Runtime Registries

AIOSv1 system hosts several registries for keeping track of the running blocks, clusters and vDAGs.

These are the following registries in AIOSv1:

1. Clusters registry - Clusters registry stores the information of all the clusters that are currently on-boarded into the network.

2. Blocks registry - Blocks registry stores the information of all the blocks that are currently running across all the clusters in the network.

3. vDAGs registry - vDAGs registry stores all the vDAGs that are currently created in the network across multiple blocks.

4. vDAG controller registry - vDAG controller registry stores all the vDAG controllers that are created to serve vDAG inference requests.

Clusters registry:

Clusters registry contains the information about the on-boarded clusters in the network.

Here is the schema of the cluster entry in clusters registry:

```
const clusterSchema = new Schema({
    // Unique ID for the cluster
    id: { type: String, required: true, unique: true },

    // Optional region or network ID where the cluster is deployed
    regionId: { type: String, required: false },

    status: { type: String, required: true },

    // Aggregated and per-node information for all nodes in the cluster
    nodes: {
        // Total number of nodes in the cluster
        count: { type: Number, required: true },
        // Detailed info for each individual node
        nodeData: [{
            // Unique ID for the node
            id: { type: String, required: true },
            // GPU details for the node
            gpus: {
                // Number of GPUs in the node
                count: { type: Number, required: true },
                // Total GPU memory in MB
                memory: { type: Number, required: true },
            // List of GPU models with individual memory sizes
            gpus: [{
                modelName: { type: String, required: true }, // GPU model name
                memory: { type: Number, required: true }     // Memory per GPU in MB
            }],
            // Optional GPU features (e.g., CUDA versions)
            features: [String],
            // List of distinct GPU model names
            modelNames: [String]
        },
        // Virtual CPU details
        vcpus: {
            count: { type: Number, required: true } // Number of vCPUs in the node
        },
        // Total memory in MB
        memory: { type: Number, required: true },
        // Total swap space in MB
        swap: { type: Number, required: true },
        // Storage info per node
        storage: {
            disks: { type: Number, required: true }, // Number of disks
            size: { type: Number, required: true }   // Total storage size in MB
        },
        // Network interface stats per node
        network: {
            interfaces: { type: Number, required: true },   // Number of network interf
            txBandwidth: { type: Number, required: true },  // Transmit bandwidth (MBps
            rxBandwidth: { type: Number, required: true }   // Receive bandwidth (MBps)
        }
    }]
},

// Total GPU stats across all nodes
gpus: {
    count: { type: Number, required: true },   // Total number of GPUs in the cluster
    memory: { type: Number, required: true }   // Total GPU memory in MB
},

// Total vCPU count across the cluster
vcpus: {
    count: { type: Number, required: true }
},

// Total memory across the cluster in MB
memory: { type: Number, required: true },
// Total swap space across the cluster in MB
swap: { type: Number, required: true },

// Aggregated storage details for the cluster
storage: {
    disks: { type: Number, required: true }, // Total number of disks
    size: { type: Number, required: true }   // Total storage size in MB
},

// Aggregated network configuration
network: {
    interfaces: { type: Number, required: true },  // Total number of interfaces
    txBandwidth: { type: Number, required: true }, // Total TX bandwidth
    rxBandwidth: { type: Number, required: true }  // Total RX bandwidth
},

// Configuration used by the cluster controller
config: {
    type: new Schema({
        policyExecutorId: { type: String, required: false, default: "" },      // Optio
        policyExecutionMode: { type: String, required: false, default: "local" }, // Exe
        customPolicySystem: { type: Schema.Types.Mixed, required: false },     // Any cu
        publicHostname: { type: String, required: true },                      // Publi
        useGateway: { type: Boolean, required: false, default: true },         // Wheth
        actionsPolicyMap: { type: Schema.Types.Mixed, required: false },       // Mappi
        // URLs to internal/external services in the cluster
        urlMap: {
            controllerService: { type: String, required: true },     // URL for control
            metricsService: { type: String, required: true },        // URL for metrics
            blocksQuery: { type: String, required: true },           // URL for querying
            publicGateway: { type: String, required: true },         // Public-facing ga
            parameterUpdater: { type: String, required: true }       // URL for model/cl
        }
    }),
    required: true
},

// List of user-defined tags or labels
tags: { type: [String], required: true },

// Human-readable metadata about the cluster
clusterMetadata: {
    type: new Schema({
        name: { type: String, required: true },                   // Friendly name of th
        description: { type: String, required: true },            // Purpose or use-case
        owner: { type: String, required: true },                  // Who owns or manages
```

```
            email: { type: String, required: false },                 // Optional contact em
            countries: { type: [String], required: false },           // Countries associat
            miscContactInfo: { type: Schema.Types.Mixed, required: false },  // Additional 
            additionalInfo: { type: Schema.Types.Mixed, required: false }    // Any extra m
        }),
        required: true
    },

    // Reputation score or reliability indicator for the cluster (not yet used anywhere in 
    reputation: { type: Number, required: false }
});
```

