Assets Management

Assets Management services provides a out of the box storage solution for storing the assets like models, policy files and misc. files that are used by the AIOSv1 instance logic and the Policies system. Assets Registry can also store asset specific metadata, versions and tags to support search and retrieval of assets.

Note: Assets Management is completely optional, any public registry with a downloadable URL can be used for storing assets.

The assets management functionalities are divided into two services:

1. Assets DB
2. Assets DB Registry

1. Assets DB:

1. Assets DB service is used for storing the assets and it's metadata.
2. Any user can spin up the assets DB server with their own S3 compatible storage backend (like S3 itself, Ceph OBS, Minio etc).
3. The deployment needs to be supported by a DB to store the metadata of the assets, the deployment can also point to the global DB if the assets globally discoverable and searchable.
4. Assets DB service should have a public URL using which other services/users can access the service.

2. Assets DB registry service:

1. This is a registry of asset DBs.
2. To make the Assets DB service discoverable to the public, it needs to be registered in the Assets DB registry service.
3. Assets DB registry provides APIs to search DB services available.
4. Assets DB registry also provides API to check the health of asset DBs.

Asset DB Schema:

Here is the python data-class of AssetObject:

```
@dataclass
class AssetObject:
    asset_uri: str = ''
    asset_name: str = ''
    asset_id: str = ''
    assets_db_id: str = ''
    asset_version: Dict[str, str] = field(
        default_factory=lambda: {"version": "", "tag": ""})
    asset_metadata: Dict[str, str] = field(default_factory=dict)
    asset_public_url: str = ''
```

```
asset_file_info: Dict[str, str] = field(default_factory=dict)
asset_tags: List[str] = field(default_factory=list)
```

Here's the updated explanation with **assets_registry_id** included and two points for each field:

1. **asset_uri (str)**:
   - A unique identifier constructed as "asset_name:asset_version-asset_tag" if all values are present.
   - Defaults to an empty string if any component (asset_name, asset_version, or asset_tag) is missing.
2. **asset_name (str)**:
   - Represents the name of the asset, used in asset_uri.
   - Helps identify the asset meaningfully within a system.
3. **asset_id (str)**:
   - Assigned by the system if public URL is not provided by the user and the user is using the asset storage provided by the assets management system.
   - An internal ID used for representing the asset in the storage system.
4. **asset_version (Dict[str, str])**:
   - Stores versioning details with "version" and "tag", both used in asset_uri.
   - Versions are used to generate a unique URI for the asset, so different assets can exist with same name but different versions.
5. **asset_metadata (Dict[str, str])**:
   - Holds additional information like descriptions, author, or creation date.
   - Metadata's schema is not fixed, users can include their own fields.
6. **asset_public_url (str)**:
   - A URL where the asset can be accessed publicly if applicable.
   - Useful for sharing assets externally - this is the downloadable URL of the asset.
   - If the asset file is uploaded using the assets DB then the public URL will be: http://<asset-db-url>/asset-object/<asset-id>
7. **asset_file_info (Dict[str, str])**:
   - Stores details about the asset file, such as size, format, or checksum.
   - The metadata of the asset-file can be included in this field.
8. **asset_tags (List[str])**:
   - A list of keywords or categories associated with the asset.

- These tags are used for search and filtering of assets.

9. **`assets_db_id` (str)**:
   - A unique identifier containing the ID of the asset DB where this asset resides.

   - By default it will be the ID of the asset DB service where the asset is stored.
   - This field will be helpful if the asset DB service is using a global assets DB to store the asset metadata.

Asset DB APIs:

1. Upload Asset API

Endpoint:
`POST /upload_asset`

Description:
This API uploads an asset file along with its metadata. The file is stored, and metadata is saved in the database. The request must be a `multipart/form-data` request containing the file and metadata.

