**Machine Config Operator (MCO)**   
  
  
In OpenShift, changes to the underlying OS of cluster nodes (which run RHCOS) are done declaratively using MachineConfig objects. These are managed by the Machine Config Operator (MCO) and are used for:

* Adding files
* Changing kernel parameters
* Installing system packages (via rpm-ostree)
* Configuring SSH keys
* Customizing systemd services

**What is MachineConfig?**   
A MachineConfig defines low-level host configurations (like /etc, /var, systemd units, kernel args) for OpenShift nodes. It’s only used on RHCOS nodes, and changes trigger a reboot of affected nodes.   
  
**Common Use Cases for MachineConfig**   
**Example**

* Add SSH key to a node To enable controlled admin access
* Configure a custom file Add /etc/myconfig.conf
* Set sysctl values Tune performance/kernel behavior
* Install additional packages e.g., tcpdump via rpm-ostree install
* Modify systemd unit files Change kubelet config or custom units

**Example 1: Add SSH Public Key to All Worker Nodes**

apiVersion: [machineconfiguration.openshift.io/v1](https://machineconfiguration.openshift.io/v1)   
kind: MachineConfig   
metadata:   
  name: worker-ssh-key   
  labels:   
    [machineconfiguration.openshift.io/role](https://machineconfiguration.openshift.io/role): worker   
spec:   
  config:   
    ignition:   
      version: 3.2.0   
    passwd:   
      users:   
        - name: core   
          sshAuthorizedKeys:   
            - ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQ... your\_key\_here  
  
This adds your SSH key to the default core user on all worker nodes.   
  
**Example 2: Create a File on All Nodes**

apiVersion: [machineconfiguration.openshift.io/v1](https://machineconfiguration.openshift.io/v1)   
kind: MachineConfig   
metadata:   
  name: add-custom-config   
  labels:   
    [machineconfiguration.openshift.io/role](https://machineconfiguration.openshift.io/role): worker   
spec:   
  config:   
    ignition:   
      version: 3.2.0   
    storage:   
      files:   
        - path: /etc/myconfig.conf   
          mode: 0644   
          contents:   
            source: data:text/plain;charset=utf-8;base64,bXkgb3BlbnNoaWZ0IGNvbmZpZw==  
  
This creates a file /etc/myconfig.conf with the content my openshift config   
(Base64-encoded content)   
  
**How It Works**

* You apply a MachineConfig using oc apply -f ....
* The Machine Config Operator (MCO) detects the change.
* MCO updates the MachineConfigPool (e.g., worker, master).

**Each affected node:**

* Cordons and drains itself
* Applies the config (via Ignition-style config)
* Reboots
* Rejoins the cluster

**View Status**   
oc get machineconfig   
oc get machineconfigpool

**Watch status with:**   
oc describe mcp worker

**Notes**   
MachineConfigs are powerful but risky — one misconfigured file can break all nodes in the pool.   
Always test changes on a single node pool first if possible.   
You can roll back a config by removing or editing the MachineConfig   
  
  
**MachineConfigPool (MCP)**

In OpenShift, MachineConfigPool (MCP) is a controller object that manages a set of nodes with similar configuration requirements (e.g., worker, master). MCPs work with the Machine Config Operator (MCO) to apply MachineConfig changes like OS updates, SSH key additions, kernel parameter changes, etc.   
  
**Concept Details**   
What is MCP? A group of nodes sharing a common configuration profile   
Who manages MCP? Machine Config Operator   
When is it created? Automatically created for default node roles (worker, master)   
When should you create one manually? When you want to apply specific MachineConfig to a custom node group (e.g., infra, edge, gpu)  
  
**Default MCPs**   
OpenShift clusters come with two built-in MCPs:   
master → manages control plane nodes   
worker → manages compute (app) nodes   
These are sufficient for most workloads.   
  
