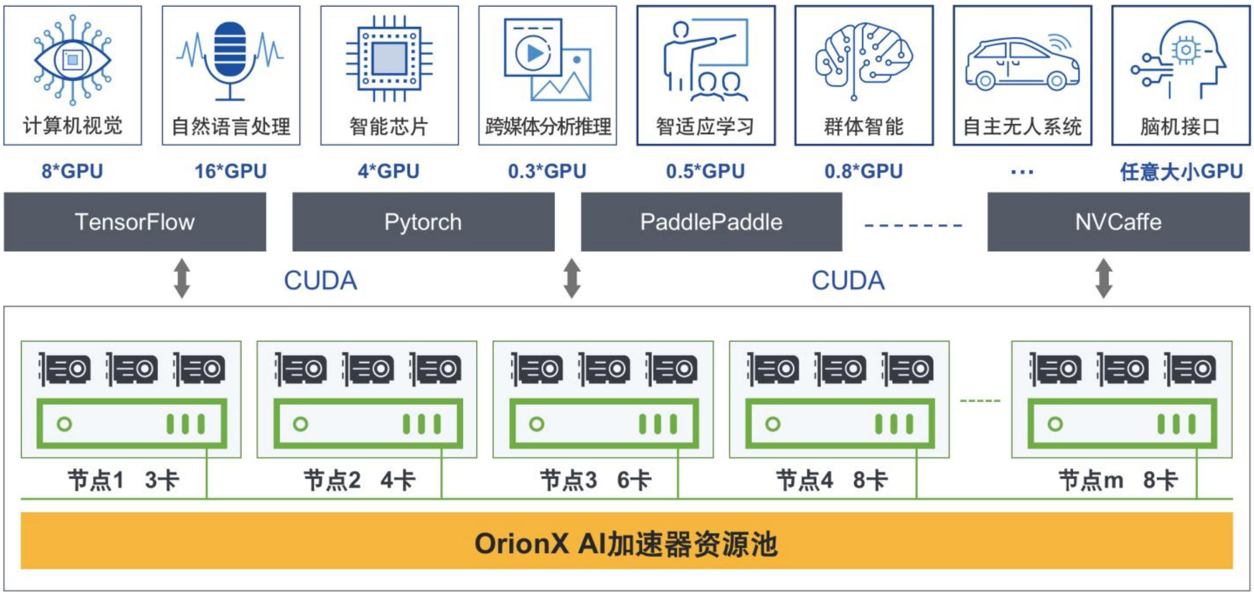
OrionX 社区版部署手册

目　　录

1. 产品简介

趋动科技专注于为用户打造业界领先的数据中心级 AI 加速器虚拟化及资源管理调度软件。目前，已有多家行业巨头，涵盖电信运营商、金融、教育、互联网、政府及公有云等领域，选择采用 OrionX AI 加速器资源池解决方案。

OrionX AI 加速器资源池解决方案在 AI 算力资源池化技术方面取得了革命性的进展。该方案采用软件定义 AI 算力的先进方法，将 GPU 资源池化的能力扩展至整个数据中心范围，实现了 AI 应用与 GPU 服务器硬件的解耦，并支持 vGPU 资源的动态扩展与灵活调度。



