Machine Configuration

Machine Configuration provides centralized management of the underlying operating system, allowing the platform to define and distribute OS configurations (such as kernel parameters, system services, and network settings) in a declarative manner. This ensures consistent configuration and runtime environments across all nodes, reducing operational complexity.

Machine Configuration applies only to clusters whose nodes run Alauda OS. On these clusters, the node operating system is immutable, and configuration updates are applied through controlled images or configuration policies, ensuring repeatability and system stability. Machine Configuration is not supported on nodes that run a non-Alauda OS.

Machine Configuration is delivered with Alauda OS clusters. It does not require a separate installation and does not have a separate release cadence. Its changes are described in the Alauda Container Platform Release Notes.

Overview

Managing Node Configuration

Managing F

Tuning Node Kernel Parameters

Node Disruption Policies

Overview

TOC

How Machine Configuration Works

Key Concepts

Configuration Types

Node Update Process

Verify Machine Configuration Is Running

Checking MachineConfigPool Status

How Machine Configuration Works

Machine Configuration handles file updates, systemd unit management, and SSH public key deployment across cluster nodes. The system provides a MachineConfig Custom Resource Definition (CRD) for writing configuration files to hosts, and a MachineConfigPool CRD for organizing nodes into configuration groups.

Each MachineConfigPool governs a set of nodes and their associated MachineConfigs. Node roles determine MachineConfigPool membership—pools manage nodes based on their role labels.

Machine Configuration manages only nodes that run Alauda OS. Control plane nodes are managed by the control plane pool and all other nodes by the worker pool.

Machine Configuration is delivered with Alauda OS clusters and does not require a separate installation. When the Machine Configuration Operator first starts on a cluster, it automatically creates two MachineConfigPools (control plane and worker) along with two empty MachineConfigs (00-master and 00-worker). The control plane pool manages the 00-master configuration, while the worker pool manages the 00-worker configuration.

You can create custom MachineConfigPools for worker nodes that require specialized configurations. Control plane nodes cannot use custom pools.

Custom MachineConfigPools inherit all configurations from the worker pool and add their own specific settings. Any changes to the worker pool automatically propagate to custom pools. Machine Configuration does not support custom pools that don't inherit from the worker pool.

The operator also creates a default MachineConfiguration CR named "cluster" for setting global node update policies. See the Node Disruption Policy documentation for details.

Sometimes node configurations drift from their intended state. The machine-config-daemon continuously monitors for configuration drift and marks affected nodes as Degraded until an administrator resolves the issue. Degraded nodes remain operational but cannot receive updates.

Key Concepts

Configuration Processing MachineConfigs are processed alphabetically. The first configuration serves as the base, with subsequent configs layered on top. Each MachineConfigPool renders its managed configs into a single MachineConfig named: `render-<pool-name>-<content-hash>`, which gets applied to all nodes in that pool.

Update Strategy Machine Configuration updates nodes by age, starting with the oldest. The `maxUnavailable` field in each MachineConfigPool controls how many nodes update simultaneously.

Scope of Management Machine Configuration only manages explicitly configured items. Manual system changes remain untouched by the Machine Configuration Operator.

Configuration Format All MachineConfigs use the Ignition v3.4.0 specification format.

Drift Detection When Machine Configuration-managed files change outside the system, machine-config-daemon marks the node as Degraded but doesn't overwrite the modified files.

Pool Benefits MachineConfigPools ensure that new nodes automatically receive the correct configuration when they join the cluster.

Supported Modifications

- Regular files (in writable, non-root directories)
- systemd units and their configurations
- SSH public keys for the boot user only

Machine Configuration doesn't create users or groups. You must create the boot user and group before configuring SSH keys.

Important: Avoid manual node modifications—they can cause configuration conflicts.

Configuration Types

Files Create or modify file content and permissions. Files can only be managed if their containing partition is writable.

Systemd Units Define new systemd services or extend existing ones with additional configuration.

SSH Public Keys Configure SSH access for the boot user. Keys for other users are rejected as invalid.

