DYNAVERTO
DIGITAL SOLUTIONS

Dynamics 365 CE & Power Platform ALM Checklist

A field guide for teams who would rather not spend Saturday fixing Friday's import

Application Lifecycle Management in Dynamics 365 Customer Engagement is not a tooling problem, it is a discipline problem. The platform will happily let you customise production directly, stack unmanaged changes forever, and deploy by exporting a zip file from one browser tab and importing it in another. Every one of those conveniences is a loan, and the interest rate is brutal. This checklist is grounded in delivery and recovery experience and aligned with Microsoft's current ALM guidance. Tick honestly — the items you skip past fastest are usually the ones that are on fire.

1 · Environment Strategy

Most teams treat environments as an afterthought until a "quick test" in production creates thousands of junk records. An environment strategy is not a diagram on a wiki; it is the set of things that are physically impossible for a developer to do to production.

- [ ] **Nobody customises directly in production — no exceptions, not even admins.** A "temporary" unmanaged change in production masks the managed behaviour beneath it for the properties it touches, and future updates of that component quietly stop taking effect until the layer is removed. Forgotten hotfixes of this kind routinely survive for years.
- [ ] **Dev, test, and production exist as separate environments with separate security roles.** When test and production share an environment, load testing becomes production load, and your first performance test doubles as your first outage.
- [ ] **Developer environments are disposable and rebuilt from source, not treasured.** If losing a dev environment costs more than a day, your customisations live in the environment instead of in source control — which means they live nowhere.
- [ ] **Sandbox refreshes are scheduled, announced, and preceded by a data export.** A refresh silently replaces the sandbox with a production copy. Carefully constructed test scenarios disappear with it, and UAT stalls while testers rebuild data by hand.
- [ ] **Production restore procedure is documented and has been rehearsed at least once.** Point-in-time restore exists, but the first time you use it should not be at 02:00 with a director on the call asking for an ETA you cannot give.

2 · Solution Architecture

The default failure mode is one giant solution containing everything, because it "keeps things simple". It does — right up until a multi-hour import fails near the end and leaves the target org in a half-deployed state on a Friday evening. Solution design is deployment design.

- [ ] **Solution boundaries are deliberate, reflecting scope, release cadence, and team ownership.** A single solution can serve a smaller implementation well; modularise when scale or independent deployment demands it. What fails is the unexamined monolith — hundreds of components where every deployment carries every risk, and one failed import means diagnosing all the rest.
- [ ] **One publisher, one prefix for your own customisations, enforced everywhere.** After two partners work in the same org you end up with new_, abc_, and cr7d3_ fields on the same table, and nobody can tell which fields are safe to delete. That archaeology never gets funded; the mess just compounds.
- [ ] **Downstream environments receive managed solutions only.** Unmanaged imports into test and production land in the single unmanaged layer that sits above every managed layer, and they cannot be cleanly uninstalled. Removing a component afterwards means hand-deleting dependencies in the right order, which is nobody's idea of an afternoon.
- [ ] **Unmanaged layers on key components are audited regularly and removed.** Anything sitting in the unmanaged layer keeps masking the managed values shipped underneath it. Check the "solution layers" view on your most important components — if you have never looked, budget a full day for what you find.
- [ ] **Cross-solution dependencies are mapped and deployment order is written down.** Undocumented dependencies surface as "missing component" import failures at deploy time, in production, with the change window closing.

3 · Source Control and Automation

If your solution lives only inside a Dataverse environment, you do not have source control — you have hope. Teams get this wrong because the platform makes manual export feel adequate for years, and then a corrupted environment or a departed developer proves it was not.

- [ ] **Every solution is represented in Git in human-readable source form.** Native Dataverse Git integration or pac solution unpack — either works; what matters is that without solution source in Git, code review is impossible and "what changed between v1.4 and v1.5" is answered by diffing zip files, which is to say never.
- [ ] **Builds are produced by a pipeline, not exported from a maker's browser.** Manual exports pick up whatever half-finished experiments were sitting in the dev environment that day, including the stray draft flow nobody meant to ship.
- [ ] **Deployments use one governed release path per solution — Power Platform Pipelines, extended where required, or your own Azure DevOps / GitHub pipeline.** Half-adopting two deployment mechanisms gives you two partial paths and zero authoritative ones. Which tool matters less than the rule: exactly one path is authoritative, and nothing ships around it.
- [ ] **Solution Checker runs on every build and blocks on critical findings.** Skipping it means deprecated API calls and unsupported client-side hacks accumulate silently until a release wave enforcement turns them off for you, on Microsoft's schedule rather than yours.
- [ ] **Plug-in and PCF source code is versioned alongside the solution.** An assembly in Dataverse with no matching source in Git is a black box you will eventually need to change and cannot. Decompiling your own product is a special kind of humiliation.

4 · Configuration, Connections, and Identity

Configuration and identity failures are uniquely nasty because they tend to break many things at once, in production only, and usually while the one person who understands them is on leave.

