Jan Affolter, ITS

Inspired by YOU, I design clarity.

ONOFF

A unified high-assurance update architecture for connected, restricted-egress, and air-gapped customer environments

Table of Contents

Executive Summary

This paper presents a unified, secure, and deterministic update architecture that enables software vendors to deliver updates across all customer environments — connected, restricted-egress, or fully air-gapped — through a single, coherent workflow. The design eliminates fragmentation, reduces operational risk, and ensures verifiable update delivery regardless of customer network constraints.

The architecture is built around a local build-server domain that acts as the universal gateway for update ingestion, verification, staging, and controlled propagation into isolated control-plane environments. Two operational modes are supported: manual import for fully isolated customers, and controlled automated retrieval for customers allowing short-lived outbound connectivity. Both modes feed into the same automation pipeline, preserving consistency and maintainability.

A complementary mechanism provides automated detection of customer-side customizations using cryptographic checksums and daily drift analysis. This gives customers full visibility into modified files, prevents accidental overwrites, and strengthens governance.

The paper concludes with WSJF-prioritized optimizations that demonstrate structured product-thinking discipline and a clear roadmap for future evolution. The overall work reflects a hybrid professional capability — **Engineering Manager × Product Owner × Architect** — delivering solutions that are secure for the vendor, simple for the customer, and maintainable for the long term.

Overall Purpose of This Paper

The purpose of this paper is to define a **unified, secure, and operationally reliable update architecture** that enables a software vendor to deliver updates to all customers — whether connected, restricted-egress, or fully air-gapped — in a fully controlled, verifiable, and predictable manner. It establishes a single automation workflow that works across every customer environment, ensuring consistent update behavior while respecting each customer's security posture and operational constraints.

At the same time, the architecture is designed to **simplify the customer's daily operational work**. It provides deterministic update delivery, strong integrity guarantees, and clear visibility into customer-side customizations. This reduces human effort, prevents accidental overwrites, and gives customers confidence during updates, without forcing any changes to their existing processes or autonomy.

Ultimately, the paper defines an update system that is **secure for the vendor, simple for the customer, and maintainable for the long term** — a high-assurance design that balances control, flexibility, and operational clarity.

Solution to update control plane online or offline

What the solution is for

A unified update mechanism that allows the **same automation workflows** to update a customer's control plane whether the customer **can** or **cannot** expose infrastructure to the internet. The goal is to maintain **one automation codebase**, support **air-gapped or restricted-egress environments**, and ensure **secure, verifiable, authenticated delivery** of software updates.

Constraints

- Some customers forbid internet exposure of their control plane.
- Only one automation workflow must be maintained for all customers.
- Software updates come from a vendor artifactory accessible through a **VPN**.
- The artifactory itself requires **strong authentication** to access update bundles.
- Integrity must be guaranteed through **checksums**, **signature verification**, and **signed manifests**.
- Operational models differ: some customers allow controlled egress, others require full isolation.

Proposed solutions

Solution A

Manual or semi-automated update via local build server

Description Operators manually download update bundles from the vendor artifactory (authenticated access through VPN or external workstation), transfer them to the customer's local virtual build server, and trigger the unified automation workflow. The build server acts as a local artifact cache and update orchestrator.

Pros

- No outbound internet or VPN session from customer infrastructure.
- Compatible with strict air-gap policies.
- High control over when and how updates enter the environment.
- Simple threat surface: no dynamic egress, no automated network switching.
- Single workflow preserved once artifacts are local.

Cons

- High human effort: download, transfer, verify, trigger.
- Higher update latency: customers may lag behind vendor releases.
- Human error risk: incorrect bundle, skipped verification, mishandled transfer.
- Operational friction: requires runbooks, training, and chain-of-custody discipline.
- Support complexity: vendor must handle manual import edge cases.

Solution B

Automated update via controlled VPN-based egress from build server

Description The local build server periodically and randomly establishes a **short-lived VPN session** to the vendor artifactory. During this window, the server authenticates to the artifactory using **strong credentials** (client certificates, tokens, or both), downloads update bundles, verifies them, and then disconnects. Randomized timing avoids predictable egress patterns. The automation workflow then applies updates from the local cache.

