Spec Store

Spec store is a data-store used for storing static specifications of blocks, vDAGs, LLM Models and Clusters which can be readily used, spec store can also be used to search, query the ready-made specifications which can be customized as per the custom requirements.

Spec store schema:

Here is the python data-class representing a spec:

```python
@dataclass
class SpecStoreObject:
    specUri: str = '' # specName:specVersion.version-specVersion.tag
    specType: str = ''
    specName: str = ''
    specVersion: Dict[str, str] = field(default_factory=lambda: {"version": "", "tag": ""})
    specDescription: str = ''
    specMetadata: Dict[str, Any] = field(default_factory=dict)
    specTags: List[str] = field(default_factory=list)
    specData: str = ''
```

Field Explanations

1. `specUri: str`

   - **Default Value**: `''`
   - **Description**:
     A unique identifier for the specification, automatically generated using `specName` and `specVersion`.
     The format used is:

     `specUri = "{specName}:{version}-{tag}"`

     This ensures each specification can be uniquely referenced.

---

2. `specType: str`

   - **Default Value**: `''`
   - **Description**:
     Specifies the type of the specification. It can be used to categorize different kinds of specifications such as API specs, data models, or configuration schemas.

3. `specName: str`

- **Default Value**: `''`
- **Description**:
  The name of the specification. This serves as the base identifier before versioning is applied.

4. `specVersion: Dict[str, str]`

- **Default Value**: `{"version": "", "tag": ""}`
- **Description**:
  A dictionary storing version information. It contains:
  - `version`: The version number of the specification.
  - `tag`: An additional tag for distinguishing versions (e.g., `stable`, `beta`).

5. `specDescription: str`

- **Default Value**: `''`
- **Description**:
  A human-readable description of the specification. It provides context on what the specification represents.

6. `specMetadata: Dict[str, Any]`

- **Default Value**: `{}` (empty dictionary)
- **Description**:
  A dictionary containing metadata related to the specification. This can include additional details such as:
  - Author
  - Creation date
  - Dependencies
  - Any other custom properties

7. `specTags: List[str]`

- **Default Value**: `[]` (empty list)
- **Description**:
  A list of tags associated with the specification. Tags help categorize and filter specifications efficiently.

8. `specData: str`

- **Default Value**: `''`
- **Description**:
  Stores the actual content or data of the specification. This can be raw JSON, YAML, or any structured format required by the specification.

Spec store APIs:

Create a Specification

Endpoint

- **URL**: `/spec`
- **Method**: `POST`
- **Description**: Creates a new specification and stores it in the database.

```
curl -X POST http://<api-url>/spec \
     -H "Content-Type: application/json" \
     -d '{
          "specName": "MySpec",
          "specType": "API",
          "specVersion": {"version": "1.0", "tag": "stable"},
          "specDescription": "This is an API specification.",
          "specMetadata": {"author": "John Doe"},
          "specTags": ["v1", "stable"],
          "specData": "{\"endpoints\": [{\"path\": \"/users\", \"method\": \"GET\"}]}"
      }'
```

Get a Specification

Endpoint

- **URL**: `/spec/<specUri>`
- **Method**: `GET`
- **Description**: Retrieves a specification based on its `specUri`.

Path Parameter

Parameter	Type	Description	Example
`specUri`	string	The unique URI of the specification	`MySpec:1.0-stable`

```
curl -X GET http://<api-url>/spec/MySpec:1.0-stable
```

Update a Specification

Endpoint

- **URL**: `/spec/<specUri>`
- **Method**: `PUT`

- **Description**: Updates an existing specification based on its `specUri`.

Path Parameter

Parameter	Type	Description	Example
`specUri`	string	The unique URI of the specification	`MySpec:1.0-stable`

Request Body (JSON)

- You can update any of the fields except `specUri`. Here is an example:

```
{
    "specDescription": "Updated description for the API.",
    "specMetadata": {"author": "Jane Doe"},
    "specTags": ["v1", "stable", "updated"]
}
```

CURL command:

```
curl -X PUT http://<api-url>/spec/MySpec:1.0-stable \
     -H "Content-Type: application/json" \
     -d '{
          "specDescription": "Updated description for the API.",
          "specMetadata": {"author": "Jane Doe"},
          "specTags": ["v1", "stable", "updated"]
      }'
```

Delete a Specification

Endpoint

- **URL**: `/spec/<specUri>`
- **Method**: `DELETE`
- **Description**: Deletes an existing specification based on its `specUri`.

Path Parameter

Parameter	Type	Description	Example
`specUri`	string	The unique URI of the specification	`MySpec:1.0-stable`

```
curl -X DELETE http://localhost:5000/spec/MySpec:1.0-stable
```

Query Specifications

Endpoint

- **URL**: `/specs`
- **Method**: `POST`
- **Description**: Queries the database for specifications using MongoDB-style query filters.

Request Body (JSON)

- The request follows MongoDB query format, allowing advanced filtering.

```
{
    "specType": "API",
    "specVersion.version": {"$gte": "1.0"},
    "specTags": {"$in": ["stable"]}
}
```

CURL command:

```
curl -X POST http://localhost:5000/specs \
     -H "Content-Type: application/json" \
     -d '{
          "specType": "API",
          "specVersion.version": {"$gte": "1.0"},
          "specTags": {"$in": ["stable"]}
      }'
```

Supported Spec types (specType field):

1. “block”: Used for representing a block specification.
2. “vdag”: Used for representing a vDAG specification.
3. “cluster”: Used for representing a cluster specification.
4. “component”: Used for representing a component in component registry.
5. “search”: Used for representing similarity search spec.
6. “mgmtCommand”: Used for representing a management command.

Refer to the documentation of parser for how to use these specifications along with parser.