Nodes

Overview

Add Nodes to On-Premises Clu

Platform administrators can add nodes to s

Manage No

Supports updati

Plan an In-Place OS Upgrade for Nodes

Plan and verify an in-place operating system upgrade on self-built cluster nodes.

Cluster Node Planning

Overview

Nodes are the fundamental building blocks of a cluster. Nodes can be virtual machines or physical servers, and each node runs the components required to host Pods, including kubelet, kube-proxy, and the container runtime.

Users with platform management permissions can add, manage, monitor, and plan nodes under clusters.

> **NOTE**
>
> Adding nodes to imported clusters or deleting nodes from imported clusters is not supported.

TOC

Node types

- **Control plane nodes** run cluster components such as kube-apiserver, kube-scheduler, kube-controller-manager, etcd, container networking, and some platform management

components.

- **Compute nodes** host business Pods running on the cluster. Plan compute node count based on workload demand.

Node operation paths

Need	Continue with
Add nodes to a cluster.	Add nodes
Manage labels, taints, tolerations, and node scheduling state.	Manage nodes
Monitor node status.	Monitor nodes
Plan infra nodes or custom role nodes.	Node planning
Plan in-place operating system upgrades for eligible nodes.	In-place OS upgrade
Tune CPU, memory, or topology manager policies for performance-sensitive workloads.	Resource manager policies

Linux node availability check

Before building an on-premises cluster, complete the node checks and confirm that every node meets the requirements. Cluster deployment can fail when prerequisites are not satisfied.

Supported operating systems and CPU models

For supported operating systems and CPU models, see Supported Operating Systems and CPU Models.

Add Nodes to On-Premises Clusters

When a cluster needs to scale up or when abnormal nodes on the cluster need to be replaced with new nodes, you can add control plane nodes and compute nodes to existing **on-premises** workload clusters on the platform by adding nodes.

TOC

Constraints and Limitations

- Prepare nodes before adding them to the cluster. Use the Node Availability Check Reference to verify the nodes. Cluster deployment can fail when required conditions are not met.
- The hardware architecture of nodes to be added must be consistent with the cluster's hardware architecture.

- To avoid unpredictable errors, the operating system type of nodes to be added should be consistent with other nodes in the cluster.
- SSH ports and authentication information for nodes added in the same **Add Node** dialog must be unified.
- **Cluster planning guideline**: A cluster must have at least 1 control plane node. Setting exactly 2 control plane nodes is not supported. With 3 or more control plane nodes, the cluster becomes a high-availability cluster (for high-availability clusters, it is recommended to use an odd number of nodes, preferably 3 or 5). **Note**: This requirement applies only when adding or changing control plane capacity; you can safely add worker/compute nodes without being forced to add control plane nodes.
- A node can be part of only one cluster. Nodes that already belong to another cluster cannot be added.

Prerequisites

- When the global cluster cannot directly access nodes to be added to the cluster through SSH service and needs to access through a proxy, prepare the proxy service in advance. Currently, only SOCKS5 proxy is supported.

Procedure

1. In the left navigation bar, click **Clusters** > **Clusters**.
2. Click the ***cluster name*** of type **On-Premises** where you want to add nodes.
3. Under the **Nodes** tab, click **Add Node**.
4. Configure the relevant parameters. For parameter details, see Node Configuration Parameters.
5. Click **Add** to perform availability check on the nodes.
   After the check passes, node addition begins, and the nodes are in **Adding** state.

Follow-up Operations

View Execution Progress

On the node list page, you can view the list information of added nodes. For nodes in **Adding** state, you can view the execution progress.

Procedure

1. Click **View Execution Progress** on the right side of nodes in **Adding** state.
2. In the pop-up execution progress dialog, you can view the node execution progress (`status.conditions`).
   Tip: When a certain type is executing or has a failed state with a reason, you can view detailed information about the reason (`status.conditions.reason`) by hovering the cursor over the corresponding reason (displayed in blue text).

Re-add Failed Nodes

After adding nodes, if some nodes fail to be added, a prompt will appear above the node list. Click the **Re-add** button in the prompt box to re-add the failed nodes.

