SODA Open Context Specification (OCS)

Enabling Enriched Data Contexts for AI

Introduction

This document provides the SODA Open Context Specification, which describes aspects and guidelines for building data context or enhancing existing context for AI Models and Agents to get accurate and faster results at scale.

Purpose

The document provides guidelines and considerations for the effective construction of data contexts for various data sources and types. This serves as the input document to lay out various data attributes to generate context artifacts, such asa context schema for specific data sources

Audience

Architects and Developers building AI based applications for data inferences and insights

Open Context Specification: The Background

There is no standard way of communication to AI to get things done! Hence, the data inference and insights suffer from:

a) Low Accuracy
   - The accuracy of results varies drastically based on the nature of data and inputs
   - Mixing guesses and different sources of knowledge confuses AI

b) Inconsistency
   - Hallucination is a key known issue with AI

c) High Latency
   - Based on the type of query and volume of data, it fails to give ontime results

d) Huge Cost
   - Iterations to get close results and verification add costs

e) Lack of Scale
   - Works for a small amount of data or 1 agent, when it comes to scale, it fails

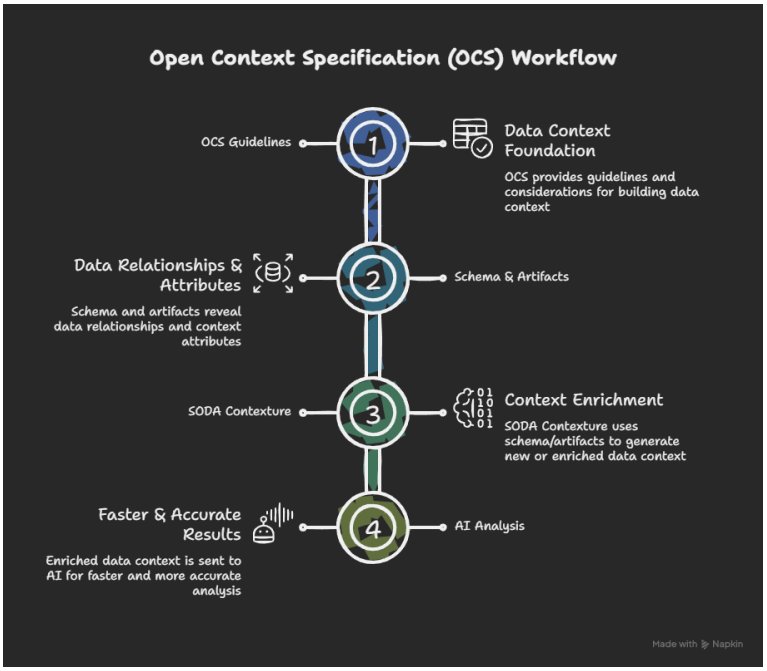
f) Low Reliability
   - Due to uncertain results, AI is not fully dependable

One of the solutions to these problems is to provide the right context to the AI, for it to better fetch the right pieces of data to derive the right inference. However this is not easy. Because, the data relationships and types can vary. Hence, it is important to build better context or enrich the existing ones.
SODA Contexture is an open source engine for building operational context for AI systems. It addresses a common issue in AI agents: missing or fragmented context across systems.
Rather than relying only on prompts, Contexture provides structured context derived from operational data. This helps improve accuracy while keeping latency and cost under control.

Open Context Specification is initiated to provide guidelines and considerations for building data context for various data sources and types.

So, OCS enables a structured way of building context schemas and other artifacts to build better data context or enrich the existing context

OCS Position and Flow



The 5 Pillars of Data Context

I. Identity & Origin (The "Who" and "Where")

This defines the unique fingerprint of the data source. AI models need this to distinguish between similar metrics coming from different environments.

- **Provider/Source:** (e.g., Prometheus, CloudWatch, NetApp, MongoDB).
- **Environment:** (e.g., Production, Staging, Dev).
- **Namespace/Domain:** The logical grouping (e.g., k8s-cluster-01 or billing-db-instance).

II. Dimensionality & Topology (The "Relationship")

This is the most critical part for AI reasoning. It defines how this metric relates to other components (e.g., "This disk belongs to this node, which hosts this pod").

- **Resource Type:** (e.g., Pod, PersistentVolume, Table).
- **Parent/Child Links:** Hard-coded relationships (e.g., NodeID for a Pod).
- **Labels/Tags:** Key-value pairs that allow for filtering (e.g., app: web-server, region: us-east-1).

III. Metric Semantics (The "What")

You must define what the number actually represents to avoid the AI comparing apples to oranges.

- **Metric Name:** A human-readable, descriptive name.

- **Unit of Measure:** (e.g., Percentage, Bytes, Milliseconds, Requests/sec).
- **Metric Type:** * *Gauge:* Snapshots (Current Memory).
  - *Counter:* Cumulative (Total requests).
  - *Histogram/Summary:* Latency distributions.

IV. Temporal Context (The "When")

AI needs to know if it's looking at a "point-in-time" value or a trend.

- **Granularity/Resolution:** (e.g., 10-second intervals vs. 5-minute averages).
- **Timestamp:** The UTC epoch of the last update.
- **Retention Policy:** The duration for which the context remains valid.

V. Operational Constraints (The "How")

This tells the AI how to interpret the health of the metric.

- **Thresholds:** (e.g., Warning @ 80%, Critical @ 90%).
- **Polarity:** Is a high value good (e.g., Uptime) or bad (e.g., Latency)?
- **Aggregator:** Should the AI use Sum, Avg, or Max when looking at a group?

Example Context Schema

Below is a JSON-style representation of how the spec guidelines and considerations can be translated for a specific data source. Here, the data is Kubernetes Pod metrics and data source could be time series data base like Prometheus

JSON

```
{
  "spec_version": "1.0",
  "context_definitions": [
    {
      "resource_id": "k8s-pod-auth-service-7f8",
      "domain": "compute.k8s",
      "identity": {
        "cluster": "prod-us-west-2",
        "namespace": "identity-iam",
        "node_owner": "ip-10-0-2-45.ec2.internal"
      },
      "metrics": [
        {
          "name": "memory_utilization",
          "type": "gauge",
          "unit": "percentage",
          "description": "Current memory usage against pod limits",
          "aggregation_logic": "average",
          "health_config": {
```

```
          "critical_threshold": 95,
          "polarity": "high_is_bad"
        }
      }
    ],
    "topology": {
      "belongs_to": "deployment/auth-service",
      "storage_dependency": "pvc-099-x2",
      "db_dependency": "postgre-main-cluster"
    }
  },
  {
    "resource_id": "db-prod-main-01",
    "domain": "storage.database",
    "identity": {
      "engine": "PostgreSQL",
      "version": "15.4",
      "tier": "mission-critical"
    },
    "metrics": [
      {
        "name": "active_connections",
        "type": "gauge",
        "unit": "count",
        "description": "Total number of active backend processes",
        "health_config": {
          "warning_threshold": 400,
          "max_capacity": 500
        }
      }
    ]
  }
 ]
}
```

Why This Helps the AI

When the user asks: *"Is the auth service slow because of the database?"*, the AI doesn't just search for "slow." It uses your spec to:

1. **Identify** the auth-service Pod.
2. **Trace** the topology.db_dependency to find the specific DB instance.
3. **Compare** the metrics.latency (if defined) of both using the unit and health_config to see which is currently in a "Critical" state.