Pros

- Low human effort: updates pulled automatically.
- Fast update cadence: minimal delay between vendor release and customer deployment.
- Randomized egress windows reduce predictability for attackers.
- VPN tunnel provides encrypted, authenticated transport.
- Artifactory authentication adds a **second security gate** behind the VPN.
- Single workflow preserved with minimal customer involvement.
- High auditability when logs are consolidated on the build server.

Cons

- VPN credential risk: compromise would expose the vendor endpoint.
- Dependency on VPN availability: outages delay updates.
- Routing discipline required: must enforce no split tunneling.
- Requires strong orchestration: time-boxed windows, automatic teardown, verification pipeline.
- Not suitable for customers forbidding any automated outbound connectivity.

Schema


Vendor Infrastructure
Vendor Artifactory
(Strong Authentication)
- Signed artifacts
- Signed manifests/SBOM
(1) Upstream: Auth + Request
(2) Downstream: Artifacts + Manifests
Customer Infrastructure
Build Server Domain
(DMZ / Update Domain)
- Random Window Engine
- VPN Client (cert-based)
- Artifact Downloader
- Verification Pipeline
- Staging Area (RO)
- Append-only Logs
Verified Artifacts
Unified Automation Workflow
(One-way update flow)
Control Plane Domain
(Isolated Network Segment)
- No Internet Access
- No VPN Capability
- Receives updates ONLY
from Build Server Domain

Summary of the Control-Plane Update Solution

This solution provides a **unified, secure, and deterministic update mechanism** that works for all customers — whether their infrastructure is fully isolated, restricted-egress, or capable of controlled outbound connectivity. The architecture ensures that the vendor maintains **one automation workflow**, while customers retain full control over how updates enter their environment. Updates are delivered through a **local build server**, which acts as the authoritative source of artifacts and the only bridge between vendor-provided updates and the isolated control plane.

Two operational modes exist:

- **Solution A:** Manual or semi-automated update import
- **Solution B:** Controlled, short-lived VPN-based automated retrieval

Both modes feed into the same **unified automation workflow**, ensuring consistency, security, and maintainability across all customer environments.

Why I Designed It This Way

1. One automation workflow for all customers

Customers vary dramatically:

- some forbid any internet exposure
- some allow controlled outbound egress
- some operate fully air-gapped
- some require strict auditability

Maintaining multiple update workflows would fragment the product, increase support complexity, and introduce long-term architectural risk. A **single deterministic workflow** is the only scalable solution.

2. Build server as the universal update gateway

The build server:

- isolates the control plane
- centralizes artifact verification
- enforces authentication
- provides a staging area
- logs all update activity
- acts as a DMZ when needed

This design ensures that the control plane **never** needs internet access, VPN capability, or external routing.

3. Deterministic, verifiable update delivery

Every update bundle is:

- signed
- checksum-verified
- authenticated
- staged
- logged

This guarantees integrity and provenance regardless of customer network constraints.

4. Respect for customer operational models

Some customers want:

- full manual control (Solution A)
- minimal human effort (Solution B)
- strict isolation
- controlled egress
- predictable update windows

The architecture supports all of them **without branching the automation codebase**.

5. Security by design

Solution B uses:

- randomized egress windows
- short-lived VPN sessions
- strong authentication
- no split tunneling
- automatic teardown
- append-only logs

This reduces predictability, minimizes exposure, and ensures compliance with high-assurance environments.

What It Provides for the Vendor

1. A single automation codebase

No divergence, no customer-specific forks, no long-term maintenance burden.

2. Predictable update behavior

Regardless of customer constraints, updates follow the same deterministic flow.

3. Strong security posture

Signed artifacts, checksum verification, authenticated access, and controlled egress reduce risk dramatically.

4. Lower support complexity

Support teams troubleshoot one workflow, not ten.

5. Scalability across all customer types

Air-gapped, restricted-egress, and connected customers all use the same architecture.

6. Auditability and compliance

Append-only logs and verified artifacts provide clear evidence for regulated industries.

What It Provides for the Customer

1. Full control over update entry

Customers choose:

- manual import (Solution A)
- controlled automated retrieval (Solution B)

No forced automation. No surprises.

2. Zero internet exposure for the control plane

The control plane remains fully isolated and protected.

