Best practices for avoiding rate limiting

ON THIS PAGE

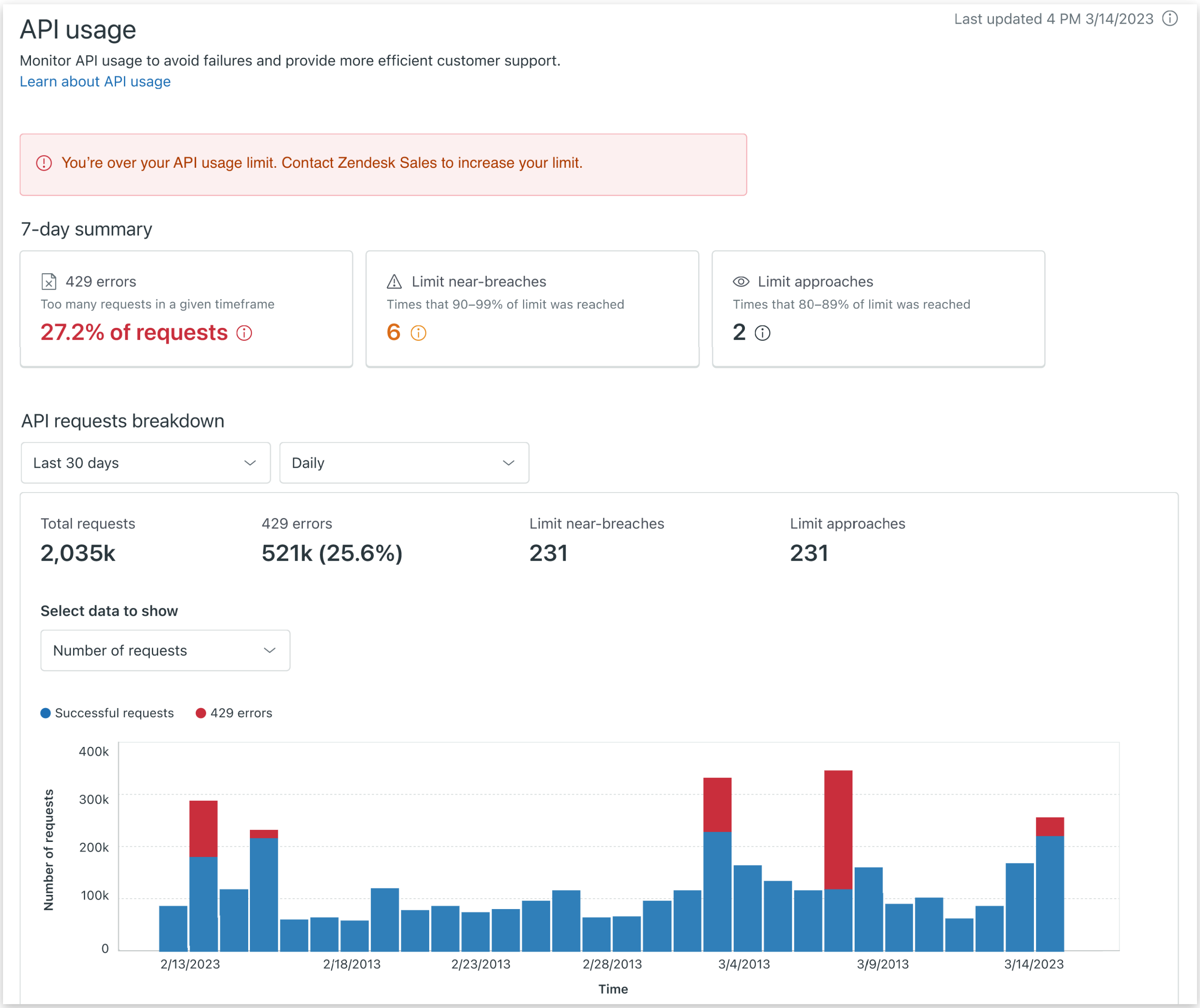
If you make a lot of Zendesk API requests in a short amount of time, you may bump into the API rate limit. When you reach the limit, the API stops processing any more requests until a certain amount of time has passed.

