Microsoft

Service protection API limits

Dynamics 365 Finance and Operations apps

Jared Hall, Principal Program Manager
Rahul Agarwal, Principal Engineering Manager
Sourav Kumar, Principal Software Engineer
Sneh Prajapati, Principal Program Manager

Service protection API limits

Ensure consistent *availability*, *reliability*, and *performance* of the Finance and Operations service protecting against extraordinary usage

Limit types

Resource

Based on thresholds of environment resource consumption

Allows throttling prioritization

Enabled since v10.0.19 (PU43)

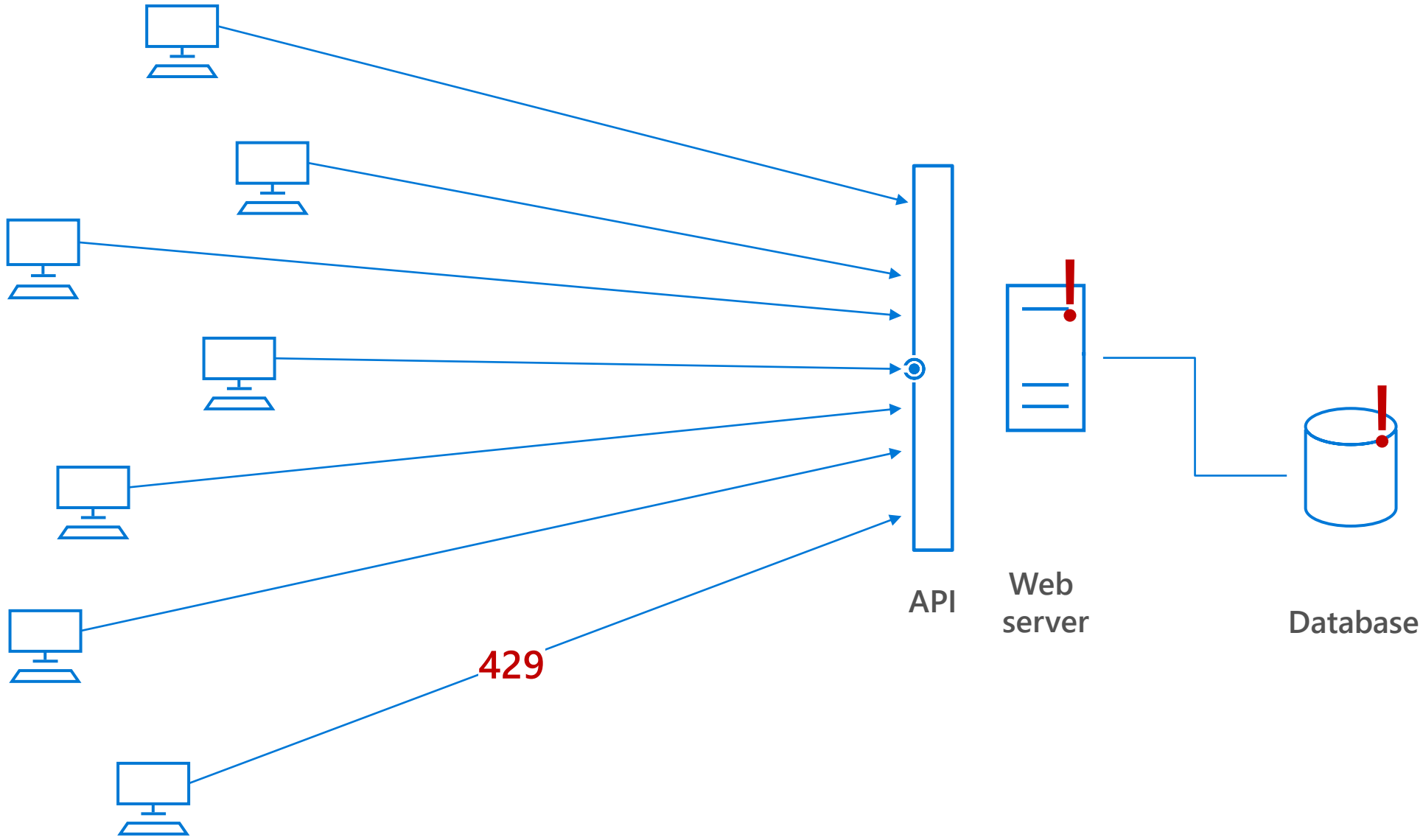
User

Protects against individual users/integrations harming system performance and availability

