



# WSO2 API Manager 4.2.0 Developer Fundamentals

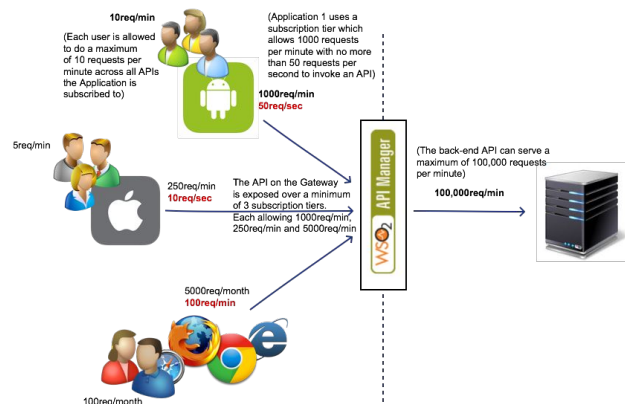
Rate Limiting Policies



WSO2 Training

CC by 4.0

## Rate Limiting Policies

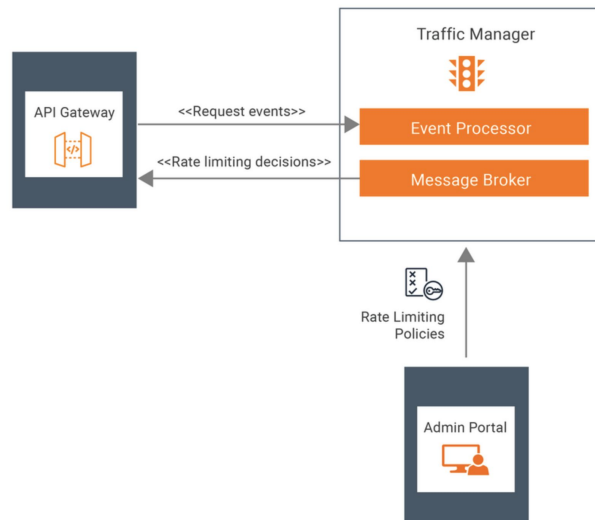


Link - [Working with Rate Limiting](#)

### Why is Rate limiting important ?

- Rate limiting is a key functionality of API Management
- API backends can't serve unlimited requests
- Rate limiting plays an important part in monetizing an API
- Ensuring that your business APIs can be exposed to the public with a regulation and ensure that each user gets his fair share
- Shaping the traffic as your business changes
- Allow more traffic to your business partners and distribute the rest of it among other users
- Control Unusual API invocation of End Users

## Introducing Central Policy Server : Traffic Manager



The Traffic Manager processes data of each API request and makes rate limiting decisions based on the applicability of available rate limiting policies.

Rate limiting decisions are made by the Siddhi runtime and published to the JMS topic.

The gateways subscribed to this JMS topic get instantly notified about the decisions.

Other features :

- Includes powerful Siddhi runtime based decision engine
- Extensible and flexible to define advanced rules based on API properties such as headers, users, JWT Claims, etc...
- Flexibility to design rate limiting policies based on both request count and bandwidth
- Supporting Instantaneous request blocking based on User, IP Address, Application and API

## Maximum Backend Throughput Limit

The screenshot displays the API Manager interface for the 'PizzaShackAPI : 1.0.0' (PUBLISHED State). The left sidebar shows the 'Runtime' configuration selected under 'Develop'. The main panel is titled 'Runtime Configurations' and is divided into 'Request' and 'Backend' sections. The 'Backend' section is highlighted with a red box, showing the 'Backend Throughput' configuration. The 'Maximum Throughput' is set to 'Specify', with 'Max Production TPS' and 'Max Sandbox TPS' both set to 'TPS'. The 'Message Mediation' is set to 'none'.

Link - [Maximum Backend Throughput Limit](#)

The Maximum backend throughput setting limits the total number of calls a particular API in the API Manager is allowed to make to the backend.

We can set the maximum throughput under rate limiting settings when publishing the API through UI.