Manage Nodes

TOC

Update Node Labels

Labels↗ are key-value pairs attached to nodes that can define node attributes. After setting labels for nodes, you can easily filter or select nodes by labels. For example: directing Pods to be scheduled to specific nodes.

Supports updating node labels for nodes in normal state, adding or removing custom node labels.

Procedure

1. In the left navigation bar, click **Cluster Management** > **Clusters**.
2. Click the ***cluster name*** where the node with labels to be updated is located.
3. Under the **Nodes** tab, click **Update Node Labels** on the right side of the node with labels to be updated.
4. Add, modify, or delete node labels.
5. Click **OK**.
   After successfully updating node labels, the number of node labels changes. You can view all label information of the node in the **Node Labels** item in the **Node** information bar.

Stop/Resume Node Scheduling

By setting the scheduling state of nodes, you can control whether newly created Pods in the cluster are allowed to be scheduled to the node.

- **Stop Scheduling**: Newly created Pods are not allowed to be scheduled to the node, but existing Pods running on the node are not affected.
- **Resume Scheduling**: Newly created Pods are allowed to be scheduled to the node.

Procedure

1. In the left navigation bar, click **Clusters** > **Clusters**.
2. Click the ***cluster name*** where the node to stop/resume scheduling is located.
3. Under the **Nodes** tab, click **Stop Scheduling/Resume Scheduling** on the right side of the node to set scheduling state.

4. Click **OK**.

Evict Pods

Evict all Pods except those managed by DaemonSet (daemon set) from nodes in normal state to other nodes in the cluster, and set the node to unschedulable state.

Note: Eviction removes data stored locally in Pods. Confirm the data impact before evicting Pods.

Procedure

1. In the left navigation bar, click **Cluster Management** > **Clusters**.
2. Click the ***cluster name*** where the node to evict Pods is located.
3. Under the **Nodes** tab, click the ***node name*** to evict Pods.
4. In the upper right corner, click **Actions** > **Evict Pods**.
5. Review the information of Pods to be evicted, then click **Evict**.

Set Taints

Set taint information for nodes in normal state.

Taints are a property of nodes that allow nodes to refuse to run certain types of Pods or even evict Pods. Taints work together with tolerations on Pods to prevent Pods from being assigned to inappropriate nodes. One or more taints can be applied to each node, and Pods that cannot tolerate these taints will not be accepted by the node.

For example: For a node where we find its memory utilization has reached 91%, it is not recommended to continue scheduling new Pods to this node. We can set a taint for it. After setting the taint, Kubernetes will not schedule Pods to this node.

For Kubernetes taint and toleration behavior, see Taints and Tolerations↗.

Procedure

1. In the left navigation bar, click **Cluster Management** > **Clusters**.
2. Click the ***cluster name*** where the node to set taints is located.
3. Under the **Nodes** tab, click **Set Taints** on the right side of the node to set taints.
4. Set the key, value, and effect of taints. Multiple taints can be added to a node.
   Taint attributes consist of `key=value [effect]`.
   `key=value` is used to match Pod tolerations. The taint indicates that the node has been contaminated by `key=value`, and Pod scheduling is not allowed or should avoid scheduling to this node, unless the Pod can tolerate (Tolerations) the `key=value` taint.
   effect is the effect of the taint, with the following three options:

   - **NoSchedule**: Indicates scheduling is not allowed, and already scheduled resources are not affected.
   - **PreferNoSchedule**: Indicates try not to schedule.
   - **NoExecute**: Indicates scheduling is not allowed, and already scheduled resources will be deleted after `tolerationSeconds`.
5. Click **OK**.

Label and Taint Management

The platform supports batch setting of labels and taints for nodes.

Constraints and Limitations

- Before setting device labels, you need to deploy device plugins on the cluster first, such as NVIDIA GPU MPS device plugin, NVIDIA GPU device plugin, GPU Manager device plugin, etc.
  Tip: Device labels are actually node labels. For your convenience, the platform categorizes node labels that device plugins depend on as device labels for quick configuration.

