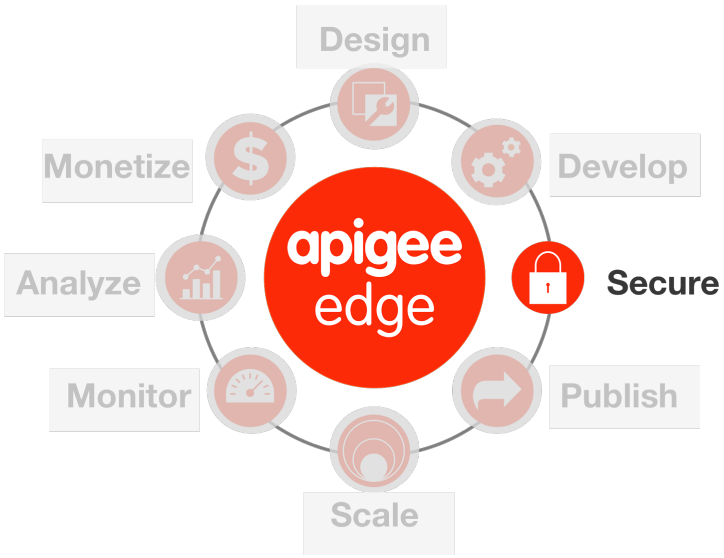
Lab 2 - Protecting APIs (Traffic Management)



Overview

To maintain performance and availability across a diverse base of client apps, it's critical to maintain app traffic within the limits of the capacity of your APIs and backend services. It's also important to ensure that apps don't consume more resources than permitted.

Apigee Edge provides three mechanisms that enable you to optimize traffic management to minimize latency for apps while maintaining the health of backend services. Each policy type addresses a distinct aspect of traffic management. In some cases, you might use all three policy types in a single API proxy.

Spike Arrest Policy

This policy smooths traffic spikes by dividing a limit that you define into smaller intervals. For example, if you define a limit of 100 messages per second, the Spike Arrest policy enforces a limit of about 1 request every 10 milliseconds (1000 / 100); and 30 messages per minute is smoothed into about 1 request every 2 seconds (60 / 30). The Spike Arrest limit should be close to capacity calculated for either your backend service or the API proxy itself. The limit should also be configured for shorter time intervals, such as seconds or minutes. This policy should be used to prevent sudden traffic bursts caused by malicious attackers attempting to disrupt a service using a denial-of-service (DOS) attack or by buggy client applications.

See Spike Arrest policy - http://apigee.com/docs/ja/api-services/reference/spike-arrest-policy

Quota Policy

This policy enforces consumption limits on client apps by maintaining a distributed 'counter' that tallies incoming requests. The counter can tally API calls for any identifiable entity, including apps, developers, API keys, access tokens, and so on. Usually, API keys are used to identify client apps. This policy is computationally expensive so, for high-traffic APIs, it should configured for longer time intervals, such as a day or month. This policy should be used to enforce business contracts or SLAs with developers and partners, rather than for operational traffic management.

See Quota policy - http://apigee.com/docs/ja/api-services/reference/quota-policy

Concurrent Rate Limit Policy

This policy enables traffic management between API Services and your backend services. Some backend services, such as legacy applications, may have strict limits on the number of simultaneous connections they can support. This Policy enfoces a limit on the number of requests that can be sent at any given time from API services to your backend service. This number is counted across all of the distributed instances of API Services that may be calling your backend service. Policy limits and time duration should be configured to match the capacity available for your backend service.

See Concurrent Rate Limit policy - http://docs.apigee.com/api-services/reference/concurrent-rate-limit-policy

Caching Policies

Apigee Edge supports different caching policies enabling you to:

- **Reduce latency:** A request satisfied from the cache gets the representation and displays it in a shorter time. The server is more responsive with requests satisfied from the cache, which is closer to the client than the origin server.
- **Reduce network traffic:** Representations are reused, reducing the impact of processing duplicate or redundant requests. Using a cache also reduces the amount of bandwidth you use.
- **Persist data across transactions:** You can store session data for reuse across HTTP transactions.
- **Support security:** You may need "scope" access to the contents of a cache so it can only be accessed in a particular environment or by a specific API proxy.

