机器配置

机器配置提供对底层操作系统的集中管理，使平台能够以声明式方式定义和分发 OS 配置（例如内核参数、系统服务和网络设置）。这可确保所有节点的配置和运行时环境保持一致，从而降低运维复杂性。

机器配置仅适用于运行 Alauda OS 的节点所在的集群。在这些集群中，节点操作系统不可变，配置更新通过受控映像或配置策略应用，以确保可重复性和系统稳定性。机器配置不支持运行非 Alauda OS 的节点。

机器配置随 Alauda OS 集群一同交付。它无需单独安装，也没有独立的发布节奏。其变更记录在 灵雀云容器平台 Release Notes 中。

概览 | 管理节点配置 | 管理防火墙

调整节点内核参数（**sysctl**） | 设置节点时区 | 节点中断策略

概览

目录

机器配置的工作原理

关键概念

配置类型

节点更新过程

验证 Machine Configuration 是否正在运行

检查 MachineConfigPool 状态

机器配置的工作原理

Machine Configuration 负责在集群节点之间更新文件、管理 systemd 单元以及部署 SSH 公钥。系统提供 MachineConfig Custom Resource Definition (CRD)，用于将配置文件写入主机；同时提供 MachineConfigPool CRD，用于将节点组织到配置组中。

每个 MachineConfigPool 管理一组节点及其关联的 MachineConfig。节点角色决定其 MachineConfigPool 成员身份——池根据节点的角色标签管理节点。

Machine Configuration 仅管理运行 Alauda OS 的节点。控制平面节点由控制平面池管理，其他所有节点由 worker 池管理。

Machine Configuration 随 Alauda OS 集群一起提供，无需单独安装。当 Machine Configuration Operator 首次在集群上启动时，它会自动创建两个 MachineConfigPool（控制平

面和 worker），以及两个空的 MachineConfig（00-master 和 00-worker）。控制平面池管理 00-master 配置，而 worker 池管理 00-worker 配置。

您可以为需要专用配置的 worker 节点创建自定义 MachineConfigPool。控制平面节点不能使用自定义池。

自定义 MachineConfigPool 会继承 worker 池的所有配置，并添加自身的特定设置。对 worker 池的任何更改都会自动传播到自定义池。Machine Configuration 不支持不继承 worker 池的自定义池。

operator 还会创建一个名为 "cluster" 的默认 MachineConfiguration CR，用于设置全局节点更新策略。有关详细信息，请参阅节点中断策略文档。

有时，节点配置会偏离其预期状态。machine-config-daemon 会持续监控配置偏差，并将受影响的节点标记为 Degraded，直到管理员解决问题。Degraded 节点仍可正常运行，但无法接收更新。

关键概念

配置处理

MachineConfig 按字母顺序处理。第一个配置作为基础，后续配置逐层叠加。每个 MachineConfigPool 都会将其管理的配置渲染为一个名为 `render-<pool-name>-<content-hash>` 的 MachineConfig，并将其应用于该池中的所有节点。

更新策略

Machine Configuration 按节点年龄更新节点，从最旧的节点开始。每个 MachineConfigPool 中的 `maxUnavailable` 字段控制同时更新的节点数量。

管理范围

Machine Configuration 仅管理明确配置的项目。手动进行的系统更改不会受到 Machine Configuration Operator 的影响。

配置格式

所有 MachineConfig 均使用 Ignition v3.4.0 规范格式。

偏差检测

当 Machine Configuration 管理的文件在系统外部发生更改时，machine-config-daemon 会将节点标记为 Degraded，但不会覆盖已修改的文件。