Procedure

1. In the left navigation bar, click **Clusters** > **Clusters**.
2. Click the ***cluster name*** where you want to manage labels and taints.

3. Under the **Nodes** tab, multi-select the nodes you want to manage, and click the **Label and Taint Management** button.
   Tip: You can enter the node labels you care about in the search box on the node list page to quickly filter out the list of nodes you want to manage labels and taints for.
4. In **Batch Operations**, add and fill in the operations you want to perform, then click OK to submit the batch operations to the cluster.

   - **Node Labels**: You can **add/update** specified labels for selected nodes, or **delete** specified labels. When selecting delete, the platform will filter out all label lists on the selected nodes. When the value is set to **Any**, it represents deleting labels on all nodes containing the specified label key.
   - **Taints**: You can **add/update** specified taints for selected nodes, or **delete** specified taints. When selecting delete, the platform will filter out all taint lists on the selected nodes. When the value is set to **Any**, it represents deleting taints on all nodes containing the specified taint key.
   - **Device Labels**: You can set the devices you want to use for selected nodes, where the device list comes from device plugins you have deployed in this cluster.

Enable/Disable Virtualization Switch

When nodes in an on-premises cluster are physical machines, you can control whether Kubernetes is allowed to schedule virtual machines (VMI, VirtualMachineInstance) to the node by enabling/disabling the node virtualization switch.

When the switch is enabled, newly created virtual machines are allowed to be scheduled to the physical machine node; when the switch is disabled, newly created virtual machines are prohibited from being scheduled to the physical machine node, but this does not affect virtual machines already running on the node.

Tip: For related operations and precautions, see Prepare Virtualization Environment.

Delete On-Premises Cluster Nodes

Supports deleting nodes in clusters of type on-premises. For example: deleting failed nodes in on-premises clusters.

Constraints and Limitations

- Nodes in imported clusters are not supported for deletion.
- When there is only one control plane node in the cluster, deleting this control plane node is not supported.

Procedure

1. In the left navigation bar, click **Cluster Management** > **Clusters**.
2. Click the ***cluster name*** of type **On-Premises** where the node to be deleted is located.
3. Under the **Nodes** tab, click **Delete** on the right side of the node to be deleted.
   Tip: If you need to clean up resources under the node after deleting a Linux node, click **Download Cleanup Script** at the bottom of the dialog to download the cleanup script to local. After the node is successfully deleted, log in to the node and execute the cleanup script.
4. Enter the node name, then click **Delete**.

Plan an In-Place OS Upgrade for Nodes

Cluster administrators can use this checklist to plan an in-place operating system upgrade for nodes in ACP self-built clusters. Complete the ACP checks before and after the operating system changes.

Scope and support boundary

Use this checklist together with the operating system provider's supported upgrade procedure during a maintenance window. Confirm the target operating system and kernel against the ACP support matrix before you upgrade any node. The checklist does not guarantee that every operating system upgrade path is supported.

TOC

Scope and Limitations

Use this procedure only when all of the following conditions are met:

- The cluster is an on-premises or self-built cluster managed by ACP.
- You can log in to the target nodes and manage the node operating system.
- You have platform administrator permissions, `kubectl` administrator permissions, and SSH or console access to each target node.
- You can drain workload Pods from the target node before the operating system upgrade.

Do not use this procedure for the following cluster types:

- Clusters that use Immutable Infrastructure. Apply operating system changes by replacing nodes with new images.
- Managed cloud Kubernetes clusters where the cloud provider manages the node or control plane operating system.
- Imported clusters where you cannot log in to nodes or control plane nodes.

Before You Upgrade

Complete the following checks before changing the operating system on any node.

1 Step 1: Confirm the target operating system and kernel

Confirm that the target operating system version, kernel version, kernel source, and CPU architecture are within the supported range. For the current support matrix and known restrictions, see Supported OS and Kernel Versions.

Apply these rules before the maintenance window:

- The operating system major and minor versions must match the supported matrix.
- The core kernel version must match the supported matrix. Only the build suffix can differ.
- The kernel must be the official kernel shipped by the operating system vendor.
- Ubuntu HWE kernels and third-party or custom-compiled kernels are not supported.
- If the target version is not in the supported matrix, contact technical support for compatibility confirmation before the upgrade.

