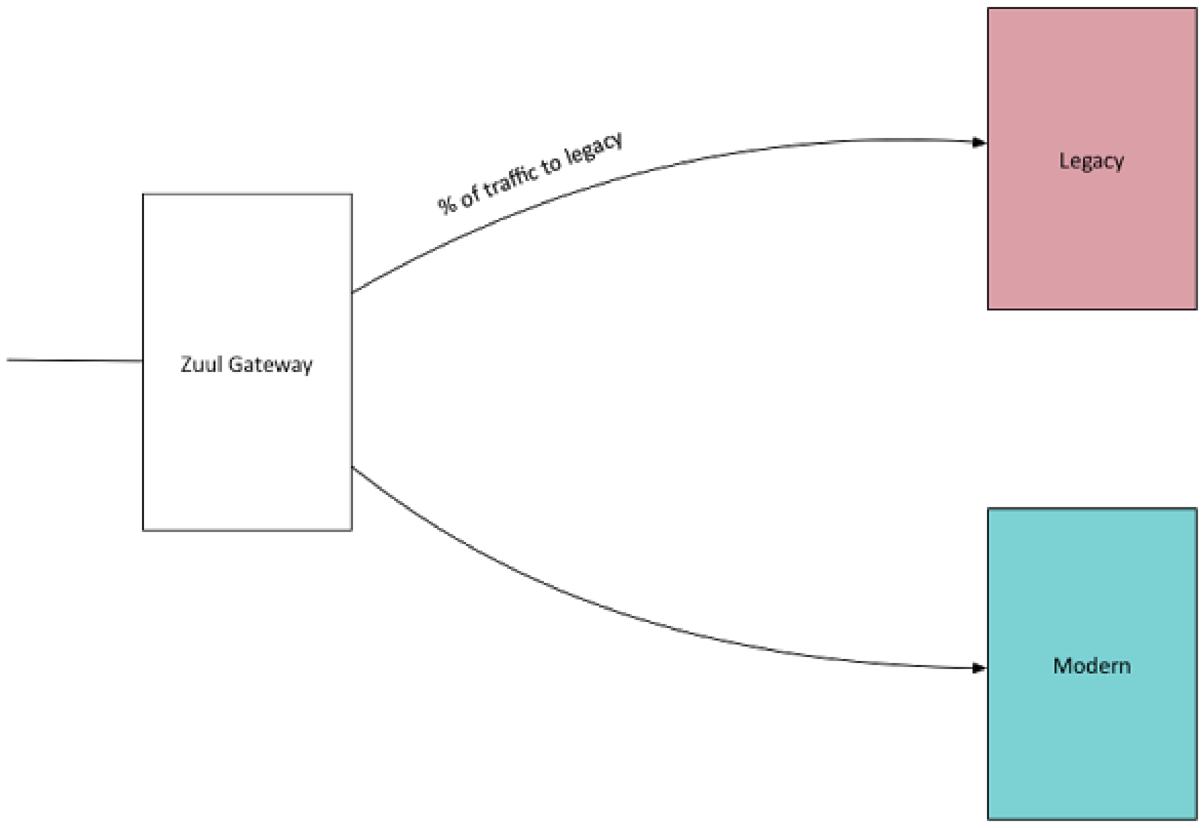
Ratio based routing to a legacy and a modern app – Netflix Zuul via Spring Cloud

javacodegeeks.com/2017/05/ratio-based-routing-legacy-modern-app-netflix-zuul-via-spring-cloud.html

A very common requirement when migrating from a legacy version of an application to a modernized version of the application is to be able to migrate the users slowly over to the new application. In this post I will be going over this kind of a routing layer written using support for Netflix Zuul through Spring Cloud . Before I go ahead I have to acknowledge that most of the code demonstrated here has been written in collaboration with the superlative Shaozhen Ding

Scenario

I have a legacy service which has been re-engineered to a more modern version(assumption is that as part of this migration the uri's of the endpoints have not changed). I want to migrate users slowly over from the legacy application over to the modern version.



Implementation using Spring Cloud Netflix – Zuul Support

This can be easily implemented using Netflix Zuul support in Spring Cloud project.