2. 术语说明

- OrionX 控制节点

  用于部署 OrionX oCenter 相关组件的 K8S 节点。

- OrionX GPU 计算节点

  配置有 GPU 设备，需要被 OrionX 集群纳管的节点。

3. 产品架构

3.1 组件说明

OrionX 组件分为控制平面组件和业务平面组件，其中控制平面组件部署于 CPU 节点，用于资源池管理、

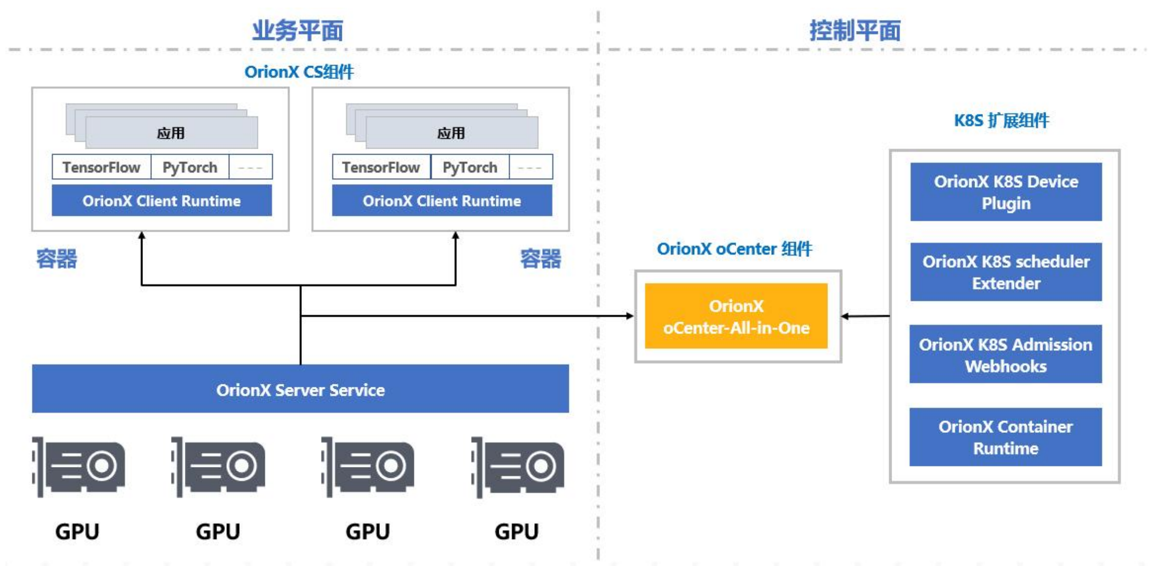
任务调度和资源监控，业务平面组件部署于 GPU 计算节点或者非 GPU 计算节点，用于业务环境初始化和为业务应用提供 vGPU 资源。

OrionX 不同组件的部署模式不同，如下表所示：**组件名称**	**容器名称**	**部署模式**	**存储要求**
OrionX oCenter All in One	orionx-ocenter-all-in-one	Deployment	动态 PV，大于 220G
OrionX GPU Exporter	orionx-gpu-exporter-*	DaemonSet	无特殊要求
OrionX Server	orionx-server-*	DaemonSet	无特殊要求
OrionX K8S Admission Webhooks	orionx-k8s-admission-webhooks-*	Deployment	无特殊要求
OrionX K8S Scheduler Extender	orionx-k8s-scheduler-*	Deployment	无特殊要求
OrionX K8S Device Plugin	orionx-k8s-device-plugin-*	DaemonSet	无特殊要求
OrionX Container Runtime	orionx-container-runtime-*	DaemonSet	无特殊要求

注：部署配置已封装在各组件对应的 YAML 文件中，用户可直接使用 Kubernetes 命令加载执行，具体操作步骤见后续部署流程。

3.2 软件架构

OrionX 产品组件分为业务平面和控制平面两类，其中控制平面又包含平台无关的 oCenter 组件和 K8S 扩展组件。OrionX oCenter 组件中的 OrionX Controller 服务是整个 OrionX 的核心管理服务，其他组件均通过网络与之通讯交互。OrionX 整体软件架构如下图所示：



4. 部署准备

4.1 部署依赖

软件依赖：

名称	部署节点	版本	备注
操作系统	所有 OrionX 节点	CentOS 7/8 Ubuntu 16 / 18 / 20 / 22 Rocky Linux 8.8 欧拉 22.03-LTS 银河麒麟 V10-SP1	
Kubernetes	所有 OrionX 节点	≥ 1.10	
容器运行时	所有 OrionX 节点	docker ≥19.0 containerd ≥ 1.5 cri-o	runc ≥1.1.5
GPU 驱动	所有 OrionX GPU 计算节点	≤545.23.08	
Nvidia Container Toolkit	所有 OrionX GPU 计算节点	推荐最新版本	

权限需求：

操作事项	权限说明	备注
OrionX 部署操作	kubectl 命令 K8S admin 权限	
OrionX 部分组件	容器 privileged 权限； 宿主机网络空间权限；	

