Fix: Stale MachineConfig Annotation Deadlock in Agent-Based Installer

Status: TESTED AND VERIFIED

Successfully tested on 2026-01-08 with OpenShift 4.18.30. The fix prevents the stale MC annotation deadlock and allows the installation to complete successfully.

Affected Versions

Version	Affected	Tested
4.18.30	Yes	Fixed and verified
4.20.4	Yes	Patched release built with fix - testing in progress

The issue is present in the upstream `assisted-installer` code and affects all OpenShift versions using agent-based installation.

Build Details (4.20.4)

- Patched orchestrator digest: `sha256:23cbb0852d139d8ea99447bf0c1763423291ba16325c13d92b432c7f21ddd1d7`
- Label: `io.openshift.release.operator=true` (required!)
- Release: 194 images (complete)

Problem Description

During agent-based OpenShift installations, the bootstrap node—also known as the **rendezvous host** (control0)—can reboot before the other control plane nodes (control1, control2) have consistent MachineConfig annotations. This creates a deadlock situation:

> **Note:** In agent-based installs, one control plane node acts as the "rendezvous host" where assisted-service runs and orchestrates the installation. This node is identified by `isBootstrap=true` in the installer code.

The Issue (High Level)

1. During installation, the Machine Config Operator (MCO) creates "rendered" MachineConfigs (e.g., `rendered-master-abc123`)
2. Nodes receive annotations pointing to these rendered configs:
3. `machineconfiguration.openshift.io/currentConfig`
4. `machineconfiguration.openshift.io/desiredConfig`
5. If MCO regenerates a new rendered config while nodes are still processing, the old rendered config may be garbage collected
6. Nodes end up with annotations pointing to **non-existent** MachineConfigs
7. MCO marks these nodes as "Degraded" with reason: `missing MachineConfig rendered-master-xxx`

8. MCO refuses to update Degraded nodes, creating a permanent deadlock

Detailed Failure Scenario Timeline

Here's exactly how the race condition occurs:

```
TIME    EVENT
──────────────────────────────────────────────────────────────────────────────
T+0     Installation begins. All 3 control plane nodes boot from agent ISO.
        control0 is the "rendezvous host" running assisted-service.

T+2m    Bootkube starts on control0. Creates initial cluster resources.
        MCO starts and creates first rendered config: rendered-master-aaa111

T+3m    control1 and control2 join the cluster.
        MCO daemon sets their annotations:
          currentConfig: rendered-master-aaa111
          desiredConfig: rendered-master-aaa111
          state: Done

T+5m    A new MachineConfig is added (e.g., from cluster-config-operator).
        MCO detects the change and renders a NEW config: rendered-master-bbb222

        MCO updates MachineConfigPool "master" to reference rendered-master-bbb222

T+5m    MCO starts updating nodes one at a time:
   +1s  - Updates control1's desiredConfig → rendered-master-bbb222
   +2s  - control1 state changes to "Working"
   +30s - control1 applies config, reboots

T+6m    control1 comes back up with:
          currentConfig: rendered-master-bbb222
          desiredConfig: rendered-master-bbb222
          state: Done

        Meanwhile control2 still has:
          currentConfig: rendered-master-aaa111  ← OLD!
          desiredConfig: rendered-master-aaa111  ← OLD!
          state: Done

T+6m    MCO garbage collector runs:
   +5s  - Sees rendered-master-aaa111 is no longer referenced by MCP
   +6s  - DELETES rendered-master-aaa111 from cluster

        ⚠️  control2's annotations now point to NON-EXISTENT config!

T+7m    assisted-installer on control0 checks:
        ✓ 2 master nodes are kubernetes "Ready"? YES (control1, control2 are Ready)
        ✓ Bootkube complete? YES
        ✓ ETCD bootstrap complete? YES
        ✓ Controller ready? YES

        → assisted-installer decides it's safe to reboot control0

T+7m    control0 (rendezvous host) REBOOTS
   +1s
        etcd loses quorum temporarily (only control1 has valid etcd)

T+8m    control0 comes back as regular master (no longer bootstrap).
        Cluster has 3 control plane nodes again.

        BUT: control2 is now MCO "Degraded":
          "unable to find rendered-master-aaa111"

        MCO refuses to update Degraded nodes.
        control2 can NEVER get the new config.

T+∞     DEADLOCK:
        - control2 stuck pointing to deleted MachineConfig
        - MCO won't update it because it's Degraded
        - Cluster operators waiting for MCO to report healthy
        - Installation hangs forever or times out
```

Why Does MCO Delete the Old Rendered Config?