2 Step 2: Check for conflicting packages

During an in-place operating system upgrade, the operating system package manager might install or overwrite container runtime, Kubernetes, or container network binaries. Before the upgrade, check and resolve packages that conflict with ACP components. For the package lists and commands, see Remove Conflicting Packages.

If conflicting packages are found, prepare an application migration plan and back up the affected data before uninstalling them.

3 Step 3: Record the current runtime and node component versions

Record the versions before the operating system upgrade. Use the records for comparison after the node is upgraded.

```
containerd --version
runc --version
crictl --version
kubelet --version
```

Also record critical node configuration files that your operating system upgrade process might update, such as `/etc/resolv.conf`, `/etc/fstab`, systemd configuration files, and container runtime configuration files.

4 Step 4: Verify cluster capacity and drain the node

Confirm that the remaining nodes have enough capacity to run the Pods evicted from the target node. Then drain the node before the operating system upgrade.

You can use the ACP console to evict Pods from the node. For the console operation, see Manage Nodes.

If you use `kubectl`, confirm the command options with your operations team before running it. For example:

```
kubectl cordon <node-name>
kubectl drain <node-name> --ignore-daemonsets --delete-emptydir-data
```

Pods managed by DaemonSets are not evicted by the drain operation. Workloads that use local storage might lose local data after eviction. Confirm the workload impact before you proceed.

Additional Checks for Control Plane Nodes

Control plane nodes run components such as `kube-apiserver`, `etcd`, `kube-scheduler`, and `kube-controller-manager`. Upgrade control plane nodes one at a time, and verify cluster health after each node is upgraded.

Before upgrading a control plane node:

- Back up etcd data. For the supported backup mechanism and restore considerations, see etcd Backup and Restore.
- Confirm that the etcd cluster is healthy and that quorum can be maintained while one control plane node is unavailable.
- Confirm that the cluster has at least three control plane nodes when you are performing rolling maintenance on control plane nodes.
- If possible, validate the procedure on compute nodes first, and then proceed with control plane nodes.

You can use the following commands as references when checking control plane health:

```
kubectl get nodes
kubectl get pods -n kube-system | grep -E "etcd|apiserver|scheduler|contr
oller"
```

If you use `etcdctl` on a control plane node, use the certificate paths from your environment:

```
ETCDCTL_API=3 etcdctl member list \
  --endpoints=https://127.0.0.1:2379 \
  --cacert=/etc/kubernetes/pki/etcd/ca.crt \
  --cert=/etc/kubernetes/pki/etcd/server.crt \
  --key=/etc/kubernetes/pki/etcd/server.key \
  --write-out=table
```

Perform the Operating System Upgrade

Follow the operating system vendor's supported upgrade procedure for the target version. The exact package commands, repository configuration, and reboot requirements are determined by the operating system vendor and your organization's operating system maintenance policy.

During the operating system upgrade:

- Do not intentionally install container runtime, Kubernetes, or container network packages that conflict with ACP components.

- Preserve node network configuration, DNS configuration, time synchronization configuration, and systemd service configuration unless the vendor procedure requires a change.
- Reboot the node when the vendor procedure requires it.
- Keep the node unschedulable until all post-upgrade checks pass.

After You Upgrade

Run the following checks on each node after the operating system upgrade and reboot are complete.

1 Step 1: Confirm the operating system and kernel

Verify that the node is running the expected operating system and kernel versions.

```
cat /etc/os-release
uname -r
```

Compare the output with the supported matrix and your approved maintenance plan.

