Hardware accelerators

Use this section to choose the right accelerator path for container workloads, understand the ACP concepts that affect resource requests, and continue to the product documentation for installation, upgrade, compatibility, and troubleshooting.

Start from the scenario closest to your task. Vendor integrations prepare hardware, drivers, runtimes, and direct allocation. Sharing and scheduling products such as HAMi build on those integrations to provide sharing, virtualization, scheduling, or resource orchestration behavior.

TOC

Choose a scenario

If you want to...	Start here
Use NVIDIA GPUs with vendor-standard components	GPU accelerator family and NVIDIA vendor integration

If you want to...	Start here
Run container workloads on Ascend NPUs with vendor-standard components	NPU accelerator family and Ascend vendor integration
Share, virtualize, or schedule GPU or NPU devices across workloads	HAMi sharing and scheduling path
Use Kubernetes DRA-based resource claims	DRA mechanism
Show accelerator resources in ACP quota pages or configure quota field metadata	Accelerator Resource Quota
Use GPU passthrough for virtual machines	Use Virtual Machine physical GPU configuration. This is a different path from container workload pGPU or vGPU usage.

Browse by category

- Overview: concepts such as Device Plugin, DRA, RuntimeClass, CDI, extended resources, quotas, and monitoring entry points.
- Accelerator families: User-facing hardware families such as GPU and NPU.
- Products: vendor integrations such as NVIDIA and Ascend, and sharing or scheduling products such as HAMi.

Related ACP pages

For quota, plugin installation, monitoring dashboards, virtual machine GPU passthrough, and billing, continue with the corresponding ACP feature documentation:

- Project cluster quota
- Namespace resource quota
- Cluster Plugin

- Monitoring dashboards
- Virtual Machine physical GPU configuration
- Alauda Cost Management

Overview

These pages explain the Kubernetes and ACP concepts that affect accelerator usage.

Use this section when you need to understand why a workload requests a certain resource name, why a runtime integration is required, or why different accelerator products expose resources in different ways. For product installation and troubleshooting, continue to the product page linked from each topic.

Device Plugin

When to use it

Related products

Related pages

DRA

How it affects workloads

Product documentation

Related products

Related pages

RuntimeCla

How it affects w

Related produc

Related pages

Accelerator Quota

Quota resources

Show quota fields in ACP

Examples

Configure quota

Quota pages

Related pages

Accelerator

What you can n

Metric sources

Related pages

Device Plugin

Device Plugin is the Kubernetes mechanism most accelerator products use to advertise device capacity to the scheduler and make selected devices available to Pods.

In ACP, Device Plugin matters because GPU and NPU products usually expose extended resources through this path. After the corresponding product is installed, users can request resources such as NVIDIA GPU, HAMi virtual GPU, or Ascend NPU resources from their workloads and manage quota through project and namespace settings.

TOC

When to use it

- Use Device Plugin based products when workloads request accelerator resources through Kubernetes extended resource names.
- Use ACP project and namespace pages for quota operations.
- Use the product documentation for installation, node labels, supported hardware, and troubleshooting.

Related products

- NVIDIA GPU Device Plugin exposes physical NVIDIA GPU resources.
- HAMi uses device plugin and scheduling components to provide shared accelerator resources.
- Ascend NPU Operator includes components that expose Ascend devices to Kubernetes workloads.

Related pages

- GPU accelerator family
- NPU accelerator family
- Resource quota

DRA

Dynamic Resource Allocation (DRA) is a Kubernetes mechanism for requesting and allocating devices through resource claim objects. It is not a GPU-only capability.

In ACP, DRA is used through a product driver. A DRA driver is an implementation of the DRA mechanism, not the mechanism itself.

TOC

How it affects workloads

DRA is relevant when:

- a workload uses `ResourceClaim` or `ResourceClaimTemplate` instead of only extended resource names;
- the accelerator product documents mention `DeviceClass`, `ResourceSlice`, or claim-based allocation;

- the cluster has both traditional Device Plugin based products and DRA based products, and you need to identify which allocation path a workload uses.

