

# Implementation design: In-place updates

Let **Stefan Büringer** know if you need write access to this doc

Related docs: [Proposal](#), [Umbrella issue](#)

Related branches/PRs

- [PR: In-place hooks API](#)
- [Alex's prototype](#)
- [MD rollout planner prototype](#)
- [KCP e2e prototype](#)

Difficulty: **Major**, **Minor**, **Done**

## Later iterations (possibly CAPI v1.13+)

- Improve how to configure to which “objects” a RuntimeExtension applies
  - Goal: We should avoid unnecessary CanUpdateMachine/MachineSet calls (e.g. for the wrong infra provider)
  - Ideas: ExtensionConfig objectSelector (like FieldSelectorRequirement), Extend response of Discovery call (e.g. objectSelector (like FieldSelectorRequirement) or “infraProviderKind”)
- Hook ordering
  - Maybe implement alphabetic ordering for CAPI v1.12.0. But make it very clear that this is not an API and nobody should rely on this order
  - Note: It's important that hooks of different update extensions are called in the same order for CanUpdateMachine/CanUpdateMachineSet and UpdateMachine hooks

## Runtime Hooks API

### CanUpdateMachine/CanUpdateMachineSet

- Principles:
  - We assume that in-place is triggered like rollout, i.e. by changing the MD/KCP or by rotating templates
  - MD/KCP already have logic to figure out when a rollout is needed, this logic must be reused (rollout.after & rollout.before will always trigger full rollouts)
  - While MD/KCP make the rollout decision relatively coarse-grained, we need more details for the CanUpdateMachine call, e.g. the current/desired Machine/BootstrapConfig/InfraMachine, including defaulting & cleanup logic
- Assumptions:
  - For KCP we are comparing every single Machine object
  - For MD we are comparing every single MachineSet.
    - > We do not want to repeat the same call for all the Machines in a MachineSet

- > To make this more explicitly, we probably need CanUpdateMachine and CanUpdateMachineSet; for implementers, they can re-use the same logic and compare spec or spec.templates.spec depending on the case
    - > this leaves open the door to the idea of deleting bootstrap config once machines are up (for scale reason)
  - Research: Kind of changes that lead to rollouts:
    - KCP
      - KCP/Machine.spec.version
        - For in-place we should already apply other in-place changes (label, annotations, ...) to current/desired Machine/KubeadmConfig/InfraMachine that we send
          - Machine: everything apart from version
          - InfraMachine/KubeadmConfig: annotations
      - KCP/Machine KubeadmConfig (\*defaulting, dropOmitEmpty...)
        - For in-place we need current/desired KubeadmConfig:
          - KubeadmConfigs have to be cleaned up
          - KubeadmConfigs have to be defaulted (reuse as a check if object can be updated)
      - KCP/Machine infrastructureRef
        - For in-place we need current/desired InfraMachine
          - InfraMachines have to be defaulted (reuse as a check if object can be updated)
      - KCP.spec.rollout.after
        - Cannot be “rolled out” with in-place
      - KCP.spec.rollout.before.certificatesExpiryDays
        - Out of scope for in-place
    - MD
      - MD/MS.spec.version
        - For in-place we should already apply other in-place changes (label, annotations, ...) to current/desired MachineSet/BootstrapConfigTemplate/InfraMachineTemplate that we send
      - MD/MS.spec.bootstrap.{configRef,dataSecretName}
        - For in-place we need current/desired BootstrapConfigTemplate
      - MD/MS.spec.infrastructureRef
        - For in-place we need current/desired InfraMachineTemplate
      - MD/MS.spec.failureDomain
      - MD.spec.rollout.after
        - Cannot be “rolled out” with in-place
  - Request:
    - CanUpdateMachineRequest
      - current/desired Machine/InfraMachine/BootstrapConfig
        - (only spec, no status, bootstrap optional)
    - CanUpdateMachineSetRequest
      - current/desired Machineset/InfraMachineTemplate/BootstrapConfigTemplate
        - (only spec, no status, bootstrap optional)

- Response
  - CanUpdateMachineResponse / CanUpdateMachineSetResponse
    - RX should modify the incoming current objects to signal which changes it can make and send them back as patch like in GeneratePatchesResponse
- Call loop:
  - We will call CanUpdateMachine/CanUpdateMachineSet for all registered update extension
  - We will use the output of the previous call as input for the next call (and repeat)
  - Once all update extensions have been called KCP/MD will figure out if the Machine/MachineSet can be updated in-place by comparing the last response with desired
    - Safeguard: For the comparison CAPI should ignore fields like labels, annotations, timeouts (fields that are already propagated in place by CAPI).