池的优势

MachineConfigPool 可确保新节点加入集群时自动获得正确的配置。

支持的修改

- 普通文件（位于可写的非根目录中）
- systemd 单元及其配置
- 仅适用于 boot 用户的 SSH 公钥

Machine Configuration 不会创建用户或组。配置 SSH 密钥之前，必须先创建 boot 用户和组。

重要信息：避免手动修改节点，否则可能导致配置冲突。

配置类型

文件

创建或修改文件内容及权限。只有在文件所在分区可写时，才能管理这些文件。

Systemd 单元

定义新的 systemd 服务，或使用其他配置扩展现有服务。

SSH 公钥

为 boot 用户配置 SSH 访问权限。其他用户的密钥会被视为无效并拒绝。

节点更新过程

应用 MachineConfig 时，Machine Configuration 会确保所有受影响的节点达到所需状态。Machine Configuration Operator 会生成新的渲染配置，machine-config-daemon 则在每个节点

上执行以下步骤：

1. 隔离 - 将节点标记为不可调度新工作负载
2. 排空 - 终止现有工作负载，并将其重新调度到其他位置
3. 应用 - 将新配置写入磁盘
4. 重启 - 重启节点以激活更改
5. 解除隔离 - 将节点标记为可再次调度

验证 **Machine Configuration** 是否正在运行

在集群上创建 `MachineConfig` 对象之前，请确认 Machine Configuration 正在运行。

1. 确认两个组件均已启动。控制器在控制平面节点上运行；守护进程在每个节点上运行：

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

2. 确认 operator 已创建其初始对象：

```
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. 如果未显示分区或基础配置，请检查控制器日志。控制器会在启动时创建所有这些对象：

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

检查 MachineConfigPool 状态

使用以下命令检查分区状态：

```
kubectl get machineconfigpool
```

示例输出：

```
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
```

字段参考：

- **NAME**：分区标识符
- **CONFIG**：应用于分区所有节点的最新配置
- **UPDATED**：所有节点均使用当前配置时为 `True`；更新期间为 `False`
- **UPDATING**：至少有一个节点正在更新时为 `True`；所有节点均为最新状态时为 `False`
- **DEGRADED**：至少有一个节点无法应用配置时为 `True`
- **MACHINECOUNT**：分区中的节点总数
- **READYMACHINECOUNT**：使用当前配置且处于健康、可调度状态的节点数
- **UPDATEDMACHINECOUNT**：已应用当前配置的节点数
- **DEGRADEDMACHINECOUNT**：标记为降级或无法协调的节点数

在此示例中，三个控制平面节点均为最新状态，而 worker 分区正在更新——其中两个节点已完成，一个节点正在更新。

获取详细的分区信息：

```
kubectl describe machineconfigpool worker
```

查看所有 MachineConfig：

```
kubectl get machineconfig
```

示例输出：

```
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
```

检查特定配置：

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

检查单个节点的状态：

```
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"
```

示例输出：

```
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
```

节点状态参考：

- **NODE**：节点标识符
- **DESIRED**：节点的目标配置
- **CURRENT**：当前已应用的配置
- **STATE**：配置状态
  - `Done`：节点健康，目标配置与当前配置匹配
  - `Working`：节点正在更新（当前配置 ≠ 目标配置）
  - `Degraded`：检测到配置漂移或应用失败——请检查日志以确定根本原因

管理节点配置

您可以使用本节所述的任务创建 `MachineConfig` 对象，以修改节点上的文件、systemd 单元和 SSH 公钥，以及恢复发生配置漂移的节点。

`MachineConfig` 支持 Ignition 规范版本 3.4。所有 `MachineConfig` 对象都必须符合此版本的要求。

在某些情况下，节点上的配置可能与当前通过 `MachineConfig` 应用的配置不完全匹配。这种情况称为配置漂移。机器配置守护进程会定期验证节点的配置是否发生漂移。如果检测到漂移，节点会被标记为 `Degraded`，并保持该状态，直到管理员恢复预期配置。

以下示例演示了如何使用 `MachineConfig` 对象管理节点配置。

目录

配置 Chrony 时间服务

禁用 Chrony 时间服务

为 `boot` 用户配置 SSH 公钥

从配置漂移中恢复

配置 **Chrony** 时间服务

要配置 Chrony 时间同步服务（`chronyd`）并指定 NTP 服务器及相关设置，可以通过 `MachineConfig` 对象更新目标节点上的 `chrony.conf` 文件。

1. 首先，创建一个包含所需 Chrony 配置的临时文件：