Three specific limits

Enabled beginning with 2022 release wave 2

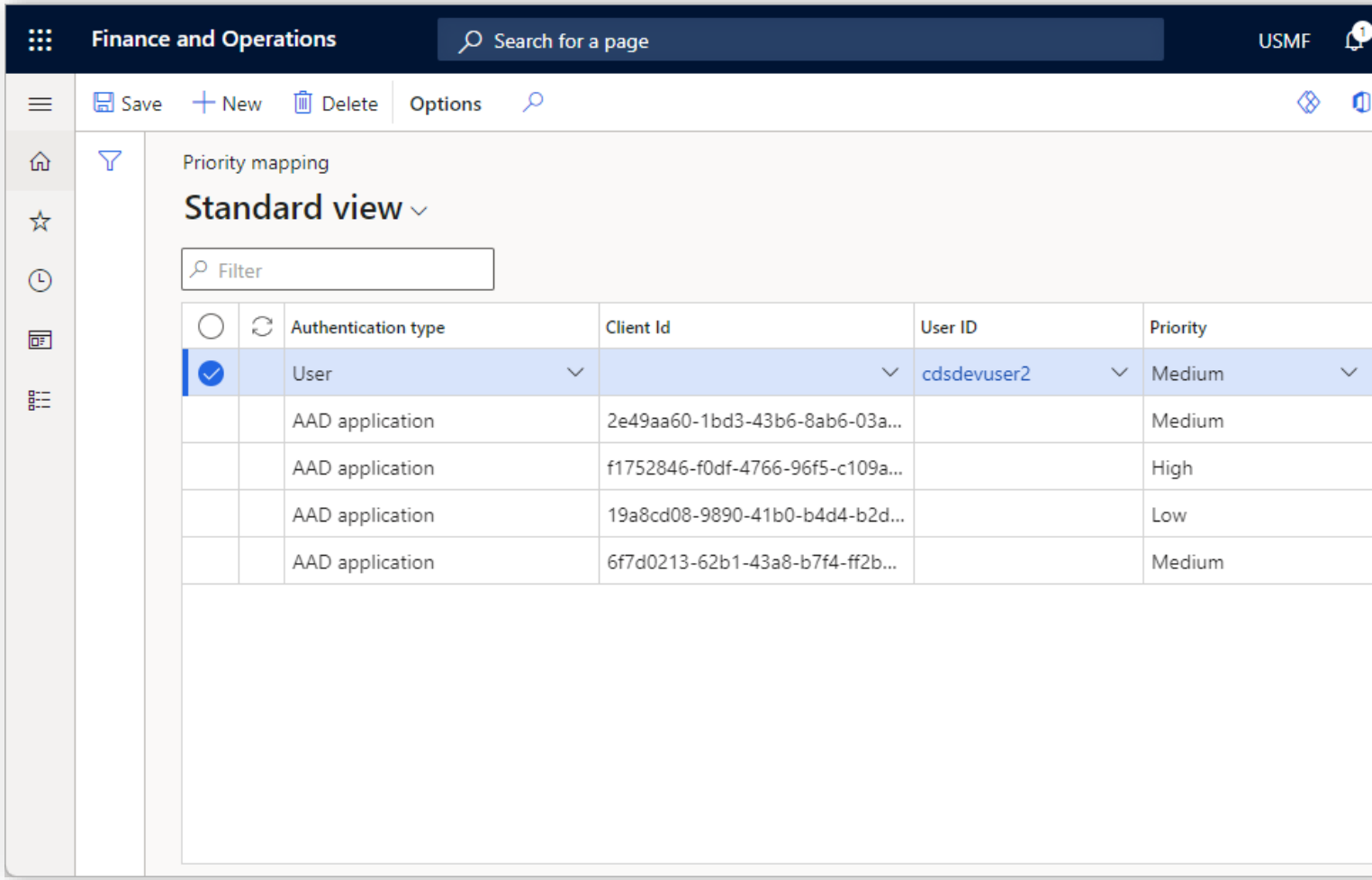
Resource-based API limits



Throttling Priority Mapping

Set prioritization for resource-based throttling

Throttling manager honors prioritization



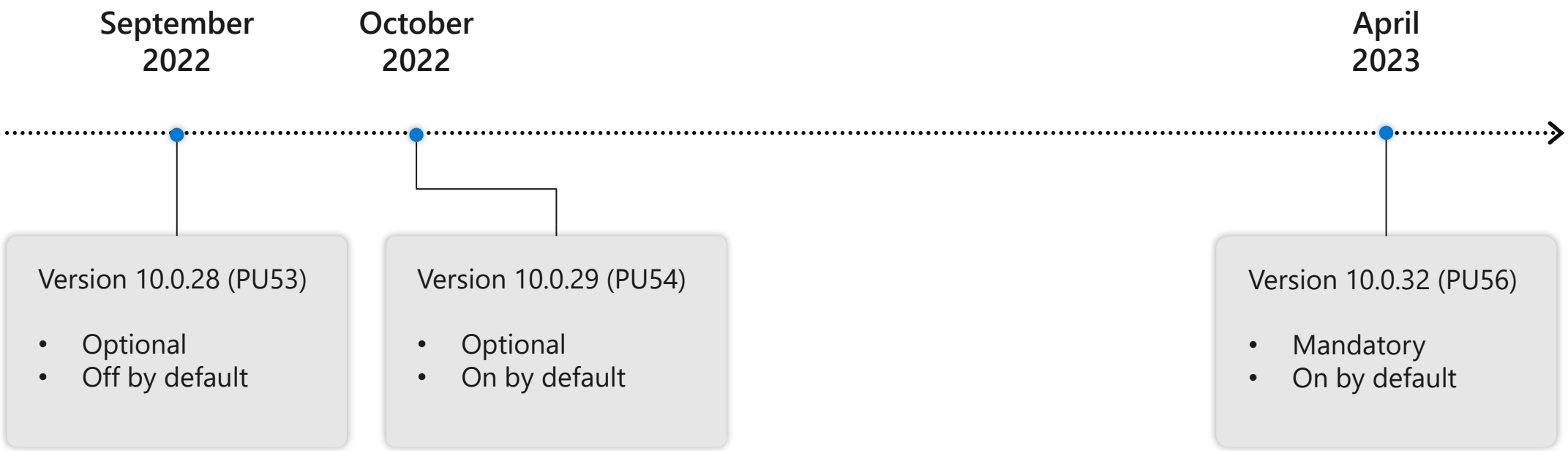
User-based service protection limits

User-based API limits are enforced *per user/service principal per web server*

Measure	Description	Service protection limit
Number of requests	The cumulative number of requests made by the user	6000 within a 5-minute sliding window
Execution time	The combined execution time of all requests made by the user	20 minutes (1200 seconds) within the 5-minute sliding window
Number of concurrent requests	The number of concurrent requests made by the user.	52

User-based API limit phased rollout

Feature Management availability



Handling API limits in your integrations

Consider integration type

Interactive clients

Anonymous users

Data integrations

Handling a 429 response

Implement retry logic for Retry-After intervals

For interactive applications:

- Display a message that the server is busy
- Provide the option to cancel the operation
- Don’t allow the user to submit additional requests until the previous request is completed

For non-interactive applications:

- Wait for the duration of the Retry-After interval before sending the request again.

