**Spring Cloud Gateway Rate Limiting and Circuit Breaker**

**Usage of SAR Platform Components within WebApps**

Software Architecture Refactoring (SAR) is focuses on enhancements that will primarily be made to mitigate the production scalability and performance challenges experienced during Filing Season 2022. SAR enhancements leverage Caching, Rate Limiting, Circuit Breaker and Spring Cloud Gateway (SCG) to build upon the capabilities defined as part of the WebApps Platform.

In the OpenShift platform, WebApps will be deployed following a micro-services paradigm, with services deployed in individual Kubernetes pods. Each service will have the ability to scale horizontally. All services will be deployed inside the Istio Service Mesh and will communicate via the envoy proxy, which are deployed within each pod. Istio will also provide the gateway components for communication with external services and clients.

The figure below (Figure 3) shows the high-level SAR Platform components in the WebApp architecture as reference for the topics discussed in the upcoming sections.



Backend

Redhat Openshift-Based Enterprise Container Platform

**Project Specific Services**

**Shared Business Services**

**Operators**

Frontend

(Portal)

**AppDynamics**

**Shared Platform Services**

Audit Microservice SiteMinder Microservice Spring Cloud Gateway

Istio Service Mesh

SADI (eAuth PII) Microservice

UserProfile Microservice

IDV

Response Microservice

OLA

Accounts

External Backend Services

IDV

OLA

Distributed Caching

Red Hat Data Grid

Eligibility Microservice

Ingress Gateway

Figure 3: WA SAR Platform Components

The figure below ([Figure 4](#bookmark1)) shows the high level overview of the interactions between the components in the ECP platform. Each component will provide specific functionality which will be discussed in detail in the upcoming sections.



Cache for frequently used

objects

gateway, entry point

for all BWAS services

3

RHDG Cache

Openshift ECP

Istio DestinationRules based Outlier Detection circuit breaker for external service routes

1

2

4

SCG

7

8

contaninerized,

horizontaly scalable

applications

Centeralized api-gateway

for external services. Provides siteminder token injection, rate limiting and routing functionality

5

6

User 1 ...n

Bucket4j (Service and

User-Based RL)

rate limiting RHDG Cache

External Service

ServiceEntry

BWAS

microservice

Ingress gateway

Figure 4: WA ECP SAR Components Interaction

**Istio Ingress Gateway**

When using Istio Service Mesh, the Ingress controller is replaced by the Ingress Gateway, VirtualService, and DestinationRule resources. The Ingress Gateway provides a single entry point into the mesh. The Gateway also scales based of the load of traffic coming into the Service Mesh.

In Istio, the “controller” is essentially the Istio control plane. The Ingress Gateway resource handles all incoming traffic and functions similarly to an Envoy Proxy running in its own deployment/pod.

A load balancer (LTM) will be configured to forward traffic to the IngressGateway Service. The IngressGateway Service is configured to listen to same ports as the load balancer and forwards traffic to the Ingress Gateway pods.

Once traffic has reached the Ingress Gateway pods, traffic is forwarded based on the DestinationRule configured. Based on the incoming request and the DestinationRule associated with that request, the Ingress Gateway forwards the request to the relevant WebApps service.

**Spring Cloud Gateway**

Spring Cloud Gateway (SCG) has been added to the WebApps architecture to centralize cross cutting functionality that is required when making external calls. At present, the following functionalities will be provided by the SCG filters for outbound requests:

• Add SiteMinder (SM) token headers to requests to external services such as SADI and CBS.

• Provide rate limiting for external services to improve application resiliency. Rate limiting is discussed in detail in the subsequent sections below.

**Rate Limiting**

Rate Limiter is an application resiliency design pattern used to dynamically limit the traffic to a service. This pattern throttles the load into a system and generates a HTTP 429 error once the set limits are exceeded.

The rate limiters are defined with a capacity and duration, where the capacity determines the number of possible interactions for the specified duration. Once the specified duration is completed, the capacity is replenished for the next duration window.

In WebApps, Rate Limiting is implemented via gateway filters in SCG as shown in Figure 5. The filters will use bucket4j APIs to store the rate limit counters to an external RHDG persistence store. Bucket4j is a java rate limiting library that is based on the token bucket algorithm.

It allows applications to define the number of tokens in a bucket. Each api call will try to retrieve a token from the bucket, and return it to the bucket upon completion. Whenever a bucket runs out of tokens, the transactions requesting the token will not be allowed to proceed and a HTTP 429 error is thrown instead.

The following rate limit types can be configured:

**User based**: in this approach, the rate is limited per user to prevent user from exceeding the set limits for the rate limiter. A token such as UUID is used to identify the user. The user based rate limiters are configured independently for different services and restrict the interaction for individual users.

**Service based**: in this approach, the overall throughput to a service is limited so as not to exceed the configured limits.

At present, Rate Limiting has only been applied for the external services. The list of services for which rate limiting has been applied can be found in Table 1 . Rate limiting has not been applied to the internal Web App service components with the understanding that each component will be horizontally scaled.

Figure 5 below shows the user based rate limiting error occurring for User 1 due to invocations made beyond the user based rate limiting thresholds (red arrows). As user based rate limiting is per user, the figure also shows the flow for Users 2..n (blue arrows), who are not impacted by the errors occurring for User 1.