The various caching policies supported by Apigee Edge are:

Response Cache Policy: Uses a cache to store and retrieve data from a backend resource, reducing the number of requests to the resource. For policy reference information, see Response Cache policy - http://apigee.com/docs/api-services/reference/response-cache-policy

Populate Cache Policy: Use the PopulateCache policy to write data to the cache. For policy reference information, see Populate Cache policy http://apigee.com/docs/api-services/reference/populate-cache-policy

Lookup Cache Policy: You can retrieve cached values with the LookupCache policy. For policy reference information, see LookupCache policy http://apigee.com/docs/api-services/reference/lookup-cache-policy

Invalidate Cache Policy: Configures how the cached values should be purged from the cache. For policy reference information, see Invalidate Cache policy http://apigee.com/docs/api-services/reference/invalidate-cache-policy

Objectives

The goal of this lesson is to introduce you to Traffic Management policies and applying a couple of these policies to the API Proxy you created in the previous lesson.

Pre-Requisites

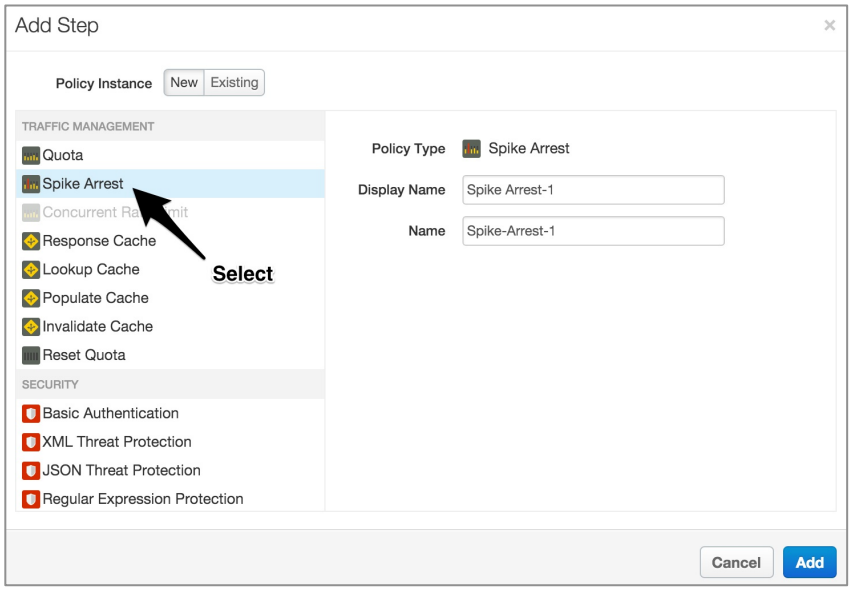
- Lab 1 is completed.

Estimated Time: 30 mins

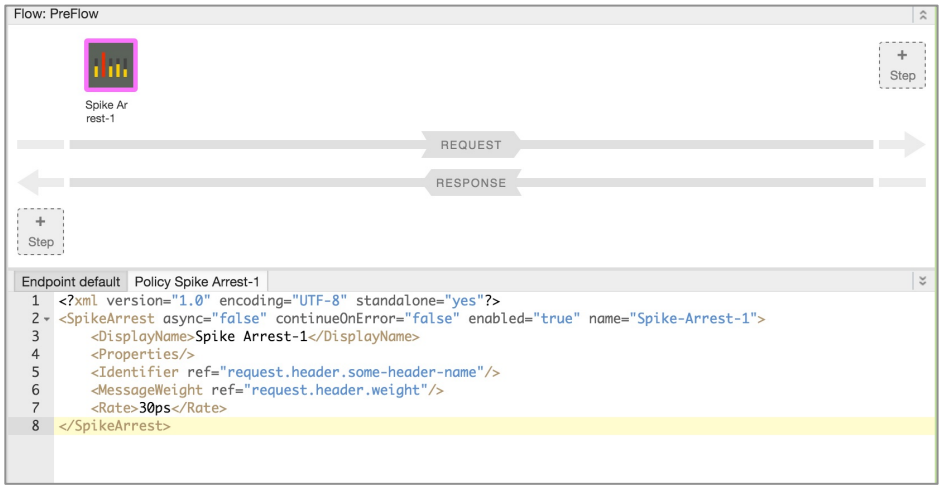
1) **Adding a Spike Arrest Policy**

   a) Go to the Apigee Edge Management UI browser tab.
   b) Select your API proxy.
   c) Switch to “Develop” tab.
   d) Go to Preflow under “Proxy EndPoints” section and Click “**+ Step**” on the Request pipeline.