4.2 环境配置

建议基础软件环境完成如下配置，以避免遇到异常：

1. **取消默认 CPU 和内存限制**

部分 K8S 平台可能会存在默认的 CPU 和内存资源限制，需要删除 orionx 命名空间默认的 CPU 和内存资源限制。

4.3 文件准备

从趋动科技获取 OrionX 产品部署文件包，并按照解压后的 tool/images.txt 文件内容，获取 OrionX 产品部署所需镜像。

4.4 证书申请

从趋动科技获取授权文件，一般为 license.txt

5. 部署实施

5.1 部署概述

基于 K8S 环境的 OrionX 部署主要流程如下：

```
环境检查 --> 部署文件上传 --> 镜像导入 --> 组件部署 --> 测试验证
```

其中 **组件部署** 环节基于 kubectl 工具，以命令行方式进行部署。

5.2 环境检查

OrionX 部署前，需要对环境做基础检查，以确保符合部署要求，检查项如下：

1. **软件环境检查**

- GPU 驱动检查

在所有 OrionX GPU 计算节点执行如下命令，检查 GPU 驱动状态是否正常，以及驱动版本是否适配：

```
nvidia-smi
#输出信息示例如下所示，表明驱动状态正常
#Driver Version 字段内容为驱动版本号，检查是否在 OrionX 支持范围内
Thu Jul  4 19:13:19 2024
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.105.17   Driver Version: 525.105.17   CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:07.0 Off |                    0 |
| N/A   32C    P8     9W /  70W |      2MiB / 15360MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

- Kubernetes 环境检查

在 K8S 管理节点执行如下命令，检查 K8S 版本和集群状态：

```
kubectl get node -owide
#输出信息示例如下
NAME          STATUS   ROLES                  AGE    VERSION   ......   CONTAINER-RUNTIME
k8s-master    Ready    control-plane,master   303d   v1.21.3   ......   docker://23.0.6
k8s-node-01   Ready    <none>                 303d   v1.21.3   ......   docker://23.0.6
k8s-node-02   Ready    <none>                 303d   v1.21.3   ......   docker://23.0.6
k8s-node-03   Ready    <none>                 229d   v1.21.3   ......   docker://23.0.6
```

检查 STATUS 字段是否全部为 Ready，VERSION 为 K8S 版本信息，CONTAINER-RUNTIME 字段为容器运行

时类型和版本，检查是否在 OrionX 支持范围。

- Nvidia Container Toolkit 检查

在所有 OrionX GPU 计算节点上执行如下命令，检查容器运行 Default Runtime 是否为 nvidia：

```
docker info | grep 'Default Runtime'
#输出信息示例如下：
 Default Runtime: nvidia
containerd config dump | grep default
#输出信息示例如下：
      default_runtime_name = "nvidia"
```

5.3 部署流程

5.3.1 部署文件上传

上传 OrionX 部署文件 yaml.tar.gz 至 K8S 管理节点任意位置，并解压，解压后的目录结构如下：

```
yaml /
├── base   #OrionX 配置文件
│    ├── namespace.yaml
│    ├── orionx-meta.yaml
│    └── runtimejson.yaml
├──deploy  # OrionX 部署文件
│    ├── orionx-ocenter-all-in-one.yaml
│    ├── orionx-k8s-scheduler-extender.yaml
│    ├── orionx-gpu-server.yaml
│    ├── orionx-k8s-device-plugin.yaml
│    ├── orionx-gpu-exporter.yaml
│    ├── orionx-k8s-admission-webhooks.yaml
│    └── orionx-container-runtime.yaml
├── readme.md
└── tools    #OrionX 部署和测试工具
    ├── images.txt
    └── labels.sh
```

上传 OrionX 镜像至所有 OrionX 部署节点，并导入镜像，或者上传 OrionX 镜像至内部镜像仓库备用。

5.3.2K8S 节点标记

编辑部署文件 tools/label.sh 脚本，按资源规划修改如下内容：

```
declare -A amd64=(
    ["control"]="k8s-cpu-1 k8s-cpu-2"    # OrionX 控制节点列表，K8S 节点名称，空格分隔
    ["gpu"]="k8s-gpu-1 k8s-gpu-2"    # OrionX GPU 计算节点列表，K8S 节点名称，空格分隔
```

