Studio e Implementazione di un Gateway API con Spring Cloud in Architetture a Microservizi

Andrea Donadello

12 giugno 2025

Sommario

Il presente lavoro di progetto universitario si concentra sull'analisi approfondita e sull'implementazione pratica di un Gateway API utilizzando Spring Cloud Gateway.
In un'era di trasformazione digitale rapida, la gestione strategica delle API è emersa come un fattore critico per le aziende che adottano architetture a microservizi, dove la complessità della comunicazione inter-servizio, della sicurezza e del routing può diventare proibitiva senza un punto di ingresso centralizzato.
Questo studio si propone di esplorare Spring Cloud Gateway come soluzione robusta e reattiva per affrontare tali sfide.

La metodologia adottata per questo progetto combina elementi di indagine ("Survey"), elaborazione ("Elaboration") e applicazione pratica ("Application"), come delineato nelle linee guida del Project Work. È stata condotta un'indagine sui concetti fondamentali dei Gateway API e sulle loro funzionalità comuni, seguita da un'elaborazione dettagliata dell'architettura e delle caratteristiche specifiche di Spring Cloud Gateway.
Il cuore del progetto consiste nell'implementazione di un prototipo di Gateway API, disponibile nel repository Git fornito (https://github.com/ddonazz/api-gateway), che dimostra funzionalità chiave quali il routing dinamico, la gestione centralizzata della sicurezza (autenticazione e autorizzazione), la limitazione del tasso di richieste (rate limiting) e la gestione delle eccezioni.

I risultati ottenuti evidenziano i significativi vantaggi di Spring Cloud Gateway, tra cui una maggiore sicurezza, un'accelerazione dell'innovazione e tempi di commercializzazione ridotti, una migliore esperienza utente e una conformità normativa più efficiente.
Tuttavia, l'analisi critica ha anche rivelato sfide intrinseche all'adozione di un Gateway API, come l'aumento della complessità architetturale e il potenziale di introduzione di latenza aggiuntiva.
Il lavoro si conclude con una discussione sui contributi del progetto e sulle prospettive future per superare le limitazioni identificate, proponendo sviluppi che potrebbero ulteriormente migliorare la robustezza e l'applicabilità di tali soluzioni.

Indice

Capitolo 1

Introduzione

1.1 Contesto e Motivazioni

Il panorama dello sviluppo software contemporaneo è profondamente influenzato dall'adozione sempre più diffusa delle architetture a microservizi. Questo modello, che scompone le applicazioni monolitiche in un insieme di servizi piccoli, autonomi e accoppiati in modo lasco, offre numerosi vantaggi in termini di scalabilità, resilienza e agilità nello sviluppo. Tuttavia, l'adozione dei microservizi introduce anche nuove sfide significative, in particolare per quanto riguarda la gestione della comunicazione tra i servizi, la sicurezza, il monitoraggio e il routing delle richieste. Man mano che il numero di microservizi aumenta, la complessità di gestire le interazioni dirette tra client e servizi backend può diventare insostenibile, portando a problemi come la logica client frammentata e la difficoltà nella gestione delle preoccupazioni trasversali.

In questo contesto, il Gateway API emerge come un componente architetturale cruciale, fungendo da «edge service» e da «reverse proxy» che centralizza l'esposizione delle API e gestisce le preoccupazioni trasversali. Questo approccio risolve il problema «N+1», dove i client dovrebbero altrimenti effettuare chiamate separate a N servizi backend, aggregando le richieste e semplificando l'interazione. La gestione strategica delle API è, infatti, un fattore critico per le aziende che perseguono la trasformazione digitale, distinguendo i leader di mercato dai ritardatari.

La scelta di Spring Cloud Gateway (SCG) come tecnologia centrale per questo studio è giustificata dalla sua rilevanza all'interno dell'ecosistema Spring, ampiamente adottato nello sviluppo di applicazioni aziendali. SCG è una soluzione leggera e reattiva, costruita su Spring WebFlux e Project Reactor, che la rende particolarmente adatta per carichi di lavoro ad alta produttività e bassa latenza. Il presente Project Work si focalizza sullo studio di questa «nuova tecnologia», come richiesto dalle linee guida, fornendo dettagli implementativi e un'analisi critica della sua applicazione pratica. La necessità di un punto di ingresso centralizzato per le API è una conseguenza diretta dell'adozione dei microservizi; senza un Gateway API, la gestione delle interazioni tra client e servizi backend, la sicurezza e il monitoraggio diventerebbero estremamente complessi e frammentati, ostacolando l'efficienza e la scalabilità del sistema distribuito.