---

cURL Command:

```
curl -X POST http://<server-url>/upload_asset \
     -H "Content-Type: multipart/form-data" \
     -F "asset=@/path/to/your/file.png" \
     -F 'asset_metadata={
           "asset_name": "SampleAsset",
           "asset_version": { "version": "1.0", "tag": "initial" },
           "asset_metadata": { "description": "Sample asset upload" },
           "asset_tags": ["image", "sample"]
         }'
```

Thanks for sharing the `AssetObject` structure. Now, let's move to the next API.

---

2. Get Asset API

Endpoint:
`GET /asset/<asset_uri>`

Description:
This API retrieves an asset using its unique `asset_uri`. The asset is fetched from the database and returned in JSON format.

---

cURL Command:

```
curl -X GET http://<api-url>/asset/SampleAsset:1.0-initial \
     -H "Content-Type: application/json"
```

---

3. Update Asset API

Endpoint:
`PUT /asset/<asset_uri>`

Description:
This API updates an existing asset using its `asset_uri`. The request body should contain the updated fields, which will be applied to the asset in the database.

---

cURL Command:

```
curl -X PUT http://<api-url>/asset/SampleAsset:1.0-initial \
     -H "Content-Type: application/json" \
     -d '{
           "asset_metadata": {"description": "Updated asset description"},
           "asset_tags": ["updated", "sample"]
         }'
```

4. Delete Asset API

Endpoint:
`DELETE /asset/<asset_uri>`

Description:
This API deletes an asset using its `asset_uri`. If the asset exists in the database, it will be removed.

---

cURL Command:

```
curl -X DELETE http://<api-url>/asset/SampleAsset:1.0-initial \
     -H "Content-Type: application/json"
```

5. Query Assets API

Endpoint:
`POST /assets`

Description:
This API allows querying multiple assets based on provided filters. The request body should contain a JSON object specifying the query criteria.

---

cURL Command:

```
curl -X POST http://<api-url>/assets \
     -H "Content-Type: application/json" \
     -d '{
           "asset_tags": ["sample"],
           "asset_metadata": {"type": "image"}
         }'
```

6. Query Assets API

Endpoint:
`POST /assets`

Description:
This API allows querying multiple assets using MongoDB-style query filters. The request body should contain a JSON object using MongoDB's query format.

---

cURL Command:

```
curl -X POST http://<api-url>/assets \
     -H "Content-Type: application/json" \
     -d '{
           "asset_tags": { "$in": ["sample", "image"] },
           "asset_metadata.type": "image",
           "asset_version.version": { "$gte": "1.0" }
         }'
```

7. Get Asset Object API

Endpoint:
`GET /asset-object/<asset_id>`

Description:
This API retrieves the asset file associated with a given `asset_id`. If the file exists, it will be returned as a downloadable response.

---

cURL Command:

```
curl -X GET http://<api-url>/asset-object/<asset_id> -o downloaded_asset
```

(Replace `<asset_id>` with the actual asset ID.)

8. Upload an external asset (using a already existing public URL)

Endpoint:
`POST /asset`

Description:
This API allows you to create a new asset by sending asset data in JSON format. The asset data is validated, converted into an `AssetObject`, and inserted into the database.

---

cURL Command:

```
curl -X POST http://<api-url>/asset \
     -H "Content-Type: application/json" \
     -d '{
           "asset_name": "Sample Asset",
           "asset_version": {"version": "1.0", "tag": "initial"},
           "asset_metadata": {"description": "Sample asset for testing"},
           "asset_tags": ["test", "sample"]
         }'
```

9. Health Check API

Endpoint:
`GET /health`

Description:
This API is used to check the health status of the service. It returns a success message indicating that the service is running properly.

