ALLIED ESM

HALO TECHNICAL GUIDE

HALO ITSM · PLATFORM OPERATIONS

Halo Releases and Upgrades in the Cloud

How Halo ships new versions to your AWS-hosted instance, and how to run a Development, Test and Production pipeline through an upgrade without breaking it.

PUBLISHED BY
Allied ESM Ltd

DATE
August 2026

CLASSIFICATION
Public

VERSION
1.1

HALO OFFICIAL PARTNER

HALO CERTIFIED PARTNER

CHAPTER ONE

Anatomy of a Halo version number

Every conversation about Halo upgrades starts with the version number, and most confusion starts there too. A Halo version such as `2.174.1` carries three separate pieces of information, and each one changes for a different reason.

DIGIT	MEANING	WHAT CAUSES IT TO CHANGE
2	**Application version**	Major overhauls to the interface or the back end, and landmark points in development. Rarely changes.
174	**Release version**	A new software release. This is the number that moves when Halo ships new functionality, and it can bring database schema changes with it.
1	**Patch version**	Bug fixes and back end improvements found during the current release period. No schema changes.

Upgrade or patch, and why it matters

Halo uses these two words precisely, and the difference determines how much attention you need to pay.

Upgrade

The middle number changes, for example `2.131.10` to `2.132.8`. May include database schema changes. Scheduled, announced, and worth planning around.

Patch

The last number changes, for example `2.132.8` to `2.132.9`. Fixes and improvements only, never schema. Applied as needed, and not announced individually.

Finding the version you are on

In the Halo agent interface, select the question mark icon in the top right and choose **About**. This shows the web application version, the database version, and your support details. The same menu offers **Show Release Details**, which opens the searchable release notes area covering both the web application and the mobile app.

> If you take one thing from this page: a moving middle digit is a change you should plan for. A moving last digit is maintenance you will usually never notice.

CHAPTER TWO

Stable or beta: choosing a release path

Halo offers two schedule paths. The path you sit on decides how often your version moves and how much notice you get. You cannot switch path yourself in the interface, so changing it is a conversation with Halo. In practice a tenancy moves as a unit, for reasons covered on the next two pages.

Stable

Quarterly upgrades, rolled out across the customer base over several weeks. Halo's tested and recommended version at any given time. The right default for production service desks.

Beta

Upgrades roughly every two weeks, rolled out over two days. Newest functionality, less time in the field. Rollouts can be delayed while the development team works on an upcoming stable.

The stable path timeline

WHEN	WHAT HAPPENS
4 weeks before	The version that will become the next stable is announced as the Release Candidate. Customers can opt into it early.
2 weeks before	Your upgrade schedule is communicated to you.
Upgrade window	Applied outside working hours for all regions, at a time agreed with you.
Between releases	Patches are applied at Halo's discretion as needed, and these can land during working hours.

Which path should you be on?

For most organisations running Halo as a live service desk, stable is the correct answer and beta is a false economy. Four weeks of Release Candidate notice plus two weeks of scheduling notice gives you a genuine window to prepare.

Mixing paths across a pipeline is the trap. Putting Development on beta while Test and Production stay on stable sounds like a way to see new functionality early, but the moment you try to push beta functionality down the pipeline it will error. If you want early sight of specific features, Halo's own recommendation is a **Sandbox instance**, which sits outside the pipeline and therefore cannot poison it.

A third label you will see

Release Candidate is not a third schedule path. It is the stable-in-waiting: a partially tested version, announced four weeks ahead, that you can opt into early to verify your configuration before it becomes the version everyone runs.

CHAPTER THREE

How an upgrade reaches your instance

Halo hosts its cloud software on Amazon Web Services. If you are a cloud customer there is nothing to download, no installer to run, no server to take offline and no database upgrade utility to execute. Halo performs the work. Your job is to know when it is coming and to have your environments ready for it.

Nothing to install

The application and database are upgraded in place by Halo. No IIS restart, no file replacement, no upgrader executable. That is the on-premises workflow, and it does not apply to you.

Email notification

Scheduled upgrades to an application or release version are notified by email with the date and time. Expect this closer to the date, because release duration varies with complexity.

Out of hours, by agreement

Halo's published service commitment is that updates take place outside business hours and are scheduled at the customer's convenience. Maintenance that affects service is carried out by agreement and scheduled in advance.

Patches arrive quietly

Patches are applied automatically as needed and are not announced individually, because they carry fixes and back end improvements rather than new features. Security patches may be applied urgently, with customers kept informed.

Reading the release notes properly

The release notes area is searchable, and searching a single word works better than a phrase because it matches against the whole string. One detail is easy to miss: any note without a version shown to the right of the blue box has **not been released yet**. Treat those as roadmap, not as available functionality, and never plan a delivery date around one.

> Because you cannot control when a patch lands, the practical protection is not scheduling. It is having a non-production instance where problems surface first, and a support relationship that can escalate quickly when something changes unexpectedly.

CHAPTER FOUR

The one rule that governs everything

Version alignment is mandatory

All instances in a configuration pipeline must be running the same version of Halo. Syncing between instances on different versions is not supported, and it is blocked by the platform. Until every instance is aligned, no configuration can move between them.

This single constraint is why an upgrade is not a background event for anyone running more than one instance. If your Development instance moves to a new version and Production has not, your pipeline stops. Work already built in Development sits there, unable to travel, until the versions match again. Nothing is lost, but nothing can ship either.

Why this changes how you plan

Halo's platform is also explicit that configuration changes must be pushed in strict chronological order. There is no supported way to skip a change or reorder the queue.