Node Update Process

When you apply a MachineConfig, Machine Configuration ensures all affected nodes reach the desired state. The Machine Configuration Operator generates a new rendered configuration and machine-config-daemon executes these steps on each node:

1. **Cordon** - Mark node unschedulable for new workloads
2. **Drain** - Terminate existing workloads and reschedule them elsewhere

3. **Apply** - Write the new configuration to disk
4. **Reboot** - Restart the node to activate changes
5. **Uncordon** - Mark node schedulable again

Verify Machine Configuration Is Running

Before you create `MachineConfig` objects on a cluster, confirm that Machine Configuration is running on it.

1. Confirm that both components are up. The controller runs on control plane nodes; the daemon runs on every node:

```
kubectl -n cpaas-system get deployment machine-config-controller
kubectl -n cpaas-system get daemonset machine-config-daemon
```

2. Confirm that the operator created its initial objects:

```
kubectl get machineconfigpool
```

```
NAME     CONFIG                   UPDATED   UPDATING   DEGRADED   MACHI
NECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT   DEGRADEDMACHINECOUN
T   AGE
master   rendered-master-06c9c4   True      False      False      3
3                  3                   0                    3m
worker   rendered-worker-f4b64    True      False      False      3
3                  3                   0                    3m
```

```
kubectl get machineconfig
```

```
NAME        IGNITIONVERSION   AGE
00-master   3.4.0             3m
00-worker   3.4.0             3m
```

```
kubectl get machineconfiguration cluster
```

3. If the pools or the base configurations do not appear, check the controller logs. The controller creates all of them at startup:

```
kubectl -n cpaas-system logs deployment/machine-config-controller
```

Checking MachineConfigPool Status

Check pool status with:

```
kubectl get machineconfigpool
```

Example output:

```
NAME     CONFIG                    UPDATED  UPDATING  DEGRADED  MACHINECO
UNT  READYMACHINECOUNT  UPDATEDMACHINECOUNT  DEGRADEDMACHINECOUNT  AGE
master   rendered-master-06c9c4    True     False     False     3
3                  3                    0                   4h42m
worker   rendered-worker-f4b64     False    True      False     3
2                  2                    0                   4h42m
```

Field Reference:

- **NAME**: Pool identifier
- **CONFIG**: Most recently applied configuration across all pool nodes
- **UPDATED**: `True` when all nodes have the current config; `False` during updates
- **UPDATING**: `True` when at least one node is updating; `False` when all are current
- **DEGRADED**: `True` when config cannot be applied to at least one node
- **MACHINECOUNT**: Total nodes in the pool
- **READYMACHINECOUNT**: Nodes with current config in healthy, schedulable state
- **UPDATEDMACHINECOUNT**: Nodes that have applied the current config

- **DEGRADEDMACHINECOUNT**: Nodes marked as degraded or unreconcilable

In this example, all three control plane nodes are current, while the worker pool is updating—two nodes are complete and one is in progress.

Get detailed pool information:

```
kubectl describe machineconfigpool worker
```

View all MachineConfigs:

```
kubectl get machineconfig
```

Example output:

```
NAME                     IGNITIONVERSION  AGE
00-master                3.4.0            3h2m
00-worker                3.4.0            3h2m
rendered-master-ccb      3.4.0            1h12m
rendered-worker-bad      3.4.0            1h20m
```

Examine specific configurations:

```
kubectl describe machineconfig 00-master
```

Check individual node status:

```
kubectl get node -o custom-columns=NODE:.metadata.name,DESIRED:.metadata.
annotations."machineconfiguration\.alauda\.io/desiredConfig",CURRENT:.met
adata.annotations."machineconfiguration\.alauda\.io/currentConfig",STAT
E:.metadata.annotations."machineconfiguration\.alauda\.io/state"
```