1.2 Obiettivi del Progetto

L'obiettivo primario di questo progetto è acquisire una comprensione approfondita, implementare e valutare criticamente Spring Cloud Gateway come soluzione per la gestione dei Gateway API in architetture a microservizi. Per raggiungere questo scopo, sono stati definiti i seguenti sotto-obiettivi specifici:

- **Conduzione di un'indagine (Survey):** Effettuare una ricerca sui concetti fondamentali dei Gateway API e sulle loro funzionalità comuni. Questo aspetto si allinea con la tipologia «Survey» del Project Work, che prevede la ricerca dei principali risultati e caratteristiche di una piattaforma tecnologica.
- **Elaborazione sull'architettura e le funzionalità di Spring Cloud Gateway:** Approfondire l'architettura di SCG, le sue caratteristiche distintive come le route, i predicati e i filtri, e i vantaggi che offre in termini di sicurezza, monitoraggio e resilienza. Questo si inquadra nella tipologia «Elaboration», concentrandosi su un argomento specifico e il suo stato dell'arte.

- **Sviluppo di un'applicazione pratica (Application):** Realizzare un'implementazione concreta di un Gateway API utilizzando Spring Cloud Gateway. Questo prototipo, disponibile nel repository Git fornito dall'utente, dimostrerà l'applicazione pratica delle funzionalità studiate, come richiesto dalla tipologia «Application» delle linee guida, che prevede l'installazione e l'uso di una tecnologia innovativa attraverso esempi semplici.
- **Analisi critica e identificazione delle limitazioni:** Condurre un'analisi obiettiva di Spring Cloud Gateway, identificando i suoi limiti, le potenziali sfide e gli ostacoli che possono emergere durante la sua adozione. Questa componente è fondamentale per soddisfare il requisito di «Critical Thinking» delle linee guida, che invita a non «innamorarsi» delle nuove tecnologie ma a cogliere anche i loro limiti.
- **Proposta di sviluppi futuri:** Sulla base dell'analisi critica, suggerire possibili evoluzioni e miglioramenti per il progetto implementato o per la tecnologia stessa, in linea con il requisito di indicare «quali saranno gli sviluppi futuri che dovranno essere affrontati per ovviare i limiti evidenziati».

L'esplicita correlazione degli obiettivi del progetto con le tipologie di Project Work definite nelle linee guida (Survey, Elaboration, Application) dimostra una comprensione approfondita dei requisiti dell'incarico. Ciò non si limita alla mera realizzazione tecnica, ma si estende alla capacità di inquadrare il lavoro all'interno di un rigoroso contesto accademico, evidenziando una competenza che va oltre la semplice implementazione.

1.3 Struttura del Rapporto

Il presente rapporto è strutturato per fornire una trattazione completa e progressiva dell'argomento, partendo dai concetti fondamentali fino all'implementazione pratica e all'analisi critica.