```
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. 然后，对文件内容进行 base64 编码：

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

3. 创建名为 `99-worker-chrony` 的 `MachineConfig` 对象。在 `.spec.config.storage.files[0].contents.source` 字段中，以 `data:text/plain;base64,<encoded-content>` 格式插入经过 base64 编码的字符串：

```
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=='
```

此配置会创建一个 `MachineConfig` 对象，将自定义的 `chrony.conf` 文件应用到与 `worker` 机器配置池关联的节点。该文件将写入每个节点上的 `/etc/chrony.conf`，文件权限设置为 `0o644`。
要使更改生效，请为 `/etc/chrony.conf` 配置一个会重启服务的节点中断策略：

```
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
```

对 `chronyd` 使用 `Restart`；该服务不支持 reload 操作。在应用 `MachineConfig` 之前，确认该策略显示在 `status.nodeDisruptionPolicyStatus` 中。

禁用 Chrony 时间服务

要禁用具有特定角色的节点上的 Chrony 时间同步服务，可以创建一个 `MachineConfig` 对象，覆盖 systemd 单元定义并禁用该服务。

配置示例：

```
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
```

此配置会将自定义版本的 `chronyd.service` 单元文件推送到 `worker` 机器配置池中的节点。该服务会被显式禁用。应用配置并重启节点后，Chrony 服务将不再自动启动。

为 `boot` 用户配置 SSH 公钥

机器配置系统允许你为受管节点上的 `boot` 用户配置 SSH 公钥。不支持为其他用户帐户配置。请注意，机器配置不会自动创建用户或组——在应用配置之前，你必须确保节点上存在 `boot` 用户和组。

配置示例：

```
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>
```

此 `MachineConfig` 会将指定的 SSH 密钥安装到 `worker` 机器配置池中节点的 `/home/boot/.ssh/authorized_keys` 文件中。

从配置漂移中恢复

如果节点的配置偏离其分配的 `MachineConfig`，则会将其标记为 `Degraded`。在此状态下，节点仍可正常运行，但在问题解决之前无法接收进一步的配置更新。

有两种方法可以将节点从此降级状态恢复：

1. 手动还原配置
   你可以手动调整节点上的文件和权限，使其与分配的 `MachineConfig` 中指定的内容完全一致。系统将检测到更正并清除降级状态。
2. 强制重新应用配置
   在受影响的节点上创建一个空文件 `/run/machine-config-daemon-force`。机器配置守护

进程将检测到此触发器，重新应用当前的 `MachineConfig` ，删除触发器文件，然后重启节点。重启后，节点将从 `Degraded` 转换回 `Done` 。

管理防火墙端口

每个节点的主机防火墙都已预配置平台所需的端口。您无需管理这些端口，也不应直接修改平台预配置的端口列表。

当工作负载需要开放额外的主机端口时（例如，某个组件直接在节点上侦听），您可以通过使用 `MachineConfig` 对象安装一个小型 systemd 单元来开放端口。该单元会在运行时通过 `firewall-cmd --add-port` 开放端口，同时保留平台预配置的端口不变。

目录

为什么使用运行时单元，而不是重新加载 Firewalld

开放其他端口

更改或移除端口

注意事项

为什么使用运行时单元，而不是重新加载 **Firewalld**

重新加载或重启 firewalld（`firewall-cmd --reload`、`systemctl restart firewalld`）会从头重建主机防火墙。在集群节点上，这可能会丢弃其他组件直接配置的数据包过滤规则，例如 Istio 的流量重定向规则、`kube-proxy` 或 CNI 规则，从而导致间歇性连接中断。为避免这种情况，此操作步骤绝不会重新加载或重启 firewalld。