Implementing retry operations

C#

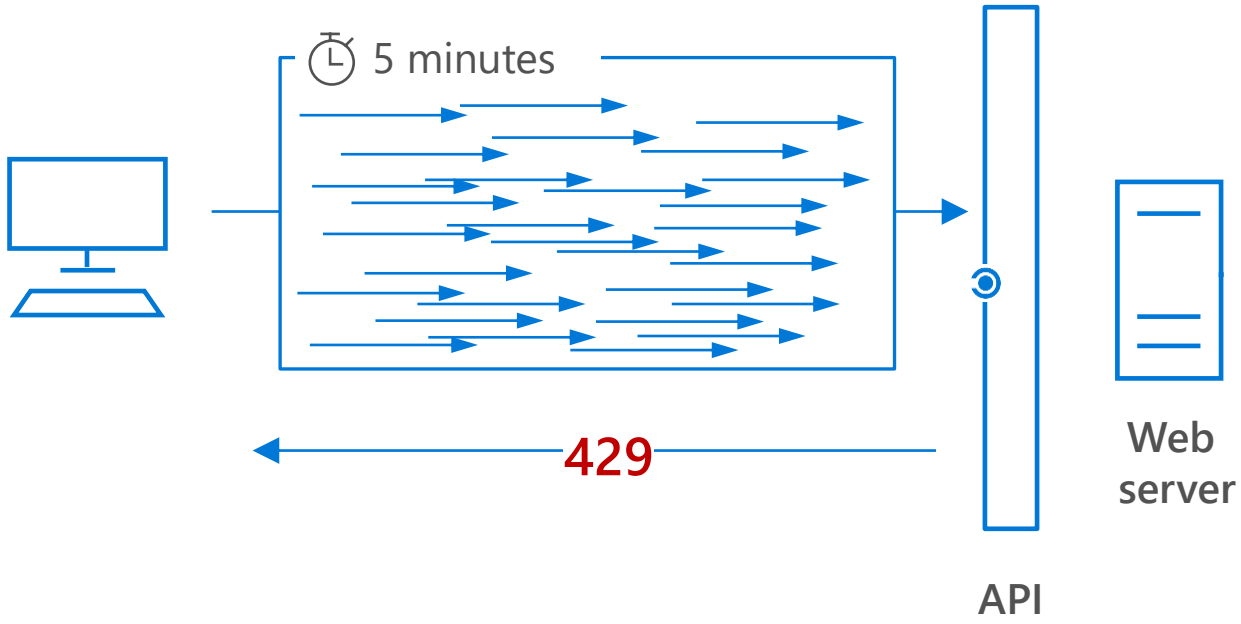
```
if (!response.IsSuccessStatusCode)
            {
                if ((int)response.StatusCode == 429)
                {
                    int seconds = 30;
                    //Try to use the Retry-After header value if it is returned.
                    if (response.Headers.Contains("Retry-After"))
                    {
                        seconds = int.Parse(response.Headers.GetValues
                         ("Retry-After").FirstOrDefault());
                    }
                    Thread.Sleep(TimeSpan.FromSeconds(seconds));

                    // Retry sending the request.
                }
            }
```

Number of requests

```
Number of requests exceeded the
limit of 6000 over the time
window of 300 seconds.
```

For interactive clients, decrease total number of records that can be selected

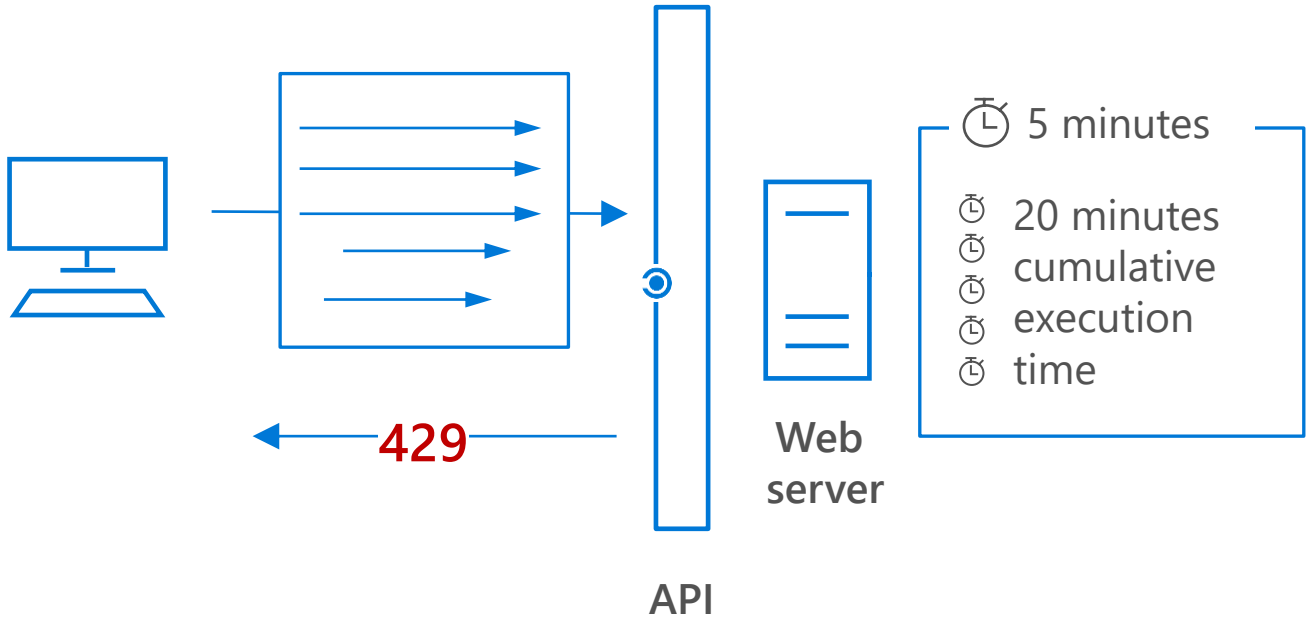
Combine selected operations into a batch (up to 5000 operations per batch)