2 Step 2: Verify base node configuration

Verify that the operating system upgrade did not revert required node settings.

```
echo "=== Base node configuration check ==="

# SELinux must be disabled on systems that use SELinux.
getenforce 2>/dev/null || echo "SELinux command not available"

# AppArmor must be disabled on systems that use AppArmor.
systemctl status apparmor 2>/dev/null || echo "AppArmor service not i
nstalled"

# Swap must be disabled.
swapon --show
free -h | grep Swap

# Firewall services must be disabled according to your cluster networ
k plan.
systemctl status firewalld 2>/dev/null || echo "firewalld not install
ed"
systemctl status ufw 2>/dev/null || echo "ufw not installed"

# /tmp must not be mounted with noexec.
mount | grep " /tmp "

# DefaultTasksMax must be infinity.
systemctl show --property=DefaultTasksMax
```

If `swap` is enabled after the operating system upgrade, disable it and remove the swap entry from `/etc/fstab` according to your operating system policy.

If SELinux or AppArmor is enabled after the operating system upgrade, disable it according to your operating system policy and the ACP node requirements.

3 Step 3: Verify runtime and node component versions

Compare the runtime and node component versions with the pre-upgrade records.

```
containerd --version
runc --version
crictl --version
kubelet --version
```

Then verify that `containerd` is running:

```
systemctl status containerd
```

If any binary was overwritten by an operating system package, stop the maintenance and contact technical support before you continue with other nodes.

4 Step 4: Verify time synchronization and DNS

Verify that time synchronization and DNS configuration were not changed by the operating system upgrade.

```
date
timedatectl
cat /etc/resolv.conf
```

The time skew between nodes must be no more than 10 seconds. If the skew is larger than 10 seconds, synchronize time before restarting workloads on the node.

5 Step 5: Restart kubelet and verify node recovery

Restart `kubelet` after the operating system upgrade is complete and the base node checks pass.

```
systemctl daemon-reload
systemctl restart kubelet
systemctl status kubelet
```

Wait for the node to return to the `Ready` state.

```
kubectl get node <node-name> --watch
```

When the node is healthy, resume scheduling.

```
kubectl uncordon <node-name>
```

You can also resume scheduling from the ACP console. For the console operation, see Manage Nodes.

6 Step 6: Validate workload recovery

Verify the node status and workload placement after scheduling is resumed.

```
kubectl get nodes
kubectl get pods -A -o wide | grep <node-name>
```

Then validate the business services that were affected by the node drain. Proceed to the next node only after the current node and related services are healthy.

Known High-Risk Scenarios

Symptom	Possible Cause	How to Check	Recommended Action
`containerd` fails to start	The operating system upgrade overwrote the ACP runtime binary	Compare `containerd --version` with the pre-upgrade record	Stop the maintenance and contact technical support
The node stays `NotReady`	`kubelet`, container runtime, CNI, DNS, or node network configuration changed	Check `systemctl status kubelet`, `systemctl status containerd`, and node events	Restore the changed configuration or contact technical support
Cluster components report TLS errors	Node time drifted during or after the upgrade	Check `timedatectl` and compare `date` across nodes	Synchronize time before continuing
DNS resolution fails	`/etc/resolv.conf` was overwritten	Check `cat /etc/resolv.conf`	Restore the approved DNS configuration

Symptom	Possible Cause	How to Check	Recommended Action
`kubelet` fails with security policy errors	SELinux or AppArmor was re-enabled	Check `getenforce` or `systemctl status apparmor`	Disable the service according to node requirements
Workloads cannot be scheduled after uncordon	The node is still unhealthy, tainted, or resource-constrained	Check `kubectl describe node <node-name>`	Resolve node conditions before continuing

Operation Checklist

Use this checklist during the maintenance window and keep the completed record after the upgrade.

Phase	Item	Owner	Status
Pre-upgrade	Confirm that the target operating system and kernel are in the support matrix		
Pre-upgrade	Confirm that the kernel is the official vendor kernel		
Pre-upgrade	Check and resolve conflicting packages		
Pre-upgrade	Record `containerd`, `runc`, `crictl`, and `kubelet` versions		
Pre-upgrade	Record DNS, time synchronization, `/etc/fstab`, and runtime configuration		
Pre-upgrade	Confirm that the cluster has enough capacity for drained workloads		