- **Capitolo 1: Introduzione** — Definisce il contesto dei Gateway API nelle architetture a microservizi, le motivazioni alla base della scelta di Spring Cloud Gateway e gli obiettivi specifici del progetto.
- **Capitolo 2: Concetti Fondamentali dei Gateway API** — Illustra la definizione e il ruolo di un Gateway API, esplorando le sue funzionalità comuni e i vantaggi architetturali che comporta.
- **Capitolo 3: Spring Cloud Gateway: Caratteristiche e Architettura** — Approfondisce i principi fondamentali di Spring Cloud Gateway, descrivendo i suoi componenti chiave (route, predicati, filtri) e le funzionalità avanzate relative a sicurezza, monitoraggio e resilienza.
- **Capitolo 4: Vantaggi e Casi d'Uso di Spring Cloud Gateway** — Sintetizza i benefici architetturali e operativi derivanti dall'adozione di SCG e presenta scenari applicativi tipici in cui questa tecnologia eccelle.
- **Capitolo 5: Implementazione Pratica: Il Progetto API Gateway** — Descrive in dettaglio l'architettura e il design del Gateway implementato, fornendo esempi concreti di configurazione, codice e dimostrazioni pratiche delle funzionalità.
- **Capitolo 6: Analisi Critica e Limitazioni** — Affronta gli ostacoli e le difficoltà incontrate durante lo studio e l'implementazione, analizzando i limiti di applicabilità di Spring Cloud Gateway e proponendo aree per futuri sviluppi.
- **Capitolo 7: Conclusioni e Sviluppi Futuri** — Riepiloga i risultati principali del progetto, evidenziando i contributi apportati e delineando le prospettive future per la ricerca e l'applicazione di Spring Cloud Gateway.
- **Biblio/Sitografia** — Elenca tutte le fonti bibliografiche e sitografiche utilizzate per la redazione del rapporto.

Questa struttura è stata concepita per guidare il lettore attraverso un percorso logico, dalla teoria alla pratica, fino a una valutazione ponderata, rispondendo in modo esaustivo a tutti i requisiti delle linee guida del Project Work.

Capitolo 2

Concetti Fondamentali dei Gateway API

2.1 Cos'è un API Gateway e il suo Ruolo nelle Architetture a Microservizi

Un API Gateway è un componente architetturale che funge da singolo punto di ingresso per tutte le richieste dei client verso un sistema di microservizi backend. In essenza, agisce come un «reverse proxy» o un «edge service» che si interpone tra i client e i servizi backend. Questo modello risolve una problematica comune nelle architetture a microservizi, nota come «N+1 problem», dove un client dovrebbe altrimenti conoscere e interagire direttamente con N servizi backend distinti per soddisfare una singola richiesta complessa. L'API Gateway aggrega queste interazioni, presentando un'unica interfaccia semplificata ai client.

Il ruolo di un API Gateway è multifunzionale e critico per la gestione efficace di sistemi distribuiti. Esso non si limita al semplice routing delle richieste al servizio appropriato, ma centralizza anche una serie di preoccupazioni trasversali che altrimenti dovrebbero essere implementate in ogni singolo microservizio. Questa centralizzazione è fondamentale per mantenere la coerenza, ridurre la ridondanza del codice e accelerare lo sviluppo.

La funzione di un API Gateway come punto di ingresso unificato semplifica intrinsecamente l'integrazione dei client e disaccoppia i client dall'evoluzione dei servizi di backend. Se i client fossero costretti a interagire direttamente con ogni microservizio, dovrebbero essere a conoscenza degli indirizzi specifici, dei protocolli e delle modifiche di ciascuno. Un Gateway API astrae questa complessità, offrendo un'interfaccia coerente. Questa separazione significa che le modifiche ai servizi di backend (ad esempio, la fusione o la divisione di servizi, o la migrazione di tecnologie) possono essere implementate senza richiedere aggiornamenti sul lato client. Ciò riduce significativamente il sovraccarico di manutenzione e accelera i cicli di sviluppo, poiché i team possono concentrarsi sulla logica di business senza preoccuparsi delle implicazioni per i client.

2.2 Funzionalità Comuni dei Gateway API

Un Gateway API robusto e completo offre una vasta gamma di funzionalità essenziali per la gestione efficiente e sicura delle API in un ambiente a microservizi. Queste funzionalità sono progettate per affrontare le sfide intrinseche dei sistemi distribuiti e per centralizzare le preoccupazioni trasversali.

Le funzionalità comuni includono:

- **Routing:** La capacità fondamentale di dirigere le richieste in ingresso al servizio backend corretto in base a regole predefinite, come il percorso dell'URL, gli header o i parametri della richiesta.
- **Autenticazione e Autorizzazione:** Applicazione centralizzata delle politiche di sicurezza. Questo include la gestione dell'autenticazione (ad esempio, Basic Auth, OAuth2, OpenID Connect) e dell'autorizzazione (controlli di accesso basati sui ruoli). La funzionalità di Single Sign-On (SSO) è spesso integrata per semplificare l'accesso tra diverse applicazioni.