Product documentation

Use this page to understand what DRA means for workload resource requests and how to distinguish DRA from traditional Device Plugin based allocation. Use the product documentation for DRA driver installation, Kubernetes version requirements, feature gates, node labels, claim examples, validation, and troubleshooting.

Related products

Only document a DRA driver in a product site after the corresponding Alauda product or package explicitly supports it.

- A vendor DRA driver belongs to the corresponding vendor integration.
- A HAMi DRA driver belongs to the HAMi sharing and scheduling path because HAMi owns the sharing, virtualization, scheduling, and resource semantics.
- Future GPU, NPU, or other accelerator products may also use DRA.

Related pages

- NVIDIA vendor integration
- HAMi sharing and scheduling path
- GPU accelerator family
- NPU accelerator family
- RuntimeClass and CDI

RuntimeClass and CDI

RuntimeClass and Container Device Interface (CDI) are runtime integration mechanisms. They determine how selected devices and required runtime files are made visible inside containers after the scheduler has selected a node and device.

These mechanisms are relevant to accelerator users because a workload can be scheduled successfully but still fail at runtime if the device runtime integration is missing or incompatible.

TOC

How it affects workloads

- Device Plugin and DRA decide how devices are advertised and allocated.
- RuntimeClass and CDI decide how the container runtime injects device files, libraries, and runtime configuration.
- Product documentation describes whether a workload must set `runtimeClassName`, whether CDI is required, and which container runtime versions are supported.

Related products

- NVIDIA products may depend on NVIDIA Container Toolkit and CDI or a NVIDIA runtime handler, depending on the selected product path.
- Ascend NPU products may use CDI or vendor runtime components to inject Ascend devices and libraries.
- HAMi may integrate with runtime components differently depending on device type and product version.

Related pages

- Device Plugin
- DRA
- GPU accelerator family
- NPU accelerator family

Accelerator Quota

Accelerator products usually expose resources through Kubernetes extended resource names. After the product is installed, ACP can use those resource names in project and namespace quota.

TOC

Quota resources

After an accelerator product is installed in a cluster, ACP can show corresponding accelerator quota fields at project or namespace scope. The exact resource names and units depend on the product.

Examples include:

- physical NVIDIA GPU count;

- HAMi virtual GPU core and memory resources;
- Ascend NPU resources;
- DRA or vendor-specific resource types.

Show quota fields in ACP

Kubernetes node allocatable resources and ACP quota fields are related but not the same. A node can report an accelerator extended resource, while the ACP quota page still needs resource metadata before it can show the resource as a selectable quota field.

If the node already reports the accelerator resource but the ACP quota page does not show it, register the resource metadata with a ConfigMap in the `kube-public` namespace.

> **NOTE**
>
> Required rules:
>
> - Create one ConfigMap for each accelerator resource name.
> - Create the ConfigMap in the `kube-public` namespace.
> - Use the `cf-crl-<groupName>-<name>` format for `metadata.name`.
> - Set `metadata.labels.features.cpaas.io/type` to `CustomResourceLimitation`.
> - Set `metadata.labels.features.cpaas.io/group` and `data.group` to the same group name.
> - Set `metadata.labels.features.cpaas.io/enabled` to `"true"`.
> - Set `data.key` to the exact Kubernetes extended resource name, such as `nvidia.com/gpualloc` or `huawei.com/Ascend910`.

Use the following fields to control how ACP displays and applies the resource in quota pages:

- `data.labelEn` and `data.labelZh`: display names shown to users.
- `data.descriptionEn` and `data.descriptionZh`: help text shown to users.
- `data.limits`: set to `required` or `optional`.

- `data.requests`: set to `disabled` or `fromLimits`.
- `data.resourceUnit`: resource unit, such as `count`.
- `data.relatedResources`: resource names that should be used together.
- `data.excludeResources`: resource names that should not be mixed with this resource.
- `data.ignoreNodeCheck`: set to `"true"` when the quota field is derived from another resource and does not need direct node allocatable checking.