Example output:

```
NODE              DESIRED                                          CURRENT
STATE
192.168.132.216   rendered-master-98db9ca4f4b4cd                   rendered-mas
ter-98db9ca4f4b4cd              Degraded
192.168.135.83    rendered-worker-05f27341ba49cf86dc4b      rendered-mast
er-e08d9cab50e383              Working
192.168.134.99    rendered-worker-05f27341ba49cf86dc4b      rendered-work
er-05f27341ba49cf86dc4b      Done
```

Node State Reference:

- **NODE**: Node identifier
- **DESIRED**: Target configuration for the node
- **CURRENT**: Currently applied configuration
- **STATE**: Configuration status
  - `Done`: Node healthy with matching desired and current configs
  - `Working`: Node updating (current ≠ desired)
  - `Degraded`: Configuration drift detected or application failed—check logs for root cause

Managing Node Configuration

You can use the tasks described in this section to create `MachineConfig` objects that modify files, systemd units, and SSH public keys on nodes, as well as to recover nodes that have experienced configuration drift.

`MachineConfig` supports Ignition specification version 3.4. All `MachineConfig` objects must be created in compliance with this version.

In certain situations, the configuration on a node may not fully match the configuration currently applied through the `MachineConfig`. This condition is referred to as configuration drift. The machine configuration daemon periodically verifies whether a node's configuration has drifted. If drift is detected, the node is marked as `Degraded` and remains in that state until an administrator restores the expected configuration.

The following examples demonstrate how to use `MachineConfig` objects to manage node configurations.

TOC

Configuring the Chrony Time Service

Disabling the Chrony Time Service

Configuring the SSH Public Key for the `boot` User

Recovering from Configuration Drift

Configuring the Chrony Time Service

To configure the Chrony time synchronization service (`chronyd`) and specify the NTP servers and related settings, you can update the `chrony.conf` file on the target nodes via a `MachineConfig` object.

1. First, create a temporary file that contains the desired Chrony configuration:

```
chrony.conf
server 0.centos.pool.ntp.org iburst
server 1.centos.pool.ntp.org iburst
makestep 1.0 3
rtcsync
logdir /var/log/chrony
```

2. Then, base64-encode the contents of the file:

```
base64 -w0 chrony.conf
```

3. Create a `MachineConfig` object named `99-worker-chrony` . In the `.spec.config.storage.files[0].contents.source` field, insert the base64-encoded string in the format `data:text/plain;base64,<encoded-content>` :

```
apiVersion: machineconfiguration.alauda.io/v1alpha1
kind: MachineConfig
metadata:
  name: 99-worker-chrony
  labels:
    machineconfiguration.alauda.io/role: worker
spec:
  config:
    ignition:
      version: 3.4.0
    storage:
      files:
        - path: /etc/chrony.conf
          mode: 0o644
          contents:
            source: 'data:text/plain;base64,c2VydmVyIDAuY2VudG9zLnBvb2w
ubnRwLm9yZyBpYnVyc3QKc2VydmVyIDEuY2VudG9zLnBvb2wubnRwLm9yZyBpYnVyc3QKbW
FrZXN0ZXAgMS4wIDMKcnRjc3luYwpsb2dkaXIgL3Zhci9sb2cvY2hyb255Cg=='
```

This configuration creates a `MachineConfig` object that applies a customized `chrony.conf` file to nodes associated with the `worker` machine configuration pool. The file will be written to `/etc/chrony.conf` on each node, with file permissions set to `0o644`.

To make the change take effect, configure a node disruption policy for `/etc/chrony.conf` that restarts the service:

```yaml
apiVersion: machineconfiguration.alauda.io/v1alpha1
kind: MachineConfiguration
metadata:
  name: cluster
spec:
  nodeDisruptionPolicy:
    files:
      - path: /etc/chrony.conf
        actions:
          - type: DaemonReload
          - type: Restart
            restart:
              serviceName: chronyd.service
    sshkey:
      actions:
        - type: None
```

Use `Restart` for `chronyd`; the service does not support a reload action. Confirm the policy appears in `status.nodeDisruptionPolicyStatus` before applying the `MachineConfig`.

