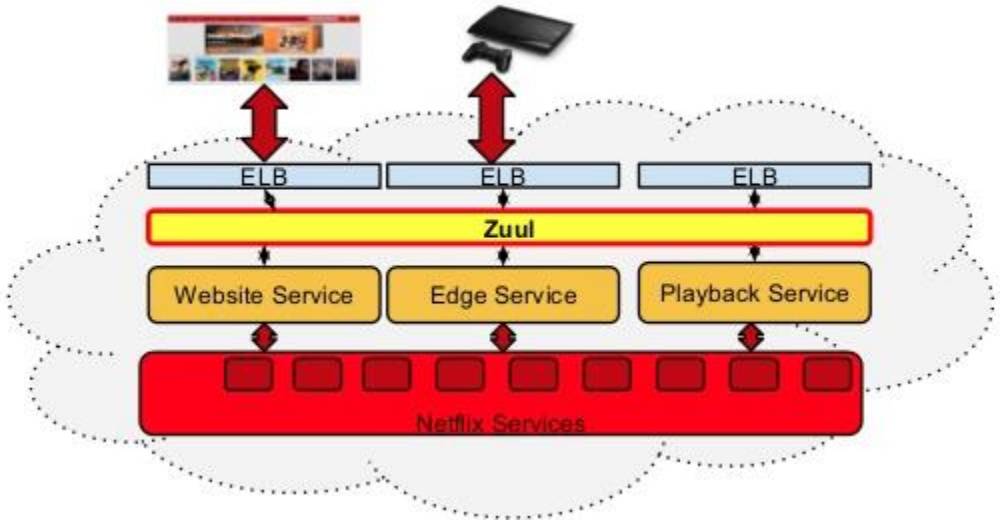
Zuul is the front door for all requests from devices and websites to the backend of the Netflix streaming application. As an edge service application, Zuul is built to enable dynamic routing, monitoring, resiliency, and security.



Routing is an integral part of a microservice architecture. For example, /api/users is mapped to the user service and /api/shop is mapped to the shop service. Zuul is a JVM-based router and server side load balancer by Netflix.

The volume and diversity of Netflix API traffic sometimes results in production issues arising quickly and without warning. We need a system that allows us to rapidly change behavior in order to react to these situations.

Zuul uses a range of different types of filters that enables us to quickly and nimbly apply functionality to our edge service. These filters help us perform the following functions:

- Authentication and Security: identifying authentication requirements for each resource.
- Insights and Monitoring: tracking meaningful data and statistics.

- **Dynamic Routing:** dynamically routing requests to different backend..
- **Stress Testing:** gradually increasing the traffic.
- **Load Shedding:** allocating capacity for each type of request and dropping requests.
- **Static Response handling:** building some responses directly.
- **Multiregion Resiliency:** routing requests across AWS regions.

Zuul contains multiple components:

- **zuul-core:** library that contains the core functionality of compiling and executing Filters.
- **zuul-simple-webapp:** webapp that shows a simple example of how to build an application with zuul-core.
- **zuul-netflix**: library that adds other NetflixOSS components to Zuul — using Ribbon for routing requests, for example.
- **zuul-netflix-webapp:** webapp which packages zuul-core and zuul-netflix together into an easy to use package.

Zuul gives us a lot of insight, flexibility, and resiliency, in part by making use of other Netflix OSS components:

- Hystrix is used to wrap calls to our origins, which allows us to shed and prioritize traffic when issues occur.
- Ribbon is our client for all outbound requests from Zuul, which provides detailed information into network performance and errors, as well as handles software load balancing for even load distribution.
- Turbine aggregates finegrained metrics in realtime so that we can quickly observe and react to problems.
- Archaius handles configuration and gives the ability to dynamically change properties.

We can create a filter to route a specific customer or device to a separate API cluster for debugging. Prior to using Zuul, we were using Hadoop to query through billions of logged requests to find the several thousand requests we were interested in.

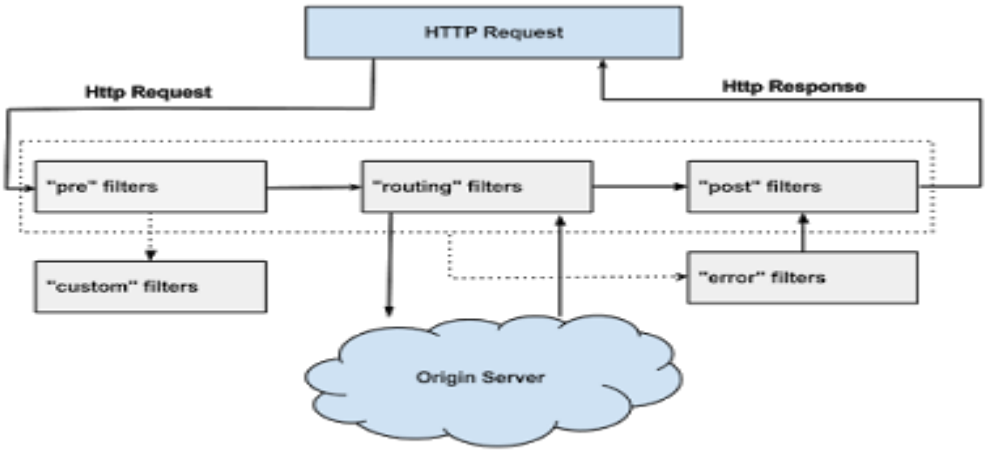
We have an automated process that uses dynamic Archaius configurations within a Zuul filter to steadily increase the traffic routed to a small cluster of origin servers. As the instances receive more traffic, we measure their performance characteristics and capacity.

Spring Cloud has created an embedded Zuul proxy to ease the development of a very common use case where a UI application wants to proxy calls to one or more back end services. This feature is useful for a user interface to proxy to the backend services it requires, avoiding the need to manage CORS and authentication concerns independently for all the backends.

To enable it, annotate a Spring Boot main class with @EnableZuulProxy, and this forwards local calls to the appropriate service. By convention, a service with the ID "users", will receive requests from the proxy located at /users.

The proxy uses Ribbon to locate an instance to forward to via discovery, and all requests are executed in a hystrix command, so failures will show up in Hystrix metrics, and once the circuit is open the proxy will not try to contact the service.

Zuul Request Lifecycle



In this picture, it is possible check that, before accessing the origin server, Zuul provides some functionality to add in requests or after requests (responses), like filtering, routing, aggregation, error treatment, etc.