## UpdateMachine

Assumptions:

- Runtime extension should treat UpdateMachine requests as desired state; it is up to them to compare with current state and do necessary actions.
  - > This is also a chance to “remediate drift”
  - Note: We will not send the list of changes to be applied (neither in the message nor via annotations).
- Request:
  - UpdateMachineRequest
    - desired Machine/InfraMachine/BootstrapConfig
      - (only spec, no status, bootstrap optional)
      - runtime.RawExtension for InfraMachine/BootstrapConfig (same as what we did in GeneratePatchesRequestItem)
- Response:
  - UpdateMachineResponse
    - CommonRetryResponse

## KCP + MD/MS controller

### How do we trigger the in-place Machine update:

Assumptions:

- We are going to overwrite Machine / InfraMachine / BootstrapConfig in-place. This seems more realistic than trying to rotate InfraMachine / BootstrapConfig as rotation would be very hard to handle for infra providers.

- Disclaimer: With some infra providers InfraMachines are immutable, this would have to be changed **if** they want to start supporting in-place updates of InfraMachines.

#### Challenges:


- Create/Patch vs SSA and fieldOwnership concerns (the following discussion is only for InfraMachine / BootstrapConfig as we already always create & update Machines with SSA):
  - Option 1: client.Create + client.Patch later
    - The problem here is that ownership tracking is not very precise, so the client.Patch might remove fields set by other controllers/providers as they are not part of our computed desired object, for example: KubeadmConfig discovery.bootstrapToken, ...
    - We could try to preserve existing fields, but this would make it impossible to unset fields. We could handle this specifically for KubeadmConfig but not generically for other types.
    - Ruled out => SSA is needed
  - Option 2: Always use client.Apply with the same fieldManager
    - This means we always have to Apply the full objects, otherwise we would unset fields.
    - This only works if BootstrapConfigTemplate / InfraMachineTemplate are still around
    - This would mean that we inadvertently always instantly update BootstrapConfig / InfraMachine if BootstrapConfigTemplate / InfraMachineTemplate are updated in-place. But we only want to update BootstrapConfig / InfraMachine for in-place updates in a controlled manner.
    - Ruled out => SSA with multiple fieldManagers is needed
  - Option 3 (preferred): Always client.Apply but with different fieldManagers (similar for MS)
    - Create InfraMachine / BootstrapConfig
      - Apply object **without** labels/annotations with fieldManager: `capi-kubeadmcontrolplane-2`
      - [opt] Maybe do Create with labels/annotations but directly drop ownership
      - Apply object labels/annotations with fieldManager: `capi-kubeadmcontrolplane` (as today)
    - syncMachines for InfraMachine / BootstrapConfig (for labels/annotations sync)
      - Apply object labels/annotations with fieldManager: `capi-kubeadmcontrolplane` (as today)
    - Trigger in-place update
      - Apply object **without** labels/annotations with fieldManager: `capi-kubeadmcontrolplane-2`
    - Upsides:
      - We can address the limitations of Option 1 & 2

- This can be implemented in a way that we eventually don't need any managedField migration anymore, and accordingly don't have to continue to store managedFields in the cache.
- Migration of managedFields of existing InfraMachines / BootstrapConfigs (from before CAPI v1.12)
  - Option 1: no cleanup
    - Leads to co-ownership of a lot of fields, won't be possible to remove any fields
    - Ruled out => Not acceptable to never be able to remove any fields that have been set with create
  - Option 2: migrate the Update entry of manager to Apply with capi-kubeadmcontrolplane-2
    - Allows removal of fields with the first in-place update
    - But because of that might also remove fields that we don't really want to remove
    - Ruled out => We should not take the risk of removing the wrong fields, Option 3 seems safer
  - Option 3 (preferred): delete the Update entry of manager
    - Orphans a lot of fields, but after the first in-place update it will be also possible to remove fields
    - Maybe that's okay because this limitation only exists for pre-existing objects
    - We chose this option for MS/Machines in the past and it worked out well
- Apply Order + When should the Machine controller start in-place update with all updated objects:
  - Order:
    - KCP/MS: Trigger in-place update
      - Add to Machine: `in-place-update-in-progress`
      - Add to InfraMachine/BootstrapConfig: `in-place-update-in-progress`
        - Update the spec in the same call
      - Add to Machine: `pending-hooks: UpdateMachine`
        - Update the spec in the same call
    - Machine: Wait for in-place update
      - Wait for Machine: `pending-hooks: UpdateMachine`
      - Wait for InfraMachine/BootstrapConfig: `in-place-update-in-progress`
    - Machine: Complete in-place update
      - Remove from Machine: `in-place-update-in-progress`
      - Remove from InfraMachine/BootstrapConfig: `in-place-update-in-progress`
      - Remove from Machine: `pending-hooks: UpdateMachine`
  - We have to make sure that the "applies" happen "atomically" so the Machine controller doesn't do a partial in-place update, e.g. if it didn't observe the latest BootstrapConfig / InfraMachine yet:

- Option 1: Use the “Patch and Wait” pattern for BootstrapConfig and InfraMachine before marking pending upgrade on the Machine
    - Ruled out => Works for MS as it is using the same cache as the Machine controller, but that doesn’t work for KCP
  - Option 2: Mark BootstrapConfig and InfraMachine as “pending” as well
    - Machine controller will only start the in-place update if all 3 objects are marked correctly
    - This works because once an in-place update is started on the Machine we won’t trigger another one on the same Machine before it is completed
- Existing InfraMachine / BootstrapConfig controllers will probably directly reconcile InfraMachine / BootstrapConfig after we applied changes
  - InfraMachine:
    - InfraMachine controller should ignore in-place changes and leave them to the RX
    - If the changes are not disruptive to other in-place updates going on in parallel it might be fine for the InfraMachine controller to apply the changes directly.
    - 3 kinds of fields on an InfraMachine:
      - entirely immutable fields
      - fields that can only be “reconciled” by the in-place extension
        - validation webhook should allow field updates
        - InfraMachine controller should not reconcile the field updates  
(InfraMachine controller has to accept changes made by the in-place update extension)
      - “status” fields written back to spec by the infra provider (e.g. ID)
  - BootstrapConfig
    - BootstrapConfig controller does not have to reconcile the BootstrapConfig as we don’t need a bootstrap data secret (e.g. cloud-init) anymore.
- Double-check where we need “Patch and wait”
- MD/MS specific:
  - Create target MS + move Machine to the target MS, Challenges:
    - update labels, ownerRef, ... (race condition free vs. old/new MS.spec.replicas)
    - **object names won’t match MS name if we move to a different MS**

## KCP controller

- **Rollout Logic:** (POC in <https://github.com/sbueringer/cluster-api/tree/pr-kcp-in-place>)
  - Figure out what do to next in upgradeControlplane to handle various circumstances
    - If we have to scale up, we’re going to create a new Machine
    - If we’re going to scale down, we’re trying to in-place update
      - If upToDateReplicas == spec.replicas => scale down (as we have enough upToDate replicas already)

- How do we trigger the in-place Machine update (  Implementation design: In-place updates )
- **Track update state / UpToDate condition handling:**
  - Should we actually hand over control of the UpToDate condition to the Machine controller and then back to KCP? (Maybe we want an additional condition as well: UpToDate true/false is not the same as Updating true/false)

## MD/MS controller

Challenges:

- Various current behaviors of the MD controller: RollingUpdate, OnDelete, sync on pause
- Race conditions (including multiple controllers reconciling objects in parallel & stale cache)
- Controller reentrancy
- We must ensure we respect MaxSurge and MaxUnavailable
- In-place can be disruptive or not. How does this impact Availability
  - => Decision: We assume in-place updates are disruptive because in-place updates can always fail and this leads to Machine deletion/remediation => this also means that if users want to do an in-place update without any Machine creations, they have to set maxUnavailable >= 1 (i.e. explicitly accept potential unavailability)
    - TBD => think about if we can/should apply the same logic for KCP

Principle: Objects should be managed only by one controller

- MD ctrl manages MS
  - MD enforces maxUnavailable, maxSurge
    - As a consequence it decides when to scale up newMS, when to scale down oldMS
  - When there is a decision to scale down, MD should check if this can be done via in-place vs delete/recreate. If in-place is possible:
    - Old MS will be informed to move machines to the newMS
- MS ctrl manages a subset of Machines
  - When scaling down, if required to move, old MS is responsible for moving a Machine to newMS
  - newMS will take over moved machine and complete the upgrade workflow

Assumptions:

- We are making the in-place decision at MS level.
  - It is a trade off that reduces complexity, has performances benefits (less RX call) and fits well in the MD ctrl responsibility and current implementation (MD manages MS)
  - Downsides of this assumption seems acceptable:
    - MS: In-place: yes can update
      - Some Machines => no cannot update => there was drift on the machine state or some state relevant for the in-place decision

that is managed outside CAPI and it is not consistent within a MS.