In order

Changes apply sequentially. Pushing out of order risks ID misalignment, cross-wiring of unrelated processes, and broken dependencies.

No cherry-picking

Selecting individual changes is technically possible but strongly discouraged. A skipped dependency causes silent misalignment that may not surface for months.

Teams are coupled

Once two teams' changes interleave in the change log they cannot be separated. Everyone pushes together, or nobody pushes.

The practical consequence

Combine the two rules and the conclusion is unavoidable. An upgrade window is a moment when your pipeline must be empty. If you are halfway through a body of work when versions diverge, you are not simply waiting, you are holding a queue of unpushed changes that grows more entangled with every day it sits there. The organisations that handle Halo upgrades well are the ones that treat the notification email as the start of a change freeze, not as background noise.

The good news is that alignment is the default. Halo upgrades all of a customer's instances in the same window unless asked to do otherwise, so divergence is normally something you choose rather than something that happens to you. The next page covers when choosing it is worth the cost.

CHAPTER FIVE

Running DEV, TEST and PROD through an upgrade

This is the sequence we use on customer engagements. It assumes a three instance pipeline and a stable path upgrade with the standard notice period.

1. **On the Release Candidate announcement, four weeks out**

   Read the release notes for anything touching your configured areas. Decide whether you want a non-production instance on the Release Candidate to verify early. Set a target date for clearing the pipeline.

2. **On the schedule confirmation, two weeks out**

   Confirm the window with Halo. By default every instance you have is upgraded in the same window, which is exactly what keeps the pipeline workable. You can ask Halo's infrastructure team to schedule them separately, typically landing within a two week spread, but understand the trade: from the moment the first instance moves until the last one lands, your versions are incompatible and nothing can be pushed. Halo's own position is that staggering is rarely necessary, because upgrades are code-on-code. Publish an internal change freeze date either way.

3. **Flush the pipeline before the freeze**

   Push everything from Development through Test and into Production. Do not leave partial work in flight. If a change set cannot be finished and pushed in time, it is better not to have started it.

4. **Freeze, then let the upgrade happen**

   Stop new configuration work in Development. Consider enabling **Disable editing of config tracked entities in production** so nothing enters Production outside the pipeline during the window.

5. **Validate in pipeline order once versions are aligned**

   Smoke-test in Test first, working through the processes the release notes touched. Create tickets, trigger workflows, run actions. Only then confirm behaviour in Production.

6. **Resume, and reconcile anything that bypassed the pipeline**

   Lift the freeze. If an emergency change was made directly in Production during the window, document it and reconcile it with the pipeline immediately rather than letting the instances drift.

Controls worth having in place first

From version 2.182 onwards you can define a formal pipeline so each instance only syncs from its designated source instance, in one direction. Set this up with **Set Source Instance** under Configuration, Instances. Combine it with the **Can push configuration changes to other instances** permission so that pushing to Production is a deliberate act by a named person.

> A note on Development instances. Development is the entry point to the pipeline, not a sandbox. Experimentation there fills the change tree with half-finished work that blocks syncing later. Prototype in a separate Sandbox instance outside the pipeline, then rebuild cleanly in Development.

CHAPTER SIX

Upgrade checklist

Before the window

- [ ] Release notes reviewed for changes affecting your configured processes and integrations
- [ ] Pipeline flushed, with no unpushed changes sitting in Development or Test
- [ ] Change freeze communicated to every team that builds configuration in Halo
- [ ] Instance sequencing agreed with Halo, and the window confirmed in writing

During the window

- [ ] No configuration changes made in any instance
- [ ] Production editing locked via **Disable editing of config tracked entities in production**
- [ ] A named owner on hand to raise a support ticket if the upgrade does not complete cleanly

After the window

- [ ] All instances confirmed on the same version via the **About** screen
- [ ] Test validated end to end before Production is trusted
- [ ] Integrations and inbound mailboxes confirmed working, as these fail quietly
- [ ] Freeze lifted and any emergency Production change reconciled with the pipeline

The entities that do not sync

Configuration change tracking does not cover everything. Around twenty entities are not tracked and therefore never travel between instances, so they must be configured manually in each one. Some, such as language packs, can at least be exported and imported individually. Build this into your upgrade and instance refresh planning, because these are the items that quietly differ between environments and cause a test to pass in one place and fail in another.

Screen Layout Profiles	Asset Templates	Call Scripts
Certificates	Language Packs	Cost Centres
Custom SQL Quantities	Software Products	Tax Rates and Rules

> If an entity is not on Halo's published synced or non-synced lists, confirm with Halo support whether it is tracked before you assume it will travel with a push.

PURE-PLAY HALO PARTNER

Allied ESM is 100% focused on Halo.

We are one of a small number of pure-play Halo ITSM partners globally. Every consultant, every project, every support call is built exclusively around Halo, with no competing platforms and no divided loyalties.

That focus is what lets us answer questions like the ones in this guide from experience rather than from documentation. If you are planning an upgrade, building a pipeline, or trying to work out why your instances will not sync, we have almost certainly seen it before.

Allied ESM holds Cyber Essentials certification and our Halo-certified delivery consultants include personnel with SC and DV clearances.

Ready to talk? Contact us at info@alliedesm.com or visit alliedesm.com

---

Sources. This guide is based on Halo's published documentation, including the Release Notes, Software Releases, Security Management and Architecture, and Configuration Change Tracking and Instance Pipeline Management guides, together with Halo Service Solutions' G-Cloud 14 service definition. The instance scheduling detail on pages 5 and 6 was confirmed directly with Halo in August 2026. Product behaviour changes between releases. Confirm specifics for your own tenancy with Halo before acting on them.