相反，一个 oneshot systemd 单元会通过 `firewall-cmd --add-port` 将端口添加到正在运行的 firewalld 中。此操作只会添加规则，不会清除任何内容。该单元的启动顺序位于 firewalld 之后，并声明为 `PartOf=firewalld.service`，因此如果 firewalld 重启，该单元会再次运行并重新添加端口（运行时 `--add-port` 本身不会在 firewalld 重启后保留）。

开放其他端口

以下示例会开放 TCP 端口 `9141`（ZooKeeper）和 `9308`（cpaas-kafka），这两个端口是日志堆栈常用的端口。请将 `--add-port` 列表更改为工作负载所需的端口。

1. 创建一个 `MachineConfig`，用于安装一个在运行时开放端口的 oneshot 单元：

```
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. 配置节点中断策略，使此单元发生更改时重启该单元（重新运行 `firewall-cmd`），而不是重启节点。将 `open-app-ports.service` 条目添加到集群范围的 `MachineConfiguration` 单例 `cluster` 的 `nodeDisruptionPolicy.units` 列表中，并在应用 `MachineConfig` 之前确认该条目已反映在 `status.nodeDisruptionPolicyStatus` 中：

```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
```

应用这两个对象后，该单元会在每个目标节点上开放端口，不会重新加载 firewalld，也不会重启节点；平台预配置的端口和其他组件的规则均保持不变。

> **WARNING**
>
> `cluster` 对象是单例，可能已包含其他单元或文件策略。请将 `open-app-ports.service` 添加到现有的 `nodeDisruptionPolicy.units` 列表中，而不是替换整个策略。此处未列出的受管单元将回退到单元的默认操作；单元发生更改时，该操作会重启节点。

更改或移除端口

- 要添加端口，请扩展单元中的 `--add-port` 列表，然后重新应用 `MachineConfig`。节点中断策略会重启该单元，并在运行时添加新端口。
- 要移除端口，仅从 `--add-port` 列表中删除端口还不够：`firewall-cmd --add-port` 只会添加端口，因此运行时已开放的端口会一直保持开放，直到 firewalld 重启。请从单元中移除该端口，然后在受影响的节点上重启 firewalld，或重启这些节点，以关闭该端口。

注意事项

- 端口会添加到 firewalld 的默认区域。如果相关接口绑定到非默认区域，请将 `--zone=<zone>` 添加到 `firewall-cmd` 命令中。
- `firewall-cmd --add-port` 具有幂等性：重新添加已开放的端口会返回警告，但仍会成功，因此可以安全地重新运行 oneshot 单元。
- 该单元要求 firewalld 正在运行。在不运行主机 firewalld 的节点上，该单元不会启动，也不会开放任何端口。
- 运行时 `--add-port` 不会写入 firewalld 的永久配置。如果 firewalld 在单元外部重新加载（`firewall-cmd --reload`），这些端口会被关闭，并且只有在 firewalld 或节点下次重启时才会重新添加。

调整节点内核参数（sysctl）

某些内核参数是按节点设置的，而不是按工作负载设置的，例如 ARP/邻居表垃圾回收阈值。这些是节点级（非命名空间）设置，因此无法通过 Pod 的 `securityContext.sysctls` 或安全上下文约束进行设置；必须在主机上应用。

可以通过使用 `MachineConfig` 对象写入 `sysctl` drop-in 文件来设置这些参数。该设置会作为单独的文件写入 `/etc/sysctl.d/`，因此不会更改节点的其他内核设置。将 `MachineConfig` 与节点中断策略配合使用，以便在不重启节点的情况下应用该值。

目录

适用场景

设置内核参数

验证更改

还原更改

适用场景

一种常见情况是，大型集群中的节点记录了 `neighbour table overflow` 错误：ARP/邻居表无法容纳如此多的对等节点，需要提高其垃圾回收阈值。以下示例提高了这些阈值；同样的操作步骤也适用于任何节点级 `sysctl` 键。

设置内核参数

以下示例提高了工作节点上的 ARP/邻居表垃圾回收阈值。

1. 创建 `sysctl` drop-in 文件。每一行都是一个 `key = value` 键值对：

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

2. 对内容进行 Base64 编码：

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

3. 创建一个 `MachineConfig` 对象，将文件写入 `/etc/sysctl.d/`：