---

cURL Command:

```
curl -X GET http://<api-url>/health
```

Assets DB registry Schema:

```python
@dataclass
class AssetsDBRegistry:
    asset_registry_id: str = field(default_factory=lambda: str(uuid.uuid4()))
    asset_registry_name: str = ''
    asset_registry_metadata: Dict[str, str] = field(default_factory=dict)
    asset_registry_tags: List[str] = field(default_factory=list)
    asset_registry_public_url: str = ''
```

Here's an explanation of each field in `AssetsDBRegistry`, with two points per field:

1. **`asset_registry_id: str`**
   - A unique identifier for each asset DB entry, generated using `uuid.uuid4()` by default.
   - Ensures that every asset DB has a globally unique ID for tracking and referencing.
2. **`asset_registry_name: str`**
   - Stores a human-readable name for the asset DB.
   - Helps in identifying and organizing asset registries within a database or system.
3. **`asset_registry_metadata: Dict[str, str]`**
   - A dictionary storing additional metadata about the asset DB, such as version, owner, or creation date.
   - Allows flexible and extensible storage of key-value pairs for descriptive information.
4. **`asset_registry_tags: List[str]`**
   - A list of tags associated with the asset DB, useful for categorization and searchability.
   - Helps in quickly filtering or grouping asset registries based on common characteristics.
5. **`asset_registry_public_url: str`**
   - A public-facing URL where the asset DB can be accessed.

Assets DB Registry APIs

1. Create Asset DB entry

Endpoint:

`POST /asset_registry`

Description:

Creates a new asset DB entry with metadata, tags, and a public URL.

cURL Command:

```
curl -X POST "http://<api-url>/asset_registry" \
     -H "Content-Type: application/json" \
     -d '{
           "asset_registry_name": "ResNet Models registry",
           "asset_registry_metadata": {
```

```
            "framework": "PyTorch",
            "version": "1.0",
            "author": "OpenAI",
            "license": "Apache-2.0"
          },
          "asset_registry_tags": ["image-classification", "deep-learning", "pytorch"],
          "asset_registry_public_url": "https://example.com/models/resnet-models-registry"
        }'
```

2. Get Asset DB entry

Endpoint:

`GET /asset_registry/<asset_db_id>`

Description:

Retrieves details of a specific asset DB entry using its unique ID.

cURL Command:

```
curl -X GET "http://<api-url>/asset_registry/123e4567-e89b-12d3-a456-426614174000" \
     -H "Content-Type: application/json"
```

3. Update Asset DB entry

Endpoint:

`PUT /asset_registry/<asset_db_id>`

Description:

Updates an existing asset DB entry with new metadata, tags, or other details.

cURL Command:

```
curl -X PUT "http://<api-url>/asset_registry/123e4567-e89b-12d3-a456-426614174000" \
     -H "Content-Type: application/json" \
     -d '{
           "asset_registry_metadata": {
             "framework": "TensorFlow" models,
             "version": "2.0"
           },
           "asset_registry_tags": ["neural-network", "tensorflow"]
         }'
```

3. Delete Asset DB entry

Endpoint:

`DELETE /asset_registry/<registry_id>`

Description:

Deletes an existing asset DB entry using its unique ID.

cURL Command:

```
curl -X DELETE "http://<api-url>/asset_registry/123e4567-e89b-12d3-a456-426614174000" \
     -H "Content-Type: application/json"
```

4. Query Asset DB entries

Endpoint:

`POST /asset_registries`

Description:

Fetches asset DB entries that match the specified query filter criteria. API accepts MongoDB type filter quries.

cURL Command:

```
curl -X POST "http://<api-url>/asset_registries" \
     -H "Content-Type: application/json" \
     -d '{
           "$and": [
             { "asset_registry_tags": { "$in": ["deep-learning"] } },
             { "asset_registry_metadata.framework": "PyTorch" }
           ]
         }'
```

5. Check Asset Registry Health

Endpoint:

`GET /check_health/<asset_db_id>`

Description:

Checks the health status of an asset registry entry by retrieving its public URL and performing a health check.

cURL Command:

```
curl -X GET "http://<api-url>/check_health/tes-123e4567-e89b-12d3-a456-426614174000" \
     -H "Content-Type: application/json"
```