Disabling the Chrony Time Service

To disable the Chrony time synchronization service on nodes with a specific role, you can create a `MachineConfig` object that overrides the systemd unit definition and disables the service.

Example configuration:

```
apiVersion: machineconfiguration.alauda.io/v1alpha1
kind: MachineConfig
metadata:
  name: 99-worker-disable-chrony
  labels:
    machineconfiguration.alauda.io/role: worker
spec:
  config:
    ignition:
      version: 3.4.0
    systemd:
      units:
        - name: chronyd.service
          enabled: false
          contents: |
            [Unit]
            Description=NTP client/server
            Documentation=man:chronyd(8) man:chrony.conf(5)
            After=ntpdate.service sntp.service ntpd.service
            Conflicts=ntpd.service systemd-timesyncd.service
            ConditionCapability=CAP_SYS_TIME

            [Service]
            Type=forking
            PIDFile=/run/chrony/chronyd.pid
            EnvironmentFile=-/etc/sysconfig/chronyd
            ExecStart=/usr/sbin/chronyd $OPTIONS
            ExecStartPost=/usr/libexec/chrony-helper update-daemon
            PrivateTmp=yes
            ProtectHome=yes
            ProtectSystem=full

            [Install]
            WantedBy=multi-user.target
```

This configuration pushes a custom version of the `chronyd.service` unit file to the nodes in the `worker` machine configuration pool. The service is explicitly disabled. Once the configuration is applied and the nodes are rebooted, the Chrony service will no longer start automatically.

Configuring the SSH Public Key for the `boot` User

The machine configuration system allows you to configure an SSH public key for the `boot` user on managed nodes. Configuration for other user accounts is not supported. Note that machine configuration will not create users or groups automatically—you must ensure that the `boot` user and group exist on the node before applying the configuration.

Example configuration:

```
apiVersion: machineconfiguration.alauda.io/v1alpha1
kind: MachineConfig
metadata:
  name: 99-worker-ssh
  labels:
    machineconfiguration.alauda.io/role: worker
spec:
  config:
    ignition:
      version: 3.4.0
    passwd:
      users:
        - user: boot
          sshAuthorizedKeys:
            - ssh-rsa <ssh-public-key>
```

This `MachineConfig` will install the specified SSH key in the `/home/boot/.ssh/authorized_keys` file on nodes in the `worker` machine configuration pool.

Recovering from Configuration Drift

If a node's configuration diverges from its assigned `MachineConfig`, it will be marked as `Degraded`. In this state, the node continues to operate but cannot receive further configuration updates until the issue is resolved.

There are two ways to restore a node from this degraded state:

1. **Manually revert the configuration** You can manually adjust the files and permissions on the node to exactly match those specified in the assigned `MachineConfig`. The system will detect the correction and clear the degraded status.
2. **Force the configuration to be reapplied** Create an empty file at `/run/machine-config-daemon-force` on the affected node. The machine configuration daemon will detect this trigger, reapply the current `MachineConfig`, delete the trigger file, and reboot the node. After rebooting, the node will transition from `Degraded` back to `Done`.

Managing Firewall Ports

Each node's host firewall is preconfigured with the ports required by the platform. You do not need to manage these ports, and you should not modify the platform's preconfigured port list directly.

When a workload needs an additional host port to be open—for example, a component that listens directly on the node—you can open it by installing a small systemd unit with a `MachineConfig` object. The unit opens the ports at runtime with `firewall-cmd --add-port`, leaving the platform's preconfigured ports untouched.

TOC

- Why This Uses a Runtime Unit, Not a Firewalld Reload
- Opening Additional Ports
- Changing or Removing Ports
- Notes

Why This Uses a Runtime Unit, Not a Firewalld Reload

Reloading or restarting firewalld (`firewall-cmd --reload` , `systemctl restart firewalld`) rebuilds the host firewall from scratch. On a cluster node this can drop packet-

filtering rules that other components program directly—such as Istio's traffic-redirection rules, `kube-proxy`, or the CNI—causing intermittent connectivity loss. To avoid that, this procedure never reloads or restarts firewalld.