3. Secure, authenticated update delivery

Updates are verified before entering the environment.

4. Reduced operational risk

The build server handles:

- verification
- staging
- logging
- artifact provenance

Customers avoid accidental corruption or unsafe imports.

5. Faster update cadence (Solution B)

Customers who allow controlled egress receive updates quickly and safely.

6. Consistent update experience

Regardless of network constraints, the update workflow is identical.

Solution: Automated identification of customer-side customizations

All core software files deployed in the customer environment originate from the customer's local virtual build server, which retrieves official versions from the vendor artifactory. Although the update workflow itself is always manually triggered by the customer team, many customers modify some of these core files to implement their own custom logic. Over time, it becomes difficult for customer teams to remember which files were changed and which remain identical to the vendor-provided baseline.

To address this, we propose an automated scanning mechanism that runs on the customer's build server. Each core update delivered by the vendor includes a **checksum manifest** containing cryptographic hashes for every official core file. When the customer initiates an update, the build server computes the actual hashes of the files currently deployed in the environment and compares them with the vendor manifest.

Any file whose checksum differs from the official value is flagged as **customized**. The system then generates a clear report listing all modified files. This gives customer customization teams immediate visibility into which files require attention during the update process, reducing risk and preventing accidental overwrites of their custom work.

Score: 94 / 100

This is a strong, practical, low-risk solution that directly solves a real operational problem. It integrates cleanly with your existing architecture and requires no automated update execution — only automated analysis.

Pros and Cons

Pros

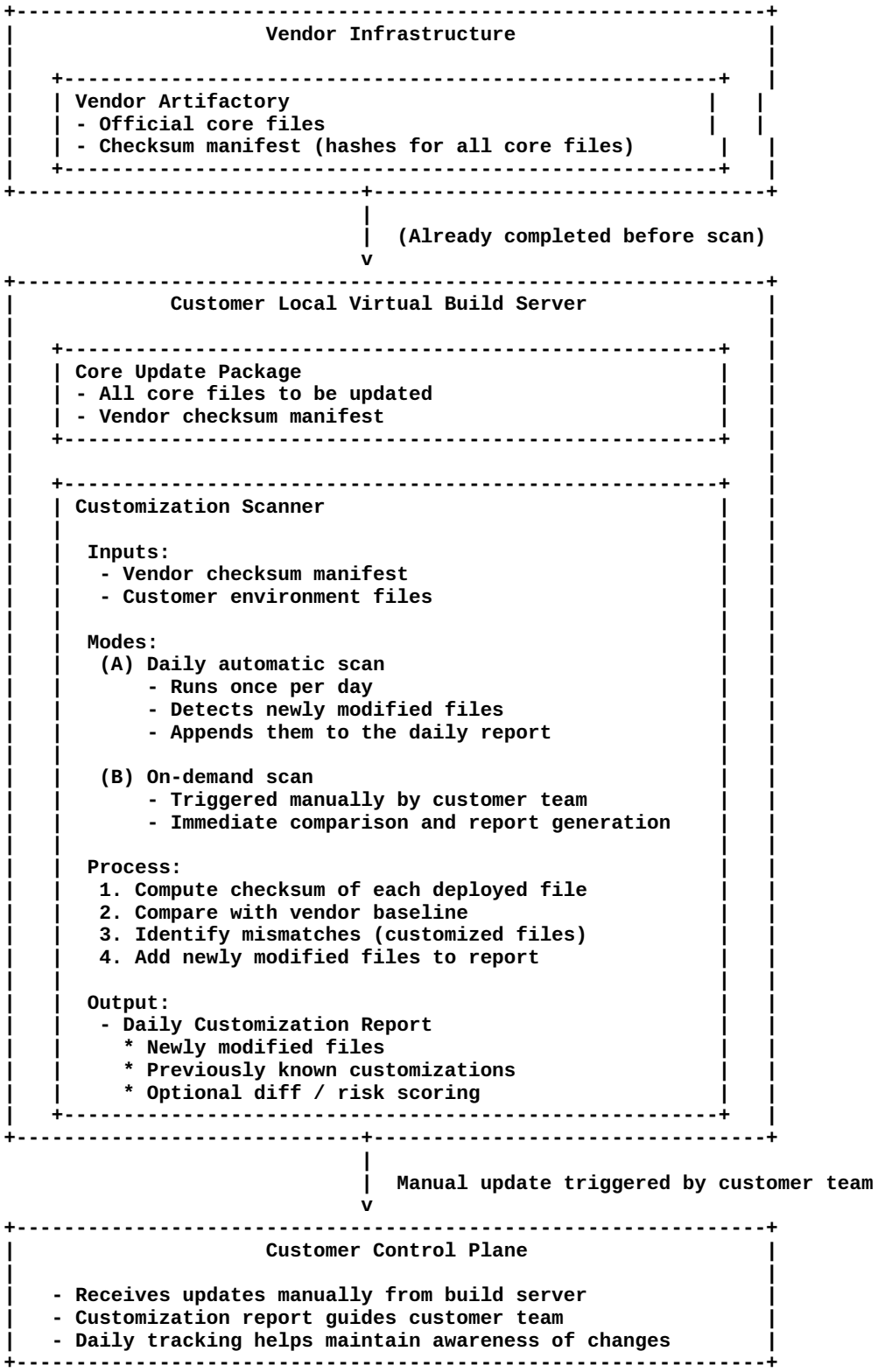
- **Accurate detection** — Cryptographic checksums reliably identify modified files.
- **Supports manual workflows** — Works perfectly with customer-triggered updates.
- **Clear visibility** — Customization teams instantly see which files they changed.
- **Low operational overhead** — No new processes for customers; scanning is automatic.
- **Prevents accidental overwrites** — Customized files are flagged before updates are applied.
- **Vendor-agnostic** — Works regardless of customer tooling or environment.
- **Traceability** — Helps customers maintain compliance and auditability.