The rate limits for the Zendesk APIs are outlined in Rate limits in the API reference. Get familiar with the limits before starting your project.

This article covers the following best practices for avoiding rate limiting.

Monitoring API activity against your rate limit

You can use the API usage dashboard in Admin Center to monitor your API activity against your rate limit. See Managing API usage in your Zendesk account in Zendesk help.


API usage

Last updated 4 PM 3/14/2023

Monitor API usage to avoid failures and provide more efficient customer support.

Learn about API usage

You're over your API usage limit. Contact Zendesk Sales to increase your limit.

7-day summary

429 errors	Limit near-breaches	Limit approaches
Too many requests in a given timeframe	Times that 90–99% of limit was reached	Times that 80–89% of limit was reached
27.2% of requests	6	2

API requests breakdown

Last 30 days — Daily

Total requests	429 errors	Limit near-breaches	Limit approaches
2,035k	521k (25.6%)	231	231

Select data to show

Number of requests

The following response headers contain the account's rate limit and the number of requests remaining for the current minute:

```
X-Rate-Limit: 700
X-Rate-Limit-Remaining: 699
```

The responses from Ticketing API list endpoints such as List Tickets or Search Users provide the following headers:

```
x-rate-limit: 700
ratelimit-limit: 700
x-rate-limit-remaining: 699
ratelimit-remaining: 699
ratelimit-reset: 41
zendesk-ratelimit-tickets-index: total=100; remaining=99; resets=41
```

The `zendesk-ratelimit-tickets-index` header is only available in list ticket endpoints. For more information, see List Tickets limits.

Using rate limit headers in your application

For each Ticketing API call you make, Zendesk includes the above account-wide rate limit information in the response headers. To utilize this information, ensure that your application properly reads and interprets these headers. Always check the account wide limits header first. If you receive a 429 response, then look for endpoint-specific headers.

In the examples below, the headers are processed to check for an endpoint rate limit. **Note**: These examples are for reference only.

Python

```python
import requests
import time

def call_zendesk_api():
    url = "https://subdomain.zendesk.com/api/v2/tickets"
    headers = {"Authorization": "Bearer YOUR_ACCESS_TOKEN"}
    response = requests.get(url, headers=headers)
    should_continue = handle_rate_limits(response)
    if should_continue:
        # Process the API response
        # Your code here...

def handle_rate_limits(response):
    account_limit = response.headers.get("ratelimit-remaining")
    endpoint_limit = response.headers.get("Zendesk-RateLimit-Endpoint")
    accountLimitResetSeconds = response.headers.get("ratelimit-reset")

    if account_limit:
        account_remaining = int(account_limit)
        if account_remaining > 0:
            if endpoint_limit:
                endpoint_remaining = int(endpoint_limit.split(";")[1].split("=")[1])
                if endpoint_remaining > 0:
                    return True
                else:
                    endpointLimitResetSeconds = int(endpoint_limit.split(";")[2].split("=")[1])
                    # Endpoint-specific limit exceeded; stop making more calls
                    handle_limit_exceeded(endpointLimitResetSeconds)
            else:
                # No endpoint-specific limit
                return True
        else:
            # Account-wide limit exceeded
            handle_limit_exceeded(accountLimitResetSeconds)
    return False

def handle_limit_exceeded(limitHeaderResetTime):
    reset_time = 60  # default time if reset time is not available from the header
    if limitHeaderResetTime:
        reset_time = limitHeaderResetTime
    wait_time = reset_time - time.time() + 1  # Add 1 second buffer
    print(f"Rate limit exceeded for {limit_header}. Waiting for {wait_time} seconds...")
    time.sleep(wait_time)
```