MCO's garbage collection logic:

1. MCO keeps rendered MachineConfigs that are:
2. Referenced by a MachineConfigPool's `spec.configuration.name`
3. Referenced by any node's `currentConfig` or `desiredConfig` annotation
4. When MCO creates a new rendered config (bbb222), it updates the MCP to point to it
5. The garbage collector runs periodically and deletes any rendered config that:
6. Is NOT the current MCP target
7. Is NOT referenced by any node
8. **The race**: If a node hasn't been updated yet (still pointing to aaa111) but MCO has already updated ALL other nodes and the MCP, the garbage collector sees aaa111 as "orphaned" and deletes it
9. The slow node (control2) now has annotations pointing to a deleted object

Why Rendezvous Host Reboot Timing Matters

The existing code in `installer.go` waits for: - 2 master nodes to be kubernetes "Ready" (`waitForMinMasterNodes`) - Bootkube completion - ETCD bootstrap completion - Controller ready

However, **kubernetes "Ready" does not mean MCO-healthy**. A node can be Ready but have stale MC annotations, causing it to be MCO Degraded.

When the rendezvous host reboots while control1/control2 have stale annotations: - etcd loses quorum temporarily - The stale annotation deadlock persists - Cluster installation fails or hangs

Solution

Add a new check before the rendezvous host reboots that verifies all master nodes have MachineConfig annotations pointing to **existing** MachineConfig objects.

Changes Made

1. **New method `GetMachineConfig(name)`** in K8SClient interface to fetch MachineConfig objects
2. **Fixed `runtimeClient` initialization bug** - The original code only initialized `runtimeClient` when `configPath == ""`, but agent-based installs pass a kubeconfig path, leaving `runtimeClient` nil and causing a panic when `GetMachineConfig()` was called
3. **New function `waitForMCAnnotationsConsistent()`** that:
4. Lists all master nodes
5. For each node, reads `currentConfig` and `desiredConfig` annotations
6. Verifies each referenced MachineConfig actually exists in the cluster
7. Waits until all annotations are consistent
8. **Call the new function** in the bootstrap path, after `waitForWorkers()` and before `finalize()` (reboot)
9. **Added `HighAvailabilityMode` flag** - Required by OCP builds but missing in upstream

Files Modified

- `src/k8s_client/k8s_client.go` - Added `GetMachineConfig()` interface method, implementation, and **fixed runtimeClient initialization**
- `src/k8s_client/mock_k8s_client.go` - Added mock for testing
- `src/installer/installer.go` - Added `waitForMCAnnotationsConsistent()` and integrated into bootstrap flow
- `src/config/config.go` - Added `HighAvailabilityMode` field and flag

Diff

src/config/config.go

```
diff --git a/src/config/config.go b/src/config/config.go
index cc38179..a9a9bc5 100644
--- a/src/config/config.go
+++ b/src/config/config.go
@@ -41,6 +41,7 @@ type Config struct {
    EnableSkipMcoReboot         bool
    NotifyNumReboots            bool
    CoreosImage                  string
+   HighAvailabilityMode         string
 }

 func printHelpAndExit(err error) {
@@ -81,6 +82,7 @@ func (c *Config) ProcessArgs(args []string) {
    flagSet.BoolVar(&c.EnableSkipMcoReboot, "enable-skip-mco-reboot", false, "indicate
assisted installer to generate settings to match MCO requirements for skipping reboot
after firstboot")
    flagSet.BoolVar(&c.NotifyNumReboots, "notify-num-reboots", false, "indicate number
of reboots should be notified as event")
    flagSet.StringVar(&c.CoreosImage, "coreos-image", "", "CoreOS image to install to
the existing root")
+   flagSet.StringVar(&c.HighAvailabilityMode, "high-availability-mode", "", "high-
availability expectations, \"Full\" which represents the behavior in a \"normal\"
cluster. Use 'None' for single-node deployment. Leave this value as \"\" for workers
as we do not care about HA mode for workers.")

    var installerArgs string
    flagSet.StringVar(&installerArgs, "installer-args", "", "JSON array of additional
coreos-installer arguments")
```