Phase	Item	Owner	Status
Pre-upgrade	Drain the target node and confirm workload impact		
Control plane only	Back up etcd data		
Control plane only	Confirm etcd health and quorum		
Upgrade	Run the operating system vendor's supported upgrade procedure		
Upgrade	Reboot the node if required by the vendor procedure		
Post-upgrade	Confirm operating system and kernel versions		
Post-upgrade	Confirm SELinux or AppArmor is disabled as required		
Post-upgrade	Confirm swap is disabled		
Post-upgrade	Confirm firewall services match the cluster network plan		
Post-upgrade	Confirm `/tmp` is not mounted with `noexec`		
Post-upgrade	Confirm `DefaultTasksMax` is `infinity`		
Post-upgrade	Confirm runtime and node component versions were not overwritten		
Post-upgrade	Confirm `containerd` and `kubelet` are healthy		

Phase	Item	Owner	Status
Post-upgrade	Confirm time skew between nodes is no more than 10 seconds		
Post-upgrade	Confirm DNS configuration is correct		
Post-upgrade	Confirm the node is `Ready`		
Post-upgrade	Resume scheduling on the node		
Post-upgrade	Confirm workloads and business services are healthy		

Node Monitoring

View node monitoring data on the node details page.

TIP

- When a cluster has more than 1 node, you can click the ***current node name*** in the resource path area on the node details page to expand the node dropdown list, then click to select a node for quick switching to other node details pages.
- When monitoring components are configured for the cluster, you can view node monitoring data including resource runtime status, resource usage, and resource trend statistics.

TOC

Procedure

1. In the left navigation bar, click **Clusters** > **Clusters**.
2. Click the ***cluster name*** where the target node is located.
3. Under the **Nodes** tab, click the target ***node name***.

4. Click the **Monitoring** tab to enter the node monitoring data display page and view relevant node monitoring data.

> **TIP**
>
> - Hover over a card and click the **Details** icon to view PromQL expressions; click the **Export** icon to export PromQL expressions for all charts on the current page.
> - When a cluster has more than 1 node, you can click the ***current node name*** in the resource path area on the node details page to expand the node dropdown list, then click to select a node for quick switching to other node details pages.

> **TIP**
>
> In the storage space statistics display area, when a node has more than 4 storage partitions:
>
> - In the partition total usage pie chart, the top 3 partitions with the highest usage are displayed separately, while remaining partitions are shown as **Others** with their total usage data displayed when hovering over the area;
> - In the partition usage bar chart, the top 3 partitions with the highest usage are displayed separately, while remaining partitions are shown as **Others** with their total usage and individual usage rates displayed when hovering over the bars.

The monitoring trend statistics are described in the following table.

Parameter	Description
CPU	**Usage rate**, **request rate**, and **limit rate** of CPU within the specified time range. **Usage rate** = CPU usage of all pods on the node / Total CPU of the node. **Note**: If the CPU usage rate of a node spikes during a certain period, you must first identify the process consuming the most CPU resources. For example, for Java custom applications, memory leaks or infinite loops in the code may cause high CPU usage.

Parameter	Description
	Request rate = CPU requests of all pods on the node / Total CPU of the node. **Note**: If the CPU request rate of a node spikes during a certain period, it may be due to unreasonable cluster oversubscription ratio settings or excessively high request values for pods running on the node, which may cause resource waste. **Limit rate** = CPU limits of all pods on the node / Total CPU of the node. **Note**: If the CPU limit rate of a node spikes during a certain period, it indicates that the limit values for pods running on the node are set too high, which may cause CPU resource waste.
Memory	**Usage rate**, **request rate**, and **limit rate** of memory within the specified time range. **Usage rate** = Memory usage of all pods on the node / Total memory of the node. Memory is one of the important components on a server and serves as a bridge for CPU communication. Therefore, memory performance has a significant impact on the machine. When programs run, data loading, thread concurrency, and I/O buffering all depend on memory. The available memory size determines whether programs can run normally and how they run. **Request rate** = Memory requests of all pods on the node / Total memory of the node. **Note**: If the memory request rate of a node spikes during a certain period, it may be due to unreasonable cluster oversubscription ratio settings or excessively high request values for pods running on the node, which may cause resource waste. **Limit rate** = Memory limits of all pods on the node / Total memory of the node. **Note**: If the memory limit rate of a node spikes during a certain period, it

