



January 2014

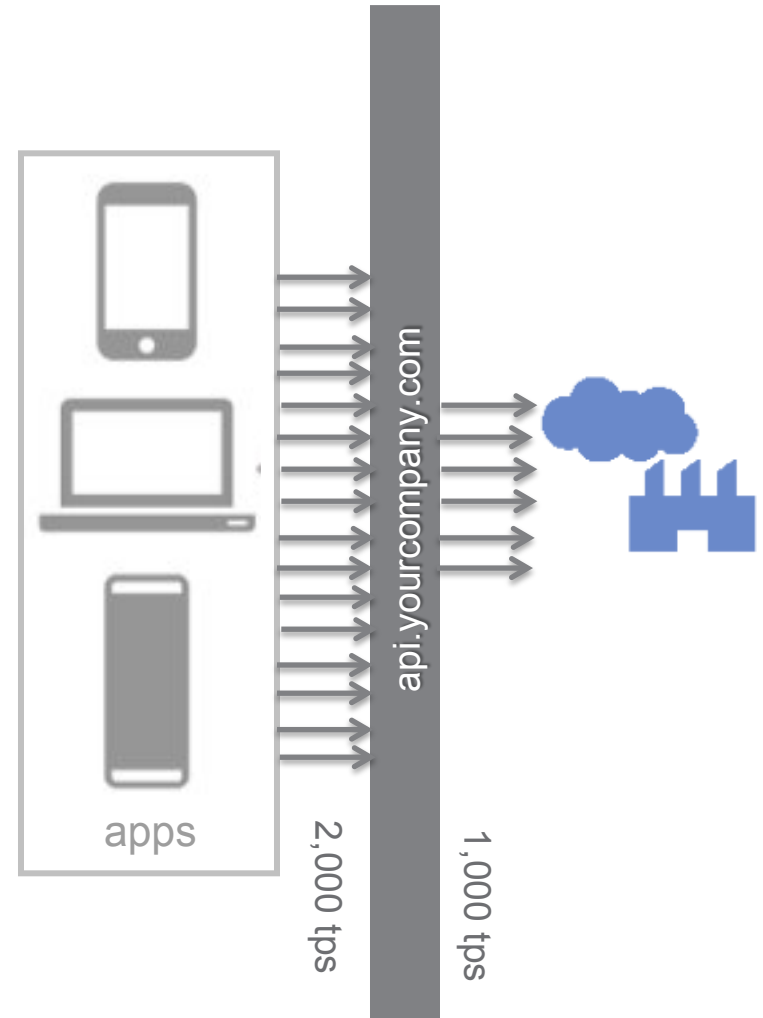
Create and Manage APIs

Spike Arrest



Main Content and Bullets

- **SpikeArrest** smooths inflow request patterns over short intervals to ensure that demand does not outstrip capacity.
- **SpikeArrest** is a technical requirement to protect the backend as opposed to a **Quota** or **Rate Limit** which is a business requirement to manage developer relationships.



Spike Arrest and Denial of Service

- To prevent the sudden influx of requests due to malicious attacks (Denial of Service) or other reasons, use the **SpikeArrest** policy to throttle the number of requests
- Put SpikeArrest policy as early as possible in request flow to minimize processing for rejected traffic
 - May need to be before authentication to avoid having anonymous, malicious traffic block legitimate traffic
- No counter is used
 - Stores time that last traffic was accepted on the current message processor
 - Traffic that follows too closely to previous request will be rejected
- Optional <Identifier> field used to keep separate tracking times for different identifiers
 - For example, if the identifier is `client_id`, the specified rate is allowed for each `client_id`
 - Times tracked separately for different environments, bundles, and/or policies

Spike Arrest and Denial of Service

- <Rate> specifies how fast traffic can be sent to a single message processor for the given identifier
 - For example 50ps = 50 per second, 12pm = 12 per minute
 - If 50 per second is specified rate, traffic can only be received every 1/50 of a second (20 ms) on each message processor
 - A second request within 20 ms on the same message processor would be rejected
 - Even with a large number allowed per second, nearly simultaneous requests will result in rejections
- Optional parameter <MessageWeight> can be used to count a request as more than one request
 - For 50ps example with message weight of 2, the next request couldn't come in until $2 \times 20 = 40$ ms elapsed