Instead, a oneshot systemd unit adds the ports to the running firewalld with `firewall-cmd --add-port`. This is purely additive and flushes nothing. The unit is ordered after firewalld and declared `PartOf=firewalld.service`, so if firewalld is ever restarted the unit runs again and re-adds the ports (a runtime `--add-port` does not survive a firewalld restart on its own).

Opening Additional Ports

The following example opens TCP ports `9141` (ZooKeeper) and `9308` (cpaas-kafka), two ports commonly needed by the logging stack. Change the `--add-port` list to the ports your workload requires.

1. Create a `MachineConfig` that installs a oneshot unit which opens the ports at runtime:

```
apiVersion: machineconfiguration.alauda.io/v1alpha1
kind: MachineConfig
metadata:
  name: 99-worker-firewall-ports
  labels:
    machineconfiguration.alauda.io/role: worker
spec:
  config:
    ignition:
      version: 3.4.0
    systemd:
      units:
        - name: open-app-ports.service
          enabled: true
          contents: |
            [Unit]
            Description=Open firewalld ports for workloads (runtime add
-port, no reload)
            After=firewalld.service
            Requires=firewalld.service
            PartOf=firewalld.service

            [Service]
            Type=oneshot
            RemainAfterExit=yes
            ExecStart=/usr/bin/firewall-cmd --add-port=9141/tcp --add-p
ort=9308/tcp

            [Install]
            WantedBy=multi-user.target
```

2. Configure a node disruption policy so that changes to this unit restart the unit—re-running `firewall-cmd`—instead of rebooting the node. Add the `open-app-ports.service` entry to the `nodeDisruptionPolicy.units` list of the cluster-scoped `MachineConfiguration` singleton named `cluster`, and confirm it is reflected in `status.nodeDisruptionPolicyStatus` **before** applying the `MachineConfig`:

```yaml
apiVersion: machineconfiguration.alauda.io/v1alpha1
kind: MachineConfiguration
metadata:
  name: cluster
spec:
  nodeDisruptionPolicy:
    units:
      - name: open-app-ports.service
        actions:
          - type: DaemonReload
          - type: Restart
            restart:
              serviceName: open-app-ports.service
    sshkey:
      actions:
        - type: None
```

After both objects are applied, the unit opens the ports on each targeted node with no firewalld reload and no node reboot; the platform's preconfigured ports and other components' rules are left intact.

> **WARNING**
>
> The `cluster` object is a singleton and may already carry other unit or file policies. Add `open-app-ports.service` to the existing `nodeDisruptionPolicy.units` list rather than replacing the whole policy. A managed unit that is **not** listed here falls back to the default action for units, which reboots the node when the unit changes.

Changing or Removing Ports

- **To add ports**, extend the `--add-port` list in the unit and re-apply the `MachineConfig`. The node disruption policy restarts the unit and the new ports are added at runtime.
- **To remove a port**, deleting it from the `--add-port` list is not sufficient by itself: `firewall-cmd --add-port` only adds, so a port that is already open at runtime stays open until firewalld is restarted. Remove the port from the unit, then restart firewalld on the affected nodes—or reboot them—to drop it.

Notes

- Ports are added to firewalld's default zone. If the relevant interface is bound to a non-default zone, add `--zone=<zone>` to the `firewall-cmd` command.
- `firewall-cmd --add-port` is idempotent: re-adding an already-open port returns a warning and still succeeds, so the oneshot unit is safe to re-run.
- The unit requires firewalld to be running. On a node that does not run host firewalld, the unit does not start and no ports are opened.
- A runtime `--add-port` is not written to firewalld's permanent configuration. If firewalld is reloaded out of band (`firewall-cmd --reload`), these ports are dropped and are re-added only the next time firewalld or the node restarts.

Tuning Node Kernel Parameters (sysctl)

Some kernel parameters are set per node rather than per workload—for example, the ARP/neighbor table garbage-collection thresholds. These are node-level (non-namespaced) settings, so they cannot be set from a Pod's `securityContext.sysctls` or a security context constraint; they must be applied on the host.

