Template Store

Template store can be used for storing the templates which can be used for validation and building custom parsers, refer to the documentation of parser for understanding how to use custom parsers using policy-rules. This documentation explains the APIs and schema of the template store.

Template store schema:

Here is the python data-class representing a template:

```
@dataclass
class TemplateObject:
    templateUri: str = ''
    templatePolicyRuleUri: str = ''
    templateMetadata: Dict[str, str] = field(default_factory=dict)
    templateName: str = ''
    templateDescription: str = ''
    templateVersion: Dict[str, str] = field(
        default_factory=lambda: {"templateVersion": "", "tag": ""})
    templateTags: List[str] = field(default_factory=list)
    templateData: str = ''
```

TemplateObject Fields

1. `templateUri: str`

   - **Default Value**: `''`
   - **Description**:
     A unique identifier for the template, automatically generated using `templateName` and `templateVersion`.
     The format used is:

     `templateUri = "{templateName}:{templateVersion}-{tag}"`

     This ensures each template is uniquely referenced.

2. `templatePolicyRuleUri: str`

   - **Default Value**: `''`
   - **Description**:
     A URI linking the template to a policy rule, allowing governance or policy enforcement.

3. `templateMetadata: Dict[str, str]`

   - **Default Value**: `{}` (empty dictionary)

- **Description**:
  A dictionary storing metadata about the template, which may include:
  - Author
  - Creation date
  - Associated policies
  - Any other custom properties

4. `templateName: str`

- **Default Value**: `''`
- **Description**:
  The name of the template, used as the base identifier before versioning is applied.

5. `templateDescription: str`

- **Default Value**: `''`
- **Description**:
  A human-readable description providing details on the template's purpose and usage.

6. `templateVersion: Dict[str, str]`

- **Default Value**: `{"templateVersion": "", "tag": ""}`
- **Description**:
  A dictionary storing versioning details, including:
  - `templateVersion`: The version number of the template.
  - `tag`: An additional tag for version tracking (e.g., `stable`, `beta`).

7. `templateTags: List[str]`

- **Default Value**: `[]` (empty list)
- **Description**:
  A list of tags associated with the template, allowing easy categorization and searchability.

8. `templateData: str`

- **Default Value**: `''`
- **Description**:
  Stores the actual content or data of the template. This could be a raw string containing:
  - JSON
  - YAML
  - Text-based configurations

Template store APIs:

Here is the documentation for the **Create Template API**, following your strict structure:

Create a Template

Endpoint

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

```
curl -X POST http://<api-url>/template \
     -H "Content-Type: application/json" \
     -d '{
          "templateName": "ExampleTemplate",
          "templateDescription": "This is a sample template.",
          "templateVersion": {"templateVersion": "1.0", "tag": "stable"},
          "templateTags": ["config", "example"],
          "templateMetadata": {"author": "John Doe", "created": "2025-03-26"},
          "templateData": "{\"key\": \"value\"}"
      }'
```

Get a Template

Endpoint

- **URL**: `/template/<templateUri>`
- **Method**: `GET`
- **Description**: Retrieves a template by its unique `templateUri`.

```
curl -X GET http://<api-url>/template/ExampleTemplate:1.0-stable \
     -H "Content-Type: application/json"
```

Here is the documentation for the **Update Template API**:

Update a Template

Endpoint

- **URL**: `/template/<templateUri>`
- **Method**: `PUT`
- **Description**: Updates an existing template identified by `templateUri`.

```
curl -X PUT http://<api-url>/template/ExampleTemplate:1.0-stable \
     -H "Content-Type: application/json" \
     -d '{
          "templateDescription": "Updated description for the template.",
          "templateTags": ["config", "updated"],
          "templateMetadata": {"modifiedBy": "Jane Doe", "modifiedAt": "2025-03-26"},
          "templateData": "{\"key\": \"newValue\"}"
      }'
```

Delete a Template

Endpoint

- **URL**: `/template/<templateUri>`
- **Method**: `DELETE`
- **Description**: Deletes a template identified by `templateUri`.

```
curl -X DELETE http://<api-url>/template/ExampleTemplate:1.0-stable \
     -H "Content-Type: application/json"
```

Here is the updated documentation for the **Query Templates API**, specifying that the query follows MongoDB syntax:

Query Templates

Endpoint

- **URL**: `/templates`
- **Method**: `POST`
- **Description**: Queries templates based on the provided filter criteria. The query follows **MongoDB syntax**, allowing for flexible filtering using operators like `$eq`, `$ne`, `$in`, `$regex`, etc.

```
curl -X POST http://<api-url>/templates \
     -H "Content-Type: application/json" \
```

```
  -d '{
       "templateMetadata.author": { "$eq": "User X" },
       "templateTags": { "$in": ["config", "stable"] }
   }'
```

Example Queries

- Find all templates created by "User X":

  ```
  { "templateMetadata.author": { "$eq": "User X" } }
  ```

- Find all templates with tags `"config"` or `"stable"`:

  ```
  { "templateTags": { "$in": ["config", "stable"] } }
  ```

- Find templates where `templateName` starts with `"Example"`:

  ```
  { "templateName": { "$regex": "^Example", "$options": "i" } }
  ```

Using template policy rules:

Templates can be bounded to a policy rule. The field `templatePolicyRuleUri` should map to a policyRuleURI in the policy DB. This policy will be used to both validate the input data and also convert the input data into the internal representation (for the format used by the parser, refer to the parser documentation to understand what is internal representation and the structure of the template policy rule code).

Template store provides an API using which you can execute the policy rule associated with the template by providing the template URI, input data and the optional parameters.

API to execute the policy rule:

Endpoint

- **URL**: `/template/execute`
- **Method**: `POST`
- **Description**: Executes a template policy using the specified `template_uri`, `input_data`, and optional `parameters`. This API processes input data based on the defined conversion policy for the given template.

```
curl -X POST http://<api-url>/template/execute \
  -H "Content-Type: application/json" \
  -d '{
       "template_uri": "example-template:1.0-stable",
       "input_data": {},
```

```
        "parameters": {}
    }'
```

Request Parameters

Parameter	Type	Required	Description
`template_uri`	`string`	Yes	The unique URI of the template to be executed.
`input_data`	`dict`	Yes	The input data that the template policy will process.
`parameters`	`dict`	No	Optional parameters to customize policy execution.