Code: Spike-Arrest

```
1 <SpikeArrest async="false" continueOnError="false" enabled="true" name="Spike-Arrest">
2   <Identifier ref="client_id"/>
3   <MessageWeight ref="request.header.weight"/>
4   <Rate>30ps</Rate>
5 </SpikeArrest>
```

Lets Add a Spike Arrest Policy

Attach Spike Arrest Policy



TRAFFIC MANAGEMENT

- Quota
- Spike Arrest**
- Concurrent Rate Limit
- Response Cache
- Lookup Cache
- Populate Cache
- Invalidate Cache
- Reset Quota

SECURITY

- Basic Authentication
- XML Threat Protection
- JSON Threat Protection
- Regular Expression Protection
- OAuth v2.0
- Get OAuth v2.0 Info
- Set OAuth v2.0 Info
- OAuth v1.0a
- Get OAuth v1.0a Info
- Verify API Key
- Access Control
- Generate SAML Assertion
- Validate SAML Assertion

Name and Select the ProxyEndpoint (Preflow) from the flows, then press Add

New Policy: Spike Arrest ×

Policy Display Name

Policy Name

Attach Policy ☒

Flow


Segment ☒ Request ☐ Response

Cancel Add













Spike Arrest Configuration Options

- **Identifier** lets you focus your spike arrest on a subset of requests based on the identifier
- **MessageWeight** lets you specify how much impact a certain request has rate calculation.
- **Rate** lets you specify the top rate allowed through this policy

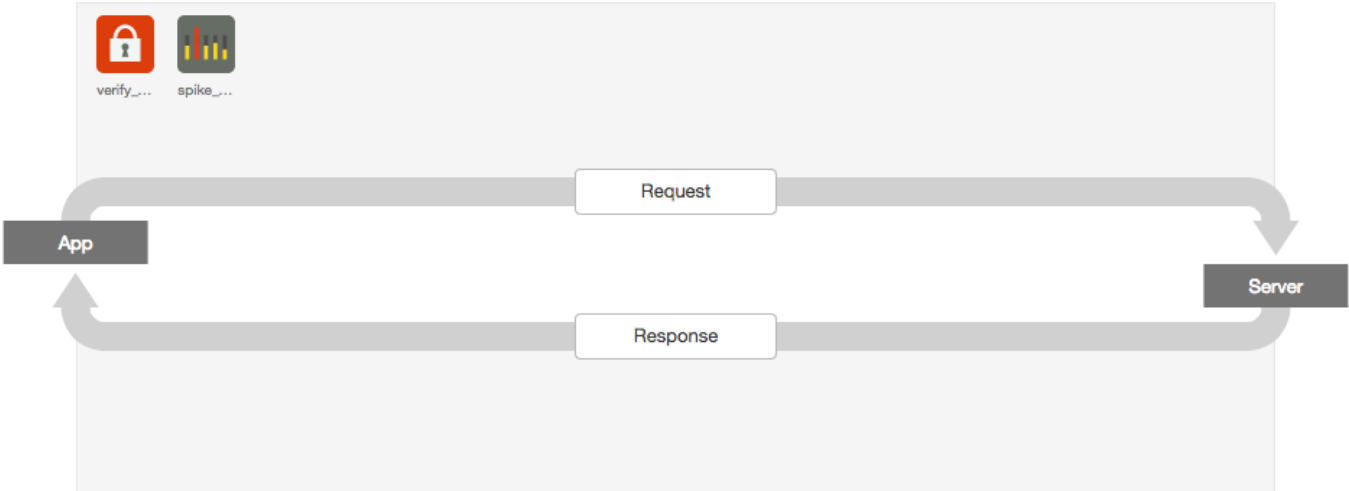
Steves-APIEater

Project ▾ Save Revision 1 ▾ New ▾ New Policy ▾ Attach Policy Tools ▾ Deployment ▾ Help for Selected  Spike Arrest Policy