You can set them by writing a `sysctl` drop-in file with a `MachineConfig` object. The setting is written as a separate file under `/etc/sysctl.d/`, so the node's other kernel settings are left unchanged. Pair the `MachineConfig` with a node disruption policy so the value is applied without rebooting the node.

TOC

When to Use This

Setting a Kernel Parameter

Verifying the Change

Reverting the Change

When to Use This

A common case is a large cluster whose nodes log a `neighbour table overflow` error: the ARP/neighbor table is too small for the number of peers, and its garbage-collection thresholds

need to be raised. The example below raises those thresholds; the same procedure applies to any node-level `sysctl` key.

Setting a Kernel Parameter

The following example raises the ARP/neighbor table garbage-collection thresholds on worker nodes.

1. Create the `sysctl` drop-in file. Each line is a `key = value` pair:

```
net.ipv4.neigh.default.gc_thresh1 = 32768
net.ipv4.neigh.default.gc_thresh2 = 65536
net.ipv4.neigh.default.gc_thresh3 = 262144
```

2. Base64-encode the contents:

```
base64 -w0 99-neigh-thresh.conf
```

3. Create a `MachineConfig` object that writes the file under `/etc/sysctl.d/`:

```yaml
apiVersion: machineconfiguration.alauda.io/v1alpha1
kind: MachineConfig
metadata:
  name: 99-worker-neigh-thresh
  labels:
    machineconfiguration.alauda.io/role: worker
spec:
  config:
    ignition:
      version: 3.4.0
    storage:
      files:
        - path: /etc/sysctl.d/99-neigh-thresh.conf
          mode: 0o644
          overwrite: true
          contents:
            source: 'data:text/plain;base64,bmV0LmlwdjQubmVpZ2guZGVmYXV
sdC5nY190aHJlc2gxID0gMzI3NjgKbmV0LmlwdjQubmVpZ2guZGVmYXVsdC5nY190aHJlc2
gyID0gNjU1MzYKbmV0LmlwdjQubmVpZ2guZGVmYXVsdC5nY190aHJlc2gzID0gMjYyMTQ0C
g=='
```

The `role` label selects which nodes the file is applied to. To target only a subset of nodes, create a custom `MachineConfigPool` and set the label to that pool's role.

4. Configure a node disruption policy that re-applies `sysctl` settings when this file changes. Because the default action for file changes is `None`, the file would otherwise be written to disk without taking effect. Add the following to the cluster-scoped `MachineConfiguration` singleton named `cluster`, and confirm it is reflected in `status.nodeDisruptionPolicyStatus` **before** applying the `MachineConfig`:

```yaml
apiVersion: machineconfiguration.alauda.io/v1alpha1
kind: MachineConfiguration
metadata:
  name: cluster
spec:
  nodeDisruptionPolicy:
    files:
      - path: /etc/sysctl.d/99-neigh-thresh.conf
        actions:
          - type: Restart
            restart:
              serviceName: systemd-sysctl.service
    sshkey:
      actions:
        - type: None
```

Restarting `systemd-sysctl.service` re-applies every file under `/etc/sysctl.d/`. The new values take effect and the node is not rebooted.

> **WARNING**
>
> Set the node disruption policy for the file path first and wait until it appears in `status.nodeDisruptionPolicyStatus`. If you apply the `MachineConfig` before the policy is in effect, the file is written to disk but the value is not applied. For the full list of disruption actions, see Node Disruption Policies.

Verifying the Change

On a target node, read back the parameter:

```
sysctl net.ipv4.neigh.default.gc_thresh2
```

The output should show the value you set:

```
net.ipv4.neigh.default.gc_thresh2 = 65536
```

Reverting the Change

Delete the `MachineConfig` object to remove the drop-in file. Values that were already applied remain in effect on the running kernel until the node is next rebooted, after which the node's default values apply.