If a product exposes multiple resource names, create one ConfigMap for each resource name. For example, shared GPU products may expose separate resource names for GPU core and GPU memory.

Examples

HAMi NVIDIA

The following example registers HAMi NVIDIA resource fields for GPU count, vGPU core, and vGPU memory:

```
apiVersion: v1
data:
  dataType: integer
  defaultValue: "1"
  descriptionEn: Declare how many physical GPUs needs
  descriptionZh: 申请的物理 gpu 个数
  group: hami-nvidia
  groupI18n: '{"zh": "HAMi NVIDIA", "en": "HAMi NVIDIA"}'
  key: nvidia.com/gpualloc
  labelEn: gpu number
  labelZh: gpu 个数
  limits: optional
  requests: disabled
  resourceUnit: "count"
  relatedResources: "nvidia.com/gpucores,nvidia.com/gpumem"
  excludeResources: "nvidia.com/mps-core,nvidia.com/mps-memory,tencen
t.com/vcuda-core,tencent.com/vcuda-memory"
  runtimeClassName: ""
kind: ConfigMap
metadata:
  labels:
    features.cpaas.io/enabled: "true"
    features.cpaas.io/group: hami-nvidia
    features.cpaas.io/type: CustomResourceLimitation
  name: cf-crl-hami-nvidia-gpualloc
  namespace: kube-public
---
apiVersion: v1
data:
  dataType: integer
  defaultValue: "20"
  descriptionEn: Declare how many cores needs per physical GPU (%)
  descriptionZh: 每个物理 GPU 申请的算力（%）
  group: hami-nvidia
  groupI18n: '{"zh": "HAMi NVIDIA", "en": "HAMi NVIDIA"}'
  key: nvidia.com/gpucores
  labelEn: vgpu cores
  labelZh: vgpu 算力
  limits: optional
  requests: disabled
  resourceUnit: "%"
  relatedResources: "nvidia.com/gpualloc,nvidia.com/gpumem"
  excludeResources: "nvidia.com/mps-core,nvidia.com/mps-memory,tencen
```

```
t.com/vcuda-core,tencent.com/vcuda-memory"
  runtimeClassName: ""
  ignoreNodeCheck: "true"
kind: ConfigMap
metadata:
  labels:
    features.cpaas.io/enabled: "true"
    features.cpaas.io/group: hami-nvidia
    features.cpaas.io/type: CustomResourceLimitation
  name: cf-crl-hami-nvidia-gpucores
  namespace: kube-public
---
apiVersion: v1
data:
  dataType: integer
  defaultValue: "4000"
  descriptionEn: Declare how many memory needs per physical GPU (Mi)
  descriptionZh: 每个物理 GPU 申请的显存（Mi）
  group: hami-nvidia
  groupI18n: '{"zh": "HAMi NVIDIA", "en": "HAMi NVIDIA"}'
  key: nvidia.com/gpumem
  labelEn: vgpu memory
  labelZh: vgpu 显存
  limits: optional
  requests: disabled
  resourceUnit: "Mi"
  relatedResources: "nvidia.com/gpualloc,nvidia.com/gpucores"
  excludeResources: "nvidia.com/mps-core,nvidia.com/mps-memory,tencen
t.com/vcuda-core,tencent.com/vcuda-memory"
  runtimeClassName: ""
  ignoreNodeCheck: "true"
kind: ConfigMap
metadata:
  labels:
    features.cpaas.io/enabled: "true"
    features.cpaas.io/group: hami-nvidia
    features.cpaas.io/type: CustomResourceLimitation
  name: cf-crl-hami-nvidia-gpumem
  namespace: kube-public
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: cf-crl-hami-config
```

```
  namespace: kube-public
  labels:
    device-plugin.cpaas.io/config: "true"
data:
  deviceName: "HAMi"
  nodeLabelKey: "gpu"
  nodeLabelValue: "on"
```