Alternatively, you can go to the synapse configuration of the API in `<APIM_HOME>/repository/deployment/server/synapse-configs` and specify the maximum backend throughput there\*.

**\* Note: The changes made in the synapse file will be overridden if the API is republished.**

## Levels of Rate Limiting

Subscription Level Rate Limiting  
(API Publisher)

Subscription Level Rate Limiting  
(API Subscriber)

Application Level Rate Limiting  
(Application Developer)

Advanced Level Rate Limiting  
(API Publisher)

☒ Request Count ☐ Request Bandwidth

Request Count \*

Number of requests allowed

Unit Time

Time configuration

Minute(s) ▼



### Why do we need Rate Limiting?

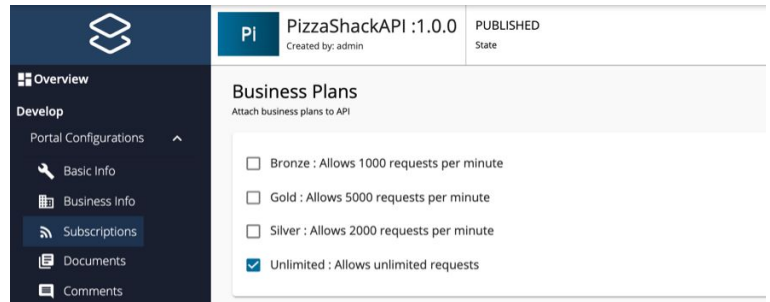
- To protect your APIs from common types of security attacks such as denial of service (DOS)
- To regulate traffic according to infrastructure availability
- To make an API, application or a resource available to a consumer at different levels of service, usually for monetization purpose

It is possible to rate limit requests for each tier based on the request count per unit time or the amount of data (bandwidth) transferred through the Gateway per unit time as shown in the last diagram.

## Subscription Rate Limiting Policies (API Publisher)

Selecting the subscription tier/s when publishing the API

- Bronze: 1000 requests per minute
- Silver: 2000 requests per minute
- Gold: 5000 requests per minute
- Unlimited: Allows unlimited access



We can add this subscription level rate limiting to the API when managing the API to publish in the Publisher portal.

## Subscription Rate Limiting Policies (API Publisher)

### Rate Limiting (Burst Control)

- Define combined tiers
- Control the usage of APIs within smaller time durations
- Protect the backend from sudden request bursts



With rate limiting, you can define tiers with a combination of, for example, a 1000 requests per day and 10 requests per second. Users are then rate limited at two layers. Enforcing a rate limit protects the backend from sudden request bursts and controls the usage at a subscription and API level.

For instance, if there's a subscription level policy enforced over a long period, you may not want users to consume the entire quota within a short time span. Sudden spikes in usage or attacks from users can also be handled via rate limiting. You can define a spike arrest policy when the subscription level tier is created.

For each subscription level throttle key, a WS policy is created on demand. The request count is calculated and rate limiting occurs at the node level. If you are using a clustered deployment, the counters are replicated across the cluster.

## Adding a New Subscription Rate Limiting Policy

Admin Portal : <https://localhost:9443/admin>

[illegible][illegible]

Link - [Subscription Rate Limiting Policy](#)

We can specify the bandwidth per unit time by adding a new subscription level tier. This can be done through the Admin Portal (Rate Limiting Policies → Subscription Policies → Add Policy)



To make changes in the rate limiting configurations, set the `enable_advanced_throttling` parameter in the `<APIM_HOME>/repository/conf/deployment.toml` file. This parameter is set to true by default. If you set it to false, you only see the available tiers.

```
[apim.throttling]
enable_advanced_throttling = true
```
















## Subscription Rate Limiting (API Subscriber)

Selecting the Rate Limiting tier when subscribing to the API

**Subscribe APIs**   

Displaying all APIs

Name	Version	Subscription Status
customer-info	1.0.0	Silver  
CustomerLeasing	1.0.0	Unlimited  
leasing	1.0.0	Silver  
PizzaShackAPI	1.0.0	Unlimited  
StarwarsAPI	1.0.0	Unlimited  

Rows per page: 10  1-5 of 5  

After subscription-level rate limiting tiers are set and the API is published, at subscription time, the consumers of the API can log in to the API Developer Portal and select which tier (out of those enabled for subscribers) they are interested in when subscribing to the API.

