la guida con un bel bug, lasciando il lettore a chiedersi perché non funzionasse nulla! Ecco perché i test sono i veri supereroi delle guide tecniche: ti salvano la reputazione prima ancora che tu premi "pubblica"! 🦸

Vantaggi dell'integrazione

- **Automazione completa:** i test mTLS vengono eseguiti automaticamente ad ogni commit, merge o rilascio.
- **Feedback immediato:** eventuali errori di configurazione, autenticazione o regressioni vengono rilevati subito.
- **Reportistica:** è possibile esportare report HTML e allegarli agli artefatti della pipeline per una consultazione rapida.
- **Sicurezza:** si garantisce che le API protette da mTLS siano sempre testate e funzionanti prima della messa in produzione.

L'immagine seguente mostra l'esecuzione di Postman CLI in una GitHub Actions, con il riepilogo dei risultati e i report HTML generati come artefatti. Questa integrazione semplifica e automatizza la validazione delle API mTLS nei workflow DevOps.

Triggered via push 6 minutes ago
amusarra pushed -> b18d361 main
Status: Success
Total duration: 30s
Artifacts: 2

run-integration-test-with-postmancli.yml
on: push

run-postman-tests 27s

run-postman-tests summary

Postman Test Report

I report HTML generati dai test di Postman sono disponibili come artifact di questa run:

- Per l'ambiente local: postman-report-local
- Per l'ambiente test: postman-report-test

Job summary generated at run-time

Artifacts

Produced during runtime

Name	Size	Digest	
postman-report-local	26.2 KB	sha256:a009dd41da6f377898d173c63acd1ea054d5ea9e9b35c971122ed2dea...	📄 📥 🗑️
postman-report-test	28.7 KB	sha256:3757710582fefbe3ca5c27e3bfa3de473ac62b2f215bfd05963494d30...	📄 📥 🗑️

Figura 6 - Esecuzione di Postman CLI in GitHub Actions - Summary

Di seguito un esempio di report HTML generato da Postman CLI usando l'opzione `--reporters html`.