Example:

```
{
  "id": "cluster-west-vision-001",
  "regionId": "us-west-2",
  "status": "live",
  "nodes": {
    "count": 2,
    "nodeData": [
      {
        "id": "node-1",
        "gpus": {
          "count": 2,
          "memory": 32768,
          "gpus": [
            { "modelName": "NVIDIA A100", "memory": 16384 },
            { "modelName": "NVIDIA A100", "memory": 16384 }
          ],
          "features": ["fp16", "tensor_cores"],
          "modelNames": ["NVIDIA A100"]
        },
        "vcpus": { "count": 32 },
        "memory": 131072,
        "swap": 8192,
        "storage": {
          "disks": 2,
          "size": 1048576
        },
        "network": {
          "interfaces": 2,
          "txBandwidth": 10000,
          "rxBandwidth": 10000
        }
      },
      {
```

```
      "id": "node-2",
      "gpus": {
        "count": 1,
        "memory": 16384,
        "gpus": [
          { "modelName": "NVIDIA V100", "memory": 16384 }
        ],
        "features": ["fp16"],
        "modelNames": ["NVIDIA V100"]
      },
      "vcpus": { "count": 16 },
      "memory": 65536,
      "swap": 4096,
      "storage": {
        "disks": 1,
        "size": 524288
      },
      "network": {
        "interfaces": 1,
        "txBandwidth": 5000,
        "rxBandwidth": 5000
      }
    }
  ]
},
"gpus": {
  "count": 3,
  "memory": 49152
},
"vcpus": {
  "count": 48
},
"memory": 196608,
"swap": 12288,
"storage": {
  "disks": 3,
  "size": 1572864
},
"network": {
  "interfaces": 3,
  "txBandwidth": 15000,
  "rxBandwidth": 15000
},
"config": {
  "policyExecutorId": "policy-exec-007",
  "policyExecutionMode": "local",
    "customPolicySystem": {
      "name": "AdvancedPolicyRunner",
      "version": "2.1.0"
    },
    "publicHostname": "cluster-west-vision-001.company.net",
    "useGateway": true,
    "actionsPolicyMap": {
      "onScaleUp": "evaluate-gpu-availability",
      "onFailure": "notify-admin"
    },

    // these fields are populated by the system:
    "urlMap": {
      "controllerService": "http://cluster-west-vision-001.company.net:32000/controller",
      "metricsService": "http://cluster-west-vision-001.company.net:32000/metrics",
      "blocksQuery": "http://cluster-west-vision-001.company.net:32000/blocks",
      "publicGateway": "http://cluster-west-vision-001.company.net:32000",
      "parameterUpdater": "http://cluster-west-vision-001.company.net:32000/mgmt"
    }
  },
  "tags": ["gpu", "production", "ml", "vision", "us-west"],
  "clusterMetadata": {
    "name": "Sample cluster",
    "description": "Dedicated to serving large-scale computer vision models in production."
    "owner": "AI Infrastructure Team",
    "email": "ai-infra@company.net",
    "countries": ["USA", "Canada"],
    "miscContactInfo": {
      "pagerDuty": "https://sample-website/ai-clusters",
      "slack": "#ml-infra"
    },
    "additionalInfo": {

    }
  },
  "reputation": 94
}
```

Creating a cluster:

For creating the cluster, refer to the documentation of Parser.

Cluster registry APIs:

Endpoint: `/clusters/:id`
Method: `GET`

Description:
Fetches a single cluster document by its unique `id`.

Example curl Command:

```
curl -X GET http://<server-url>/clusters/cluster-west-vision-001
```

Endpoint: `/clusters/:id`
Method: `PUT`
Description:
Updates a cluster document by its `id` using the payload provided in the request body. The body should use MongoDB-style update syntax.

Example curl Command:

```
curl -X PUT http://<server-url>/clusters/cluster-west-vision-001 \
  -H "Content-Type: application/json" \
  -d '{
    "$set": {
      "tags": ["gpu", "updated"],
      "reputation": 97
    }
  }'
```

Endpoint: `/clusters/:id`
Method: `DELETE`
Description:
Deletes the cluster document with the specified `id`.