## Advanced Rate Limiting (API Publisher)

Applicable in API-Level as well as Resource Level rate limiting when publishing the API

- 10KPerMin: 10,000 requests per minute
- 20KPerMin: 20,000 requests per minute
- 50KPerMin: 50,000 requests per minute
- Unlimited: Unlimited access

Resources

Operations Configuration<sup>Ⓢ</sup>

Rate limiting level

☐ API Level ☒ Operation Level

Resources

Operations Configuration<sup>Ⓢ</sup>

Rate limiting level

☒ API Level ☐ Operation Level

HTTP verbs

10KPerMin  
20KPerMin  
50KPerMin  
Unlimited

Order

Summary & Description

Create a new Order

Operation Configuration<sup>Ⓢ</sup>

Rate limiting level

☒ API Level ☐ Operation Level

Operation verbs

10KPerMin  
20KPerMin  
50KPerMin  
Unlimited

Operation verbs

10KPerMin  
20KPerMin  
50KPerMin  
Unlimited

Parameters

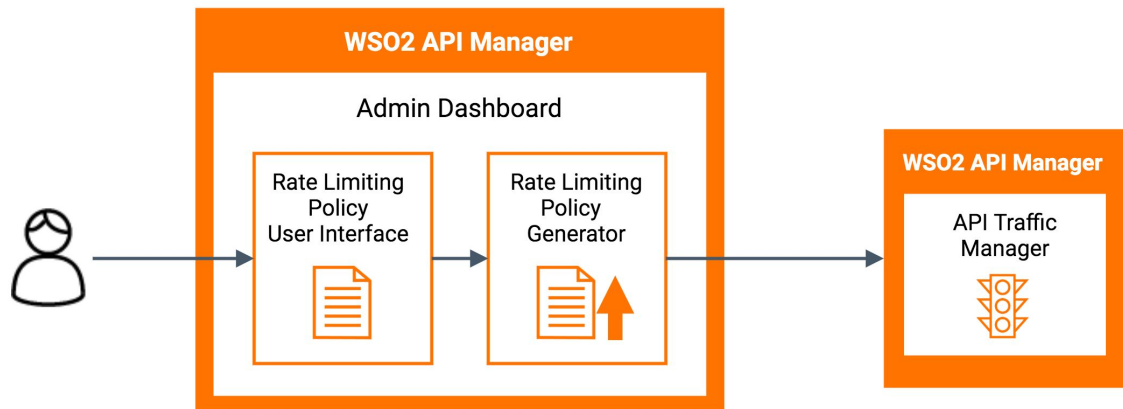
Parameter Type:  Parameter Name:  Data Type:  Required: ☐ (Select at least one required parameter)

Parameter Type	Parameter Name	Data Type	Required	Actions
Body	application/json	Text	Yes	

Advanced rate limiting policies are applicable on the API and also for Resources. They are defined when managing the API in the Publisher portal. Resource-level rate limiting tiers are set to HTTP verbs of an API's resources as shown.

## Adding New Advanced Rate Limiting Policy

Admin Portal : <https://localhost:9443/admin>



We can add new advanced Rate limiting policies through the Admin Portal by adding a new advanced rate limiting tier.

We can also set Conditional groups for the tier, such as IP conditions, header conditions, etc.

# Adding New Advanced Rate Limiting Policy

Admin Portal : <https://localhost:9443/admin>

**General Details**  
Provide name and description of the policy. The policy can be referred back the name.

Name of the throttle policy

Description of the throttle policy

**Default Limits**  
Request Count and Request Bandwidth are the two options for default limits. You can use the option according to your requirement.