Zuul is driven by a set of filters which act on a request before(pre filters), during(route filters) and after(post filters) a request to a backend. Spring Cloud adds it custom set of filters to Zuul and drives the behavior of these filters by configuration that looks like this:

```
zuul:
  routes:
    ratio-route:
      path: /routes/**
      strip-prefix:false
```

This specifies that Zuul will be handling a request to Uri with prefix "/routes" and this prefix will not be stripped from the downstream call. This logic is encoded into a "PreDecorationFilter". My objective is to act on the request AFTER the PreDecorationFilter and specify the backend to be either the legacy version or the modern version. Given this a filter which acts on the request looks like this:

```
import com.netflix.zuul.ZuulFilter;
import com.netflix.zuul.context.RequestContext;
...

@Service
public class RatioBasedRoutingZuulFilter extends ZuulFilter {

    public static final String LEGACY_APP = "legacy";
    public static final String MODERN_APP = "modern";

    private Random random = new Random();
```

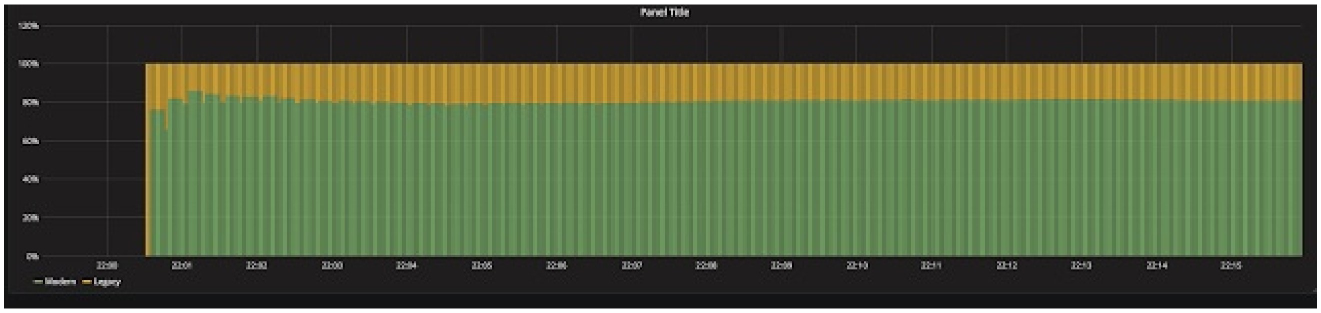
```java

    @Autowired
    private RatioRoutingProperties ratioRoutingProperties;

    @Override
    public String filterType() {
        return "pre";
    }

    @Override
    public int filterOrder() {
        return FilterConstants.PRE_DECORATION_FILTER_ORDER + 1;
    }

    @Override
    public boolean shouldFilter() {
        RequestContext ctx = RequestContext.getCurrentContext();
        return ctx.containsKey(SERVICE_ID_KEY)
```

```
                && ctx.get(SERVICE_ID_KEY).equals("ratio-route");
    }

    @Override
    public Object run() {
        RequestContext ctx = RequestContext.getCurrentContext();

        if (isTargetedToLegacy()) {
            ctx.put(SERVICE_ID_KEY, LEGACY_APP);
        } else {
            ctx.put(SERVICE_ID_KEY, MODERN_APP);
        }
        return null;
    }

    boolean isTargetedToLegacy() {
        return random.nextInt(100) < ratioRoutingProperties.getOldPercent();
    }
```

```
}
```

The filter is set to act after the “PreDecorationFilter” by overriding the filterOrder() method. The routing logic is fairly naive but should work for most cases. Once the serviceId is resolved, Spring Cloud would use Ribbon to route the request and just for variation I am using a configured url for legacy call and Eureka for the modern backend call. If you are interested in exploring the entirety of the application it is available in my github repo

With the entire set-up in place, a small test with the legacy handling 20% of the traffic confirms that the filter works effectively:



Conclusion

Spring Cloud support for Netflix Zuul makes handling such routing scenarios a cinch and should be a good fit for any organization having these kinds of routing scenarios that they may want to implement.

Reference: Ratio based routing to a legacy and a modern app – Netflix Zuul via Spring Cloud from our JCG partner Biju Kunjummen at the all and sundry blog.