Javascript

```javascript
const axios = require('axios');

async function callZendeskAPI() {
    const url = "https://subdomain.zendesk.com/api/v2/tickets";
    const headers = {"Authorization": "Bearer YOUR_ACCESS_TOKEN"};
    try {
        const response = await axios.get(url, { headers });
        const shouldContinue = handleRateLimits(response);
        if (shouldContinue) {
            // Process the API response
            // Your code here...
        }
    } catch (error) {
        // Handle other errors
        console.error(error);
    }
}

function handleRateLimits(response) {
    const accountLimit = response.headers["ratelimit-remaining"];
    const endpointLimit = response.headers["Zendesk-RateLimit-endpoint"];
    const accountLimitResetSeconds = response.headers["ratelimit-reset"]

    if (accountLimit) {
        const accountRemaining = parseInt(accountLimit);
        if (accountRemaining > 0) {
            if (endpointLimit) {
                const endpointRemaining = parseInt(endpointLimit.split(";")[1].split("=")[1]);
                if (endpointRemaining > 0) {
                    return true;
                } else {
                    const endpointLimitResetSeconds = parseInt(endpointLimit.split(";")[2].split("=")[1]);
                    // Endpoint-specific limit exceeded
                    handleLimitExceeded(endpointLimitResetSeconds);
                }
            } else {
                // No endpoint-specific limit
                return true;
            }
        } else {
            // Account-wide limit exceeded
            handleLimitExceeded(accountLimitResetSeconds);
        }
    }
    return false;
}

async function handleLimitExceeded(limitHeaderResetTime) {
    const resetTime = limitHeaderResetTime || 60; // default to 60
    const waitTime = resetTime - Math.floor(Date.now() / 1000) + 1; // Add 1 second buffer
    console.log(`Rate limit exceeded for {limitHeader}. Waiting for ${waitTime} seconds...`);
    await new Promise(resolve => setTimeout(resolve, waitTime * 1000));
}

callZendeskAPI();
```

Both examples verify that the limit is not exceeded for each response before proceeding to the next request. If the limit is exceeded, parse the reset time from the rate limit headers, calculate the wait time until the reset, and then pause until that time. After waiting, retry the API call.

Alternatively, you can choose to check only when the response status code is 429 (rate limit exceeded).

Here are some best practices for utilizing rate limit headers:

- Regularly monitor your API usage to prevent unexpected rate limit breaches.
- Design your application to gracefully handle rate limit headers, ensuring continuous service for your users.
- Implement exponential backoff strategies to effectively manage rate limit exceeded errors.

Handling errors caused by rate limiting

If the rate limit is exceeded, the API responds with a 429 Too Many Requests status code.

It's best practice to include error handling for 429 responses in your code. If your code ignores 429 errors and keeps trying to make requests, you might start getting null errors. At that point, the error information won't be useful in diagnosing the problem.

For example, a request that bumps into the rate limit might return the following response:

```
< HTTP/1.1 429
< Server: nginx/1.4.2
< Date: Mon, 04 Nov 2013 00:18:27 GMT
< Content-Type: text/html; charset=utf-8
< Content-Length: 85
< Connection: keep-alive
< Status: 429
< Cache-Control: no-cache
< X-Zendesk-API-Version: v2
< Retry-After: 93
< X-Zendesk-Origin-Server: ****.****.***.*****.com
< X-Zendesk-User-Id: 338231444
< X-Zendesk-Request-Id: c773675d81590abad33i
<
* Connection #0 to host SUBDOMAIN.zendesk.com left intact
* Closing connection #0
* SSLv3, TLS alert, Client hello (1):
Rate limit for ticket updates exceeded, please wait before updating this
 ticket again
```

The response contains the following information:

- `Status: 429`
- `Retry-After: 93`

The 429 status code means too many requests. The `Retry-After` header specifies that you can retry the API call in 93 seconds. Your code should stop making additional API requests until enough time has passed to retry.

The following pseudo-code shows a simple way to handle rate-limit errors:

```
response = request.get(url)
if response.status equals 429:
    alert('Rate limited. Waiting to retry…')
    wait(response.headers['retry-after'])
    retry(url)
```

Node.js

The following snippet shows how you can handle rate-limit errors in JavaScript for Node.js.

```javascript
const axios = require("axios")

async function requestWithRateLimit(url, username, apiToken) {
  const tokenUsername = username + '/token';
  const password = apiToken;
  const response = await axios.get(url, {
      auth: {
        tokenUsername,
        password
    }
  })
  if (response.status === 429) {
    const secondsToWait = Number(response.headers["retry-after"])
    await new Promise(resolve => setTimeout(resolve, secondsToWait * 1000))
    return requestWithRateLimit(url, username, apiToken)
  }
  return response
}
```