```
)
```

修改完 label.sh 脚本后，执行如下操作，对 K8S 集群节点进行标记：

```
bash tools/label.sh
```

5.3.3创建命名空间和配置信息对象

执行如下操作，创建名为 orionx 的命名空间，创建 OrionX 配置信息 ConfigMap 对象：

```
kubectl apply -f base/
```

检查目标资源对象是否已创建：

```
kubectl get ns | grep orionx
kubectl -n orionx get configmap
```

输出内容如下所示：

```
root@k8s-master: kubectl get ns |grep orionx
orionx                Active     113d

root@k8s-master: kubectl get ns |grep orionx
NAME                                      DATA     AGE
orionx-k8s-scheduler-config          1          20d
orionx-meta                              1          20d
runtimejson                            3          20d
```

5.3.4部署 orionx-ocenter-all-in-one 组件

执行如下操作，部署 orionx-ocenter-all-in-one 组件：

```
kubectl apply -f deploy/orionx-ocenter-all-in-one.yaml
```

执行如下操作，检查组件状态是否为 Running：

```
kubectl -n orionx get pod -owide | grep orionx-ocenter-all-in-one
#正常状态示例如下
root@k8s-master:~# kubectl -n orionx get pod -owide | grep orionx-ocenter-all-in-one
orionx-ocenter-all-in-one -5c4977c7d6-d22cd   1/1   Running  0   21d   10.7.1.11   k8s-node2    <none>
<none>
```

查看组件日志，组件服务正常情况下，没有 ERR 级别日志：

```
kubectl -n orionx logs orionx-ocenter-all-in-one -5c4977c7d6-d22cd   #替换成真实 Pod 名称
```

5.3.5导入证书

通过浏览器访问 http://<k8s-node-ip>:30125，打开 OrionX GUI 管理页面，通过 **激活** 页面，**添加激活文件** 按钮导入证书，证书导入后，会显示证书信息，如下图所示：

5.3.6部署 orionx-k8s-scheduler 组件

配置说明：

配置项	ConfigMap 名称	说明	备注
KubeSchedulerConfiguration	orionx-k8s-scheduler-config	K8S≤1.18 时使用 apiVersion 为 kubescheduler.config.k8s.io/v1alpha1 的配置内容； K8S≥1.19 时使用 apiVersion 为 kubescheduler.config.k8s.io/v1beta1 的配置内容： • 1.23≤K8S<1.25 时将 v1beta1 修改为 v1beta2； • K8S≥1.25 时将 v1beta1 修改为 v1；	

执行如下操作，部署 orionx-k8s-scheduler 组件：

```
kubectl apply -f deploy/orionx-k8s-scheduler-extender.yaml
```

执行如下操作，检查组件状态是否为 Running：

```
kubectl -n orionx get pod -owide | grep orionx-k8s-scheduler
#正常状态示例如下
root@k8s-master:~# kubectl -n orionx get pod -owide | grep orionx-k8s-scheduler
orionx-k8s-scheduler-5c4977c7d6-d22cd    1/1   Running   0   21d    10.7.1.11    k8s-node2    <none>
<none>
```

查看组件日志，组件服务正常情况下，没有 ERR 级别日志：

```
kubectl -n orionx logs orionx-k8s-scheduler-5c4977c7d6-d22cd -c scheduler-extender   #替换成真实 Pod 名称
```

5.3.7部署 orionx-server 组件

配置说明：

环境变量	说明	备注
ORION_CONTROLLER	自定义字符串，OrionX Controller SVC 地址 默认配置："orionx-controller:9123"	通常保持默认值即可
ORION_BIND_ADDR	自定义字符串，OrionX Server 监听地址 默认配置：<K8S Node IP> *ORION_BIND_ADDR 和 ORION_BIND_NET 二选一*	单一网络平面环境，保持默认即可

ORION_BIND_NET	自定义字符串，OrionX Server 监听网口 默认配置：n/a *ORION_BIND_ADDR 和 ORION_BIND_NET 二选一*	多网络平面环境下，按需修改，指定 OrionX 业务平面
ORION_SERVER_PORT	正整数，OrionX Server 监听端口 默认配置："9960"	通常保持默认即可，可按需修改
ORION_VGPU_COUNT	正整数，GPU 切分数量上限 默认配置："12"	按需修改
ORION_VGPU_EXPORTER_LISTEN_PORT	正整数，vGPU Exporter 服务监听端口 默认配置："9401"	通常保持默认即可，可按需修改

按需修改 orionx-gpu-server.yaml 文件中环境变量配置：