src/k8s_client/k8s_client.go

```
diff --git a/src/k8s_client/k8s_client.go b/src/k8s_client/k8s_client.go
index 584fda6..5862ea7 100644
--- a/src/k8s_client/k8s_client.go
+++ b/src/k8s_client/k8s_client.go
@@ -38,7 +38,6 @@ import (
    certificatesClient "k8s.io/client-go/kubernetes/typed/certificates/v1"
    "k8s.io/client-go/tools/clientcmd"
    runtimeclient "sigs.k8s.io/controller-runtime/pkg/client"
-   runtimeconfig "sigs.k8s.io/controller-runtime/pkg/client/config"

    "github.com/openshift/assisted-installer/src/ops"
    "github.com/openshift/assisted-installer/src/utils"
@@ -89,6 +88,7 @@ type K8SClient interface {
    IsClusterCapabilityEnabled(configv1.ClusterVersionCapability) (bool, error)
    UntaintNode(name string) error
    PatchMachineConfigPoolPaused(pause bool, mcpName string) error
+   GetMachineConfig(name string) (*mcfgv1.MachineConfig, error)
 }

 type K8SClientBuilder func(configPath string, logger logrus.FieldLogger) (K8SClient,
error)
@@ -133,32 +133,31 @@ func NewK8SClient(configPath string, logger logrus.FieldLogger)
(K8SClient, erro
    if err != nil {
        return &k8sClient{}, errors.Wrap(err, "creating openshift config client")
    }
-   var runtimeClient runtimeclient.Client
-   if configPath == "" {
-       scheme := runtime.NewScheme()
-       err = clientgoscheme.AddToScheme(scheme)
-       if err != nil {
-           return &k8sClient{}, errors.Wrap(err, "failed to add scheme to")
-       }
+   // Always create runtime client with full scheme support
+   scheme := runtime.NewScheme()
+   err = clientgoscheme.AddToScheme(scheme)
+   if err != nil {
+       return &k8sClient{}, errors.Wrap(err, "failed to add scheme to")
+   }

-       err = metal3v1alpha1.AddToScheme(scheme)
-       if err != nil {
-           return &k8sClient{}, errors.Wrap(err, "failed to add BMH scheme")
-       }
-       err = machinev1beta1.AddToScheme(scheme)
-       if err != nil {
-           return &k8sClient{}, errors.Wrap(err, "failed to add Machine scheme")
-       }
+   err = metal3v1alpha1.AddToScheme(scheme)
+   if err != nil {
+       return &k8sClient{}, errors.Wrap(err, "failed to add BMH scheme")
+   }
+   err = machinev1beta1.AddToScheme(scheme)
+   if err != nil {
+       return &k8sClient{}, errors.Wrap(err, "failed to add Machine scheme")
+   }

-       err = mcfgv1.AddToScheme(scheme)
-       if err != nil {
-           return &k8sClient{}, errors.Wrap(err, "failed to add MCP scheme")
-       }
+   err = mcfgv1.AddToScheme(scheme)
+   if err != nil {
+       return &k8sClient{}, errors.Wrap(err, "failed to add MCP scheme")
+   }

-       runtimeClient, err = runtimeclient.New(runtimeconfig.GetConfigOrDie(),
runtimeclient.Options{Scheme: scheme})
-       if err != nil {
-           return &k8sClient{}, errors.Wrap(err, "failed to create runtime client")
-       }
+   // Use the config we already loaded (works with both in-cluster and kubeconfig)
+   runtimeClient, err := runtimeclient.New(config, runtimeclient.Options{Scheme:
scheme})
```

```
+	if err != nil {
+		return &k8sClient{}, errors.Wrap(err, "failed to create runtime client")
	}

	return &k8sClient{logger, client, ocClient, csvClient, runtimeClient, csrClient,
@@ -713,3 +712,12 @@ func (c *k8sClient) PatchMachineConfigPoolPaused(pause bool,
mcpName string) err
	c.log.Infof("Setting pause MCP %s to %t", mcpName, pause)
	return c.runtimeClient.Patch(context.TODO(), mcp,
runtimeclient.RawPatch(types.MergePatchType, pausePatch))
 }
+
+func (c *k8sClient) GetMachineConfig(name string) (*mcfgv1.MachineConfig, error) {
+	mc := &mcfgv1.MachineConfig{}
+	err := c.runtimeClient.Get(context.TODO(), types.NamespacedName{Name: name}, mc)
+	if err != nil {
+		return nil, err
+	}
+	return mc, nil
+}
```

