Specification: The Codon SDK `Workload` Class

1. Core Design & Purpose

The `Workload` class is the central, user-facing component of the SDK. It serves as a unified interface to register, define, execute, and track an agent's logic. Its design is guided by three principles:

- **Decoupling:** It separates the agent's portable **logic** from its execution **deployment context**.
- **Developer Experience:** It provides a seamless, single-flow workflow that minimizes friction and boilerplate.
- **Flexibility:** It supports both custom-built agents and those created with popular frameworks via adapters.

---

2. Revised ID Structure

The `Workload` class orchestrates the registration and use of a decoupled ID system.

- **The Logic Group 📜:** Represents the agent's immutable, portable code.
  - **`Agent Class ID`:** The top-level, human-readable identifier for a family of agents (e.g., "InvoiceProcessor v2").
  - **`NodeSpec ID`:** An environment-agnostic blueprint of a single computational step.
  - **`Logic ID`:** The single, universal identifier for the agent's complete logical structure, derived from the `Agent Class ID` and its constituent `NodeSpec`s.
- **The Deployment Group 🌍:** Represents a specific execution context.
  - **`Deployment ID`:** A unique identifier for a specific runtime environment. It combines the infrastructure manifest (the original `CIM` concept) with the agent's specific configuration (the original `Agent Instance` concept).

---

3. Client-Facing API

The API is designed to be intuitive and flexible.

- **Instantiation:** `my_workload = Workload(name, version, description, tags)`
  - **`name` (str):** The agent's family name.

    - **`version` (str):** The specific version of the agent's logic.
    - **`description` (str, optional):** A human-readable description.
    - **`tags` (List[str], optional):** A list of standardized tags for explicit, cross-customer classification (e.g., `["finance", "ocr"]`).
- **Graph Building (for custom agents):**
    - `my_workload.add_node(function, name, role)`
    - `my_workload.add_edge(source_name, destination_name)`
- **Convenience Adapters:**
    - `@classmethod Workload.from_langgraph(graph, name, version, ...)`
    - `@classmethod Workload.from_crewai(crew, name, version, ...)`
- **Execution:**
    - `result = my_workload.execute(input, deployment_id)`

---

4. Key Responsibilities & Workflow

The `Workload` object manages the entire agent lifecycle.

1. **Implicit Registration:** Upon instantiation (`Workload(...)`), the SDK automatically handles the registration of the **Logic Group**. It uses the provided metadata to register the `Agent Class ID`, defines the `NodeSpec`s, and generates the final, portable `Logic ID`. This eliminates any separate, manual registration steps for the developer.
2. **Auto-Instrumentation:** When a developer adds a node (`.add_node(...)`), the `Workload` object automatically wraps the raw function with all necessary OpenTelemetry instrumentation logic. This removes the need for decorators and cleanly separates business logic from observability concerns.
3. **Bind-at-Runtime:** The `.execute()` method triggers the agent run. It requires a `deployment_id` to be passed in, which **binds** the portable `Logic ID` to a specific `Deployment ID` for that single execution. This is where the SDK performs its pre-flight validation (e.g., checking if the logic is authorized to run in that deployment) and generates the ephemeral `WorkloadRunID`.