Container Registry

Container registries are used to store the container images of AI/Computational workload instances on AIOSv1. These container images are pulled and instantiated as instances under a block whenever a block is created or instances are added to the block due to auto-scaling based on demand.

Features of the container registry system:

1. Any user can spin up their own container registry either on one of the clusters in a network or independently outside of the network.
2. A container registry DB is provided where users can list their container registries for public listing and discovery.
3. Developers can push their AIOS instance images to one or more of these registries to publish their code and then register their container image metadata in the components registry.
4. It is also possible to link any external container registry.
5. The Docker registry should be compatible with the OCI protocol.

Container Registry DB:

The container registry provides APIs to onboard a new container registry, query it, and perform update and delete operations. Here is the schema of the container registry:

```
from dataclasses import dataclass, field
from typing import Dict, List, Any
import uuid

@dataclass
class ContainerRegistry:
    # Unique identifier for the container registry entry (auto-generated UUID)
    container_registry_id: str = field(default_factory=lambda: str(uuid.uuid4()))

    # Human-readable name for the container registry
    container_registry_name: str = ''

    # Arbitrary metadata about the registry (e.g., environment, owner, labels)
    container_registry_metadata: Dict[str, Any] = field(default_factory=dict)

    # List of tags used for search, filtering, or categorization
    container_registry_tags: List[str] = field(default_factory=list)

    # Publicly accessible URL of the Docker container registry (e.g., http://registry.examp
    container_registry_public_url: str = ''

    # Used for prefixing the images, example: registry.example.com/my-image:v1
    container_registry_image_prefix: str = ''

    # Registry configuration settings (authentication, mirrors, endpoints, etc.)
    container_registry_config: Dict[str, Any] = field(default_factory=dict)

    # Storage-related metadata for the container registry:
    # {
    #     "storage_type": str       # "s3", "object-store", "fs", or "unknown"
    #     "storage_size": int       # In MB, or -1 for unlimited
    #     "memory_size": int        # In MB, or -1 if unknown or unlimited
    #     "registry_type": str      # "aios" for internal registry, "external" for 3rd-party
    # }
    container_registry_storage_info: Dict[str, Any] = field(default_factory=dict)
```

Deploying Your Own Container Registry:

The container registry can be deployed either on a Kubernetes cluster or outside of the cluster using the Docker `registry:2` container registry.

To deploy the container registry on the cluster, follow the steps below:

1. Switch to the following directory from the project root:

```
cd v1_deploy/container-registry
```

2. Use the cluster's kubectl to deploy the registry

```
# create namespace
kubectl create ns registry

# create pv
kubectl create -f pv.yaml

# create pvc
kubectl create -f pvc.yaml

# create deployment
kubectl create -f deployment.yaml

# create svc
kubectl create -f svc.yaml
```

You can modify these files to customize the storage, deployment port, etc. By default, the service creation will expose the `nodePort` 32633 on the cluster.

If you don't want to use Kubernetes, you can deploy the container image `registry:2` from Docker Hub using Docker on bare-metal:

```
docker run --rm -d --name="registry" -p "5000:5000" registry:2
```

It is not mandatory to use `registry:2`; you can use any container registry that implements the full OCI spec.

Onboarding the registry to the DB for listing:

1. Prepare the JSON Example:

```json
{
  "container_registry_id": "b3c38f52-8f2e-43a3-9aa9-bd9a4c75e509",
  "container_registry_name": "internal-ai-registry",
  "container_registry_metadata": {
    "environment": "production",
    "owner": "mlops-team",
    "region": "us-west-2"
  },
  "container_registry_tags": [
    "ai",
    "training",
    "internal",
    "fast-pull"
  ],
  "container_registry_public_url": "http://10.13.21.22:32633",
  "container_registry_image_prefix": "10.13.21.22:32633",
  "container_registry_config": {
    "auth_required": true,
    "mirror_enabled": false,
    "storage_backend": "s3",
    "max_layer_size_mb": 500
  },
  "container_registry_storage_info": {
    "storage_type": "s3",
    "storage_size": -1,
    "memory_size": 8192,
    "registry_type": "aios"
  }
}
```

2. Create an entry in the DB using the create API

```
curl -X POST http://registry-db-url/container_registry \
     -H "Content-Type: application/json" \
     -d @entry.json
```

Container registry DB APIs:

Endpoint: `/container_registry`
Method: `POST`
Description:
This endpoint creates a new container registry entry. It accepts a full JSON payload describing the registry, including metadata, configuration, tags, public URL, and storage information.

Example curl Command:

```
curl -X POST http://<registry-db-url>/container_registry \
     -H "Content-Type: application/json" \
     -d @entry.json
```

Endpoint: `/container_registry/<registry_id>`
Method: `GET`
Description:
Retrieves a single container registry entry using its unique `registry_id`. If the registry is found, the full data is returned as JSON. If not, an error message is returned.

Example curl Command:

```
curl -X GET http://<registry-db-url>/container_registry/<registry_id>
```

Endpoint: `/container_registry/<registry_id>`
Method: `PUT`
Description:
Updates fields of an existing container registry by its `registry_id`. The request body should include a JSON object with only the fields that need to be updated.

Example curl Command:

```
curl -X PUT http://<registry-db-url>/container_registry/<registry_id> \
     -H "Content-Type: application/json" \
     -d '{ "$set": {"container_registry_name": "updated-name"}}'
```

Endpoint: `/container_registry/<registry_id>`
Method: `DELETE`
Description:

Deletes the container registry entry identified by `registry_id`. If the registry exists, it will be removed from the database.

Example curl Command:

```
curl -X DELETE http://<registry-db-url>/container_registry/<registry_id>
```

Endpoint: `/container_registries`
Method: `POST`
Description:
Queries container registries based on filters provided in the request body. Supports flexible JSON queries for metadata, tags, or any field.

Example curl Command:

```
curl -X POST http://<registry-db-url>/container_registries \
     -H "Content-Type: application/json" \
     -d '{"container_registry_tags": {"$in": ["ai", "internal"]}}'
```

Endpoint: `/check_container_health/<registry_id>`
Method: `GET`
Description:
Performs a connectivity check for the container registry identified by `registry_id`. It verifies that the public URL is reachable and returns connection status and metadata.

Example curl Command:

```
curl -X GET http://<registry-db-url>/check_container_health/<registry_id>
```