Cisco Observability Platform Knowledge Store

Exported on 07/09/2024

The Knowledge Store is an advanced storage system used to create, manage, and store solutions as well as their components. For an introduction to solutions, see Solutions. This page introduces basic Knowledge Store concepts.

Knowledge Object

A JSON or YAML object that contains data in the form of key/value pairs. A knowledge object is an artifact that enriches, customizes, or alters the behavior of data ingestion, processing, and visualization in a solution. It can be created by a developer, end user, or solution.

Examples of knowledge objects include solution configurations, entity types, metric types, and event types.

Knowledge Type

A JSON or YAML schema file within your solution. It defines the structure of a knowledge object and defines properties such as how the object ID will be generated, how the display name will be generated, whether any of the object properties contain secrets, and more.

An example of a knowledge type is a UI template that users can use to create knowledge objects that represent UI components.

Knowledge Store

An advanced JSON and YAML document storage system that manages the lifecycle of knowledge objects, knowledge types, and solutions.

The Knowledge Store stores knowledge objects and types in a database that can be accessed by end users and services. It enables solutions to define and manage domain-specific business data, which is a key enabler of differentiated solutions. For example, a solution may use health rules and FMM entity modeling to detect network intrusions. Using the Knowledge Store, the solution could introduce a concept such as an “investigation” to the platform, allowing its users to create and manage the complete lifecycle of a network intrusion investigation from creation to remediation.

The ownership of knowledge objects is transparently managed by the Knowledge Store. Whether the knowledge object is a user UI template or a Tenant pipeline configuration, the Knowledge Store automatically recognizes the identity principal of the user and uses that information to target the objects owned by that principal. This enables user-owned data by allowing user-aware platform configurations to be retrieved by user interfaces used by an end user.

The Knowledge Store also provides an interface in the form of a REST API. As a result, the caller is abstracted from the physical database that is backing the Knowledge Store. See Knowledge Store API.

Layers

A layer represents a level of information where knowledge objects can be created, stored, and accessed. The Cisco Observability Platform currently supports three layers: Solution, Tenant, and Local-user.

Layer	Description	Who can create knowledge objects at this layer?	Who can access knowledge objects at this layer?
Solution	A layer that is replicated globally across cells. Suitable for small pieces of information that need to be shared globally.	Solution	• Tenants subscribed to the solution • All users within the Tenant
Tenant	A layer that is replicated across users in the same Tenant. Suitable for larger pieces of information that need to be shared between a group of users in the same Tenant.	User	Users within the Tenant based on their access rights
Local-user	A layer that is only accessible to the local user. Suitable for larger pieces of information that are only relevant to a single user.	User	The user that created the object

Layering Use Cases

Layers enable simple solution management by facilitating efficient Solution- and Tenant-wide changes. With layers, a solution owner can make a change in one layer and synchronize that change to all Tenants and users if desired. For example, instead of adjusting defaults in every individual Tenant, solution owners can use the Solution layer to make one adjustment and rely on the Knowledge Store to ensure that every Tenant receives the new value.

Layers also support different use cases for user-specific content, which are described in the following section.

Layering Methods

There are two methods of layering: layering with fragments and layering with complete objects.

Layering with Fragments

Fragments are partial knowledge objects that represent different levels of ownership. Fragments allow for mutable defaults that can support knowledge objects, such as user preferences, at different layers.

In this method, a patch is used to define a fragment on a lower layer with a target object in any of the upper layers. The target object is the fragment or complete object that the fragment is trying to patch. At read-time, the knowledge object is assembled as a hierarchy of fragments. When the fragments are assembled in layers, they create a complete knowledge object. Knowledge objects in one layer can override values of knowledge objects in upper layers based on the knowledge type definition.

An example of layering with fragments is a solution that provides default UI preferences at the Solution layer. At the Tenant layer, each Tenant can save a fragment that overrides a field, such as a time zone setting. At the Local-user layer, each end user can choose between dark mode or light mode. When users query the Knowledge Store for their UI settings, they receive a settings knowledge object that is dynamically assembled from all of the required layers.

Layering with Complete Objects

In this method, a complete knowledge object exists at one layer independently of the same knowledge object(s) at different layers. This method can be used in situations where mutable defaults are not preferred.

An example of layering with complete objects is a scenario where a solution provides default health rules to users. The Solution-layer health rules can be used by the Tenants subscribed to the solution, but can also be modified by the solution developer at any time. A user can clone the health rules as complete objects at the Tenant layer to prevent the Solution-layer changes from being automatically applied to their health rules. A user could also modify those complete objects to create custom health rules at the Tenant layer.