Setting the Node Time Zone

All nodes of a cluster should run with one agreed time zone, so that platform logs, audit records and monitoring timelines line up with the operator's wall clock. On Alauda OS nodes you set the time zone with a `MachineConfig` object — never by logging in to nodes and changing them one by one.

The time zone cannot be shipped as a file: `/etc/localtime` is a symbolic link, and machine configuration manages file contents, systemd units, and SSH public keys — not symbolic links. Instead, a oneshot systemd unit runs `timedatectl set-timezone` on each node.

TOC

Configure the Node Disruption Policy First

Apply the Time Zone

Verify

Related Notes

Configure the Node Disruption Policy First

Without a node disruption policy, a systemd unit change falls back to the default action — **rebooting the node** — so applying the `MachineConfig` below would roll every node in the pool through a drain and reboot. Configure the policy first, confirm it took effect, and only then apply the `MachineConfig`.

1. Add `set-timezone.service` to the `nodeDisruptionPolicy.units` list of the cluster-scoped `MachineConfiguration` singleton named `cluster`:

```
apiVersion: machineconfiguration.alauda.io/v1alpha1
kind: MachineConfiguration
metadata:
  name: cluster
spec:
  nodeDisruptionPolicy:
    units:
      - name: set-timezone.service
        actions:
          - type: DaemonReload
          - type: Restart
            restart:
              serviceName: set-timezone.service
```

> **WARNING**
>
> `cluster` is a singleton that may already carry policies for other units or files — for example the `/etc/chrony.conf` entry from Configuring the Chrony Time Service. **Append to the `units` list; do not replace the existing policy.** Any unit not listed falls back to the default reboot action.

2. Confirm the policy is active before going on:

```
kubectl get machineconfiguration cluster \
  -o jsonpath='{.status.nodeDisruptionPolicyStatus}' | grep set-timezon
e
```

Apply the Time Zone

Create one `MachineConfig` per role — the two objects differ only in `metadata.name` and the role label. Replace `Asia/Shanghai` with the agreed tz database name; do not deliver the example value unchanged.

```
apiVersion: machineconfiguration.alauda.io/v1alpha1
kind: MachineConfig
metadata:
  name: 99-worker-timezone # name the control plane copy 99-master-timezo
ne
  labels:
    machineconfiguration.alauda.io/role: worker # and label it master
spec:
  config:
    ignition:
      version: 3.4.0
    systemd:
      units:
        - name: set-timezone.service
          enabled: true
          contents: |
            [Unit]
            Description=Set node timezone
            After=multi-user.target

            [Service]
            Type=oneshot
            ExecStart=/usr/bin/timedatectl set-timezone Asia/Shanghai
            RemainAfterExit=yes

            [Install]
            WantedBy=multi-user.target
```

Verify

```
kubectl get machineconfigpool
```

Wait until each pool reports `UPDATED=True` and `DEGRADED=False`, and confirm no node went through an unexpected drain or reboot. Then spot-check nodes:

```
timedatectl                              # time zone and NTP state
systemctl status set-timezone.service    # ran once, exited successfully
```

Platform logs, audit records and monitoring timelines should now agree with the configured time zone.

Related Notes

- Time zone at install time is a host prerequisite — see the bare metal page's Host Time Synchronization requirements. This page is for normalizing or changing the time zone on a running cluster.
- Time synchronization is a separate concern, configured through Configuring the Chrony Time Service.

Node Disruption Policies

TOC

Understanding Node Disruption in Machine Configuration

What You Can Control with Node Disruption Policies

Example: Default Node Disruption Policy

Example: Customizing File Behavior

Example: Applying Policy to a Specific File

Understanding Node Disruption in Machine Configuration

By default, when you make certain changes to the `systemd` units section of a `MachineConfig` object, the Machine Configuration Operator will drain and reboot the nodes associated with that `MachineConfig`. However, changes to regular file entries typically do **not** cause a reboot, which may result in the configuration not taking effect as expected. To address this, you can define node disruption policies to specify which types of changes should trigger node reboots or other disruption actions. These policies are configured in the cluster-scoped `MachineConfiguration` singleton named `cluster`. See the example below for configuring a node disruption policy.