src/installer/installer.go

```
diff --git a/src/installer/installer.go b/src/installer/installer.go
index 22fd20d..b428534 100644
--- a/src/installer/installer.go
+++ b/src/installer/installer.go
@@ -201,6 +201,18 @@ func (i *installer) InstallNode() error {
		if err = i.waitForWorkers(ctx); err != nil {
			return err
		}
+
+		// Wait for MachineConfig annotations on all master nodes to be consistent
+		// before rebooting the bootstrap. This prevents the stale annotation deadlock
+		// where nodes point to non-existent MachineConfigs.
+		kc, err := i.kcBuilder(KubeconfigPath, i.log)
+		if err != nil {
+			i.log.Error(err)
+			return err
+		}
+		if err = i.waitForMCAnnotationsConsistent(ctx, kc); err != nil {
+			return err
+		}
	}

	//upload host logs and report log status before reboot
@@ -899,6 +911,59 @@ func (i *installer) waitForNodes(ctx context.Context, minNodes
int, role string,
    }
 }

+const (
+   mcCurrentConfigAnnotation = "machineconfiguration.openshift.io/currentConfig"
+   mcDesiredConfigAnnotation = "machineconfiguration.openshift.io/desiredConfig"
+   mcStateAnnotation         = "machineconfiguration.openshift.io/state"
+)
+
+// waitForMCAnnotationsConsistent waits for all master nodes to have MachineConfig
annotations
+// that reference existing MachineConfig objects. This prevents the bootstrap from
rebooting
+// while nodes have stale annotations pointing to non-existent MachineConfigs.
+func (i *installer) waitForMCAnnotationsConsistent(ctx context.Context, kc
k8s_client.K8SClient) error {
+   i.log.Infof("Waiting for MachineConfig annotations to be consistent on all master
nodes")
+
+   return utils.WaitForPredicate(waitForeverTimeout, generalWaitInterval, func() bool
{
+       nodes, err := kc.ListNodesByRole("master")
+       if err != nil {
+           i.log.Warnf("Failed to list master nodes: %v", err)
+           return false
+       }
+
+       for _, node := range nodes.Items {
+           currentConfig := node.Annotations[mcCurrentConfigAnnotation]
+           desiredConfig := node.Annotations[mcDesiredConfigAnnotation]
+           state := node.Annotations[mcStateAnnotation]
+
+           // Skip if annotations are not set yet
+           if currentConfig == "" || desiredConfig == "" {
+               i.log.Infof("Node %s has no MC annotations yet, waiting...",
node.Name)
+               return false
+           }
+
+           // Check if currentConfig exists
+           if _, err := kc.GetMachineConfig(currentConfig); err != nil {
+               i.log.Warnf("Node %s has currentConfig %s which does not exist
(state=%s), waiting...",
+                   node.Name, currentConfig, state)
+               return false
+           }
+
+           // Check if desiredConfig exists
+           if _, err := kc.GetMachineConfig(desiredConfig); err != nil {
+               i.log.Warnf("Node %s has desiredConfig %s which does not exist
(state=%s), waiting...",
+                   node.Name, desiredConfig, state)
+               return false
+           }
+
+           i.log.Infof("Node %s MC annotations are consistent (current=%s,
desired=%s, state=%s)",
+               node.Name, currentConfig, desiredConfig, state)
+       }
+
+       i.log.Infof("All master nodes have consistent MachineConfig annotations")
+       return true
+   })
+}
+
 func (i *installer) getInventoryHostsMap(hostsMap
map[string]inventory_client.HostData) (map[string]inventory_client.HostData, error) {
    var err error
    if hostsMap == nil {
```