Navigator

- ▾ Policies
 -  basic_authentication
 -  kvm_get_credentials
 -  spike_arrest_inbound
 -  verify_apikey
- ▾ Proxy Endpoints
 - ▾ default
 -  PreFlow
 -  trucks
 -  chefs
 -  cuisines
 -  routes
 -  PostFlow
- ▾ Target Endpoints
 - ▾ default
 -  PreFlow
 -  PostFlow
- ▾ Scripts

Map: Endpoint default, Flow PreFlow



```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SpikeArrest async="false" continueOnError="false" enabled="true" name="spike_arrest_inbound">
  <DisplayName>spike_arrest_inbound</DisplayName>
  <FaultRules/>
  <Properties/>
  <Identifier ref="request.header.some-header-name" />
  <MessageWeight ref="request.header.weight" />
  <Rate>30ps</Rate>
</SpikeArrest>
```

Day2 – Lab 10 Spike Arrest

- Add a spike Arrest policy to Proxy Endpoint PreFlow

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

```
<SpikeArrest async="false" continueOnError="false" enabled="true" name="Spike-Arrest">
```

```
  <DisplayName>Spike Arrest</DisplayName>
```

```
  <Properties/>
```

```
  <Rate>5pm</Rate>
```

```
</SpikeArrest>
```

- Test using postman
 - Send more than 5 request / minute to make sure spike arrest triggers

Basic Quota Policy



Quota

- Use the **Quota** policy to configure the number of requests that an app can make to an API over the course of an hour, day, week, or month
- The Quota policy utilizes counters, unlike SpikeArrest
 - No burst traffic limits over the time period
 - Once limit is reached, subsequent API calls are rejected
 - Quotas can be reset using the **ResetQuota** policy
- <Identifier> can be used to have separate counters for separate apps
 - Counts are automatically separate for different environments, bundles, and policy names
 - Identifier can further split counts
 - Identifier is generally used to identify an app (client_id), but can use any different variable
- <MessageWeight> is used to count a request as more than one request toward the quota limit

Quota Usage

- Quota usage is specified as an allowed number of requests per number of time periods (example: 1000 requests per 1 month)

Example Product Products give developers access to your APIs. [Learn more](#)

Product Details

Display Name Example Product

Description

Environment ☒ test ☒ prod

Access Public

Key Approval Type Automatic

Quota 10000 requests every 1 month

Allowed OAuth Scopes

- Number of requests = <Allow>
- Number of time periods = <Interval>
(default is 1)
- Time unit = <TimeUnit>
(default is month)

- It is common to take these values from the API Product, but they can be taken from developer attributes as well
- For each item, you can supply a variable from which to retrieve the value, and a default value to use if the variable is null, doesn't exist or isn't supplied

Code: Quota-1

```
1 <Quota async="false" continueOnError="false" enabled="true" name="Quota-1">
2   <Allow count="1000" countRef="apiproduct.product.quota.limit"/>
3   <Interval ref="apiproduct.product.quota.interval">1</Interval>
4   <TimeUnit ref="apiproduct.product.quota.timeunit">month</TimeUnit>
5 </Quota>
```

Quota Type

- Quota type indicates when the quota counter starts counting usage
 - **calendar** (default if not specified)
 - Quota counting has an explicit start time
 - `<StartTime>` is specified and quota counting starts at the `StartTime` and is reset based on the specified `<Interval>` and `<TimeUnit>`
 - **rollingwindow**
 - Quota uses a rolling window that resets based on the `<Interval>` and `<TimeUnit>`
 - The start time is the time the first message is received matching the `<Identifier>`
 - `rollingwindow` allows a quota that resets on every interval
 - **flexi**
 - Start time is dynamic for each `<Identifier>` based on first message being received
 - Calls can be used until interval has elapsed
 - Quota does not automatically reset at the end of the interval
 - `flexi` allows access for a specified period of time
 - Quota type specified in **type** attribute of Quota root element

Distributed Quotas

- `<Distributed>` specifies whether the count is shared among all message processors (`Distributed = true`) or maintained separately for each message processor (`Distributed = false`)
 - Default is false, but should be set to true if the developer expects a particular number of allowed messages
 - Unpredictable count if set to false, since traffic for a specific developer won't generally be distributed evenly among the message processors
- Distributed quotas can't have per-second `<TimeUnit>`
 - per-minute is the lowest for distributed quotas
- `<Synchronous>` specifies whether the distributed quota counter is updated synchronously
 - Setting `<Synchronous>` to true gives a more up-to-date count, and should be used if it is essential to not allow calls over the quota
 - Setting `<Synchronous>` to false (the default) provides better performance, but it is possible some API calls exceeding the quota could go through