Parameter	Description
	indicates that the limit values for pods running on the node are set too high, which may cause memory resource waste.
Storage	**Space usage rate** and **inode usage rate** within the specified time range. **Space usage rate** = Storage space used / Total storage space. By monitoring historical disk space data, you can evaluate disk usage during a given time period. When disk usage is high, you can free up disk space by cleaning up unnecessary images or containers. **Inode usage rate** = Inode storage used / Total inode storage. **Note**: Every file must have an inode to store file metadata such as file creator and creation date. Inodes also consume disk space, and many small cache files can easily lead to inode resource exhaustion. Additionally, when inodes are exhausted but the disk is not full, new files cannot be created on the disk.
System Load	Average CPU load over 1 minute, 5 minutes, and 15 minutes. The value is the ratio of the total number of processes currently being executed by the CPU and waiting to be executed by the CPU to the maximum number of processes the CPU can execute, which is an important indicator of system busy/idle status. **Note**: If the 1-minute/5-minute/15-minute curves are similar over a certain period, it indicates that the cluster's CPU load is relatively stable. If the 1-minute value is much greater than the 15-minute value at a certain time period or specific time point, it indicates that the load in the recent 1 minute is increasing and needs continued observation. Once the 1-minute value exceeds the number of CPUs, it may indicate system overload. You need to further analyze the root cause of the problem. If the 1-minute value is much smaller than the 15-minute value at a certain time period or specific time point, it indicates that the system load is decreasing in the recent 1 minute and generated high load in the previous 15 minutes.

Parameter	Description
Disk Throughput	Disk throughput within the specified time range refers to the speed of data flow transmission by the disk, where transmission data is the sum of read and write data.
Disk IOPS	Disk IOPS within the specified time range is the sum of continuous reads and writes per second, representing a performance metric of the number of read and write operations per second by the disk.
Network Traffic Rate	Network traffic inflow and outflow rates within the specified time range, counted by the node's physical network interface.
Network Packet Rate (packets/sec)	Network packet receive and send rates within the specified time range, counted by the node's physical network interface.

Cluster Node Planning

A cluster uses Kubernetes node role labels in the format `node-role.kubernetes.io/<role>` to assign different roles to nodes. In this topic, `role label` refers to this type of label.

By default, a cluster contains two types of nodes: control plane nodes and worker nodes, used to host control plane workloads and application workloads, respectively.

In a cluster:

- The control plane nodes are labeled with the role label `node-role.kubernetes.io/control-plane`.

  > **Note:**
  > Prior to Kubernetes v1.24, the community also used the label `node-role.kubernetes.io/master` to mark control plane nodes. For backward compatibility, both labels are considered valid for identifying control plane nodes.

- The worker nodes, by default, have no role labels. However, you can explicitly assign the role label `node-role.kubernetes.io/worker` to a worker node if desired.

In addition to these default role labels, you can also define custom role labels on worker nodes to further classify them into different functional types. For example:

- You can add the role label `node-role.kubernetes.io/infra` to designate a node as an infra node, intended for hosting infrastructure components.
- You can add the role label `node-role.kubernetes.io/log` to designate a node as a log node, specialized for hosting logging components.

Use role labels and taints to create infra nodes or custom role nodes, then move selected workloads to those nodes when workload isolation requires it.

TOC

Creating Infra Nodes on Non-Immutable Cluster

By default, a cluster only includes control plane nodes and worker nodes. If you want to designate certain worker nodes as infra nodes dedicated to hosting infrastructure components, you need to manually add the appropriate role label and taint to those nodes.

> **Note:**

> The operations in this section apply only to non-immutable clusters. Do not use these operations for provider-managed cloud clusters, third-party clusters, or clusters where the nodes use an immutable OS.

Adding Infra Nodes

Step 1: Add the Infra Role Label to the Node resources

```
kubectl label nodes 192.168.143.133 node-role.kubernetes.io/infra="" --overwrite
```

This command adds the infra role label to the Node 192.168.143.133: `node-role.kubernetes.io/infra: ""`, indicating that the node is an infra node.

