# Cluster controller and Nodes onboarding:

Cluster controllers and nodes to the existing cluster can be on-boarded by submitting a cluster controller spec, these specifications are validated and submitted to the cluster controller gateway, which runs the pre-checks and on-boards the cluster.

## Cluster specification:

Before onboarding the cluster, the basic kubernetes infrastructure needs to be setup by the cluster onboarding entity on the target cluster, refer to the "Onboarding cluster" documentation in Onboarding document for more details. Once the cluster infrastructure is setup, the specification needs to be prepared as per the template of choice and can be on-boarded using the parser API.

Perfect! Below is the **Cluster Specification Documentation** followed by a table of **Suggested Policies** that can be written to operate on or validate cluster specs.

---

## Cluster Specification

The cluster specification defines the structure and requirements for provisioning a new cluster in the system. It includes identifiers, resource allocation, configuration metadata, and policy-related runtime information.

### Top-Level Fields

| Field | Type | Required | Description |
|---|---|---|---|
| `id` | string | Yes | Unique identifier for the cluster. |
| `regionId` | string | Yes | The deployment region for the cluster. |
| `nodes` | object | Yes | Details about the nodes within the cluster. |
| `gpus` | object | Yes | Aggregate GPU count and memory across the cluster. |
| `vcpus` | object | Yes | Total vCPU count available in the cluster. |
| `memory` | number | Yes | Total RAM (in MB) available in the cluster. |
| `swap` | number | No | Total swap memory (in MB). |
| `storage` | object | Yes | Aggregate storage configuration. |
| `network` | object | Yes | Network interface and bandwidth configuration. |
| `config` | object | No | Runtime configuration and policy integration data. |
| `tags` | array | No | Tags to classify the cluster (e.g., `ml`, `gpu`, `production`). |
| `clusterMetadata` | object | No | Human-readable metadata for documentation and ownership. |

| Field | Type | Required | Description |
|---|---|---|---|
| `reputation` | number | No | System-defined reputation score for this cluster (e.g., 0-100). |

---

### Nested Field Structures

**`nodes`**

| Field | Type | Required | Description |
|---|---|---|---|
| `count` | number | Yes | Number of nodes in the cluster. |
| `nodeData` | array | No | Array of detailed node specifications. |

Each entry in `nodeData` includes:

- `id`, `gpus`, `vcpus`, `memory`, `swap`, `storage`, `network`

**`gpus`**

| Field | Type | Required | Description |
|---|---|---|---|
| `count` | number | Yes | Total number of GPUs. |
| `memory` | number | Yes | Total GPU memory in MB. |

**`vcpus`**

| Field | Type | Required | Description |
|---|---|---|---|
| `count` | number | Yes | Total vCPU cores. |

**`storage`**

| Field | Type | Required | Description |
|---|---|---|---|
| `disks` | number | Yes | Number of physical disks. |
| `size` | number | Yes | Total size in MB. |

**`network`**

| Field | Type | Required | Description |
|---|---|---|---|
| `interfaces` | number | Yes | Number of network interfaces. |
| `txBandwidth` | number | Yes | Transmission bandwidth in Mbps. |
| `rxBandwidth` | number | Yes | Reception bandwidth in Mbps. |

`config`

| Field | Type | Required | Description |
|---|---|---|---|
| `policyExecutorId` | string | No | Identifier of the policy executor. |
| `policyExecutionMode` | string | No | Execution mode (`local`, `distributed`). |
| `customPolicySystem` | object | No | Details of custom runtime (e.g., name/version). |
| `publicHostname` | string | No | Hostname used for exposing services. |
| `useGateway` | boolean | No | Indicates if the cluster is exposed via a gateway. |
| `actionsPolicyMap` | object | No | Maps events (e.g., `cluster policies`) to policies. |
| `urlMap` | object | No | Auto-populated service URLs (system-generated). |

Excellent! Here's a clean and structured documentation section you can include under your cluster spec or config section to document the `actionsPolicyMap` inside `config`.

---

`config.actionsPolicyMap`

The `actionsPolicyMap` is an optional configuration field under `config` that maps specific system-level **actions** to corresponding **policy rule URIs**. These policies are invoked automatically during various control plane or runtime operations (e.g., block creation, scaling, parameter updates).