HAMi Ascend vNPU

The following example registers HAMi Ascend vNPU resource fields for Ascend 310P vNPU count and vNPU memory:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: cf-crl-hami-ascend310p
  namespace: kube-public
  labels:
    features.cpaas.io/enabled: "true"
    features.cpaas.io/group: hami-ascend-vnpu
    features.cpaas.io/type: CustomResourceLimitation
data:
  dataType: integer
  defaultValue: "1"
  descriptionEn: Number of Ascend 310P vNPU devices requested
  descriptionZh: 申请的 Ascend 310P vNPU 数量
  group: hami-ascend-vnpu
  groupI18n: '{"zh":"HAMi Ascend vNPU","en":"HAMi Ascend vNPU"}'
  key: huawei.com/Ascend310P
  labelEn: Ascend 310P vNPU
  labelZh: Ascend 310P vNPU
  limits: optional
  requests: disabled
  resourceUnit: "count"
  relatedResources: "huawei.com/Ascend310P-memory"
  runtimeClassName: ""
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: cf-crl-hami-ascend310p-memory
  namespace: kube-public
  labels:
    features.cpaas.io/enabled: "true"
    features.cpaas.io/group: hami-ascend-vnpu
    features.cpaas.io/type: CustomResourceLimitation
data:
  dataType: integer
  defaultValue: "3072"
  descriptionEn: Ascend 310P vNPU memory requested per device
  descriptionZh: 每个 Ascend 310P vNPU 申请的显存
  group: hami-ascend-vnpu
  groupI18n: '{"zh":"HAMi Ascend vNPU","en":"HAMi Ascend vNPU"}'
  key: huawei.com/Ascend310P-memory
  labelEn: Ascend 310P vNPU memory
```

```
    labelZh: Ascend 310P vNPU 显存
    limits: optional
    requests: disabled
    resourceUnit: "Mi"
    relatedResources: "huawei.com/Ascend310P"
    runtimeClassName: ""
    ignoreNodeCheck: "true"
  ---
  apiVersion: v1
  kind: ConfigMap
  metadata:
    name: cf-crl-hami-ascend-vnpu-config
    namespace: kube-public
    labels:
      device-plugin.cpaas.io/config: "true"
  data:
    deviceName: "HAMi Ascend vNPU"
    nodeLabelKey: "ascend"
    nodeLabelValue: "on"
```

The memory quota field uses `ignoreNodeCheck: "true"` because it is used together with the vNPU device count field instead of being checked as an independent node allocatable device count.

Ascend 910

The following example registers the Ascend 910 resource reported as `huawei.com/Ascend910` and the device display metadata for Huawei NPU nodes:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: cf-crl-huawei-ascend910
  namespace: kube-public
  labels:
    features.cpaas.io/enabled: "true"
    features.cpaas.io/group: huawei-npu
    features.cpaas.io/type: CustomResourceLimitation
data:
  dataType: integer
  defaultValue: "1"
  descriptionEn: Number of Ascend 910 NPUs requested
  descriptionZh: 申请的 Ascend 910 NPU 数量
  group: huawei-npu
  groupI18n: '{"zh":"Huawei NPU","en":"Huawei NPU"}'
  key: huawei.com/Ascend910
  labelEn: Ascend 910
  labelZh: Ascend 910
  limits: optional
  requests: disabled
  resourceUnit: "count"
  runtimeClassName: ""
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: cf-crl-huawei-npu-config
  namespace: kube-public
  labels:
    device-plugin.cpaas.io/config: "true"
data:
  deviceName: "Huawei NPU"
  nodeLabelKey: "ascend"
  nodeLabelValue: "on"
```

For example, even if the NPU Operator reports `huawei.com/Ascend910` in node allocatable resources, ACP can show `Ascend 910` in quota pages only after the corresponding `CustomResourceLimitation` metadata is available.

```
{
  "huawei.com/Ascend910": "8"
}
```