Once defined, the Machine Configuration Operator validates the policy to detect issues such as invalid formatting. Then, it populates the policy into the `status.nodeDisruptionPolicyStatus` field of the `MachineConfiguration` object. These user-defined policies override the cluster's default disruption settings.

The Machine Configuration Operator creates this resource when it first starts, with the default policy shown below.

> **WARNING**
>
> Because the default action for file changes is `None`, a `MachineConfig` object that only writes or updates a file is applied to disk but does **not** restart the affected service—so the change may not take effect. When a file change requires a service to reload or restart (for example, kubelet or firewalld configuration), define a node disruption policy for that file path, and confirm it appears in `status.nodeDisruptionPolicyStatus` **before** applying the `MachineConfig`.

What You Can Control with Node Disruption Policies

Node disruption policies allow you to define what happens when changes are made to the following configuration areas:

- **Files**: You can define behavior for file changes (excluding changes to the root directory). By default, file changes do not trigger any disruption. You can modify this behavior using the `spec.defaultNodeDisruptionPolicySpecAction.files` field.
- **Systemd Units**: You can create or modify `systemd` services, including enabling, disabling, or changing their state. By default, changes to `systemd` units trigger a node drain and reboot.
- **SSH Public Keys**: You can add or update SSH keys for the `boot` user. These changes are applied immediately by default and do not trigger a reboot or drain.

Each change is evaluated against the node disruption policy, which can trigger one or more of the following actions:

- `Reboot`: Drain and reboot the node.
- `None`: No disruption is triggered; the change is applied silently.
- `Drain`: Drain the node without rebooting.
- `Reload`: Reload the specified `systemd` service without restarting it.
- `Restart`: Restart the specified `systemd` service.
- `DaemonReload`: Reload all `systemd` unit configurations.

Example: Default Node Disruption Policy

The following is the default configuration for the `cluster` `MachineConfiguration` resource after installation:

```yaml
apiVersion: machineconfiguration.alauda.io/v1alpha1
kind: MachineConfiguration
metadata:
  name: cluster
spec:
  defaultNodeDisruptionPolicySpecAction:
    files:
    - type: None
    units:
    - type: Reboot
  nodeDisruptionPolicy:
    sshkey:
      actions:
      - type: None
```

This configuration means:

- File changes trigger no action.
- Systemd unit changes cause a node drain and reboot.
- SSH key changes are applied without disruption.

Example: Customizing File Behavior

You can change the default action for file changes to trigger a reboot:

```yaml
apiVersion: machineconfiguration.alauda.io/v1alpha1
kind: MachineConfiguration
metadata:
  name: cluster
spec:
  defaultNodeDisruptionPolicySpecAction:
    files:
    - type: Reboot
    units:
    - type: Reboot
  nodeDisruptionPolicy:
    sshkey:
      actions:
      - type: None
```

With this configuration, any change to a managed file will cause the Machine Configuration Operator to drain and reboot the affected node.

Example: Applying Policy to a Specific File

You can also define disruption actions for a specific file path. In the following example, changes to `/usr/local/bin/myapp.sh` will not trigger a node reboot, but instead reload the systemd configuration and restart the related service:

```yaml
apiVersion: machineconfiguration.alauda.io/v1alpha1
kind: MachineConfiguration
metadata:
  name: cluster
spec:
  defaultNodeDisruptionPolicySpecAction:
    files:
    - type: Reboot
    units:
    - type: Reboot
  nodeDisruptionPolicy:
    files:
    - path: /usr/local/bin/myapp.sh
      actions:
      - type: DaemonReload
      - type: Restart
        restart:
          serviceName: myapp.service
    sshkey:
      actions:
      - type: None
```

In this case, when `/usr/local/bin/myapp.sh` is updated, the Machine Configuration Operator will reload all `systemd` units and restart the `myapp.service`—without draining or rebooting the node.