Step 2: Add a Taint to the Node resources

Add a taint to prevent other workloads from being scheduled onto the infra node.

```
kubectl taint nodes 192.168.143.133 node-role.kubernetes.io/infra=reserved:NoSchedule
```

This command adds the taint `node-role.kubernetes.io/infra=reserved:NoSchedule` to Node 192.168.143.133, indicating that only applications that tolerate this taint can be scheduled onto this node.

Step 3: Verify the Label and Taint

Check whether the node has been assigned the infra role label and taint:

```
# kubectl describe node 192.168.143.133
Name:               192.168.143.133
Roles:              infra
Labels:             node-role.kubernetes.io/infra=""
                    ...
Taints:             node-role.kubernetes.io/infra=reserved:NoSchedule
```

The output indicates that the Node `192.168.143.133` has been configured as an infra node and has the `node-role.kubernetes.io/infra=reserved:NoSchedule` taint.

Migrating Pods to Infra Nodes

If you want to schedule a specific Pod onto infra nodes, you need to make the following configurations:

- A nodeSelector targeting the infra role label.
- Corresponding tolerations for the infra node's taint.

Below is an example Pod manifest configured to run on the infra node.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: infra-pod-demo
  namespace: default
spec:
  ...
  nodeSelector:
    node-role.kubernetes.io/infra: ""
  tolerations:
  - effect: NoSchedule
    key: node-role.kubernetes.io/infra
    value: reserved
    operator: Equal
  ...
```

The nodeSelector ensures the Pod is only scheduled on nodes with the label `node-role.kubernetes.io/infra: ""`, and the toleration allows the Pod to tolerate the taint `node-role.kubernetes.io/infra=reserved:NoSchedule`.

With these configurations, the Pod will be scheduled onto the infra node.

> **Note:**
> Moving pods installed through OLM Operators or Cluster Plugins to an infra node is not always possible. The capability to move these pods depends on the configuration of each

> Operator or Cluster Plugin.

Custom Node Planning

Beyond infra nodes, you may want to designate worker nodes for other specialized purposes — such as hosting logging components, storage services, or monitoring agents.

You can achieve this by assigning more custom role labels and corresponding taints to worker nodes, effectively turning them into custom role nodes.

General Steps for Defining Custom Role Nodes

The process is similar to creating infra nodes.

Step 1: Add a Custom Role Label

```
kubectl label nodes <node> node-role.kubernetes.io/<role>="" --overwrite
```

Replace <role> with your desired role name, such as monitoring, storage, or log.

Step 2: Add a Corresponding Taint

```
kubectl taint nodes <node> node-role.kubernetes.io/<role>=<value>:NoSchedule
```

Replace <role> with your custom role name and replace <value> with a meaningful descriptor, such as reserved or dedicated. This value is optional but can help operators understand the scheduling intent.

Step 3: Verify the Configuration

```
kubectl describe node <node>
```

Ensure the Labels and Taints fields reflect your custom role configuration.

Example: Create A Node Dedicated To Logging Components

If you want to create a node specifically for installing logging components, you can add the log role. In this case, create the log node as follows.

Step 1: Add the Log Role Label

```
kubectl label nodes 192.168.143.133 node-role.kubernetes.io/log="" --over
write
```

This label indicates that the node is designated for log-related workloads.

Step 2: Add a Taint to the Node

```
kubectl taint nodes 192.168.143.133 node-role.kubernetes.io/log=reserved:
NoSchedule
```

This taint prevents unscheduled workloads from being deployed to the node.

Step 3: Verify the Label and Taint

```
Name:               192.168.143.133
Roles:              log
Labels:             node-role.kubernetes.io/log=""
                    ...
Taints:             node-role.kubernetes.io/log=reserved:NoSchedule
```

This confirms that the node has been successfully configured as a log node with the appropriate label and taint.

Use role labels and taints to partition Kubernetes nodes by purpose, improve workload isolation, and schedule selected components onto appropriately configured nodes.