e) Select the 'Spike Arrest' from the list of policies and click on Add.



f) This will add the Spike Arrest policy to Proxy PreFlow section on Request pipeline.



By applying the policy to the Flow PreFlow, this Spike Arrest policy will get enforced for all the resources defined for this Proxy.

g) Edit the Spike Arrest policy configuration XML to allow only 10 request per minute. The XML for the policy should look something like this:

Alternatively you can copy the below policy XML and paste into your policy editor.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SpikeArrest async="false" continueOnError="false" enabled="true"
name="Spike-Arrest-1">
    <DisplayName>Spike Arrest-1</DisplayName>
    <Properties/>
    <Rate>10pm</Rate>
</SpikeArrest>
```

h) Save the changes to the proxy and ensure that it is deployed successfully to the 'test' environment.

Think of Spike Arrest as a way to generally protect against traffic spikes (system wide) rather than as a way to limit traffic to a specific number of requests for certain users. Your APIs and backend can handle a certain amount of traffic, and the Spike Arrest policy helps you smooth traffic to the general amounts you want.

The runtime Spike Arrest behavior differs from what you might expect to see from the literal per-minute or per-second values you enter.

For example, say you enter a rate of 30pm (30 requests per minute). In testing, you might think you could send 30 requests in 1 second, as long as they came within a minute. But that's not how the policy enforces the setting. If you think about it, 30 requests inside a 1-second period could be considered a mini spike in some environments.

What actually happens, then? To prevent spike-like behavior, Spike Arrest smooths the allowed traffic by dividing your settings into smaller intervals:

- **Per-minute** rates get smoothed into requests allowed intervals of **seconds**. For example, 30pm gets smoothed like this: 60 seconds (1 minute) / 30pm = 2-second intervals, or about 1 request allowed every 2 seconds. A second request inside of 2 seconds will fail. Also, a 31st request within a minute will fail.
- **Per-second** rates get smoothed into requests allowed in intervals of **milliseconds**.For example, 10ps gets smoothed like this: 1000 milliseconds (1 second) / 10ps = 100-millisecond intervals, or about 1 request allowed every 100 milliseconds . A second request inside of 100ms will fail. Also, an 11th request within a second will fail.

2) Testing the Spike Arrest Policy

a) Switch to Trace tab of your API proxy. Alterntively call your API from a brower using the following URL.

http://**{your-edge-org}**-test.apigee.net/v1/**{your_initials}**_hotels

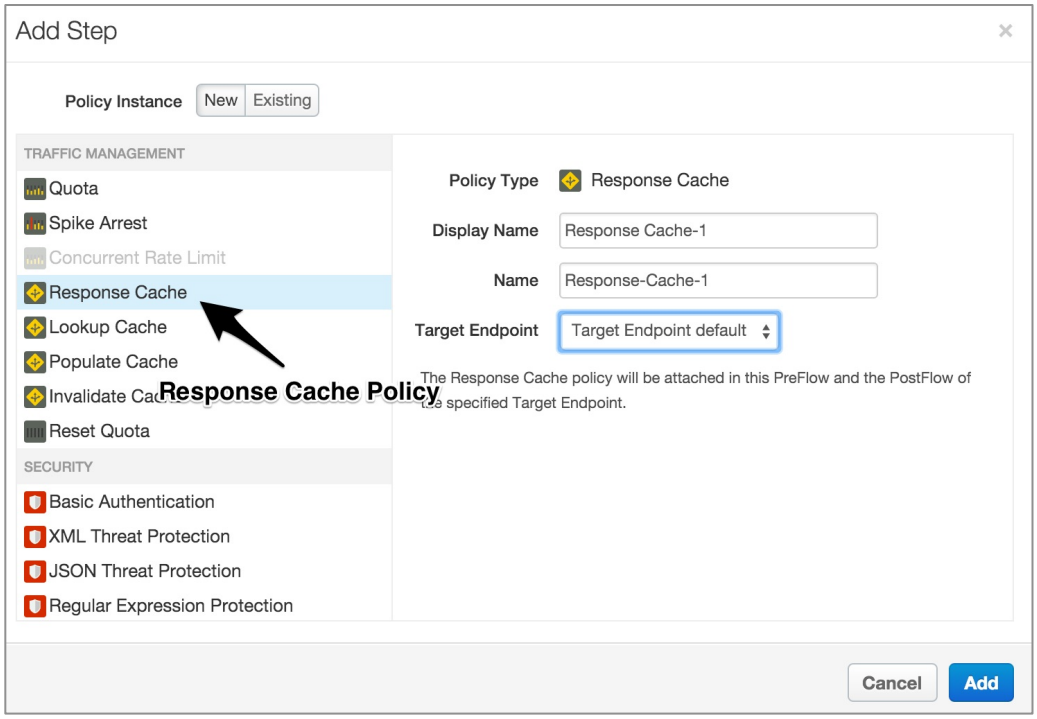
Replace **{your-edge-org}** with the actual name of your Edge organization. Replace **{your_initials}** with your initials. This should match with your API proxy name.

b) Start a trace session and quickly send more than 10 requests in a minute and observe that certain requests will receive an error with the errorCode “policies.ratelimit.SpikeArrestViolation”

3) Adding Response Cache Policy to reduce external service calls, reduce network traffic and improve performance

a) Switch to the ‘Develop’ tab.
b) Go to Preflow under “Proxy EndPoints” section and Click “**+ Step**” on the Request pipeline.

c) Select the 'Spike Arrest' from the list of policies and click on Add.

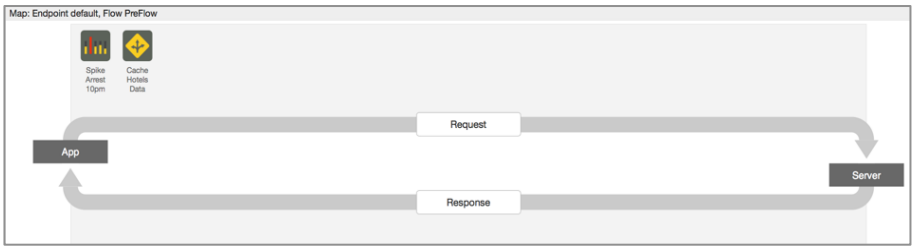


Once the 'Cache Hotels Data' policy appears in the 'Navigator' panel, review its properties. Since everything else except the name was left as a default, you will notice that the Expiration Timeout in Seconds is set to 3600 seconds. The timeout property along with other properties should be modified as per your use cases. For policy reference information, see Response Cache policy - http://apigee.com/docs/api-services/reference/response-cache-policy