NVIDIA GPU

The following example registers the physical NVIDIA GPU resource reported as `nvidia.com/gpu`:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cf-crl-nvidia-gpu
  namespace: kube-public
  labels:
    features.cpaas.io/enabled: "true"
    features.cpaas.io/group: nvidia-gpu
    features.cpaas.io/type: CustomResourceLimitation
data:
  dataType: integer
  defaultValue: "1"
  descriptionEn: Number of NVIDIA GPUs requested
  descriptionZh: 申请的 NVIDIA GPU 数量
  group: nvidia-gpu
  groupI18n: '{"zh":"NVIDIA GPU","en":"NVIDIA GPU"}'
  key: nvidia.com/gpu
  labelEn: NVIDIA GPU
  labelZh: NVIDIA GPU
  limits: optional
  requests: disabled
  resourceUnit: "count"
  runtimeClassName: ""
```

For other accelerator products, keep the same ACP metadata model and replace the resource `key`, labels, display names, unit, and device information with the values exposed by that product.

Configure quota

Use ACP quota pages for operations such as adding a cluster to a project, creating a namespace, or changing namespace quota. Use accelerator product documentation to understand which resource names a product exposes and what each unit means.

Quota pages

- Project cluster resource allocation
- Namespace resource quota

Related pages

- GPU accelerator family
- NPU accelerator family

Accelerator Monitoring

Accelerator monitoring in ACP connects product-provided metrics with ACP observability features.

TOC

What you can monitor

The exact metric names and dashboards depend on the accelerator product installed in the cluster. ACP observability pages explain how to view dashboards, query metrics, and configure alerts.

Metric sources

- NVIDIA GPU metrics may come from DCGM Exporter or product-specific collectors.
- HAMi may provide virtual GPU allocation and usage metrics.
- Ascend NPU products may provide NPU exporter metrics.

Related pages

- Use ACP Observability documentation for monitoring concepts and dashboard management.
- Use Alauda Cost Management documentation for GPU or NPU billing models.
- Use each accelerator product site for product-specific metric names and dashboards.

Accelerator Families

Start here when you know the type of accelerator hardware you want to use, such as GPU or NPU, and need to choose a product path.

GPU

- Choose a GPU path
- Containers and VMs
- Related pages

NPU

- Choose an NPU path
- Use NPU resources
- Product documentation
- Related pages

GPU

Use this page when you want to run container workloads on GPUs or choose the right GPU product path in ACP.

Use the NVIDIA vendor integration for hardware enablement, direct GPU allocation, DRA, GPU Operator, or NVIDIA metrics. Use HAMi when the user task starts from sharing, virtualization, or scheduling behavior built on a GPU base.

TOC

Choose a GPU path

Need	Typical path
Use NVIDIA GPUs with vendor-standard components	NVIDIA vendor integration
Share, virtualize, or schedule GPU resources across workloads	HAMi sharing and scheduling path

Need	Typical path
Use claim-based GPU allocation	DRA, when the target product delivery explicitly supports a DRA driver
Use GPU passthrough for virtual machines	ACP Virtualization virtual machine creation documentation

Containers and VMs

Container workload GPU usage and virtual machine GPU passthrough are different paths.

- Container workload GPU usage is covered by this accelerator section and related product sites.
- If a GPU resource is allocatable on the node but not visible in ACP quota pages, register the accelerator quota field metadata in Accelerator resource quota.
- Virtual machine physical GPU passthrough is configured in ACP Virtualization documentation.

Related pages

- NVIDIA vendor integration
- HAMi sharing and scheduling path
- Device Plugin
- DRA

NPU

Use this page when you want to run container workloads on Ascend NPUs.

TOC

Choose an NPU path

Use the Ascend vendor integration for hardware enablement, driver lifecycle, runtime integration, CDI, direct NPU allocation, and vendor-standard operations. Use HAMi when the user task starts from sharing, virtualization, or scheduling behavior built on an Ascend base.

If a deployment prepares Ascend nodes with NPU Operator but uses HAMi Ascend Device Plugin to expose resources, use the NPU product documentation for base prerequisites and the HAMi product documentation for device exposure, resource requests, sharing semantics, and troubleshooting.

