Usage Metering and Token Allocation

Version 1.3.0-SNAPSHOT, 2026-02-17T20:12:18Z

Table of Contents

This section describes how to meter API and LLM usage per tenant and how to allocate **token pools** that can be assigned to different sets of APIs and Tools/LLM configurations for usage-based billing and quotas.

Opt-in: Token metering is **disabled by default**. To use it you must enable it in `application.properties` (see Configuration). When disabled, no usage is recorded and no quotas apply.

Chapter 1. Overview

- **Usage records** — Each API call and each LLM/agent tool invocation can be recorded with tenant, caller (or runAs user), and API/tool identity. Records are persisted so you can aggregate by tenant, user, API, or tool for billing.
- **Token allocation** — Per tenant you define **allocations**: a token type (e.g. API calls, LLM requests, input/output tokens), a **scope** (which APIs or tools this pool applies to), an allocated amount per period, and a **billing interval** and **replenishment** (frequency and amount). Consumption is debited from the matching allocation when usage is recorded. When the period ends, the allocation is replenished (consumed reset, period advanced).
- **Token types** — Different kinds of tokens can be assigned to different API sets and Tools/LLM configs. For example: one pool for "Premium API" (integration/query/find, integration/query/save), another for "LLM Standard" (tools query_find, query_save with a specific LLM config key).
- **Blocking when tokens run out** — When enforce-quota is enabled, APIs and tools covered by an allocation are **blocked** when tokens run out. The framework returns HTTP 402 (Payment Required) with a message that includes the allocation name and the **next replenishment date** when configured.

Chapter 2. Token Types

The framework supports these consumable token kinds:

| Token type | Unit | Typical scope | |---|---|---| | `API_CALL` | 1 per API request | API identifiers (area/domain/action) | | `LLM_REQUEST` | 1 per tool invocation | Tool names (e.g. query_find, query_save) | | `LLM_INPUT_TOKENS` | Token count | Tool names and/or LLM config keys | | `LLM_OUTPUT_TOKENS` | Token count | Tool names and/or LLM config keys |

You create a **TenantTokenAllocation** per tenant: name, token type, scope (which APIs/tools/LLM configs), allocated amount, **billing interval** (e.g. MONTHLY, YEARLY), and **replenishment amount** (tokens added each period). When usage is recorded, the framework finds an allocation whose scope matches the request, replenishes the allocation if its period has ended, then debits consumption.

Chapter 3. Scope: Assigning Pools to APIs and Tools/LLM Configs

Each allocation has a **UsageScope** that defines which usage consumes from that pool:

- **API scope** — List of API identifiers in the form `area/domain/action` (e.g. `integration/query/find`, `integration/query/save`). Only API calls that match one of these identifiers consume from this allocation.
- **Tool scope** — List of agent tool names (e.g. `query_find`, `query_save`). Only LLM/tool invocations that match one of these names consume from this allocation.
- **LLM config scope** — Optional list of LLM config keys. When present, only usage for those LLM configurations consumes from this allocation. This lets you assign different token pools to different LLM configurations (e.g. "Standard" vs "Premium" model).

You can combine scope types: for example, one allocation with tool scope `[query_find, query_save]` and llmConfigKeys `[standard]`, and another with the same tools and llmConfigKeys `[premium]`, so that standard and premium LLM usage are billed from different pools.

Chapter 4. Usage Records

- **ApiCallUsageRecord** — One record per API call: realm, callerUserId, area, functionalDomain, action, optional path, timestamp. Optional link to the allocation that was debited.
- **LlmUsageRecord** — One record per LLM/agent tool use: realm, runAsUserId (effective user), toolName, optional llmConfigKey, optional inputTokens/outputTokens, timestamp. Optional link to the allocation that was debited.

Records are stored in the tenant's realm database so you can query by tenant, user, time range, or allocation for billing and reporting.

Chapter 5. Billing Interval and Replenishment (per tenant)

Each **TenantTokenAllocation** can have:

- **Billing interval** — `BillingInterval.MONTHLY` or `BillingInterval.YEARLY`. Defines the period length; when the period ends, the allocation is replenished.
- **Replenishment amount** — Number of tokens to add at the start of each period (often the same as the allocated amount). When the period ends, consumed is reset to 0 and allocated amount is set to this value (or unchanged if 0).
- **Period start** / **period end** — The current period window. The framework advances the period automatically when it has ended and the allocation is next used.

Replenishment is applied lazily: when a matching API or tool is used and the allocation's period has ended, the framework resets consumed, advances the period, and then checks quota. This avoids a separate scheduler; you can still run a job to advance periods if you prefer.

Chapter 6. Blocking When Tokens Run Out

When `quantum.metering.enforce-quota` is `true` and a request would exceed the matching allocation's remaining amount:

- The framework throws `UsageMeteringService.QuotaExceededException`.
- The exception is mapped to **HTTP 402 Payment Required** with a JSON body that includes a clear message, for example: **"Usage limit reached for \"Premium API\". The associated APIs or tools are blocked until the next replenishment on 2025-03-01."**
- When no replenishment is configured (no billing interval), the message states that the associated APIs or tools are blocked and no replenishment is configured.

Clients can use the response body to show the user when the limit will reset.

Chapter 7. Configuration

Token metering is **opt-in**. By default it is **not** used; you must enable it in `application.properties` to record usage or enforce quotas.

| Property | Default | Description | |---|---|---| | `quantum.metering.enabled` | `false` | Set to `true` to enable token metering. When false, no usage is recorded and no quotas apply. | | `quantum.metering.enforce-quota` | `false` | When true (and metering enabled), APIs/tools covered by an allocation are blocked when tokens run out (HTTP 402 with message including next replenishment date when available). |

Morphia must map the usage entities: include `com.e2eq.framework.model.persistent.usage` in `quarkus.morphia.packages`.

Chapter 8. Integrating Metering

- **API calls** — Call `UsageMeteringService.recordApiCall(realm, callerUserId, area, functionalDomain, action, path)` from a request filter or after successful API handling. The service records the usage and, if a matching API_CALL allocation exists, debits it.
- **LLM/agent tool usage** — Call `UsageMeteringService.recordLlmUsage(realm, runAsUserId, toolName, llmConfigKey, inputTokens, outputTokens)` from the agent execute path (e.g. after `AgentExecuteHandler.execute`). The service records the usage and, if a matching LLM allocation exists, debits by request count or token count.

When `enforce-quota` is true, recording throws `UsageMeteringService.QuotaExceededException` if the allocation would be exceeded; the framework maps this to HTTP 402 with a message that includes the allocation name and next replenishment date when available.

Chapter 9. Billing and Reporting

- **By tenant** — Query usage records by realm and time range; optionally group by callerUserId, area/domain/action, or toolName.
- **By allocation** — Usage records can store the allocationId that was debited; you can aggregate consumption per allocation for billing.
- **Remaining quota** — `TenantTokenAllocation.getRemainingAmount()` returns allocated minus consumed for the current period. Allocations have periodStart/periodEnd and optional billing interval (MONTHLY, YEARLY) for replenishment.

See the implementation classes: `TenantTokenAllocation`, `BillingInterval`, `UsageScope`, `ApiCallUsageRecord`, `LlmUsageRecord`, `UsageMeteringService`, and the usage repos in `com.e2eq.framework.model.persistent.morphia.usage`.