Execution time

```
Combined execution time of
incoming requests exceeded the
limit of 1200 seconds over the
time window of 300 seconds.
```

Divide operations into smaller batches with multiple service principals

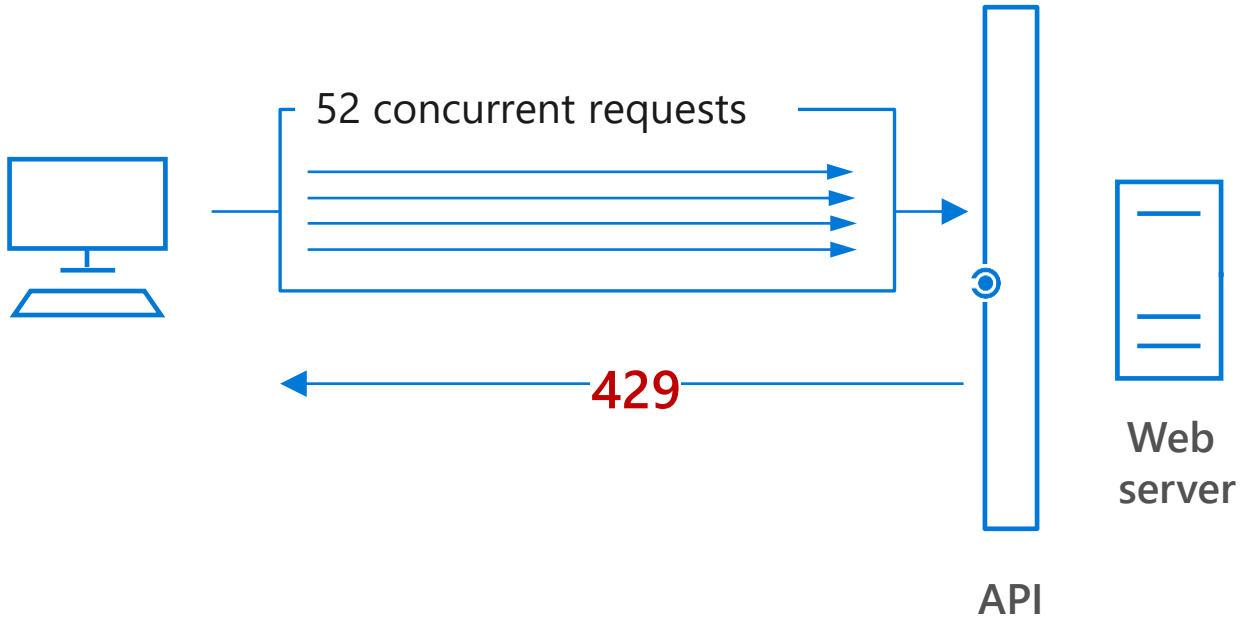


Concurrent requests

```
Number of concurrent requests
exceeded the limit of 52.
```