Use NPU resources

- Workloads request NPU resources exposed by the selected allocation product.
- Use project and namespace pages for quota operations.
- If an NPU resource is allocatable on the node but not visible in ACP quota pages, register the accelerator quota field metadata in Accelerator resource quota.
- Use observability pages for dashboard and alert configuration.
- Use NPU product documentation for Ascend base resource names, driver lifecycle, and validation workloads. Use HAMi documentation for HAMi-specific Ascend resource names or sharing behavior.

Product documentation

Use the NPU product site for installation, driver lifecycle, compatibility, release notes, and troubleshooting.

> **Note**
>
> Because Alauda Ascend NPU releases on a different cadence from Alauda Container Platform, the Alauda Ascend NPU documentation is now available as a separate documentation set at Alauda Ascend NPU ↗.

Related pages

- Ascend vendor integration
- HAMi sharing and scheduling path
- Device Plugin
- RuntimeClass and CDI
- Accelerator resource quota

Accelerator Products

Products help you choose the vendor integration or sharing and scheduling product to use after you know which accelerator family you need.

Vendor integrations prepare hardware, drivers, runtimes, direct allocation, and vendor-specific monitoring. Sharing and scheduling products build on one or more vendor integrations to provide sharing, virtualization, scheduling, queueing, or resource orchestration.

Vendor Integrations

NVIDIA

- Choose an NVIDIA path
- Related ACP pages
- Related concepts
- Product documentation

Ascend

- Choose an Ascend path
- Related ACP pages
- Product documentation
- Related concepts

Sharing and Scheduling

HAMi

- Use HAMi when
- HAMi backends

Related concepts

Product documentation

Vendor Integrations

Vendor integrations prepare accelerator hardware, drivers, runtimes, direct allocation, and vendor-specific monitoring paths.

Use this section when you want to choose the vendor base for container accelerator workloads. Continue to product documentation for installation, upgrade, compatibility, release notes, and troubleshooting.

NVIDIA

- Choose an NVIDIA path
- Related ACP pages
- Related concepts
- Product documentation

Ascend

- Choose an Ascend path
- Related ACP pages
- Product documentation
- Related concepts

NVIDIA

The NVIDIA vendor integration covers NVIDIA GPU enablement in ACP clusters, including direct GPU allocation, DRA-based GPU allocation, GPU Operator, and NVIDIA GPU metrics components such as DCGM-Exporter when those plugin products are available from the application marketplace or an approved delivery package.

TOC

Choose an NVIDIA path

Option	Use when
NVIDIA GPU Device Plugin	You need physical NVIDIA GPU resources exposed as Kubernetes extended resources.

Option	Use when
NVIDIA DRA Driver for GPUs	You need Kubernetes DRA resource claims for NVIDIA GPU allocation.
GPU Operator	You need a unified NVIDIA stack deployment path, such as driver, toolkit, device plugin, or monitoring components, when that product is available in your delivery.
DCGM-Exporter	You need NVIDIA GPU metrics for ACP monitoring, billing, or product dashboards.
HAMi on NVIDIA GPU	You need sharing, virtualization, or scheduling behavior built on NVIDIA GPU prerequisites. Use the HAMi sharing and scheduling path for the HAMi part of the workflow.

WARNING

Choose one GPU allocation path for each physical NVIDIA GPU pool.

Do not let multiple GPU allocation components manage the same physical GPUs unless that topology is explicitly documented and validated for your product versions.

Related ACP pages

Use this page to choose an NVIDIA integration path. Use the corresponding product documentation for installation, node labels, runtime prerequisites, compatibility, and troubleshooting. If HAMi exposes the GPU resources, use this path only for NVIDIA prerequisites and continue with HAMi documentation for resource requests and sharing semantics.

For generic plugin installation, see Cluster Plugin.

Related concepts

- Device Plugin

- DRA
- RuntimeClass and CDI
- Accelerator resource quota

Product documentation

NVIDIA GPU documentation

> **Note**
>