**Steps to Create a Custom MachineConfigPool**

Let’s say you want to create an MCP for nodes with the label [node-role.kubernetes.io/infra](https://web.telegram.org/a/node-role.kubernetes.io/infra)

**1.Label Target Nodes**   
Label the nodes you want to assign to this new pool:   
oc label node <node-name> [node-role.kubernetes.io/infra=](https://web.telegram.org/a/node-role.kubernetes.io/infra=)   
  
**2. Create the New MachineConfigPool YAML**

apiVersion: [machineconfiguration.openshift.io/v1](https://machineconfiguration.openshift.io/v1)   
kind: MachineConfigPool   
metadata:   
  name: infra   
spec:   
  machineConfigSelector:   
    matchExpressions:   
      - key: [machineconfiguration.openshift.io/role](https://machineconfiguration.openshift.io/role)   
        operator: In   
        values:   
          - infra   
  nodeSelector:   
    matchLabels:   
      [node-role.kubernetes.io/infra](https://node-role.kubernetes.io/infra): ""  
  
**Apply it:**   
oc apply -f mcp-infra.yaml   
  
**3. Create a Matching MachineConfig**   
  
apiVersion: [machineconfiguration.openshift.io/v1](https://machineconfiguration.openshift.io/v1)   
kind: MachineConfig   
metadata:   
  name: custom-infra-config   
  labels:   
    [machineconfiguration.openshift.io/role](https://machineconfiguration.openshift.io/role): infra   
spec:   
  config:   
    ignition:   
      version: 3.2.0   
    storage:   
      files:   
        - path: /etc/motd   
          mode: 0644   
          contents:   
            source: data:text/plain;charset=utf-8;base64,Q3VzdG9tIEluZnJhIG5vZGUgTVRPREZJTEU=  
  
The label on the MachineConfig ([machineconfiguration.openshift.io/role](https://web.telegram.org/a/machineconfiguration.openshift.io/role): infra) must match the MCP name (infra)   
  
**Apply it:**   
oc apply -f mc-infra.yaml   
  
**How the Flow Works**:

* MCO sees the new MCP (infra)
* It watches for MachineConfigs labeled for that role
* Nodes with [node-role.kubernetes.io/infra](https://web.telegram.org/a/node-role.kubernetes.io/infra) are cordoned and drained
* Config is applied (via Ignition)
* Nodes reboot and return to Ready state

**Check Status**   
oc get mcp

**Look at the UPDATING, UPDATED, DEGRADED columns**   
oc describe mcp infra   
  
**Things to Keep in Mind**:

* Every new MCP adds complexity. Use only when necessary
* Changes cause node reboots. Plan updates during maintenance windows
* Don't modify master MCP unless you really know what you're doing

**When MCP Updates a Node:**   
If a node (e.g., with role infra) is updated via MachineConfig:   
The Machine Config Operator (MCO) performs a cordon and drain operation.

**This means:**   
**Cordon**: No new pods can be scheduled on the node.   
**Drain**: Existing pods are evicted (removed), and the node is rebooted.   
  
**So Where Do the Running Pods Go?**   
**Type of Pod What --- Happens During Drain**ReplicaSet/Deployment/StatefulSet ========== Pods Re-scheduled on other eligible nodes (with matching taints/tolerations and resource availability)   
DaemonSet ========== Pods Killed and re-created after reboot on the same node (not moved)   
Static Pods (host-level, kubelet-managed) ========= Not drained; they remain   
Pods without Controller (bare pods) =========== Evicted and deleted permanently — NOT rescheduled   
  
**Example:**   
**Suppose you have this setup:**3 worker nodes (2 default worker, 1 labeled infra)   
An app is running with replicas = 3

**If the infra node is drained by MCP:**   
That node is cordoned and drained.

**The pod(s) on the infra node are:**   
Evicted   
Recreated by the controller (e.g., Deployment) on one of the remaining worker nodes, if possible.

**If there’s not enough capacity in the cluster, pods might:**   
Stay in Pending state (until a node is available)   
Fail to reschedule (if anti-affinity or taints/tolerations prevent it)

**What You Should Ensure Before MCP Reboots a Node**

* You have enough schedulable nodes to take over the evicted pods.
* Your pods are managed by a controller (Deployment, StatefulSet, etc.)