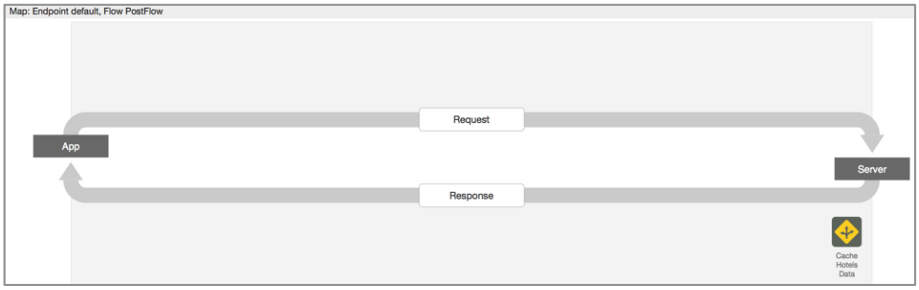
The Response Cache policy needs a Cache Resource that can be used to cache the data. Apigee Edge provides a default cache resource that can be used for

quick testing, which is what is being used in this lesson. The Cache Resource to be used by Response Cache policies should also be created and configured as per your use cases. For Cache Resource configuration information, see Manage Caches for an Environment - http://apigee.com/docs/api-services/content/manage-caches-environment

d) Your Proxy Endpoints → Default → PreFlow should look as follows:



e) Your Target Endpoints → Default → PostFlow should look as follows:

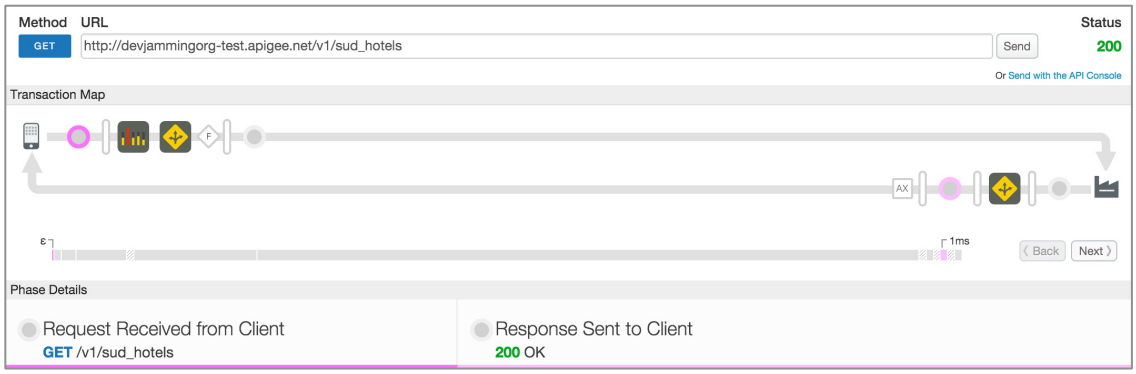


f) Save the changes to the API Proxy, wait for it to successfully deploy.

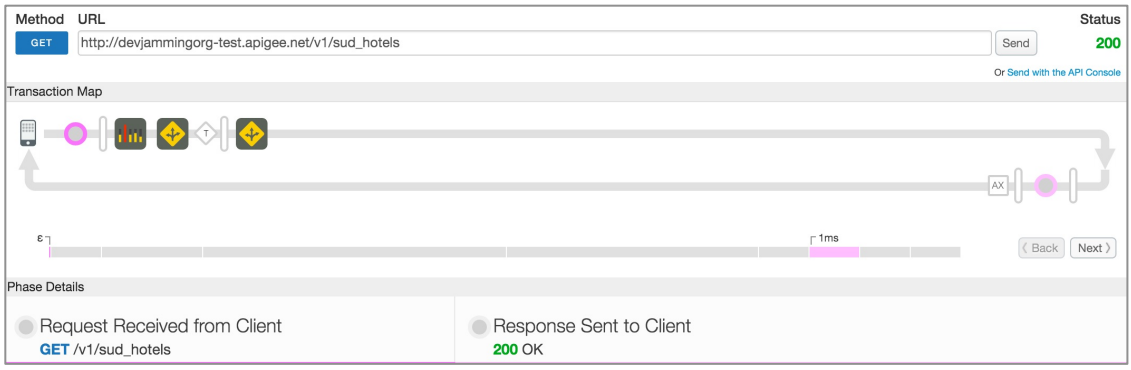
4) Testing the Response Cache Policy

a) Start the API Trace and send a make a couple of calls.
b) Wait for 6 to 10 seconds (to avoid the Spike Arrest policy from stopping your requests) and send the same request again.
c) Go back to the Trace view and review the transaction map of both the requests including the overall elapsed time to process both requests

The first transaction map should look as follows:



The second execution flow should look as follows:



After configuring the Response Cache policy, as expected, after the initial request, the second and all other requests for the next 3600 seconds will be served from the cache and hence avoid executing any other policies.

Summary

That completes this hands-on lesson. You learned how to use the Spike Arrest to protect the environment from traffic spikes and to use the Response Cache policy to provide a better overall experience for the API consumer while reducing network traffic. Obviously like any other policy, these policies must be used appropriately based upon your use cases.