Default Limit Option

☒ Request Count ☐ Request Bandwidth

Request Count

Unit Time

Minutes

**Conditional groups**  
To add throttling rules with different parameters based on IP, Header, Query Params, and JWT Claim conditions, click Add Conditional Group.

**Warning**  
Publishing Query Params, Header Data and JWT token isn't configured. If a policy configured with any of these conditions, it won't be applied.

**Condition Policies**

**IP Condition Policy**  
This configuration is used to throttle by IP address.

**Header Condition Policy**  
This configuration is used to throttle based on headers.

**Query Param Condition Policy**  
This configuration is used to throttle based on query parameters.

**JWT Condition Policy**  
This configuration is used to define JWT claims conditions.

**Default Limit Option**

☒ Request Count ☐ Request Bandwidth

Request Count

Number of requests allowed

Unit Time

Minutes

**Description**

Description of this group

Link - [Advanced Rate Limiting Policy](#)

Some Conditional Groups :

- IP Condition** - Allows you to set a rate limit for a specific IP address or a range of IP addresses.
- Header Condition** - Allows you to set a rate limit to specific headers and parameters.
- Query Param Condition** - Allows you to set a rate limit to specific query parameters.
- JWT Claim Condition** - Allows you to set a rate limit to specific claims.

## Rate Limiting Policy Definition

```
@Plan:name('carbon.super_sub_Gold')
```

Policy Name

```
@Plan:description('ExecutionPlan for sub_gold')
```

Policy Description

```
@Import('org.wso2.throttle.processed.request.stream:1.0.0')
define stream RequestStream (messageID string, appKey string, appTier string,
subscriptionKey string, apiKey string, apiTier string, subscriptionTier string,
resourceKey string, resourceTier string, userID string, apiContext string, apiVersion
string, appTenant string, apiTenant string, appId string, apiName string,
propertiesMap string);
```

Request Stream Definition

```
@Export('org.wso2.throttle.globalThrottle.stream:1.0.0')
define stream GlobalThrottleStream (throttleKey string, isThrottled bool,
expiryTimeStamp long);
```

Global Throttle Stream Definition

<http://wso2.com/library/articles/2016/09/article-introducing-wso2-api-managers-throttling-implementation-architecture/>

Once the Traffic Manager receives a request, Siddhi runtime will process the deployed policies in the traffic manager instance.

These are the sections of rate limiting policy definition.

- Policy name: name of the rate limiting policy
- Policy description: description of rate limiting policy
- Request stream definition: mapping between incoming data from rate limiting request stream to a Siddhi stream named RequestStream that can be understood by Siddhi runtime.
- Global throttle stream definition: mapping of Siddhi output stream named GlobalThrottleStream to outgoing global throttle stream.

## Rate Limiting Policy Definition

```
FROM RequestStream
SELECT messageID, ( apiTenant == 'carbon.super' and subscriptionTier == 'Gold') AS
isEligible, subscriptionKey AS throttleKey
INSERT INTO EligibilityStream;
```

Eligibility Query

```
FROM EligibilityStream[isEligible==true]#throttler:timeBatch(1 min, 0)
select throttleKey, (count(messageID) >= 5000) as isThrottled, expiryTimeStamp
group by throttleKey
INSERT ALL EVENTS into ResultStream;
```

Check Rate Limit State

```
from ResultStream#throttler:emitOnStateChange(throttleKey, isThrottled)
select *
insert into GlobalThrottleStream;
```

Notify Rate Limiting Decision

<http://wso2.com/library/articles/2016/09/article-introducing-wso2-api-managers-throttling-implementation-architecture/>



These are the sections of rate limiting policy definition.

- Eligibility query: this query will decide whether the policy will be further processed or not.
- Check Rate limit state: query that checks whether the current request is rate limited or not.
- Notify rate limiting decision: the operation that sends the rate limiting decision to the JMS Topic. Query in this section sends a message to the JMS Topic if there is a change in the Rate limiting state. Moreover, if the current request is rate limited, then the query will send a message to the JMS Topic. If the request is rate limited, gateways won't publish any data to the traffic manager until the rate limit time interval is over.