**Machine Config Operator (MCO) acts as a controller + CRDs + application logic, using your exact example:**   
**Operator = Controller + CRDs + App Logic**

**The Machine Config Operator (MCO) is a perfect example of a Kubernetes Operator:**

**Component MCO’s Role**   
Controller Watches objects like MachineConfig, MachineConfigPool, Node   
CRDs Defines custom resources: MachineConfig, MachineConfigPool, etc.   
App Logic Applies OS-level changes, reboots nodes, syncs configs across nodes   
  
**1. Controller (Watcher + Reconciler)**   
**The controller inside MCO continuously watches:**   
Changes to MachineConfig objects (e.g., adding a new file, SSH key, kernel args)

Node labels and annotations (to see which nodes belong to which MachineConfigPool)   
MachineConfigPool health status   
It compares the desired state (in the MachineConfig) with the actual state of nodes and takes action to make them match.   
  
**2. CRDs Used by MCO**   
CRD What it Represents   
MachineConfig Low-level OS config (Ignition format) applied to nodes   
MachineConfigPool A group of nodes that should receive a set of configs   
ControllerConfig (indirect) Internal config used during bootstrapping   
These are all Custom Resource Definitions (CRDs) — they extend the Kubernetes API.   
  
**3. App Logic in the Operator**   
**This is where the intelligence of MCO comes in:**

Reboot Workflow (on MachineConfig update):   
Detects that a new MachineConfig applies to a node.   
Cordons the node (no new pods scheduled).   
Drains the node (evicts existing pods).   
Applies the Ignition-based config to the node:   
Add files (e.g. /etc/motd)   
Change kernel args   
Install packages using rpm-ostree   
Reboots the node   
Waits for it to come back healthy   
Moves to the next node

This is all fully automated — you just define the desired state via YAML.

**Ignition :**

Ignition is the low-level system provisioning tool that handles the first boot configuration of a node — before the OS even fully starts.   
  
**How it works**

Purpose: Ignition is responsible for setting up the node’s OS exactly the way the cluster needs — things like:

* Writing configuration files (/etc/...)
* Adding SSH keys
* Configuring networking
* Setting up systemd units
* Partitioning and formatting disks

**When it runs:**   
It runs only once — during the first boot of a CoreOS machine.   
After the first boot, it doesn’t run again unless the node is reprovisioned.   
  
**Connection with MachineConfig**

**In OpenShift:**   
You create or modify a MachineConfig (e.g., add a custom CA, enable a kernel parameter).

**The Machine Config Operator (MCO):**   
Detects the change   
Regenerates an Ignition config for the affected MachineConfigPool   
Triggers the nodes in that pool to reboot   
  
**When the node reboots:**   
Ignition reads the generated config and applies the changes   
The OS comes up with the new configuration already in place   
  
**Why it matters**   
Without Ignition, CoreOS nodes would need manual setup. Ignition makes it declarative and automated, so the cluster’s desired state (MachineConfig) is applied consistently across all nodes.   
  
**How it fits into your example:**   
During fresh install   
When you first install OpenShift, the Machine Config Operator (MCO) generates Ignition configuration files for each MachineConfigPool (e.g., master, worker).

**These Ignition files contain:**   
OS-level settings (systemd units, SSH keys, time sync configs, kernel args, etc.)   
Pull secrets   
kubelet configuration

When the bare-metal/VM boots with RHCOS, the Ignition service runs once, reads that config from the API or a URL, and applies it before the OS fully starts.

In your mc.yaml example (post-install changes)   
You apply a new MachineConfig.   
MCO compares the desired config to what's running on each worker node.   
If the change requires an OS-level modification that needs a reboot (like changing crio.conf or adding systemd services), MCO regenerates a new Ignition-like config internally and rolls it out:   
Cordon → Drain → Write config → Reboot node → RHCOS reads & applies new config.   
When Ignition is actually invoked   
On first boot of a node (always)   
On re-provision/rebuild of a node (always)   
On MCO-triggered reboots for OS-level changes (the node effectively re-applies the MachineConfig like a mini-provision step).