Example curl Command:

```
curl -X DELETE http://<server-url>/clusters/cluster-west-vision-001
```

Endpoint: `/clusters/query`
Method: `POST`
Description:
Queries cluster documents using a MongoDB-style filter provided in the request body. Supports standard MongoDB operators such as `$eq`, `$gt`, `$in`, etc.

Example curl Command:

```
curl -X POST http://<server-url>/clusters/query \
  -H "Content-Type: application/json" \
  -d '{
    "gpus.count": { "$gte": 2 },
    "clusterMetadata.countries": { "$in": ["USA"] }
  }'
```

Blocks registry:

Blocks registry stores the information of all the blocks that are currently running across all the clusters in the network.

Here is the schema of the block:

```
const BlockSchema = new mongoose.Schema({
    // Unique identifier for the block
    id: { type: String, required: true, unique: true },

    // the component URI the block is running - taken from component registry
    componentUri: { type: String },

    // The component data of the block - copied from component registry
    component: { type: mongoose.Schema.Types.Mixed },

    // same as componentUri + block-id
    blockUri: { type: String },

    // Human-readable or structured metadata about the block - copied from component
    blockMetadata: { type: mongoose.Schema.Types.Mixed },

    // Policy configuration or rules tied to the block
    policies: { type: mongoose.Schema.Types.Mixed },

    // The cluster data of the block, copied as it is from the cluster registry
    cluster: { type: mongoose.Schema.Types.Mixed },

    // Data used to initialize the block during deployment/startup
    blockInitData: { type: mongoose.Schema.Types.Mixed },

    // Initialization settings (env vars, args, flags, etc.)
    initSettings: { type: mongoose.Schema.Types.Mixed },

    // Parameters required to configure the block's runtime behavior
    parameters: { type: mongoose.Schema.Types.Mixed },

    // Minimum number of instances this block should maintain
    minInstances: { type: Number, required: false },

    // Maximum number of instances allowed for scaling
    maxInstances: { type: Number, required: false },

    // Input interface specification (protocol - follows a template - copied from component,
    inputProtocol: { type: mongoose.Schema.Types.Mixed },

    // Output interface specification (protocol - copied from component)
    outputProtocol: { type: mongoose.Schema.Types.Mixed }
});
```

Example:

```
{
  "id": "block-object-detector-001",
  "componentUri": "",
  "component": {},
  "blockUri": "",
  "blockMetadata": {},
  "policies": {
        "resourceAllocator": {
            "policyRuleURI": "policies.resource_allocator.standard:latest",
            "parameters": {},
            "settings": {}
        },
        "loadBalancer": {
            "policyRuleURI": "policies.load_balancer.gateway.load_balancer_sep2:v0.0.1-beta'
            "parameters": {},
            "settings": {}
        },
        "loadBalancerMapper": {
            "policyRuleURI": "policies.load_balancer.mapper.loadbalancer_mapper_oct1:v1.2.0'
            "parameters": {},
            "settings": {}
        },
        "assignment": {
            "policyRuleURI": "policies.assignment.default_strategy:v1.0.3",
            "parameters": {},
            "settings": {}
        },
        "stabilityChecker": {
            "policyRuleURI": "policies.health.stability_checker:v0.1.0",
            "parameters": {},
            "settings": {}
        },
        "autoscaler": {
            "policyRuleURI": "policies.autoscaler.basic_auto_scaler:v0.3.5",
            "parameters": {},
            "settings": {}
        },
        "accessRulesPolicy": {
            "policyRuleURI": "policies.access.control.access_rules_policy:v2.0.0",
          "parameters": {},
          "settings": {}
        }
    },
  "cluster": {},
  "blockInitData": {},
  "initSettings": {},
  "parameters": {},
  "minInstances": 1,
  "maxInstances": 5,
  "inputProtocol": {},
  "outputProtocol": {}
}
```

Creating a block:

For creating the block, refer to the documentation of Parser.

Block registry APIs:

Endpoint: `/blocks`
Method: `GET`
Description:
Fetches all block documents in the database.

Example curl Command:

```
curl -X GET http://<server-url>/blocks
```

Endpoint: `/blocks/:id`
Method: `GET`
Description:
Fetches a single block document by its unique `id`.

Example curl Command:

```
curl -X GET http://<server-url>/blocks/block-object-detector-001
```

Endpoint: `/blocks/:id`
Method: `PUT`
Description:
Updates a block document by its `id` using MongoDB-style update syntax in the request body.

Example curl Command:

```
curl -X PUT http://<server-url>/blocks/block-object-detector-001 \
  -H "Content-Type: application/json" \
  -d '{
    "$set": {
      "blockMetadata.description": "Updated description for object detection block",
      "minInstances": 2
    }
  }'
```

Endpoint: `/blocks/:id`
Method: `DELETE`
Description:
Deletes the block document with the specified `id`.