# Application Rate Limiting (Application Developer)

## Defined when creating the application in the API Developer Portal

### Create an application

Create an application providing name and quota parameters. Description is optional.  
Required fields are marked with an asterisk (\*)

Application Name \*

PizzaApp

Enter a name to identify the Application. You will be able to pick this application when subscribing to APIs

Shared Quota for Application Tokens \*

10PerMin

Assign API request quota per access token. Allocated quota will be shared among all the subscribed APIs of the application.

Application Description

(512) characters remaining

Application Groups

Type a group and enter

SAVE

CANCEL

### Subscribe

Subscribe to an application and generate credentials

Application

PizzaApp

Select an Application to subscribe

Business Plan

Unlimited

Available Policies - Unlimited

SUBSCRIBE

An application is available to a consumer at different levels of service. For example, if you have infrastructure limitations in facilitating more than a certain number of requests to an application at a time, the rate limiting tiers can be set accordingly so that the application can have a maximum number of requests within a defined time.

## Adding New Application Rate Limiting Policy (Per Token Quota)

APIM ADMIN PORTAL

Dashboard

Rate Limiting Policies

Advanced Policies

Application Policies

Subscription Policies

Custom Policies

Proxy Policies

Gateway

API Categories

Key Managers

Tools

User Creation

Application Creation

Subscription Creation

Subscription Update

Application Registration

API State Change

Rate Limiting Policies / Application Policies

Application Rate Limiting Policies

Search by Application Policy name

ADD POLICY

Name	Quota	Unit Time	Actions
50PerMin	50	1 min	<a href="#">Edit</a> <a href="#">Delete</a>
20PerMin	20	1 min	<a href="#">Edit</a> <a href="#">Delete</a>
10PerMin	10	1 min	<a href="#">Edit</a> <a href="#">Delete</a>

Rows per page: 10 1-3 of 3

Add Policy

General Details

Name \*

100PerMin

Name of the throttle policy

Description

Allows 100 requests per minute

Description of the throttle policy

Quota Limits

☒ Request Count ☐ Request Bandwidth

Request Count \*

100

Number of requests allowed

Unit Time

1

Minute(s)

Time configuration

Cancel

Save

Admin Portal : <https://localhost:9443/admin>

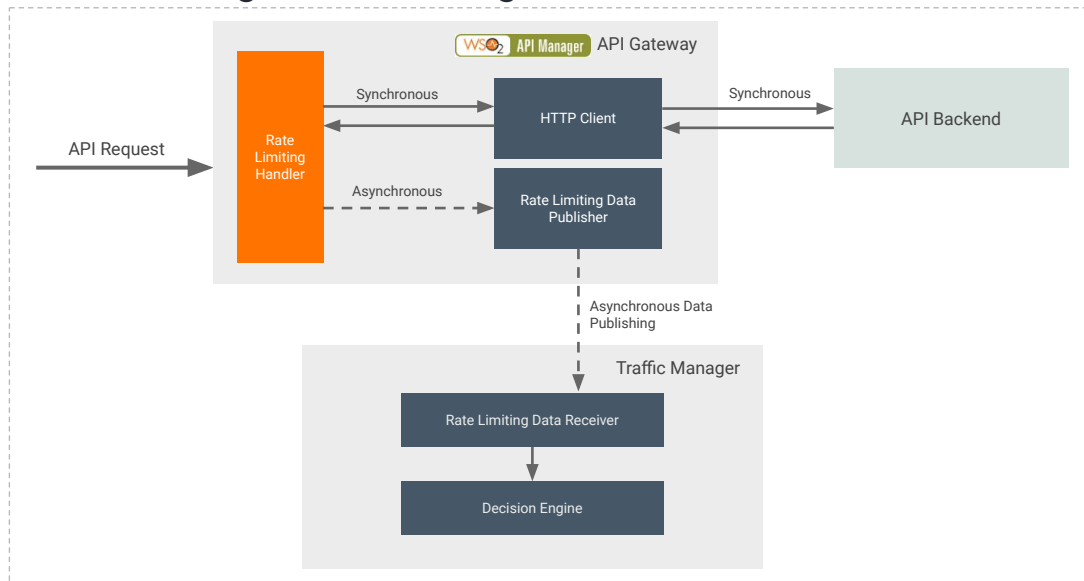
Link - [Application Rate Limiting Policy](#)



Application-level rate limiting policies are applicable per access token generated for an application.