```
env:
- name : ORION_CONTROLLER
  value : "orionx-controller:9123"
- name: ORION_BIND_ADDR
  valueFrom:
    fieldRef:
      fieldPath: status.hostIP
#- name : ORION_BIND_NET
#  value : "eth0"
- name : ORION_SERVER_PORT
  value : "9960"
- name : ORION_VGPU_COUNT
  value : "12"
- name: ORION_VGPU_EXPORTER_LISTEN_PORT
  value: "9401"
```

执行如下操作，部署 orionx-server 组件：

```
kubectl apply -f deploy/orionx-gpu-server.yaml
```

执行如下操作，检查组件状态是否为 Running，Pod 实例数量符合规划，部署于所有 OrionX GPU 计算节点：

```
kubectl -n orionx get pod -owide -l app=orionx-server
#正常状态示例如下
root@k8s-master:~# kubectl -n orionx get pod -owide -l app=orionx-server
NAME                  READY   STATUS    RESTARTS   AGE   IP            NODE
orionx-server-2tcn7   1/1     Running   0          20d   10.12.102.5   k8s-node-01
```

查看组件日志，组件服务正常情况下，没有 ERROR 级别日志：

```
kubectl -n orionx logs orionx-server-2tcn7    #替换成真实 Pod 名称
```

5.3.8 部署 orionx-k8s-device-plugin 组件

配置说明：

环境变量	说明	备注
ORION_CONTROLLER	自定义字符串，OrionX Controller SVC 地址 默认配置："orionx-controller:9123"	通常保持默认值即可
REPORT_ONLY_LOCAL_VGPU	布尔值，只上报本地节点资源特性开关 默认值：false false：K8S 节点 vGPU 资源量为节点总量； true：K8S 节点 vGPU 资源量为集群全局总量；	通常保持默认即可，按需修改

执行如下操作，部署 orionx-k8s-device-plugin 组件：

```
kubectl apply -f deploy/orionx-k8s-device-plugin.yaml
```

执行如下操作，检查组件状态是否为 Running，Pod 实例数量符合规划，部署于所有 OrionX GPU 计算节点：

```
kubectl -n orionx get pod -o wide -l app=orionx-k8s-device-plugin
#正常状态示例如下
root@k8s-master:~# kubectl -n orionx get pod -owide -l app=orionx-k8s-device-plugin
NAME                               READY   STATUS    RESTARTS   AGE   IP             NODE
orionx-k8s-device-plugin-xsdfk     1/1     Running   0          26d   10.12.102.2    k8s-node-01
```

查看组件日志，组件服务正常情况下，没有 ERROR 级别日志：

```
kubectl -n orionx logs orionx-k8s-device-plugin- xsdfk  #替换成真实 Pod 名称
```

5.3.9 部署 orionx-gpu-exporter 组件

配置说明：

环境变量	说明	备注
DCGM_EXPORTER_LISTEN	自定义字符串，gpu-exporter 服务监听端口 默认配置：":9400"	通常保持默认即可，可按需修改
ORION_BIND_PORT	正整数，访问 gpu-exporter 服务端口 默认配置："9400"	通常保持和 DCGM_EXPORTER_LISTEN 配置端口一致。 确保 orinox-ocenter-all-in-one 组件可以访问
ORION_BIND_IP	自定义字符串，访问 gpu-exporter 服务 IP 默认配置：<K8S Node IP>	通常保持默认即可。 确保 orinox-ocenter-all-in-one 组件可

		以访问
ORION_SERVICE_REGISTRY_ADDR	自定义字符串，OrionX Controller SVC 地址 默认配置："orionx-controller-proxy:9123 "	通常保持默认即可

执行如下操作，部署 orionx-gpu-exporter 组件，会部署于所有 OrionX GPU 计算节点：