Asynchronous Distributed Quotas

- **<AsynchronousConfiguration>** specifies how an asynchronous distributed quota is updated
 - **<SyncIntervallInSeconds>** indicates that the distributed counter should be updated every n seconds
 - **<SyncMessageCount>** indicates that the distributed counter should be updated after every nth message received
 - The two settings are mutually exclusive
 - If no **<AsynchronousConfiguration>** setting is provided for an asynchronous distributed quota, the default is to update the distributed counter every 10 seconds
- If **<PreciseAtSecondsLevel>** is set to true, the quota will be enforced with an accuracy of a second, even if the **<TimeUnit>** is set at a unit longer than a second
 - If set to false (the default), the quota will be accurate to a minute for all **<TimeUnit>** settings greater than a second

Attach Quota Policy



TRAFFIC MANAGEMENT

- Quota
- Spike Arrest
- Concurrent Rate Limit
- Response Cache
- Lookup Cache
- Populate Cache
- Invalidate Cache
- Reset Quota

SECURITY

- Basic Authentication
- XML Threat Protection
- JSON Threat Protection
- Regular Expression Protection
- OAuth v2.0
- Get OAuth v2.0 Info
- Set OAuth v2.0 Info
- OAuth v1.0a
- Get OAuth v1.0a Info
- Verify API Key
- Access Control
- Generate SAML Assertion
- Validate SAML Assertion

Name and Select the ProxyEndpoint (Preflow) from the flows, then press Add

New Policy: Quota ×

Policy Display Name

quota_proxy

Policy Name

quota_proxy

Attach Policy

☒

Flow

Flow PreFlow, Proxy Endpoint default ▾

Segment

☒ Request ☐ Response

Cancel

Add

Quota Flow Variables

- Flow variables are created after a Quota policy executes
 - Variables are prefixed with "ratelimit.{policy_name}."
 - Examples are:
 - ratelimit.{policy_name}.available.count (available quota count)
 - ratelimit.{policy_name}.used.count (used quota count)
 - ratelimit.{policy_name}.expiry.time (time in ms when quota resets)
 - ratelimit.{policy_name}.identifier (identifier for the policy)

Code: Quota-2

```
1 <Quota async="false" continueOnError="false" enabled="true" name="Quota-2" type="calendar">
2   <Allow count="2000" countRef="apiproduct.developer.quota.limit"/>
3   <Interval ref="apiproduct.developer.quota.interval">1</Interval>
4   <TimeUnit ref="apiproduct.developer.quota.timeunit">month</TimeUnit>
5   <Distributed>true</Distributed>
6   <Synchronous>false</Synchronous>
7   <StartTime>2014-10-22 12:00:00</StartTime>
8   <AsynchronousConfiguration>
9     <SyncMessageCount>5</SyncMessageCount>
10  </AsynchronousConfiguration>
11  <PreciseAtSecondsLevel>true</PreciseAtSecondsLevel>
12 </Quota>
```

Quotas set by Products •



Quota Policy and Product

New Product Products give developers access to your APIs. [Learn more](#)

Product Details

Display Name

Description

Environment ☒ test ☒ prod

Access ☐ Internal only — Visible only to developers at trainingmats during app registration
☐ Private — Visible only to external developers with explicit permission during app registration
☒ Public — Visible only to any registered developer during app registration

Key Approval Type ☒ Automatic ☐ Manual

[Learn more](#)

Quota requests every

Allowed OAuth Scopes

Comma separated scope names. [Learn more](#)

