Semaphores microservice

Overview

A locking mechanism was created for data resources in Turbine. It is needed to avoid concurrency problems. The locking mechanism is based on strictly virtual (not enforced) semaphores, stored in a DB table and accessed via REST API. On Turbine process side the API is being used from a dedicated object instantiated per process run.

Endpoints related to semaphores are exposed by the separate microservice.

Repository

https://dev.azure.com/dh-platforms-devops/app-cdl-turbine/_git/semaphore.api

Resources

DEV

App Service: **https://portal.azure.com/#@pgone.onmicrosoft.com/resource/subscriptions/cbf879fb-2d14-474a-9fd5-cf145bd4759d/resourceGroups/AZ-RG-CoreDataPlatform-Utilities-Metadata-DevOps-01/providers/Microsoft.Web/sites/app-cdl-semaphore-api-n101/appServices**

Database: **https://portal.azure.com/#@pgone.onmicrosoft.com/resource/subscriptions/cbf879fb-2d14-474a-9fd5-cf145bd4759d/resourceGroups/AZ-RG-CoreDataPlatform-Utilities-Metadata-DevOps-01/providers/Microsoft.Sql/servers/app-cdl-semaphore-api-db-server-n101/databases/app-cdl-semaphore-api-db-n101/overview**

Key Vault: **https://portal.azure.com/#@pgone.onmicrosoft.com/resource/subscriptions/cbf879fb-2d14-474a-9fd5-cf145bd4759d/resourceGroups/AZ-RG-CoreDataPlatform-Utilities-Metadata-DevOps-01/providers/Microsoft.KeyVault/vaults/kv-coreutil-metav2-n101/overview**

Service Bus: **https://portal.azure.com/#@pgone.onmicrosoft.com/resource/subscriptions/cbf879fb-2d14-474a-9fd5-cf145bd4759d/resourceGroups/AZ-RG-CoreDataPlatform-Utilities-Metadata-DevOps-01/providers/Microsoft.ServiceBus/namespaces/sbn-metadatav2-n101/overview**

Configuration

Below keyVault secrets are needed to run the microservice:

Environment variable	Desc
semaphore-db-url	Database URL
semaphore-db-user	Database user
semaphore-db-password	Database password
turbine-service-bus-connection-string	Connection string to a service bus

Local configuration:

To run it locally you need to specify variables responsible for connection with keyVault: azure.keyvault.client-id, azure.keyvault.client-key and azure.keyvault.tenant-id. If it's done, you should be able to run the application.

Details

Endpoints

GET /api/v1/semaphores

Fetch existing semaphores. It can be filtered by processRunKey and resourcePath.

POST /api/v1/semaphores

Acquire semaphore for given processRunKey, resourcePath and resourcePathPattern. Additional user is able to set semaphoreType, requestTimespan and currentProcessLimit if needed.

PUT /api/v1/semaphores/release

Release semaphore. User is able to release all semaphores matching criteria: processRunKey, resourcePath and resourcePathPattern

PUT /api/v1/semaphores/renew

Renew semaphore. User is able to renew all semaphores matching criteria: processRunKey, resourcePath and requestedTimespan

Service bus (release-semaphore-queue)

Semaphore microservice is listening release-semaphore-queue. It's another way to release semaphores. In a message user is allowed to pass processRunKey, resourcePath and resourcePathPattern.

Usage

Semaphores are a mechanism for handling concurrency that is supposed to prevent two main scenarios:

- multiple processes overwriting the same data simultaneously and
- one process reading data that is being overwritten by another process

A typical scenario for semaphore use is merging POS data.

Interface

Semaphore use is built into two operations: FileLoaderTabular and FilePublisher, for which semaphores are managed automatically but there's also a separate Semaphore operation that allows to acquire and release semaphores on demand. You add the Semaphore operation block and

choose the type of semaphore (shared/exclusive) and the operation (acquire/release). It is an option for advanced users only.

There is also an option to acquire a semaphore for the entire process using process parameters `SEMAPHORE_PATH` and `SEMAPHORE_TYPE`:

[process semaphores] | *../../resources/images/semaphores/process_semaphores.png*

The "PATH" doesn't have to be an actual path, any string value is accepted. This mechanism allows to limit the number of concurrent processes of a particular type running in parallel.

Semaphore options

Semaphores can be "shared" or "exclusive".

Multiple processes can have shared semaphores on the same resource and run concurrently without waiting (e.g. for reading the same data). It's the default type for FileLoader.

Only one process can have an exclusive semaphore and while an exclusive semaphore is acquired for a resource, no other processes can get even a shared semaphore on it. Exclusive is the default for FilePublisher.

For FilePublisher there is one more option for semaphore type, "already acquired". It should be used if the semaphore for particular output path was already acquired in previous operations (Loader or Semaphore).

Internals

Regardless of the operation from which semaphores are used, communication with Semaphore API is managed by an instance of a `SemaphoreManager` object. All semaphores for a process run are released when the process ends, regardless of the status it ends with.