```
kubectl apply -f deploy/orionx-gpu-exporter.yaml
```

执行如下操作，检查组件状态是否为 Running，Pod 数量是否正确：

```
kubectl -n orionx get pod -o wide -l app=orionx-gpu-exporter
#正常状态示例如下
root@k8s-master:~# kubectl -n orionx get pod -owide -l app=orionx-gpu-exporter
NAME                         READY   STATUS    RESTARTS   AGE   IP            NODE          ......
orionx-gpu-exporter-twlxm    1/1     Running   0          26d   10.12.102.3   k8s-node-01   ......
```

查看组件日志，组件服务正常情况下，没有 error 级别日志：

```
kubectl -n orionx logs orionx-gpu-exporter- twlxm   #替换成真实 Pod 名称
```

5.3.10 部署 orionx-k8s-admission-webhooks 组件

配置说明：

环境变量	说明	备注
ORION_CONTROLLER	自定义字符串，OrionX Controller Proxy SVC 地址 默认配置："orionx-controller-proxy:9123"	通常保持默认值即可

执行如下命令，部署 orionx-k8s-admission-webhooks 组件：

```
kubectl apply -f deploy/orionx-k8s-admission-webhooks.yaml
```

执行如下操作，检查组件状态是否为 Running

```
kubectl -n orionx get pod -owide -l app=orionx-k8s-admission-webhooks
#正常状态示例如下
root@k8s-master:~# kubectl -n orionx get pod -l app=orionx-k8s-admission-webhooks
NAME                                              READY   STATUS    RESTARTS   AGE
orionx-k8s-admission-webhooks-6fbd654ddc-5ktkd    1/1     Running   0          26d
```

查看组件日志，组件服务正常情况下，没有 error 级别日志（TLS handshake error 日志除外）：

```
kubectl -n orionx logs orionx-k8s-admission-webhooks-7cc6fddbbb-5ktkd   #替换成真实 Pod 名称
```

5.3.11 部署 orionx-container-runtime 组件

配置说明：

环境变量	说明	备注

HOST_ENGINE	固定字符串，指定容器运行时类型 默认值：docker 可选值：docker、containerd、crio	根据K8S平台底层容器运行时类型配置，目前支持：docker、containerd 和 cri-o

执行如下操作，部署 orionx-container-runtime 组件：

```
kubectl apply -f deploy/orionx-container-runtime.yaml
```

执行如下操作，检查组件状态是否为 Running，Pod 实例数量符合规划，部署于所有 OrionX GPU 计算节点：

```
kubectl -n orionx get pod -o wide -l app=orionx-container-runtime
#正常状态示例如下
root@k8s-master:~# kubectl -n orionx get pod -owide -l app=orionx-container-runtime
NAME                              READY   STATUS    RESTARTS   AGE   IP          NODE
orionx-container-runtime-h8ncl    1/1     Running   0          20d   10.7.1.10   k8s-node-03
orionx-container-runtime-d4src    1/1     Running   0          21d   10.7.1.11   k8s-node-01
orionx-container-runtime-h8ncl    1/1     Running   0          20d   10.7.1.10   k8s-node-02
orionx-container-runtime-d4src    1/1     Running   0          21d   10.7.1.11   k8s-master
```

查看组件日志，组件服务正常情况下，没有 ERROR 级别日志：