- **Limitazione del Tasso (Rate Limiting):** Controllo del numero di richieste che un client può effettuare in un determinato periodo di tempo per prevenire abusi, attacchi Denial of Service (DoS) e garantire un uso equo delle risorse.
- **Bilanciamento del Carico (Load Balancing) e Tolleranza ai Guasti (Fault Tolerance):** Distribuzione uniforme del traffico in ingresso tra più istanze di un servizio backend per migliorare le prestazioni e la disponibilità, garantendo che il sistema rimanga reattivo anche in caso di guasti o sovraccarichi di singoli servizi.
- **Monitoraggio e Logging:** Raccolta centralizzata di metriche e log relativi al traffico API, alle prestazioni e agli errori, fornendo visibilità sull'operatività del sistema e facilitando il debugging.
- **Trasformazione di Protocollo e Formato Dati:** Capacità di convertire richieste e risposte tra diversi protocolli (ad esempio, HTTP a gRPC) o formati di dati (ad esempio, JSON a XML), facilitando l'integrazione tra sistemi eterogenei.
- **Versionamento delle API:** Gestione di più versioni di un'API, consentendo agli sviluppatori di introdurre nuove funzionalità o modifiche senza interrompere i client esistenti.
- **Caching:** Memorizzazione di risposte a richieste frequenti per ridurre il carico sui servizi backend e migliorare i tempi di risposta.
- **Circuit Breaker (Interruttore di Circuito):** Un pattern di resilienza che previene i guasti a cascata in un sistema distribuito, isolando i servizi che non rispondono e fornendo risposte di fallback.
- **Header di Sicurezza:** Applicazione automatica di header di sicurezza (ad esempio, Cache-Control, X-Content-Type-Options, Strict-Transport-Security) per rafforzare la postura di sicurezza complessiva.

L'aggregazione di queste preoccupazioni trasversali all'interno del Gateway API le trasforma da responsabilità individuali di ciascun servizio in politiche centralizzate e gestibili. Questo approccio riduce il carico di sviluppo sui team, che possono concentrarsi sulla creazione di valore di business piuttosto che sulla ricostruzione di componenti infrastrutturali. Il risultato è un'accelerazione dell'innovazione e del tempo di commercializzazione, oltre a una conformità più efficiente e un'esperienza cliente superiore, derivanti dall'applicazione coerente di best practice di sicurezza e gestione del traffico.

Tabella 2.1: Funzionalità Comuni di un API Gateway (con librerie standard)

Funzionalità	**Descrizione**	**Beneficio Principale**
Routing	Inoltra le richieste in ingresso al servizio backend appropriato.	Semplifica l'integrazione client e la gestione del traffico.
Autenticazione & Autorizzazione	Applica politiche di sicurezza centralizzate per l'accesso alle API.	Migliora la sicurezza e la conformità, riduce il rischio.
Limitazione del Tasso (Rate Limiting)	Controlla il numero di richieste per prevenire abusi e sovraccarichi.	Garantisce stabilità del sistema e equità d'uso.
Bilanciamento del Carico	Distribuisce il traffico tra le istanze del servizio per ottimizzare le prestazioni.	Aumenta la disponibilità e la scalabilità.
Tolleranza ai Guasti	Gestisce i fallimenti dei servizi backend per mantenere la disponibilità.	Aumenta la resilienza del sistema.
Monitoraggio & Logging	Raccoglie dati sul traffico API, le prestazioni e gli errori.	Fornisce visibilità operativa e facilita il debugging.
Trasformazione Protocollo/Dati	Converte richieste/risposte tra diversi formati o protocolli.	Facilita l'integrazione con sistemi eterogenei.
Versionamento API	Gestisce più versioni di un'API contemporaneamente.	Consente evoluzione delle API senza interrompere i client esistenti.
Caching	Memorizza le risposte per richieste frequenti.	Riduce la latenza e il carico sui servizi backend.
Circuit Breaker (Interruttore di Circuito)	Isola i servizi problematici per prevenire guasti a cascata.	Migliora la resilienza del sistema in ambienti distribuiti.
Header di Sicurezza	Applica automaticamente header HTTP per rafforzare la sicurezza.	Migliora la postura di sicurezza complessiva.