- in-place update might success or fail => if failure is reported, remediation will kick in
- A possible improvement, is that RX, when an in-place upgrade is starting, checks again feasibility, and if not it fails fast
- MS: In-place: no cannot update
  - Some Machines => yes can update => will go through a full replacement, slightly less efficient than possible
- In place is always considered as potentially disruptive
  - in place must respect maxUnavailable
  - if maxUnavailable is zero, a new machine must be created, then as soon as there is "buffer" for in-place, in-place update is done
- when in-place is possible, the system should try to in-place update as many machines as possible.
  - maxSurge is not fully used (it is used only for scale up by one if maxUnavailable =0)
  - Also, if there is a scale up in the middle of a rollout, creation of new machines must be limited taking into account machines that can be updated in-place.

## Machine controller

- How to know when to start in-place update
  - => Only start the in-place update if Machine/InfraMachine/BootstrapConfig are marked correctly
- How to track completion of the in-place update
  - Remove annotations from all objects
- How to track the progress of the in-place update
  - Phase: Updating
  - Condition: Updating (will be included to compute UpToDate)
    - Option 1: Reuse UpToDate condition
      - It's hard to share a condition between KCP/MS/Machine controller
      - It's hard to surface all states in a single condition:
        - UpToDate true/false
        - In-place update in progress true/false
    - Option 2 (preferred): Introduce an additional Updating condition
      - Machine controller owns Updating condition
      - KCP/MS controller own UpToDate condition
      - Updating condition is also used to compute UpToDate condition
      - Updating condition can be used by MHC to trigger remediation if an in-place update is taking too long (this would not work with the UpToDate condition)
- What do we do if the update fails?

- Retry & MHC takes over

No guarantees below this line, wildly brainstorming



## MD/MS controller

- Are maxSurge/maxUnavailable applicable? If yes, how? (in-place rollouts vs scale up/down)
  - Only in-place updates => maxSurge/maxUnavailable doesn't matter
    - We could add a new maxConcurrentInPlaceUpdates eventually (and use 1 for now)
  - Only regular rollout => maxSurge/maxUnavailable as of today
  - Mixed Mode => (probably mixed over old MS, not mixed over time)
- Where exactly to call CanUpdate + fallback to configured strategy
  - When we fallback, how do we guarantee we fallback for the same Machine for which we asked CanUpdate? (Can we really assume that the answer is the same for all Machines of a specific old/new MS pair?)
    - Note: MD controller might roll out multiple Machines at once (maxSurge, ..)
  - MD controller does not have an understanding of a "single Machine" update
  - What about the OnDelete strategy?
    - OnDelete does not make sense as the idea of something else deleting Machines is incompatible with in-place updating Machines
  - When we update in-place how do we make sure that the Machine controller will later reconcile towards the expected desired state (e.g. we have to ensure that the target doesn't change when KCP changes)
- What about pre-checks?
- How do we trigger the in-place Machine update (
  - ☰ Implementation design: In-place updates )
- Track update state / UpToDate condition handling:
  - Should we actually hand over control of the UpToDate condition to the Machine controller and then back to MS? (Maybe we want an additional condition as well: UpToDate true/false is not the same as Updating true/false)
- How do we handle if MD is changed again while previous update is still ongoing

## e2e testing: e2e test, in-place CAPD Runtime Extension

- TODO

## Ensure MHC works smoothly during in-place updates

- MHC should not accidentally interrupt in-place updates
- MHC should catch failed in-place updates even if the Machine does not become unhealthy (via nodeUnhealthyConditions? maybe via machineUnhealthyConditions?)

- KCP:
  - [remediation.go](#): tl;dr In-place update is blocking remediation except for the in-place updating Machine
    - We should prioritize in-place updating Machine for remediation

## Autoscaler

- autoscaler should not accidentally try to delete Machines that are going through in-place update

[Defer] More fancy MHC behavior