```
kubectl -n orionx logs orionx-container-runtime-h8ncl  #替换成真实 Pod 名称
```

6. 测试验证

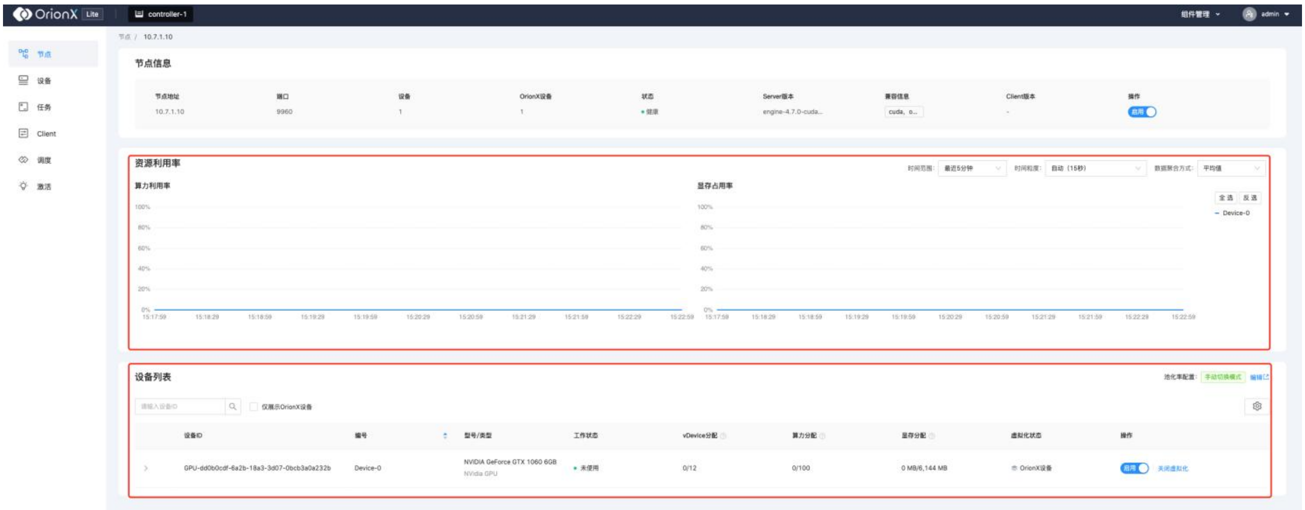
6.1 GUI 状态检查

浏览器访问 http://<k8s-node-ip>:30125 地址，打开 OrionX GUI，登录信息：admin / GM@VirA1。登录 OrionX GUI 后，检查如下页面及信息：

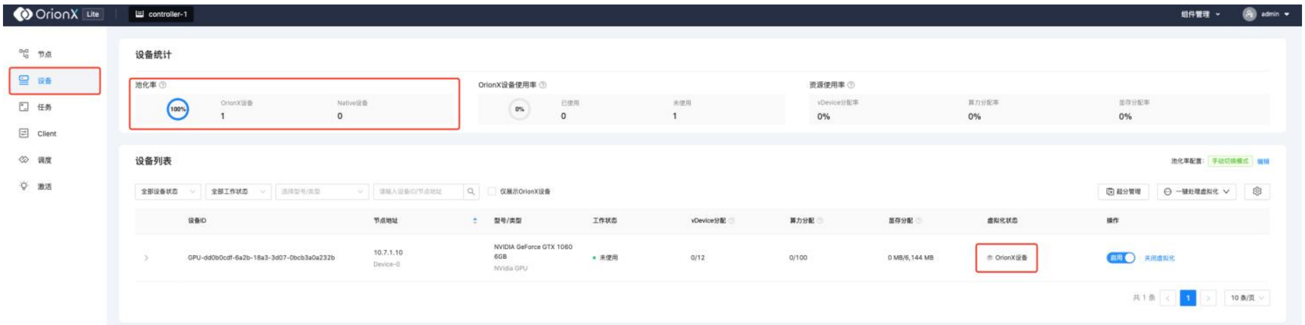
打开 **节点** 管理页面展示了 OrionX GPU 计算节点信息，检查节点信息是否正确，状态是否健康，是否为可调度状态，如下图所示：

分别进入各节点 **详情** 页面，如下图：

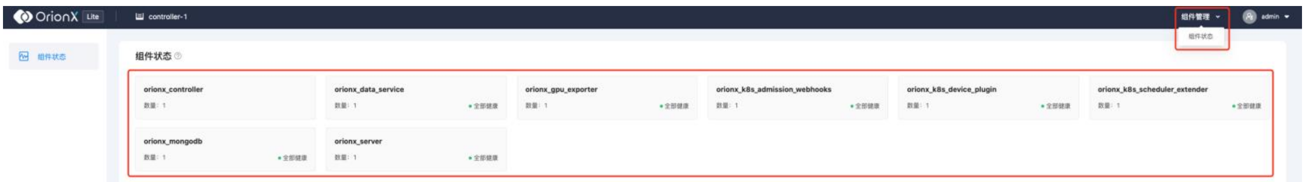
检查各节点 **资源利用率** 统计图是否正常显示，**设备列表** 中 GPU 信息是否正确，如下图所示：



进入 **设备** 管理页面，检查资源池化率是否为 100%，证书包含的 GPU 设备会自动池化，虚拟化状态为 **OrionX 设备**，如下图所示：



进入 **组件管理** 页面，检查 OrionX 组件状态是否健康，如下图所示：



6.2 基础测试

测试镜像：

harbor.virtaitech.com/orionx/native:tensorflow2.4.1-hvd0.21.3-cu11.1-cudnn8-py3.6-ubuntu18.04

测试工具：Tensorflow Benchmarks

基于原生 K8S 资源申请方式，申请 OrionX vGPU 资源，创建测试容器，配置如下所示：