- [ ] **All environment-specific values live in environment variables, never hard-coded.** A hard-coded test URL that ships to production sends real customer data to your sandbox API. It happens quietly, and you find out from an auditor.
- [ ] **Environment variable values are set at deployment time and documented.** Per-environment values deliberately survive solution updates. That persistence works in your favour when values are applied through the release process and recorded — and against you when they are edited ad hoc and never written down.
- [ ] **Business-critical flows have non-personal ownership, with connection references mapped to controlled connections.** Use a service principal where supported, or a governed service account where connector authentication requires it. When a flow owner leaves and IT disables their account, every connection tied to that person fails at once — and the cleanup is measured in days, not minutes, because nobody knows which connections were theirs.
- [ ] **Flow and app ownership is inventoried, with no single human as sole owner.** Orphaned flows keep running until they need re-authentication, then fail with an error message only their departed owner would have received.
- [ ] **Service account credentials and secrets are in a vault with a rotation plan.** A password in a OneNote page titled "IMPORTANT DO NOT DELETE" is not a secrets strategy, though it is a remarkably common one.

5 · Release Discipline

Deployments fail in predictable ways, which means most deployment disasters are choices. The teams that get burnt are the ones treating each release as a unique artisanal event rather than a boring, rehearsed, reversible routine.

- [ ] **Every production deployment is rehearsed in an environment that mirrors production.** The import that works in a clean sandbox and fails in production is failing because of production's layer stack. If your test environment does not share that stack, you are rehearsing a different play.
- [ ] **No production deployments on Friday afternoon, ever.** The failure will surface Saturday morning, support queues run on business days, and your weekend is now a bridge call.
- [ ] **A rollback plan exists before the import starts, not after it fails.** Managed solution upgrades cannot always be reversed by "importing the old version". Know before you start whether your path back is downgrade, restore, or hotfix-forward — deciding at midnight goes badly.
- [ ] **Deployments use holding solutions or upgrade mode when deleting components.** A plain update never removes components, so the fields you deleted in dev live on in production indefinitely, still filled by legacy integrations nobody remembers.
- [ ] **Release wave plans are reviewed, and auto-enabled changes are tested in a sandbox first.** Part of every wave ships enabled by default, ready or not. Opting in early on a sandbox costs a day twice a year; being surprised in production costs considerably more.

6 · Operations and Observability

Many Dynamics teams discover production problems the same way: a user emails a screenshot. That is not monitoring, it is crowdsourced incident detection with a latency of hours. What you cannot see, you will explain to a steering committee later.

- ☐ **Application Insights is connected to production and someone actually reads it.** Without telemetry, "the system is slow" investigations start from zero every time. With it, the plug-in whose execution time exploded after a data migration is visible the same day.
- ☐ **Flow failures alert a shared channel, not the owner's personal inbox.** A quietly failing integration flow can drop records for weeks before anyone notices — failure notifications that land in an unmonitored mailbox might as well not exist.
- ☐ **Storage capacity, request entitlements, and throttling are monitored as separate things.** They fail differently: storage exhaustion blocks operations, entitlement overage becomes a licensing conversation, and service-protection throttling returns 429s that your integrations must handle by honouring Retry-After. Buying more storage fixes none of the other two.
- ☐ **Deprecation announcements and wave release plans have a named reader.** Microsoft tells you months in advance which legacy endpoints and features die. The teams that get "surprised" simply had nobody assigned to read the announcements.
- ☐ **There is a living runbook for the top five failure scenarios.** Institutional knowledge that lives in one senior developer's head has the same availability SLA as that developer's employment contract.

Critical red flags — these override any score

- ! Direct customisation in production
- ! Solutions that exist only inside environments, with no source control
- ! Business-critical connections and flows owned by one person's user account
- ! No tested restore or rollback path
- ! Unmanaged solutions imported into production

If any of these is true, treat your score as zero until it is fixed. A team can tick twenty-five boxes and still be one resignation or one bad import away from a very long week.

How to read your score

This is a self-assessment aid, not a maturity certification. Count your honest ticks — honest meaning "we do this every time", not "we did it once". Ten or fewer: you are in triage; stop adding features for a sprint and fix the bleeding, starting with source control and connection ownership. Eleven to twenty: you are stabilising — one departure or one bad import can still cost you a week, so pick the three unticked items that scare you most and schedule them. Twenty-one or more: you are running an engineering practice rather than a sequence of near misses; your remaining gaps are known trade-offs instead of surprises, which is the most anyone can honestly claim.

If the list made you wince

Most of the failures above are cheap to prevent and expensive to survive, and the gap between the two is usually one honest architecture review. DynaVerto runs exactly that: a few days inside your environments, layers, and pipelines, ending with a prioritised plan rather than a slide deck. If more than a dozen boxes stayed empty, it is worth a conversation before the platform schedules one for you.

Book an ALM architecture review.

info@dynaverto.si · dynaverto.si

DynaVerto d.o.o. · Šišenska cesta 2, 1000 Ljubljana, Slovenia

Sources & further reading

Solution layers · Organize solutions · Dataverse Git integration · Environment variables · Service principal-owned flows · Dataverse API limits · Power Platform Pipelines