Capitolo 3

Spring Cloud Gateway: Caratteristiche e Architettura

3.1 Panoramica e Principi Fondamentali

Spring Cloud Gateway (SCG) si presenta come un Gateway API leggero e reattivo, costruito sopra il framework Spring. È stato progettato per fornire un metodo semplice ma efficace per instradare le richieste alle API e per gestire le preoccupazioni trasversali quali sicurezza, monitoraggio/metriche e resilienza. La sua architettura è saldamente radicata su tecnologie moderne dell'ecosistema Spring: Spring Framework 5, Project Reactor e Spring Boot 2.0. Questa fondazione è cruciale, poiché conferisce a SCG la sua natura reattiva e non bloccante, un aspetto distintivo che lo rende particolarmente adatto per gestire un elevato numero di richieste concorrenti con bassa latenza.

La scelta di basare SCG su Project Reactor e Spring WebFlux non è un mero dettaglio tecnico, ma una decisione architetturale fondamentale che lo distingue dai gateway tradizionali basati su modelli di programmazione bloccanti. I modelli di programmazione reattiva sono intrinsecamente progettati per gestire un gran numero di connessioni concorrenti con un numero ridotto di thread, ottimizzando l'utilizzo delle risorse e migliorando le prestazioni sotto carichi pesanti. Questo è particolarmente vantaggioso in ambienti a microservizi, dove la capacità di gestire efficacemente un flusso elevato di traffico è essenziale per garantire «carichi di lavoro a bassa latenza e alta produttività».

Il ciclo di vita di una richiesta all'interno di Spring Cloud Gateway segue un flusso ben definito:

1. **Client Request:** Una richiesta viene inviata dal client al Gateway API.
2. **Gateway Handler Mapping:** Il Gateway determina se la richiesta corrisponde a una route definita.
3. **Gateway Web Handler:** Se una corrispondenza viene trovata, la richiesta viene inoltrata a questo handler.
4. **Filter Chain:** L'handler esegue la richiesta attraverso una catena di filtri specifici per quella richiesta. Questi filtri possono modificare la richiesta prima che venga inviata al servizio di destinazione (pre-filtri) o la risposta che ritorna al client (post-filtri).
5. **Downstream Service:** La richiesta modificata viene inviata al servizio backend appropriato.
6. **Response:** La risposta dal servizio backend ritorna attraverso la catena di filtri (post-filtri) prima di essere inviata al client.

Questo modello di elaborazione basato su filtri e route consente una grande flessibilità e modularità nella gestione del traffico API.

3.2 Componenti Chiave: Route, Predicati e Filtri

Spring Cloud Gateway si basa su tre componenti fondamentali per la sua operatività: Route, Predicati e Filtri. La modularità di questi elementi consente una configurazione dichiarativa e un'estensibilità notevole, permettendo di gestire logiche di routing complesse e preoccupazioni trasversali con grande flessibilità.

Route: Una Route è il «blocco di costruzione fondamentale» del gateway. È definita da un ID univoco, un URI di destinazione (il servizio backend a cui la richiesta deve essere inoltrata), una collezione di predicati e una collezione di filtri. Una route viene considerata «matched» (corrispondente) se l'operazione logica AND su tutti i suoi predicati restituisce true.

Esempio di configurazione (YAML):

Listing 3.1: Esempio di configurazione di una route in YAML

```yaml
spring:
  cloud:
    gateway:
      routes:
        - id: book-service-route
          uri: http://localhost:8081 # URI del servizio backend
          predicates:
            - Path=/api/books/** # Predicato di percorso
          filters:
            - StripPrefix=2 # Rimuove /api/books dal percorso prima di inoltrare
```

Predicate: I Predicati sono funzioni booleane (Java 8 Function Predicate) che permettono di far corrispondere le route a qualsiasi attributo della richiesta HTTP. Questo include header, parametri, metodi HTTP, host, e persino il tempo. È possibile combinare più predicati con operatori logici AND per creare regole di routing altamente specifiche.