Example curl Command:

```
curl -X DELETE http://<server-url>/blocks/block-object-detector-001
```

Endpoint: `/blocks/query`
Method: `POST`
Description:
Queries block documents using a MongoDB-style filter provided in the JSON body. Supports standard MongoDB operators such as `$eq`, `$gt`, `$in`, etc. Optional `options` can be passed for sorting, pagination, etc.

Example curl Command:

```
curl -X POST http://<server-url>/blocks/query \
  -H "Content-Type: application/json" \
  -d '{
    "query": {
      "cluster.reputation": { "$gt": 90 },
      "policies.autoscaler.policyRuleURI": { "$ne": "" }
    },
    "options": {
      "sort": { "id": 1 },
      "limit": 10
    }
  }'
```

Endpoint: `/vdag/:vdagURI`
Method: `PUT`
Description:
Updates fields in the vDAG document identified by the given `vdagURI` using MongoDB-style update syntax.

Example curl Command:

```
curl -X PUT http://<server-url>/vdag/sample-vdag:1.0-stable \
     -H "Content-Type: application/json" \
     -d '{
           "$set": {
             "status": "active",
             "metadata.owner": "team-ml"
           }
         }'
```

Endpoint: `/vdag/:vdagURI`
Method: `DELETE`
Description:
Deletes the vDAG document identified by the given `vdagURI`.

Example curl Command:

```
curl -X DELETE http://<server-url>/vdag/sample-vdag:1.0-stable
```

Endpoint: `/vdags`
Method: `POST`
Description:
Queries multiple vDAG documents using a MongoDB-style filter object.

Example curl Command:

```
curl -X POST http://<server-url>/vdags \
     -H "Content-Type: application/json" \
     -d '{
           "status": "pending",
           "metadata.owner": "team-ml"
         }'
```

---

vDAG controllers registry:

vDAG controller registry stores all the vDAG controllers that are created to serve vDAG inference requests.

Here is the schema of a vDAG controller:

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

@dataclass
class vDAGController:
    # Unique identifier for the vDAG controller instance
    vdag_controller_id: str = ''
    # Associated vDAG URI this controller is managing
    vdag_uri: str = ''
    # Publicly accessible URL for interacting with the controller
    public_url: str = ''
    # Identifier of the cluster where the controller is deployed
    cluster_id: str = ''
    # Arbitrary metadata for storing additional information
    metadata: Dict[str, Any] = field(default_factory=dict)
    # Configuration parameters used by the controller
    config: Dict[str, Any] = field(default_factory=dict)
    # Tags used for search and discovery of the controller
    search_tags: List[str] = field(default_factory=list)
```

Creating a vDAG controller:

For creating the vDAG controller, refer to the documentation of Parser.

vDAG controllers registry APIs:

Endpoint: `/vdag-controller/:controller_id`
Method: `GET`
Description:
Fetches the vDAG Controller document identified by the given `controller_id`.

Example curl Command:

```
curl -X GET http://<server-url>/vdag-controller/controller-123
```

Endpoint: `/vdag-controller/:controller_id`
Method: `PUT`
Description:
Updates fields in the vDAG Controller document identified by the given `controller_id` using MongoDB-style update syntax.

Example curl Command:

```
curl -X PUT http://<server-url>/vdag-controller/controller-123 \
     -H "Content-Type: application/json" \
     -d '{
           "$set": {
             "metadata.owner": "team-alpha",
             "public_url": "https://controller.example.com"
           }
         }'
```

Endpoint: `/vdag-controller/:controller_id`
Method: `DELETE`
Description:
Deletes the vDAG Controller document identified by the given `controller_id`.

Example curl Command:

```
curl -X DELETE http://<server-url>/vdag-controller/controller-123
```

Endpoint: `/vdag-controllers`
Method: `POST`
Description:
Queries multiple vDAG Controller documents using a MongoDB-style filter object.

Example curl Command:

```
curl -X POST http://<server-url>/vdag-controllers \
     -H "Content-Type: application/json" \
     -d '{
           "cluster_id": "cluster-west-1",
           "metadata.owner": "team-alpha"
         }'
```

Endpoint: `/vdag-controllers/by-vdag-uri/:vdag_uri`
Method: `GET`
Description:
Fetches all vDAG Controller documents associated with the given `vdag_uri`.

Example curl Command:

```
curl -X GET http://<server-url>/vdag-controllers/by-vdag-uri/sample-vdag:1.0-stable
```