> Because Alauda NVIDIA GPU releases on a different cadence from Alauda Container Platform, the Alauda NVIDIA GPU documentation is now available as a separate documentation set at Alauda NVIDIA GPU ↗.

GPU Operator

Use the GPU Operator product documentation when that product is available in your delivery.

Ascend

The Ascend vendor integration covers Huawei Ascend NPU enablement in ACP clusters, including NPU Operator, driver lifecycle, runtime integration, CDI, direct NPU allocation, and vendor-specific monitoring components when they are delivered in the target release.

TOC

Choose an Ascend path

NPU Operator manages the Ascend NPU software stack and can expose NPU resources to Kubernetes workloads through the vendor-standard path. Use the NPU product documentation for installation, driver lifecycle, upgrade, compatibility, release notes, and troubleshooting.

If a deployment uses NPU Operator only to prepare the Ascend base and uses HAMi Ascend Device Plugin to expose resources, use the HAMi product documentation for device exposure, resource requests, sharing semantics, and troubleshooting.

Related ACP pages

- Use ACP accelerator pages to understand where Ascend fits in GPU/NPU capability navigation.
- Use ACP quota pages for project and namespace quota operations.
- Use ACP observability pages for dashboard and alert operations.
- Use NPU product documentation for version-specific Ascend base behavior.
- Use HAMi documentation for HAMi on Ascend NPU or HAMi on Ascend vNPU behavior.

Product documentation

Use the NPU product site for installation, driver lifecycle, compatibility, release notes, and troubleshooting.

> **Note**
>
> Because Alauda Ascend NPU releases on a different cadence from Alauda Container Platform, the Alauda Ascend NPU documentation is now available as a separate documentation set at Alauda Ascend NPU↗.

Related concepts

- Device Plugin
- RuntimeClass and CDI
- Accelerator resource quota
- NPU accelerator family
- HAMi sharing and scheduling path

Sharing and Scheduling

Sharing and scheduling products build on one or more vendor integrations to provide accelerator sharing, virtualization, scheduling, queueing, or resource orchestration behavior.

Use this section when the user task starts from sharing or scheduling behavior instead of basic device enablement.

HAMi

Use HAMi when

HAMi backends

Related concepts

Product documentation

HAMi

HAMi is a sharing and scheduling product path for accelerator sharing, virtualization, and scheduling behavior. It builds on vendor integrations such as NVIDIA GPU or Ascend NPU, depending on the product version and backend support.

TOC

Use HAMi when

- multiple workloads need to share accelerator devices;
- users need accelerator sharing modes such as virtual GPU, virtual NPU, hard partitioning, or soft partitioning when supported by the target backend;
- the product documentation confirms support for the target hardware, runtime, and ACP version.

HAMi backends

HAMi documentation owns the HAMi part of a combined deployment: HAMi installation, scheduler behavior, HAMi device plugins, resource keys, sharing semantics, monitoring integration, and troubleshooting.

HAMi backend pages should point to the vendor integration prerequisites they depend on:

- HAMi on NVIDIA GPU depends on the NVIDIA GPU base path for driver, runtime, toolkit, and NVIDIA-specific prerequisites.
- HAMi on Ascend NPU or Ascend vNPU depends on the Ascend base path for NPU nodes, driver lifecycle, runtime integration, and CDI prerequisites.

For accelerator allocation without HAMi sharing semantics, use the corresponding vendor integration:

- For NVIDIA GPU allocation without HAMi sharing, use the NVIDIA vendor integration.
- For Ascend NPU allocation without HAMi sharing, use the Ascend vendor integration.

Related concepts

- Device Plugin
- DRA
- RuntimeClass and CDI
- Accelerator resource quota
- GPU accelerator family
- NPU accelerator family

Product documentation

Use the HAMi product site for installation, compatibility, release notes, troubleshooting, and backend-specific behavior.

Note

Because Alauda Build of HAMi releases on a different cadence from Alauda Container Platform, the Alauda Build of HAMi documentation is now available as a separate documentation set at Alauda Build of HAMi↗.