Sending concurrent requests can help maximize throughput

ParallelOptions.MaxDegree OfParallelism should not be greater than 52

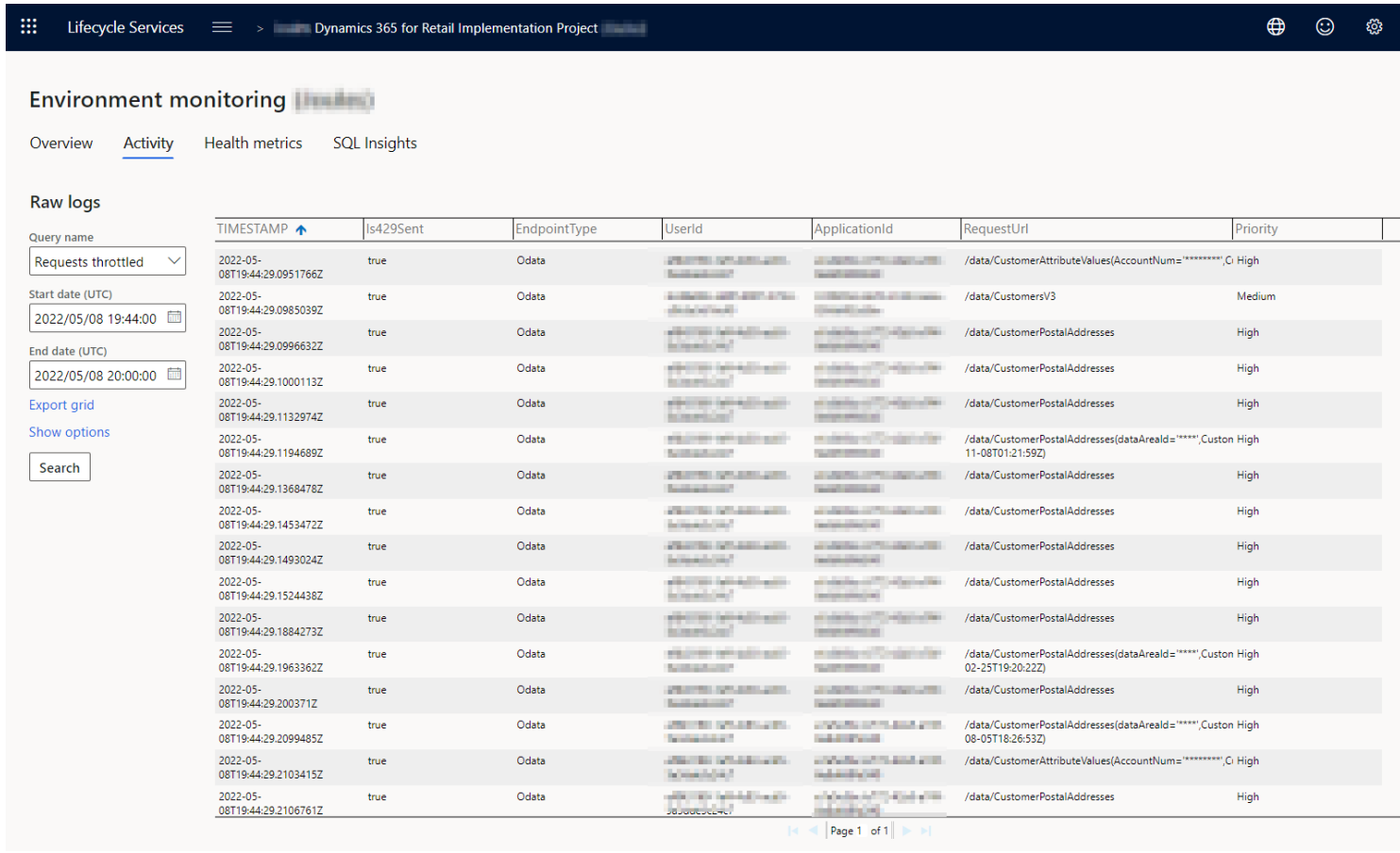


Maximizing API throughput

Monitoring API throttling

Use the **Requests throttled** query in LCS

API usage report coming soon



Maximizing throughput

- **Let the server tell you when to retry the request**
- Use multiple threads
- Distribute workloads across multiple service principals
- Avoid large batches
- Remove the affinity cookie
- Optimize your connection