Esempi comuni di predicati includono: Path, Host, Method, Query, Header, Cookie, RemoteAddr, After, Before, Between. I predicati After, Before e Between sono particolarmente utili per scenari come le finestre di manutenzione, permettendo di instradare il traffico in base a intervalli temporali specifici.

Filter: I Filtri forniscono un meccanismo per modificare le richieste HTTP in ingresso e le risposte HTTP in uscita. Esistono due tipi principali di filtri:

- **GatewayFilter:** Applicati a una specifica route.
- **GlobalFilter:** Applicati a tutte le route. Spring Cloud Gateway include molti filtri predefiniti (ad esempio, AddRequestHeader, AddResponseHeader, RateLimiter, RewritePath). È anche possibile implementare filtri personalizzati per esigenze di business uniche. Ad esempio, un filtro personalizzato potrebbe essere utilizzato per aggiungere un timestamp alla richiesta in ingresso prima che venga inoltrata (AddRequestTimeHeaderPreFilter) o per simulare una validazione e bloccare la richiesta se un header di autorizzazione non è presente.

L'enfasi ricorrente su Route, Predicati e Filtri sottolinea la loro centralità nel potere di Spring Cloud Gateway. La capacità di definire le route in modo dichiarativo, di combinare i predicati per una corrispondenza granulare e di applicare i filtri per la modifica delle richieste/risposte significa che le politiche di gestione delle API, anche le più complesse, possono essere costruite a partire da blocchi più piccoli e riutilizzabili. Questa modularità rende il gateway altamente configurabile ed estensibile, consentendo agli sviluppatori di aggiungere facilmente nuove funzionalità o di implementare filtri personalizzati per requisiti specifici. Questo principio di design è fondamentale per la sua adattabilità in diversi ambienti a microservizi.

3.3 Funzionalità Avanzate: Sicurezza, Monitoraggio, Resilienza

Oltre alle capacità fondamentali di routing e trasformazione, Spring Cloud Gateway eccelle nella gestione di funzionalità avanzate che sono cruciali per la robustezza e la sicurezza delle architetture a microservizi. Queste includono la sicurezza centralizzata, il monitoraggio e le metriche, e la resilienza del sistema.

Sicurezza: SCG centralizza le preoccupazioni di sicurezza, riducendo la necessità di implementare logiche di sicurezza in ogni singolo microservizio. Questo include:

- *Autenticazione e Autorizzazione:* Supporta vari meccanismi di autenticazione come Basic Auth e OAuth2/OpenID Connect. Le funzionalità di Single Sign-On (SSO) sono particolarmente rilevanti, in quanto possono essere configurate una sola volta a livello di gateway, eliminando la necessità di implementazioni diverse per ogni sistema backend. Ciò consente un controllo degli accessi basato sui ruoli e semplifica notevolmente la conformità normativa, riducendo i costi di conformità fino al 40% per le organizzazioni con controlli di sicurezza centralizzati.

Tabella 3.1: Esempi di Predicati di Route in Spring Cloud Gateway

Predicato	**Descrizione**	**Parametri Esempio**	**Esempio di Configurazione**
Path	Corrisponde al percorso dell'URL della richiesta.	`/api/users/**`	`predicates: - Path=/api/users/**`
Host	Corrisponde all'header Host della richiesta.	`*.example.com`	`predicates: - Host=*.example.com`
Method	Corrisponde al metodo HTTP della richiesta.	GET, POST	`predicates: - Method=GET,POST`
Query	Corrisponde a un parametro di query specifico.	param=value	`predicates: - Query=param,value`
Header	Corrisponde a un header HTTP specifico e al suo valore.	`X-Request-Id, d+`	`predicates: - Header=X-Request-Id, d+`
After	Corrisponde alle richieste che avvengono dopo una data e ora specificate.	2017-01-20T...	`predicates: - After=2017-01-20T17:42:47.789Z[UTC]`
Before	Corrisponde alle richieste che avvengono prima di una data e ora specificate.	2017-01-21T...	`predicates: - Before=2017-01-21T17:42:47.789Z[UTC]`
Between	Corrisponde alle richieste che avvengono tra due date e ore specificate.	datetime1, datetime2	`predicates: - Between=...,` ...
RemoteAddr	Corrisponde all'indirizzo IP remoto della richiesta.	192.168.1.1/24	`predicates: - RemoteAddr=192.168.1.1/24`