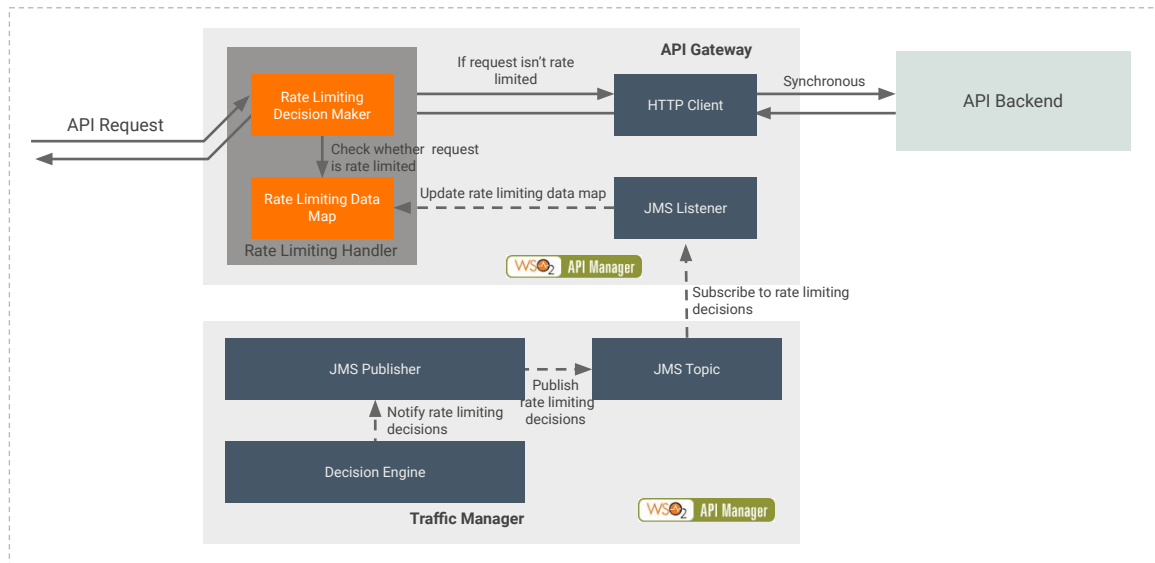
## Rate Limiting Data Publishing



Traffic Manager has the responsibility of making rate limiting decisions

- Data required to make rate limiting decisions need to be published to the Traffic Manager
- Each Gateway in a deployment asynchronously publishes data required to make rate limiting decision for every API request to the Traffic Manager

## Policy Evaluation and Notifications



- Traffic Manager has the responsibility of making rate limiting decisions
- The Siddhi Runtime in Traffic Manager processes events from gateways
- Policies deployed in traffic manager are executed on each event
- An event that triggers a condition in a policy will be notified to gateways through a JMS topic
- Each gateway maintains a rate limiting data map to check whether a request is within the allowed quota.
- Gateways update the rate limiting data map from the JMS Topic which is notified by the Traffic Manager

## Custom Rate Limiting and Denying Requests

### Custom Rate Limiting Policy - Define Policy

Name: CustomRule

Name of the throttle policy

Description: This is a custom Policy

Description of the throttle policy

Key Template: \$userId:\$apiContext:\$apiVersion

The specific combination of attributes being checked in the policy need to be defined as the key template. Allowed values are: \$userId, \$apiContext, \$apiVersion, \$resource, \$appTenant, \$appTenant, \$appId, \$scriptId