Openshift ECP

3 - BalDue not in cache

9 - cache BalDue object from CBS

1

2

4

7

8

SCG

1

2

4

5 **(User Based**

**Rate Limit Triggered for User 1)**

5

6

User-based, service based rate limiters

configured in SCG for external service calls

User 1

User 2 ...n

Bucket4j (Service and User-Based RL)

rate limiting

RHDG Cache

BalDue

BalDue

Summary

CBS

BalDue Service

CBS BalDue ServiceEntry

BalDue

RHDG Cache

Figure 5: Rate limiter

The snippet below provides an example of how a rate limit filter is configured as part of the spring boot application.yml file. In the configuration the variables are used as follows:

• Capacity – specify the number of unique concurrent requests that can be allowed by the rate limiter

• Duration – the duration in which the bucket capacity will get replenished.

Example Configuration:

**webapps: bucket4j:**

**props:**

**- limit: HEADER duration: 1m capacity:10**

The table below (Table 1) lists all the rate limit filters that have been defined and are currently used. The variables in the table below have been set based on consulation with the Business. While the configurations might change in future, the initial settings have been selected with high thresholds such that the rate limiter errors out under very high error (>60%) rates.

|  |  |  |  |
| --- | --- | --- | --- |
| Table 1: Rate Limiting Filters **Functionality** | **Service** | **Rate limit type** | **Threshold** |
| SADI UserProfile | /sadi-pii/user | User | 10 requests/ min |
| Service | | 236 requests/second | |
| CBS UserProfile | GetTaxpayerPrimaryName | User | 10 requests/ min |
| Service | | 56 requests/second | |
| CBS BalDue | GetIndividualBalanceDue | User | 10 requests/ min |
| Service | | 56 requests/second | |
| CBS Tax Records #1 | GetIndividualTaxReturnModules | User | 10 requests/ min |
| Service | | 236 requests/second | |
| CBS Tax Records #2 | GetIndividualReturnTransaction | User | 10 requests/ min |
| Service | | 56 requests/second | |

**Circuit Breaker**

Circuit Breaker is an application resiliency design pattern that encapsulates the logic of detecting failures and limiting invocations until system has stabilized again. A HTTP 503 service unavailable error will be thrown when a circuit breaker detects a fault condition.

In WebApps, Circuit Breaker has been applied to external services provided by SADI and CBS. The Circuit Breaker is configured in Istio DestinationRules which are configured against the Istio serviceEntry objects that are defined to route traffic to the external services.

Each service endpoint has been configured with it’s own circuit breaker. Once error conditions (specified number of concurrent 500 HTTP errors) are reached, the circuit enters an open state. Once the Circuit Breaker is in open state, all subsequent requests will immediately get a HTTP 503 error until the configured time expires and circuit moves back to a closed state.

At present, Circuit Breakers have only been applied for external service requests. All the services that have circuit breaker configured are listed in Table 2.

Each service has been configured with it’s own circuit breaker and fault detection and management for the services will occur independently, i.e. failure for one service will not cause another service to automatically fail. e.g. failure to the CBS User Profile service will not directly cause a failure to the CBS balance due Service

The current recommendation is to apply the circuit breaker resiliency patterns only for the external services. If multiple components in the request handling path are configured with circuit breakers, there is a possibility for the same error to cause all the circuit breakers in the path to open. So without proper handling of the HTTP 503 error from the circuit breaker, this cascading effect of multiple services going to failed state is avoided by having the circuit breaker in a single layer.

Figure 6 below shows the flow for BalanceDue Summary service along with the circuit breaker components for CBS BalDueService. When the circuit breaker is in open state, subsequent calls during the failed(open) state of the circuit breaker will generate HTTP 503 errors without any invocation to the external CBS service, as highlighted in red below.



3 account summary not in cache

Openshift ECP

Istio DestinationRules based Outlier Detection circuit breaker for external service routes

Istio service mesh

1

2

4

SCG

7

8

5

6

User 1 ...n

rate limiting RHDG Cache

CBS

BalDue Service

Bucket4j (Service and

User-Based RL)

CBS BalDue ServiceEntry

BalDue

BalDue Summary

BalDue RHDG Cache

Figure 6: Circuit breaker

The snippet below provides an example of how a circuit breaker is configured against a Destination Rule. In the configuration the variables are used as follows:

cosecutive5xxErrors - is used to specify the number of HTTP 500 errors after which circuit trips

baseEectionTime – the time for which the circuit remains open

apiVersion: networking.istio.io/v1alpha3 kind: DestinationRule metadata:

name: sadi outlierDetection: consecutive5xxErrors: 5 baseEjectionTime: 3m

The table below (Table 2) lists all the circuit breakers that have been defined and are currently used. The variables in the table below have been set based on consulation with the Business.

While the configurations might change in future, the initial settings have been selected with high thresholds such that the circuit breaker opens under very high error (>60%) rates.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table 2: Circuit Breaker Configurations **Functionality** | **Service** | **Consecutive5xxErrors** | **baseEjectionTime** | **Interval** |
| SADI UserProfile | /sadi-pii/user | 10 | 1m | 1m |
| CBS UserProfile | GetTaxpayerPrimaryName | 10 | 1m | 1m |
| CBS BalDue | GetIndividualBalanceDue | 1000 | 1m | 1m |
| CBS Tax Records #1 | GetIndividualTaxReturnModules | 1000 | 1m | 1m |
| CBS Tax Records #2 | GetIndividualReturnTransaction | 1000 | 1m | 1m |