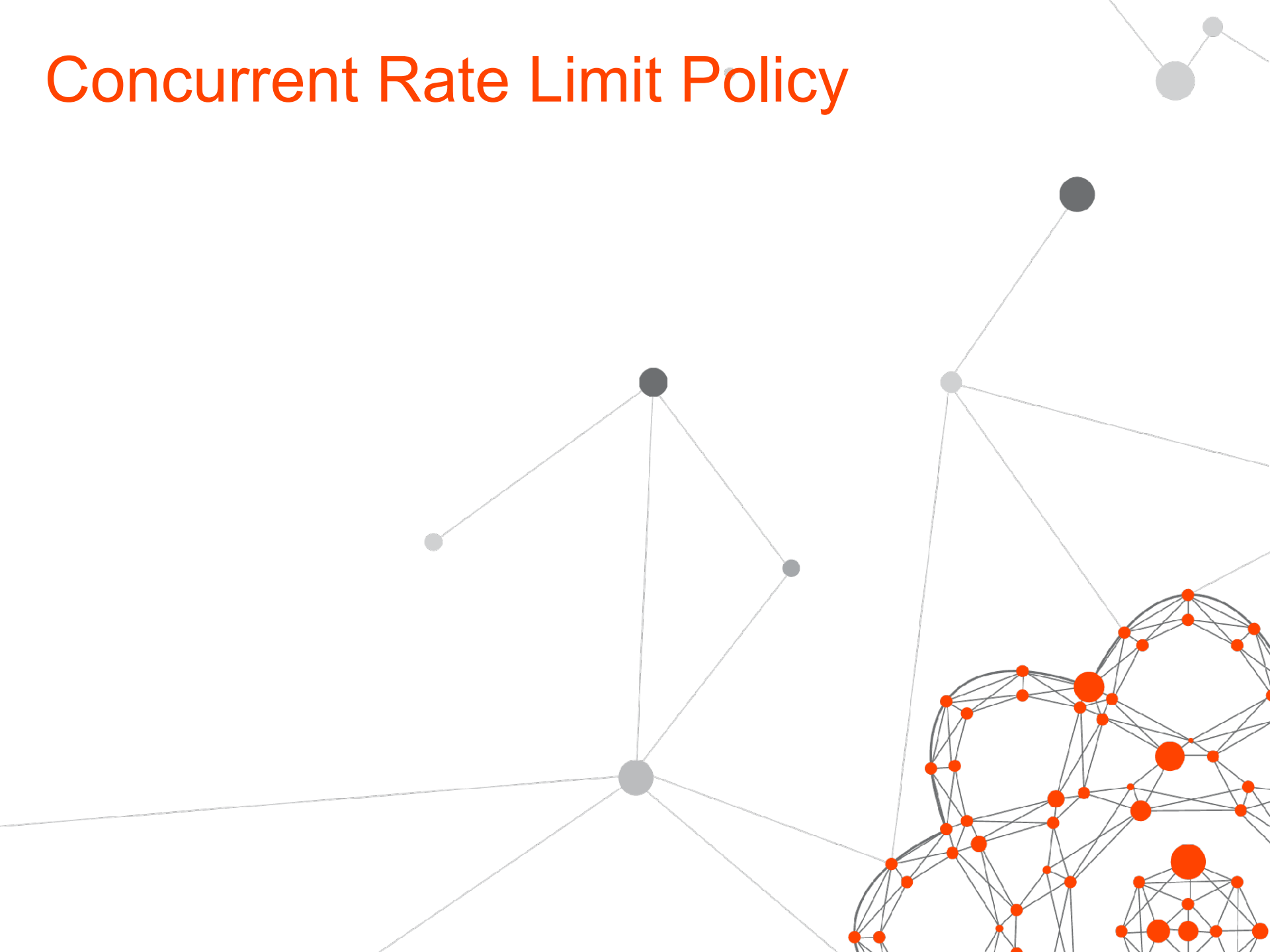
Quota Policy

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Quota async="false" continueOnError="false" enabled="true" name="quota" type="calendar">
  <DisplayName>Quota</DisplayName>
  <FaultRules/>
  <Properties/>
  <Allow count="2000" countRef="verifyapikey.verify-apikey.apipproduct.developer.quota.limit" />
  <Interval ref="verifyapikey.verify-apikey.apipproduct.developer.quota.interval">1</Interval>
  <Distributed>false</Distributed>
  <Synchronous>false</Synchronous>
  <TimeUnit ref="verifyapikey.verify-apikey.apipproduct.developer.quota.timeunit">month</TimeUnit>
  <StartTime>2013-12-8 12:00:00</StartTime>
  <AsynchronousConfiguration>
    <SyncIntervalInSeconds>20</SyncIntervalInSeconds>
    <SyncMessageCount>5</SyncMessageCount>
  </AsynchronousConfiguration>
  <Identifier ref="request.queryparam.apikey" />
</Quota>
```

Using Quota as a Security Policy

- Quotas often use the `client_id` as the identifier to count traffic against an app
 - Can use other identifiers as well
- Use the email address or username as an identifier
- Place the quota in your OAuth login flow
- Can use a fairly low limit (example: 5 per minute) to protect against brute force login attacks

Concurrent Rate Limit Policy



Concurrent Rate Limit

- The **ConcurrentRatelimit** policy is used to limit the number of simultaneous connections allowed to a single backend target
- The policy works by keeping count of active connections to the backend target
- The policy is attached to the target's request preflow, response postflow, and default fault rule with `<AlwaysEnforce>` set to true
- When the flow reaches the target's request preflow policy, the active connection count is checked to see whether there is an available connection
 - If there is an available connection, the count of available connections is decremented, and communication to the backend continues
 - If all connections are in use, 503 Service Unavailable is returned to the user
- When the connection to the backend is allowed and the flow reaches the target's response postflow policy or the policy in the default fault rule, the number of available connections is incremented

Concurrent Rate Limit

- Using the same policy for more than one target will share the counter
- Counter cannot be shared across multiple proxies
- `<AllowConnections>` attribute count sets the number of allowed connections