```
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=='
```

`role` 标签用于选择应用该文件的节点。若要仅针对部分节点，请创建自定义 `MachineConfigPool`，并将标签设置为该池的角色。

4. 配置节点中断策略，以便在文件更改时重新应用 `sysctl` 设置。由于文件更改的默认操作是 `None`，否则该文件只会写入磁盘，而不会生效。将以下内容添加到集群范围的 `MachineConfiguration` 单例 `cluster` 中，并在应用 `MachineConfig` 之前确认其已反映在 `status.nodeDisruptionPolicyStatus` 中：

```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
```

重启 `systemd-sysctl.service` 会重新应用 `/etc/sysctl.d/` 下的每个文件。新值会生效，且节点不会重启。

> **WARNING**
>
> 先为该文件路径设置节点中断策略，并等待其显示在 `status.nodeDisruptionPolicyStatus` 中。如果在策略生效前应用 `MachineConfig`，文件会写入磁盘，但不会应用该值。如需查看完整的中断操作列表，请参阅节点中断策略。

验证更改

在目标节点上读取参数：

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

输出应显示你设置的值：

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

还原更改

删除 `MachineConfig` 对象以移除 drop-in 文件。已应用的值会继续对运行中的内核生效，直到节点下次重启；重启后，节点将应用默认值。

设置节点时区

集群中的所有节点都应使用统一的时区，以便平台日志、审计记录和监控时间线与 operator 的本地时间保持一致。在 Alauda OS 节点上，应通过 `MachineConfig` 对象设置时区，切勿登录节点逐个修改。

时区不能作为文件下发：`/etc/localtime` 是符号链接，而机器配置管理的是文件内容、systemd 单元和 SSH 公钥，而非符号链接。为此，每个节点都会运行一个 oneshot systemd 单元来执行 `timedatectl set-timezone`。

目录

首先配置节点干扰策略

应用时区

验证

相关说明

首先配置节点干扰策略

如果没有节点干扰策略，更改 systemd 单元会回退到默认操作——重启节点——因此应用下面的 `MachineConfig` 会导致池中的每个节点都经过清空和重启。请先配置策略并确认其已生效，然后再应用 `MachineConfig`。

1. 将 `set-timezone.service` 添加到集群范围的 `MachineConfiguration` 单例 `cluster` 的 `nodeDisruptionPolicy.units` 列表中：

```yaml
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` 是一个单例，其中可能已包含适用于其他单元或文件的策略，例如配置 Chrony 时间服务中的 `/etc/chrony.conf` 条目。 请追加到 **`units`** 列表，不要替换现有策略。 未列出的任何单元都会回退到默认的重启操作。

2. 继续之前，请确认策略已激活：

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

应用时区

每个角色创建一个 `MachineConfig` ——这两个对象的区别仅在于 `metadata.name` 和角色标签。将 `Asia/Shanghai` 替换为统一采用的 tz 数据库名称；请勿直接使用示例值。

```
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
```

验证

```
kubectl get machineconfigpool
```

等待每个池报告 `UPDATED=True` 和 `DEGRADED=False`，并确认没有节点经历意外的排空或重启。然后抽查节点：

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

平台日志、审计记录和监控时间线现在应与配置的时区一致。

相关说明

- 安装时的时区是主机先决条件 — 请参阅裸金属页面中的 主机时间同步要求。 本页面用于在运行中的集群上规范化或更改时区。
- 时间同步是单独的事项，通过 配置 Chrony 时间服务进行配置。