Eg: \$userId:\$apiContext:\$apiVersion

Siddhi Query:

The following sample query will allow 5 requests per minute for an Admin user.  
Key Template: \$userId

```
1 FROM
2 RequestStream
3 SELECT
4   userId,
5   (userId == 'admin@carbon.super') AS isEligible,
6   throttleKey('admin@carbon.super', '') AS throttleKey
7 INSERT INTO
8   EligibilityStream;
9 FROM
10  EligibilityStream [isEligible=true] #throttle:timeBatch(1 min) SELECT throttleKey, (count(userId) > 5) AS isThrottled
```

Buttons: Add, Cancel

Select Item to Deny

Condition Type

☒ API Context ☐ Application ☐ IP Address ☐ IP Range ☐ User

Value \*

Format: \${context}

Eg: /test/1.0.0

☒ Enable Condition

Buttons: Cancel, Deny

Link - [Custom Throttling](#)

Custom policies allow users to write custom Siddhi queries. Depending on the data received from gateway API calls, these policies will be executed and will send rate limiting decisions through JMS topic to gateways. Users may write Siddhi queries to limit activities of certain users based on their API invocation data that comes to the traffic manager. The throttle key is an important part of defining the policy. The key template definition available in the policy configuration will be sent to the gateway and then replaced with actual values of the request and will decide if the current request is rate limited or not. These policies extend the flexibility of rate limiting by allowing users to write custom rate limiting policies.

Deny policy conditions provide functionality to:

- instantly block a user who invokes APIs by username
- block API calls coming from a specific IP address by specifying IP
- block an API by API context
- block API calls coming from an application by application name.

## GraphQL Query Depth Limit

```
query{  
  allFilms{  
    id  
    Species{  
      id  
      films{  
        title  
        planets{  
          id  
          residents{  
            eyeColor  
            films{  
              director  
              producers  
            }  
          }  
        }  
      }  
    }  
  }  
}  
# depth value of query : 7
```

The screenshot displays the WSO2 API Manager console interface. On the left, a sidebar menu shows navigation options like 'Rate Limiting Policies', 'Subscription Policies', and 'APIs'. The main panel is titled 'Subscription Rate Limiting Policy - Create new'. It contains several configuration sections: 'General Details' (Policy Name: 'Policy1'), 'Quota Limits' (Request Count: 2000, Request Period: 1 minute), 'Rate Control (Rate Limiting)' (Request Rate: 100 requests per second), and 'GraphQL' (Maximum Depth: 5, highlighted with a red box). Below this, the 'Methods' section shows 'Max Subscriptions'. At the bottom, a 'GraphQL' tab is active, showing a query execution result. The query is a complex nested query (the same one as in the image). The result is a JSON object with a 'status' of 'failed' and a 'message' indicating 'QUERY TOO DEEP'. The error message is highlighted with a red box.

Since GraphQL schemas often have circular relationships, the depth can grow without bounds. This relationship allows a malicious user to construct an expensive nested query. WSO2 API-Manager introduces GraphQL **Query Depth Limitation** to avoid such cyclic relationships.

The request is allowed or rejected based on the depth of the requested query, and the maximum depth value which has been configured according to the corresponding subscription policy of the API.

## GraphQL Query Complexity

```
query {  
  allFilms{  
    id          # complexity 1  
    title       # complexity 3  
    planets {  
      climate   # complexity 2  
    }  
  }  
}  
  
# total complexity = 1 + 2 + 1 + 3 + 1 = 8
```

WSO2 API-Manager PUBLISHER

Subscription Rate Limiting Policy - Create new

General Details

Policy Name:

Policy Type:

Policy Description:

Query Limits

Request Count:  Request Bandwidth:  Exceed Based on (select one):

Request Count:

Request Bandwidth:

Exceed Based on (select one):

Burst Control (Rate Limiting)


Request Rate:  Requests:

Complexity:  (highlighted with a red box)