Use the **Retry-After** interval provided by the server

Maximizing throughput

- Let the server tell you when to retry the request
- **Use multiple threads**
- Distribute workloads across multiple service principals
- Avoid large batches
- Remove the affinity cookie
- Optimize your connection

For individual operations that are relatively quick, depending on the nature of the data, increasing the number of threads may help achieve optimum throughput.

Maximizing throughput

Let the server tell you when to retry the request

Use multiple threads

» **Distribute workloads across multiple service principals**

Avoid large batches

Remove the affinity cookie

Optimize your connection

User-based service protection API limits are *per user per web server*.

Use multiple service principals for bulk data operations.

Use a separate service principal per integration

Maximizing throughput

Let the server tell you when to retry the request

Use multiple threads

Distribute workloads across multiple service principals

» **Avoid large batches**

Remove the affinity cookie

Optimize your connection

Limit is 5K operations per batch, but this shouldn’t be the default

Maximizing throughput

- Let the server tell you when to retry the request
- Use multiple threads
- Distribute workloads across multiple service principals
- Avoid large batches
- » **Remove the affinity cookie**
- Optimize your connection

Azure service affinity cookie ensure requests are routed to the same server, enabling the reuse of cached data.

When cookie is removed, requests are routed to any eligible server.

```
C#

HttpMessageHandler messageHandler = new OAuthMessageHandler(
    config,
    new HttpClientHandler() { UseCookies = false }
    );
HttpClient httpClient = new HttpClient(messageHandler)
```

Maximizing throughput

Let the server tell you when to retry the request

Use multiple threads

Distribute workloads across multiple service principals

Avoid large batches

Remove the affinity cookie

» **Optimize your connection**

Increase throughput by optimizing your connection

C#

```
//Change max connections from .NET to a remote service default: 2
System.Net.ServicePointManager.DefaultConnectionLimit = 65000;
//Bump up the min threads reserved for this app to ramp connections faster
System.Threading.ThreadPool.SetMinThreads(100, 100);
//Turn off the Expect 100 to continue message
System.Net.ServicePointManager.Expect100Continue = false;
//Can decrease overall transmission overhead
System.Net.ServicePointManager.UseNagleAlgorithm = false;
```

Temporary exemptions from API limits

Document Routing Agent

Warehouse Management mobile app (WHSMobile)

Retail Server API

Office Integration

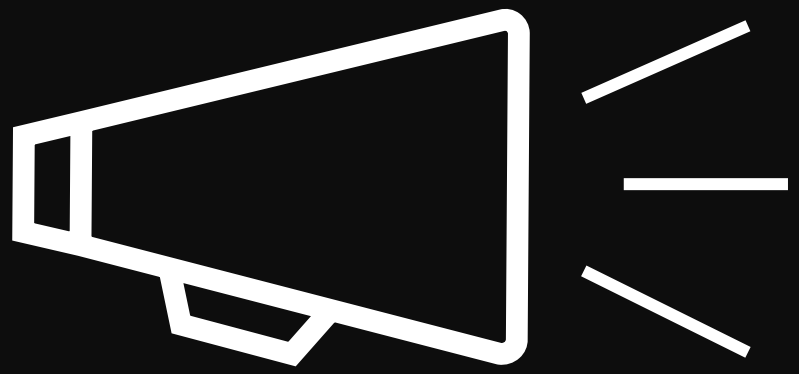
Data Import/Export Framework (DIXF)

Data Integrator

Dual-write

Power Platform virtual tables for Finance and Operations apps

Finance and Operations apps Connector



Call to action!

Implement handing for 429 response

Take steps to optimize integrations

Review additional resources and provide feedback

Additional resources

Documentation	https://aka.ms/FinOpsAPIThrottling
Email	finopsapithrottling@microsoft.com
Office hours	Coming soon!

Microsoft