节点中断策略

目录

了解 Machine Configuration 中的节点中断

节点中断策略可控制的内容

示例：默认节点中断策略

示例：自定义文件行为

示例：将策略应用于特定文件

了解 **Machine Configuration** 中的节点中断

默认情况下，当您对 `systemd` 对象的 `MachineConfig` units 部分进行某些更改时，Machine Configuration Operator 会排空并重启与该 `MachineConfig` 关联的节点。不过，对常规文件条目的更改通常不会导致重启，因此可能导致配置未按预期生效。为解决此问题，您可以定义节点中断策略，指定哪些类型的更改应触发节点重启或其他中断操作。这些策略在集群范围的 `MachineConfiguration` 单例 `cluster` 中配置。有关如何配置节点中断策略，请参见以下示例。

定义策略后，Machine Configuration Operator 会验证策略，以检测格式无效等问题。然后，它会将策略填入 `MachineConfiguration` 对象的 `status.nodeDisruptionPolicyStatus` 字段中。这些用户定义的策略会覆盖集群的默认中断设置。

Machine Configuration Operator 首次启动时会创建此资源，并采用如下所示的默认策略。

WARNING

由于文件更改的默认操作为 `None`，因此，仅写入或更新文件的 `MachineConfig` 对象会将文件应用到磁盘，但不会重启受影响的服务，因此更改可能不会生效。当文件更改需要重新加载或重启服务（例如 kubelet 或 firewalld 配置）时，请为该文件路径定义节点中断策略，并在应用 `MachineConfig` 之前确认该策略已显示在 `status.nodeDisruptionPolicyStatus` 中。

节点中断策略可控制的内容

节点中断策略允许您定义对以下配置区域进行更改时的处理方式：

- 文件：您可以定义文件更改（不包括对根目录的更改）的处理方式。默认情况下，文件更改不会触发任何中断。您可以使用 `spec.defaultNodeDisruptionPolicySpecAction.files` 字段修改此行为。
- **Systemd Units**：您可以创建或修改 `systemd` 服务，包括启用、禁用或更改其状态。默认情况下，对 `systemd` units 的更改会触发节点排空和重启。
- **SSH** 公钥：您可以为 `boot` 用户添加或更新 SSH 密钥。默认情况下，这些更改会立即应用，且不会触发重启或排空。

每项更改都会根据节点中断策略进行评估，该策略可触发以下一项或多项操作：

- `Reboot`：排空并重启节点。
- `None`：不触发中断；静默应用更改。
- `Drain`：排空节点，但不重启。
- `Reload`：重新加载指定的 `systemd` 服务，但不重启该服务。
- `Restart`：重启指定的 `systemd` 服务。
- `DaemonReload`：重新加载所有 `systemd` unit 配置。

示例：默认节点中断策略

以下是安装后 `cluster` `MachineConfiguration` 资源的默认配置：

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

此配置表示：

- 文件更改不会触发任何操作。
- Systemd unit 更改会导致节点排空并重启。
- SSH 密钥更改会直接应用，不会造成中断。

示例：自定义文件行为

您可以更改文件更改的默认操作，使其触发重启：

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

使用此配置时，任何受管理文件的更改都会导致 Machine Configuration Operator 排空并重启受影响的节点。

示例：将策略应用于特定文件

您也可以为特定文件路径定义中断操作。在以下示例中，对 `/usr/local/bin/myapp.sh` 的更改不会触发节点重启，而是重新加载 systemd 配置并重启相关服务：

```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
```

在这种情况下，更新 `/usr/local/bin/myapp.sh` 时，Machine Configuration Operator 将重新加载所有 `systemd` units 并重启 `myapp.service`，而不会排空或重启节点。