Webhooks

Webhook URL:

Webhook Method:

GraphiQL  Prettyfy History Explorer Complexity Analysis

```
1 query{  
2   allHumans(first:6){  
3     id  
4     name  
5     friendsConnection(first:5){  
6       totalCount  
7       friends{  
8         name  
9       }  
10    }  
11  }  
12 }
```

```
{  
  "fault": {  
    "code": 900913,  
    "message": "QUERY TOO COMPLEX",  
    "description": "[maximum query complexity exceeded]"  
  }  
}
```

Often, limiting only the depth of a query is not sufficient to protect a GraphQL service from complex queries. The reason for this is that some fields in a GraphQL schema are more costly to compute than others. WSO2 API-Manager introduces

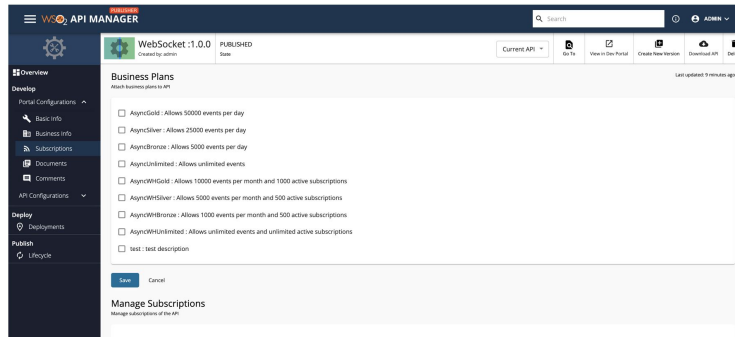
**“Query Complexity Limitation”** to address such cases.

With this strategy, a request allowed or rejected based on the complexity of the query, and the configured max complexity value of the subscription policy for the corresponding API.

Here we introduced the **complexity values for each of the Fields** in the schema. That **describes the computation cost of resolving the particular field**. This is configurable via the runtime configurations tab of the publisher portal.

## Rate Limiting for Streaming APIs

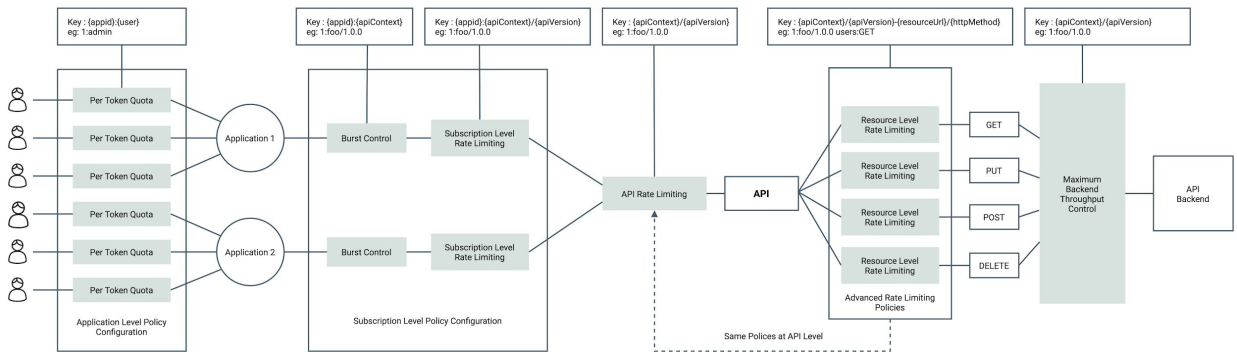
- Count-based Rate Limiting Policy
- Time-based Rate Limiting Policy
- Count-time Hybrid Rate Limiting Policy



Link - [Rate Limiting for Streaming APIs](#)



## Rate Limiting Policies Applicable at Different Levels



<http://wso2.com/library/articles/2016/09/article-introducing-wso2-api-managers-throttling-implementation-architecture/>



This is how the above discussed policies are applied in different levels of rate limiting.

**Let's try it out!**

Working with Rate Limiting Policies