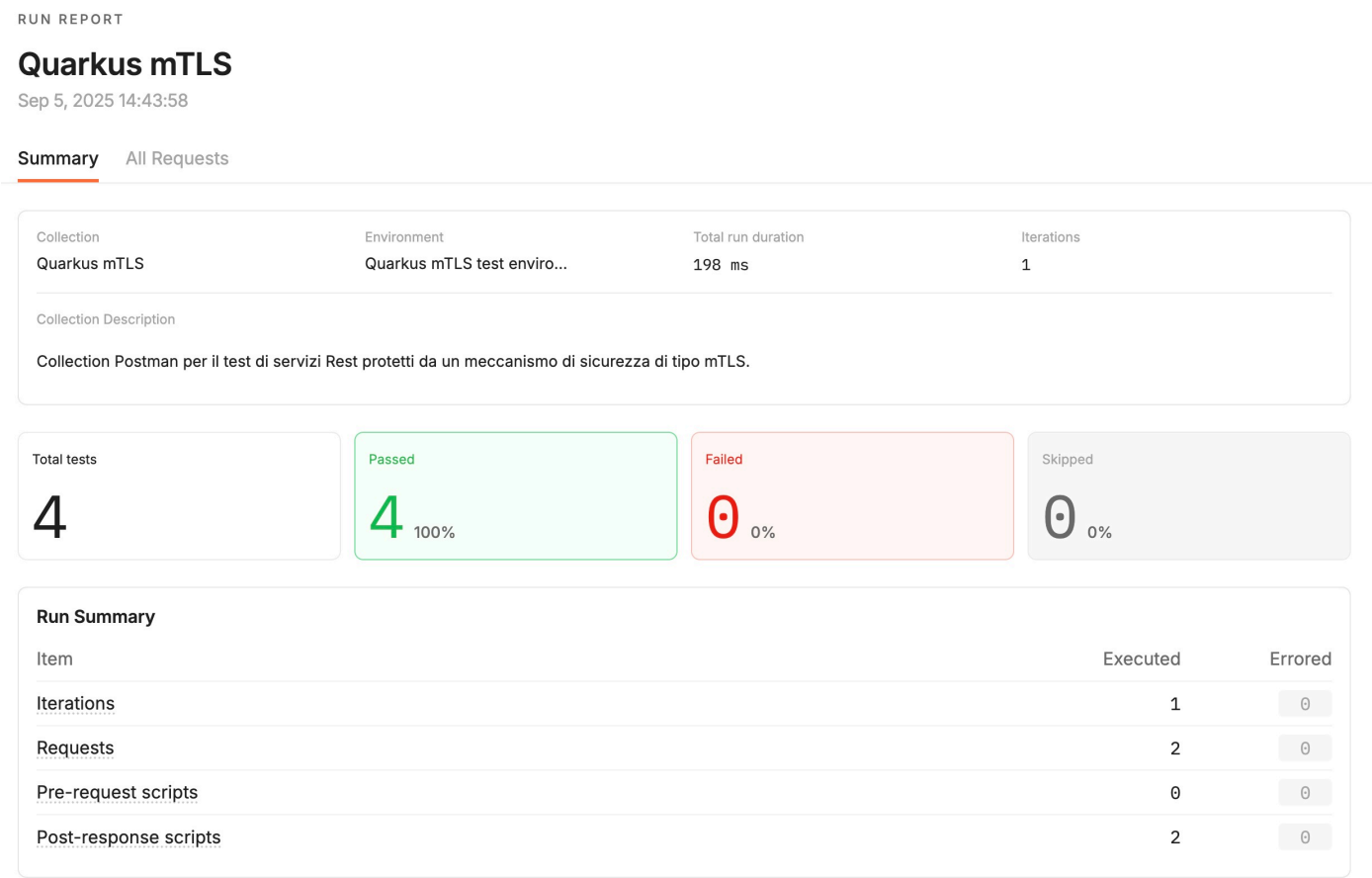


Figura 7 - Report HTML generato da Postman CLI - Summary

Summary **All Requests**

Iteration 1

GET connection-info/info 200 OK • 143 ms • 4.79 KB 3 -
<https://blog.quarkus.dontesta.it/api/v1/connection-info/info>

> Request headers (7)

> Response headers (2)

> Response body

> Tests (3)

GET connection-info/user-identity 401 Unauthorized • 23 ms • 85 B 1 -
<https://blog.quarkus.dontesta.it/api/v1/connection-info/user-identity>

> Request headers (7)

> Response headers (2)

Response body

```
{
  "statusCode": 401,
  "message": "Decoded roles do not match the expected pattern: Role="
}
```

> Tests (1)

Status	Test
PASS	Status code is 401

Figura 8 - Report HTML generato da Postman CLI - Dettaglio richieste

Questo report mostra in modo grafico e dettagliato lo stato delle richieste, i test eseguiti, le asserzioni superate e fallite, i tempi di risposta e tutte le informazioni utili per la validazione automatica delle API mTLS.

Conclusioni

In questa guida abbiamo visto come configurare e utilizzare Postman CLI per testare endpoint protetti da mTLS, sfruttando ambienti, variabili e certificati client multipli.

Abbiamo imparato a:

- Organizzare la struttura di progetto per Postman CLI.
- Definire ambienti e variabili per gestire facilmente configurazioni diverse.
- Configurare e selezionare certificati client in modo centralizzato tramite file JSON.
- Creare collection Postman con test automatizzati che variano in base all'ambiente.
- Eseguire le collection da CLI, verificare il risultato tramite exit code e generare report HTML.
- Integrare i test mTLS in pipeline CI/CD per garantire la qualità e la sicurezza delle API.

Questi strumenti e procedure ci permettono di automatizzare e rendere ripetibile la validazione di API protette da mTLS, migliorando la produttività e la robustezza dei processi di sviluppo e delivery.

Premio per aver seguito la guida fino a qui!

Trovi l'intero progetto pronto all'uso su GitHub:

 <https://github.com/amusarra/postman-mtls>

Nel repository troverai tutte le configurazioni, gli script, le collection Postman, i certificati di esempio, la GitHub Actions di esempio e molto altro. Puoi clonare il progetto, adattarlo alle tue esigenze e iniziare subito a testare API protette da mTLS con Postman CLI.

Risorse utili

Ecco una raccolta di risorse e riferimenti utili per approfondire i temi trattati nella guida:

Postman

- [Documentazione ufficiale Postman CLI](#)
- [Installazione Postman CLI](#)
- [Gestione ambienti in Postman](#)
- [Gestione certificati in Postman](#)
- [Scrivere test in Postman](#)
- [Report con Postman CLI](#)

GitHub

- [Quickstart quarkus-mtls-auth](#)
- [Repository di esempio: postman-mtls](#)
- [GitHub Actions Documentation](#)

Articoli e Guide

- [Implementazione di TLS Mutual Authentication \(mTLS\) con Quarkus](#) - pubblicato sul blog di [Antonio Musarra](#)
- [Autenticazione mTLS con Quarkus](#) - pubblicato sul podcast di Antonio Musarra
- [Liferay 7.2: Esempio di Two-Way SSL/TLS Mutual Authentication Client](#) - pubblicato sul blog di [Antonio Musarra](#)
- [RabbitMQ 4.1 mTLS e AMQP 1.0: Guida essenziale per sviluppatori](#) - pubblicato sul blog di [Antonio Musarra](#)

Wikipedia

- [Mutual TLS \(Wikipedia\)](#)
- [Certificate Authority \(Wikipedia\)](#)
- [Privacy Enhanced Mail \(PEM\)](#)

Altre Risorse

- [Debug TLS/SSL con OpenSSL](#)

Queste risorse ti aiuteranno ad approfondire la configurazione, l'automazione e il troubleshooting di mTLS e Postman CLI.