Browser JavaScript

The following snippet shows how you can handle rate-limit errors in client-side JavaScript for the browser.

```javascript
async function requestWithRateLimit(url, accessToken) {
  const options = {
    method: "GET",
    headers: {
      Authorization: `Bearer ${accessToken}`
    }
  }
  const response = await fetch(url, options)
  if (response.status === 429) {
    const secondsToWait = Number(response.headers.get("retry-after"))
    await new Promise(resolve => setTimeout(resolve, secondsToWait * 1000))
    return requestWithRateLimit(url, accessToken)
  }
  return response
}
```

Note: To make an authenticated request from the browser to a Zendesk API, you must authenticate the request using an OAuth access token. For more information, see Making client-side CORS requests to the Ticketing API.

Python

The following snippet shows how you can handle rate-limit errors in Python.

```python
import requests
import time
import os

# Store the API token in an environment variable for security reasons
ZENDESK_API_TOKEN = os.getenv('ZENDESK_API_TOKEN')

def request_with_rate_limit(url, username, token):
    auth = (f'{username}/token', token)
    response = requests.get(url, auth=auth)
    if response.status_code == 429:  # Check if the rate limit has been reached
        seconds_to_wait = int(response.headers["Retry-After"])  # Get the number of seconds to wait from the Retry-After header
        time.sleep(seconds_to_wait)  # Sleep for that duration
        return request_with_rate_limit(url, username, token)  # Recursively call the function again until the rate limit is lifted
    return response

# Usage example:
# Assuming the API endpoint you're trying to access is located at `api/some_endpoint`

url = 'https://your_subdomain.zendesk.com/api/some_endpoint'
response = request_with_rate_limit(url, ZENDESK_USER_NAME, ZENDESK_API_TOKEN)  # Pass in the URL, username, and API token
if response.status_code == 200:
    # Success
    print(response.json())
else:
    # Error handling
    print(f'Error occurred: {response.status_code} - {response.text}')
```

Reducing the number of API requests

Make sure you make only the requests that you need. Here are areas to explore for reducing the number of requests:

1. Optimize your code to eliminate any unnecessary API calls.

   For example, are some requests getting data items that aren't used in your application? Are retrieved data items being put back to your Zendesk product instance with no changes made to them?
2. Cache frequently used data.

   You can cache data on the server or on the client using DOM storage. You can also save relatively static information in a database or serialize it in a file.
3. Use bulk and batch endpoints, such as Update Many Tickets, that let you update up to 100 tickets with a single API request.

Regulating the request rate

If you regularly exceed the rate limit, update your code to distribute requests more evenly over a period of time. This is known as a throttling process or a throttling controller. Regulating the request rate can be done statically or dynamically. For example, you can monitor your request rate and regulate requests when the rate approaches the rate limit.

To determine if you need to implement a throttling process, monitor your request errors. How often do you get 429 errors? Does the frequency warrant implementing a throttling process?

Frequently asked questions

- **Is there an endpoint that returns all endpoints for a resource, including the different rate limits and times to retry?**

  No, we don't provide a rate-limit endpoint.

  However, you can use the following response headers to monitor your account's rate limit and the number of requests remaining for the current minute:

  ```
  X-Rate-Limit: 700
  X-Rate-Limit-Remaining: 699
  ```

  Tickets use the following response headers:

  ```
  x-rate-limit: 700
  ratelimit-limit: 700
  x-rate-limit-remaining: 699
  ratelimit-remaining: 699
  ratelimit-reset: 41
  zendesk-ratelimit-tickets-index: total=100; remaining=99; resets=41
  ```

- **What happens when the rate limit is reached?**

  The request isn't processed and a response is sent containing a 429 response code and a `Retry-After` header. The header specifies the time in seconds that you must wait before you can try the request again.
- **Will batching calls reduce the number of API calls? How about making parallel calls?**

  No, batching calls won't reduce the number of API calls.

  However, using a bulk or batch endpoint, such as Update Many Tickets or Ticket Bulk Import, will reduce calls.