Each action listed below is recognized by a specific system component and may trigger a policy execution during the cluster or block lifecycle.

**Supported Actions**

| **Action** | **Description** | **Triggered By** |
|---|---|---|
| `remove_block` | Removes a specified block from the system. | Cluster Controller Gateway |
| `create_block` | Creates a new block using the specified configuration. | Cluster Controller Gateway |

| Action | Description | Triggered By |
|---|---|---|
| `parameter_update` | Updates parameters of an existing component or block. | Management Command Executor |
| `scale` | Adjusts the number of block instances for scaling up or down. | Auto-scaler, Cluster Controller Gateway |
| `dry_run` | Simulates an operation without executing it, for validation purposes. | Cluster Controller Gateway |
| `remove_instance` | Removes a specific runtime instance from the system. | Cluster Controller Gateway |
| `init_create_status_update` | Updates the status during the initialization phase of an LLM container. | Cluster Controller Gateway (LLM Support) |
| `query_init_container_data` | Queries current state or metadata from the LLM init container. | Cluster Controller Gateway (LLM Support) |
| `reassign-instances` | Reassigns instances between blocks/components for load balancing or failover. | Dynamic Infrastructure Scanner |

---

**Example Usage:**

```
"actionsPolicyMap": {
  "create_block": "policies.cluster.block-creation-policy:v1",
  "scale": "policies.autoscaling.default-scaler-policy:v2",
  "parameter_update": "policies.params.param-validator:v1",
  ...
}
```

---

`clusterMetadata`

| Field | Type | Description |
|---|---|---|
| `name` | string | Human-friendly name for the cluster. |
| `description` | string | Description of the cluster's purpose. |
| `owner` | string | Owner/team responsible for this cluster. |
| `email` | string | Contact email for support. |
| `countries` | array | Allowed countries for usage. |
| `miscContactInfo` | object | Additional contacts (e.g., Slack, PagerDuty). |
| `additionalInfo` | object | Free-form extension metadata. |

---

**Example specification:**

```
{
  "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"
    }
  },
```

```
  "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
}
```

---

## Pre-check Policies

**Pre-check policies** are customizable rule sets, authored in Python, that evaluate and authorize actions prior to their execution. These policies serve as a governance mechanism, enabling cluster administrators and developers to enforce cluster-specific constraints and compliance rules.

By implementing pre-check policies, the system ensures that only authorized operations are performed.

---

**Writing a pre-check policy:**

The pre-check policy rule should return a dict containing following fields:

```
{
    "allowed": True,
    "input_data": input_data  # the modified input data, if not return the input data as it
}
```

If not allowed:

```
{
    "allowed": False,
    "input_data": <message or dict containing the reason data of why the action was not allo
}
```

The Boolean key `allowed` tells whether the execution of the given action should proceed or not, also the input_data that is passed to the policy rule can be

tweaked by the pre-check policy rule, thus the input_data field should contain the updated version of the input dictionary passed to the policy rule, if no modifications are made, return the input_data as it is in this field. Here is the structure of the policy rule that can be used as a pre-check:

```
class AIOSv1PolicyRule:

    def __init__(self, rule_id, settings, parameters):

        """
            Initializes an AIOSv1PolicyRule instance.
            Args:
            rule_id (str): Unique identifier for the rule.
            settings (dict): Configuration settings for the rule.
            parameters (dict): Parameters defining the rule's behavior.

        """
        self.rule_id = rule_id
        self.settings = settings
        self.parameters = parameters

    def eval(self, parameters, input_data, context):

        """
            Evaluates the policy rule.
            This method should be implemented by subclasses to define the   rule's logic.
            It takes parameters, input data, and a context object to perform  evaluation.
            Args:
                parameters (dict): The current parameters.
                input_data (any): The input data to be evaluated.
                context (dict): Context (external cache), this can be used for  storing and
        """
        # the input_data dict can be modified by the policy
        # make input_data dict modifications here

        return {
            "allowed": True,
            "input_data": input_data
        }
```

---

## Node Onboarding Specification

Nodes are onboarded into an existing cluster by submitting their hardware and system configuration to the Parser API. The structure below defines the required fields for node onboarding.