src/k8s_client/mock_k8s_client.go

```
diff --git a/src/k8s_client/mock_k8s_client.go b/src/k8s_client/mock_k8s_client.go
index b27a5b2..9534009 100644
--- a/src/k8s_client/mock_k8s_client.go
+++ b/src/k8s_client/mock_k8s_client.go
@@ -17,6 +17,7 @@ import (
    v1 "github.com/openshift/api/config/v1"
    v1beta1 "github.com/openshift/api/machine/v1beta1"
    ops "github.com/openshift/assisted-installer/src/ops"
+   mcfgv1 "github.com/openshift/machine-config-operator/pkg/apis/
machineconfiguration.openshift.io/v1"
    v1alpha10 "github.com/operator-framework/api/pkg/operators/v1alpha1"
    gomock "go.uber.org/mock/gomock"
    v10 "k8s.io/api/batch/v1"
@@ -551,6 +552,21 @@ func (mr *MockK8SClientMockRecorder)
PatchMachineConfigPoolPaused(pause, mcpName
    return mr.mock.ctrl.RecordCallWithMethodType(mr.mock,
"PatchMachineConfigPoolPaused", reflect.TypeOf((*MockK8SClient)
(nil).PatchMachineConfigPoolPaused), pause, mcpName)
 }

+// GetMachineConfig mocks base method.
+func (m *MockK8SClient) GetMachineConfig(name string) (*mcfgv1.MachineConfig, error)
{
+   m.ctrl.T.Helper()
+   ret := m.ctrl.Call(m, "GetMachineConfig", name)
+   ret0, _ := ret[0].(*mcfgv1.MachineConfig)
```

```
+	ret1, _ := ret[1].(error)
+	return ret0, ret1
+}
+
+// GetMachineConfig indicates an expected call of GetMachineConfig.
+func (mr *MockK8SClientMockRecorder) GetMachineConfig(name any) *gomock.Call {
+	mr.mock.ctrl.T.Helper()
+	return mr.mock.ctrl.RecordCallWithMethodType(mr.mock, "GetMachineConfig",
reflect.TypeOf((*MockK8SClient)(nil).GetMachineConfig), name)
+}
+
 // PatchNamespace mocks base method.
 func (m *MockK8SClient) PatchNamespace(namespace string, data []byte) error {
 	m.ctrl.T.Helper()
```

Deployment

To use this fix:

Step 1: Build the patched image

CRITICAL: You MUST include the `io.openshift.release.operator=true` label. Without this label, `oc adm release new` will silently drop the image from the release!