Tabella 3.2: Esempi di Filtri in Spring Cloud Gateway

Filtro	**Tipo**	**Descrizione**	**Caso d'Uso Esempio**
AddRequestHeader	GatewayFilter	Aggiunge un header alla richiesta in uscita.	`filters: - AddRequestHeader=X-Request-Foo, Bar`
AddResponseHeader	GatewayFilter	Aggiunge un header alla risposta in uscita.	`filters: - AddResponseHeader=X-Response-Bye, Bye`
StripPrefix	GatewayFilter	Rimuove un prefisso dal percorso della richiesta.	`filters: - StripPrefix=1` per /api/v1/users → /users
RewritePath	GatewayFilter	Riscrive il percorso della richiesta usando un'espressione regolare.	`filters: - RewritePath=/foo/(?<segment>.*), /${segment}`
RequestRateLimiter	GatewayFilter	Limita il tasso di richieste per utente/IP.	`filters: - RequestRateLimiter=` (con config.)
CircuitBreaker	GatewayFilter	Implementa un pattern Circuit Breaker per la resilienza.	`filters: - CircuitBreaker=myServiceCircuit`
SecureHeaders	GlobalFilter	Applica globalmente header di sicurezza HTTP.	`tanzu: api-gateway: secure-headers: deactivated: false`
CustomGlobal-ExceptionHandler	GlobalFilter	Gestisce eccezioni a livello globale per risposte uniformi.	Intercetta HttpClientErrorException per risposte di errore consistenti
Authorization Filter	GlobalFilter	Filtro personalizzato per autenticazione/autorizzazione.	Verifica token OAuth2 o credenziali Basic Auth
AddRequestTime-HeaderPreFilter	GlobalFilter	Filtro personalizzato che aggiunge un timestamp alla richiesta.	`filters: - AddRequestTimeHeaderPreFilter`

- *Header di Sicurezza:* L'applicazione automatica di header di sicurezza (es. Cache-Control, X-Content-Type-Options, Strict-Transport-Security) a livello globale per tutte le route è una best practice che rafforza la postura di sicurezza complessiva.
- *Gestione Globale delle Eccezioni:* Implementare una gestione globale delle eccezioni in SCG è essenziale per catturare e gestire in modo uniforme gli errori che si verificano all'interno del gateway. Questo previene risposte di errore frammentate e incoerenti, migliorando l'esperienza utente e facilitando il debugging.

Monitoraggio/Metriche: SCG può essere integrato con strumenti di monitoraggio e logging esterni come Splunk o ELK Stack. Questa integrazione fornisce una visibilità centralizzata sul traffico API, sulle prestazioni e sugli errori. La capacità di registrare ogni eccezione per analisi future è una best practice fondamentale.

Resilienza: La resilienza è vitale in ambienti distribuiti per prevenire i guasti a cascata. SCG offre funzionalità come:

- *Integrazione con Circuit Breaker:* L'integrazione con circuit breaker come Hystrix (o Resilience4j in versioni più recenti) consente di isolare i servizi che non rispondono, evitando che un singolo guasto si propaghi attraverso l'intero sistema.
- *Gestione Intelligente del Traffico e Tolleranza ai Guasti:* SCG offre funzionalità avanzate di gestione del traffico e alta disponibilità che possono mantenere prestazioni consistenti anche durante i picchi di traffico. Questo include il bilanciamento del carico e la gestione dei fallimenti.

La centralizzazione di funzionalità avanzate come la sicurezza e la resilienza a livello di gateway le trasforma da correzioni reattive in salvaguardie architetturali proattive. Questo approccio migliora significativamente la robustezza complessiva del sistema e la sua postura di conformità. Gestendo questi aspetti in modo centralizzato, il sistema diventa più resistente ai guasti e agli attacchi, e la conformità normativa è più facile da gestire, riducendo potenziali sanzioni. Questo non si limita alla mera funzionalità, ma si estende all'impatto strategico di tali funzionalità sull'integrità operativa del sistema e sui risultati di business.