The `add-node` action expects a `nodeData` object that represents the node's hardware and runtime characteristics.

---

**Top-Level Structure**

| Field | Type | Required | Description |
|---|---|---|---|
| `id` | string | Yes | Unique identifier of the node. Typically injected from the environment. |
| `clusterId` | string | Yes* | ID of the cluster to which the node belongs. Required if called via Parser. *(Not part of the auto-register script but required in IR)* |
| `gpus` | object | Yes | GPU device configuration and summary. |
| `vcpus` | object | Yes | Logical CPU count. |
| `memory` | number | Yes | Total physical RAM (in MB). |
| `swap` | number | No | Total swap memory (in MB). |
| `storage` | object | Yes | Storage configuration including disk count and size. |
| `network` | object | Yes | Network configuration including interface count. |
| `tags` | array | No | Classification tags for the node (e.g., "gpu", "fp16"). |
| `nodeMetadata` | object | No | Additional metadata for traceability and diagnostics. |

---

**Nested Structures**

`gpus`

| Field | Type | Required | Description |
|---|---|---|---|
| `count` | number | Yes | Total number of GPU devices. |
| `memory` | number | Yes | Total GPU memory in MB. |
| `gpus` | array | Yes | List of individual GPUs with model and memory. |
| `modelNames` | array | No | Unique set of GPU model names. |
| `features` | array | No | Optional list of GPU features (e.g., "tensor_cores"). |

**Individual GPU object:**

```
{
  "modelName": "NVIDIA A100",
  "memory": 16384
}
```

**vcpus**

| Field | Type | Required | Description |
|---|---|---|---|
| `count` | number | Yes | Number of logical CPU cores. |

**storage**

| Field | Type | Required | Description |
|---|---|---|---|
| `disks` | number | Yes | Number of physical disk partitions. |
| `size` | number | Yes | Total storage size in MB. |

**network**

| Field | Type | Required | Description |
|---|---|---|---|
| `interfaces` | number | Yes | Number of network interfaces detected. |
| `txBandwidth` | number | No | Placeholder for transmit bandwidth (0). |
| `rxBandwidth` | number | No | Placeholder for receive bandwidth (0). |

**tags**

| Type | Description |
|---|---|
| array | Optional list of classification labels (e.g., `["gpu"]`). |

**nodeMetadata**

| Type | Description |
|---|---|
| object | Optional key-value metadata. Useful for tracking vendor, rack, etc. |

**Example Node Onboarding Payload**


```
{
  "header": {
    "templateUri": "Parser/V1",
    "parameters": {}
  },
  "body": {
    "spec": {
      "values": {
        "clusterId": "cluster-west-vision-001",
        "id": "node-1",
        "gpus": {
          "count": 2,
          "memory": 32768,
          "gpus": [
            { "modelName": "NVIDIA A100", "memory": 16384 },
            { "modelName": "NVIDIA A100", "memory": 16384 }
          ],
          "modelNames": ["NVIDIA A100"],
          "features": ["tensor_cores"]
        },
        "vcpus": { "count": 32 },
        "memory": 131072,
        "swap": 8192,
        "storage": { "disks": 2, "size": 1048576 },
        "network": { "interfaces": 2, "txBandwidth": 0, "rxBandwidth": 0 },
        "tags": ["gpu", "fp16", "production"],
        "nodeMetadata": {
          "vendor": "Supermicro",
          "location": "Rack 2 - DC1",
          "notes": "Installed 2024-12"
        }
      }
    }
  }
}
```


---

## Using the parser:

Sure! Below are the `curl` requests for **adding a node** and **creating a cluster** using the Parser's API. Each uses a local JSON file as input (`node_data.json` and `cluster.json` respectively).

---

**1. Add Node – `node_data.json`**

```
curl -X POST http://<parser-host>:<port>/api/addNode \
  -H "Content-Type: application/json" \
  -d @node_data.json
```

Replace `<parser-host>:<port>` with your actual parser API endpoint.

---

**2. Create Cluster – `cluster.json`**

```
curl -X POST http://<parser-host>:<port>/api/createCluster \
  -H "Content-Type: application/json" \
  -d @cluster.json
```

Make sure `node_data.json` and `cluster.json` follow the proper Parser API request format (with `header` and `body.spec.values` fields).