Menu

Machine Configuration

Overview

TOC

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.

During cluster installation, the system automatically creates two MachineConfigPools (master and worker) along with two empty MachineConfigs (00-master and 00-worker). The master 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. Master 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 cluster includes 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

Checking MachineConfigPool Status

Check pool status with:

```
kubectl get machineconfigpool
```

Example output:

```
NAME     CONFIG                    UPDATED  UPDATING  DEGRADED  MACHINECOUNT
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 master 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/c
```

Example output:

```
NODE              DESIRED                                      CURRENT
STATE
192.168.132.216   rendered-master-98db9ca4f4b4cd               rendered-master-
98db9ca4f4b4cd             Degraded
192.168.135.83    rendered-worker-05f27341ba49cf86dc4b       rendered-master-
e08d9cab50e383             Working
192.168.134.99    rendered-worker-05f27341ba49cf86dc4b       rendered-worker-
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 with MachineConfig

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

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>` :

```yaml
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: 0644
          contents:
            source:
'data:text/plain;base64,c2VydmVyIDAuY2VudG9zLnBvb2wubnRwLm9yZyBpYnVyc3QKc2VydmVyIDEuY2Vud
```

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 `0644`.

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:

```yaml
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:

```yaml
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`.

Node Disruption Policies

TOC

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 `MachineConfiguration` object located in the `cpaas-system` namespace. 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.

A default `MachineConfiguration` custom resource named `cluster` is installed with the cluster. You can configure node disruption behavior on this resource.

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.
- `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.