- `<AllowConnections>` attribute ttl indicates that a connection that is still in use should be made available after the configured number of seconds has elapsed, even if the backend call has not yet returned
- When `<Distributed>` is true, the counter is intended to be shared among all message processors
 - For performance reasons, it is recommended to use `<Distributed> = false`

Concurrent Rate Limit

- If <StrictOnTtl> is set to true (default is false), each active connection should only be made available after the configured number of seconds, even if the backend call returns earlier than that
 - <StrictOnTtl> = true is generally not needed

Code: LimitBackendConnections

```
1 <ConcurrentRatelimit async="false" continueOnError="false" enabled="true" name="LimitBackendConnections">
2   <AllowConnections count="10" ttl="5"/>
3   <Distributed>false</Distributed>
4   <StrictOnTtl>false</StrictOnTtl>
5   <TargetIdentifier name="counter1"/>
6 </ConcurrentRatelimit>
```

Access Control (IP Filtering)

- The **AccessControl** policy can be used to whitelist (allow) and blacklist (deny) access based upon specific IP addresses and/or IP ranges
- The policy should generally be placed early in the request preflow to avoid unnecessary processing when incoming IP address is not accepted
- IP filtering should not be the only protection against illegal access
 - A blocked IP can make a successful request using a proxy
 - It is also possible to spoof an IP address
- If X-Forwarded-For header exists in incoming request, it is used to determine the IP address
- If no X-Forwarded-For header is retrieved, the actual IP address of the connection is used

Access Control (IP Filtering)

Code: ACL

```
1 <AccessControl name="ACL">
2   <IPRules noRuleMatchAction="ALLOW">
3     <MatchRule action="ALLOW">
4       <SourceAddress mask="32">10.10.10.20</SourceAddress>
5     </MatchRule>
6     <MatchRule action="DENY">
7       <SourceAddress mask="24">10.10.10.00</SourceAddress>
8     </MatchRule>
9   </IPRules>
10 </AccessControl>
```

- <IPRules> noRuleMatchAction attribute specifies the default action if the IP address doesn't match any <MatchRule>
- The <MatchRule> list is evaluated in order from top to bottom
 - Each <SourceAddress> entry is checked for a match
 - <SourceAddress> mask attribute specifies the net mask length
 - If <SourceAddress> matches, the <MatchRule> action is chosen
 - If ALLOW, flow continues
 - If DENY, a fault is raised and 403 Forbidden is returned
- Example above denies all traffic with IP 10.10.10.* except 10.10.10.20, with other traffic allowed

Thank you

apigee