```
resources：
  limits：
    virtaitech.com/gpu：1      #OrionX vGPU 实例个数
    virtaitech.com/gmem：10   #单个 OrionX vGPU 实例显存资源量，默认单位 GB，有效值 1≤gmem≤物理显存资源量
    virtaitech.com/ratio：100    #单个 OrionX vGPU 实例算力资源量，单位%，有效值 1≤ratio≤100
```

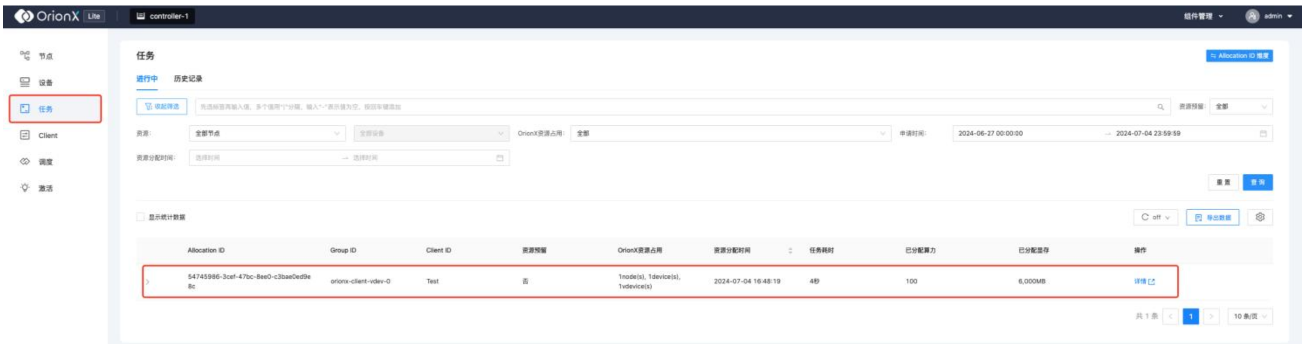
进入测试容器，执行如下测试代码：

```
cd /root
export ORION_CLIENT_ID="Test"
python3 benchmarks/scripts/tf_cnn_benchmarks/tf_cnn_benchmarks.py \
  --data_name=imagenet \
  --model=resnet50 \
  --forward_only=True \
  --num_gpus=1 \
  --num_batches=500 \
  --batch_size=32 \
  --allow_growth=true
```

测试任务输出内容示例如下：

```
……
450 images/sec：103.4 +/- 0.2 (jitter = 1.4) 0.000 0.000 0.000
460 images/sec：103.4 +/- 0.2 (jitter = 1.4) 0.000 0.000 0.000
470 images/sec：103.5 +/- 0.2 (jitter = 1.4) 0.000 0.000 0.000
480 images/sec：103.5 +/- 0.2 (jitter = 1.4) 0.000 0.000 0.000
490 images/sec：103.5 +/- 0.2 (jitter = 1.4) 0.000 0.000 0.000
500 images/sec：103.5 +/- 0.2 (jitter = 1.4) 0.000 0.000 0.000
----------------------------------------------------------------
total images/sec：103.12
----------------------------------------------------------------
```

查看 OrionX GUI **任务** 管理页面，可以看到对应的任务信息，Client ID 为 Test，已分配算力为 100，已分配显存位 10240MB，如下图所示：



7. 产品使用

OrionX 产品使用方法参考《OrionX 社区版使用手册》。