Use a unique tag like `orchestrator-patched` to avoid conflicts when re-mirroring overwrites stock images:

```
cd upstream/assisted-installer
docker build --platform linux/amd64 -f Dockerfile.assisted-installer . \
  -t registry.gw.lo/openshift/release:orchestrator-patched \
  --label "io.openshift.release.operator=true"
```

Step 2: Push to local registry

If your registry uses self-signed certificates, use skopeo:

```
# Save image to tar
docker save registry.gw.lo/openshift/release:orchestrator-patched -o /tmp/
orchestrator-patched.tar

# Copy to registry host and push with skopeo
scp /tmp/orchestrator-patched.tar root@registry.gw.lo:/tmp/
ssh root@registry.gw.lo "skopeo copy \
  docker-archive:/tmp/orchestrator-patched.tar \
  docker://registry.gw.lo/openshift/release:orchestrator-patched \
  --dest-tls-verify=false"
```

Step 3: Get the digest

The digest is required for the release - **never use tags** in release images:

```
DIGEST=$(ssh root@registry.gw.lo "skopeo inspect \
  docker://registry.gw.lo/openshift/release:orchestrator-patched \
  --tls-verify=false --format '{{.Digest}}'")
echo "Patched orchestrator digest: $DIGEST"
```

Step 4: Create patched release (run on registry host)

```
oc adm release new \
  --from-release=registry.gw.lo/openshift/release:4.20.4-x86_64 \
  --to-image=registry.gw.lo/openshift/release:4.20.4-x86_64 \
  --insecure=true \
  agent-installer-orchestrator=registry.gw.lo/openshift/release@${DIGEST}
```

Step 5: Verify the release

Ensure the patched release has all images (194 for 4.20.x):

```
oc adm release info registry.gw.lo/openshift/release:4.20.4-x86_64 --insecure 2>&1 |
head -20
```

Verify orchestrator uses digest not tag:

```
oc adm release info registry.gw.lo/openshift/release:4.20.4-x86_64 --insecure 2>&1 |
grep orchestrator
```

Should show: `agent-installer-orchestrator registry.gw.lo/openshift/release@sha256:...`

Step 6: Update openshift-install cache

```
ssh root@registry.gw.lo "rm -f /var/lib/openshift-cache/openshift-install-4.20.4"
ssh root@registry.gw.lo "oc adm release extract --command=openshift-install \
  --to=/var/lib/openshift-cache \
  registry.gw.lo/openshift/release:4.20.4-x86_64 --insecure=true"
ssh root@registry.gw.lo "ln -sf openshift-install-linux-4.20.4 /var/lib/openshift-
cache/openshift-install-4.20.4"
```

Important Notes

- **CRITICAL - Label required**: Images MUST have label `io.openshift.release.operator=true` or they are silently dropped from the release
- **Use unique tag**: Use a tag like `orchestrator-patched` to avoid conflicts when mirroring overwrites stock images
- **Always use digest**: Release images must reference components by `@sha256:...` digest, never by tag
- **Component name**: Agent-based installs use `agent-installer-orchestrator`, NOT `baremetal-installer` (which is for IPI installs)
- **Mirror warning**: Running `oc mirror` or `oc adm release mirror` can overwrite your patched release! Use a unique tag and re-apply the patch after mirroring
- Both are built from the same repo (`assisted-installer`) using different Dockerfiles
- The release image embeds the digest, so you must re-extract `openshift-install` after creating a new release

Source Code Repository

The patched code is maintained in a GitHub fork of the upstream repository:

Repository	Location
Upstream	https://github.com/openshift/assisted-installer
Fork (glennswest)	https://github.com/glennswest/assisted-installer
Branch	`master`
Patch Commit	`a6a0fc3` - "Fix stale MachineConfig annotation deadlock in agent-based installs"

Modified Files in Fork

```
upstream/assisted-installer/
├── src/
│   ├── config/
│   │   └── config.go                # Added HighAvailabilityMode flag
│   ├── installer/
│   │   └── installer.go             # Added waitForMCAnnotationsConsistent()
│   └── k8s_client/
│       ├── k8s_client.go            # Fixed runtimeClient init + GetMachineConfig()
│       └── mock_k8s_client.go       # Added mock for GetMachineConfig
```

Related Issues

- This fix addresses a timing issue where MCO annotations become stale before the rendezvous host reboots
- The existing 1-minute reboot delay (for OCPBUGS-5988 RBAC bug) does not prevent this issue
- The original `runtimeClient` initialization bug would cause a nil pointer panic when trying to use `GetMachineConfig()`