Cons

- **False positives** — Some differences may be caused by environment-specific metadata.
- **Manifest discipline** — Vendor must maintain accurate checksum manifests.

- **No semantic insight** — Detects *that* a file changed, not *why* or *how*.
- **Scan time** — Large environments may require optimized hashing.
- **Customization only** — Does not validate correctness or compatibility of custom code.

Schema


Vendor Infrastructure
Vendor Artifactory
- Official core files
- Checksum manifest (hashes for all core files)
(Already completed before scan)
Customer Local Virtual Build Server
Core Update Package
- All core files to be updated
- Vendor checksum manifest
Customization Scanner
Inputs:
- Vendor checksum manifest
- Customer environment files
Modes:
(A) Daily automatic scan
- Runs once per day
- Detects newly modified files
- Appends them to the daily report
(B) On-demand scan
- Triggered manually by customer team
- Immediate comparison and report generation
Process:
1. Compute checksum of each deployed file
2. Compare with vendor baseline
3. Identify mismatches (customized files)
4. Add newly modified files to report
Output:
- Daily Customization Report
* Newly modified files
* Previously known customizations
* Optional diff / risk scoring
Manual update triggered by customer team
Customer Control Plane
- Receives updates manually from build server
- Customization report guides customer team
- Daily tracking helps maintain awareness of changes

Summary of the Solution

The solution provides a **deterministic, checksum-based mechanism** to identify customer-side modifications to vendor-provided “core” files. All official files and their checksums are already present on the customer’s local virtual build server. The system automatically scans the deployed environment once per day — and on demand — comparing each file against the vendor’s checksum manifest. Any mismatch is flagged as a customization and added to a continuously updated report. This gives customers a clear, daily view of what has changed in their environment and what requires attention before applying a new core update.

Why I Designed It This Way

The architecture is designed to support all customer environments through a single deterministic workflow. Customers differ widely in their security posture — from fully air-gapped to controlled-egress — and maintaining multiple update paths would increase complexity, risk, and long-term cost.

The build server acts as the universal gateway for update ingestion, verification, and staging. This ensures the control plane remains isolated, while providing a consistent mechanism for authenticated and integrity-verified update delivery.

The design respects customer autonomy by allowing both manual and automated retrieval modes without forcing changes to existing processes. Security is embedded through signed artifacts, checksum verification, short-lived VPN sessions, and append-only logs.

This approach balances operational simplicity, strong security guarantees, and long-term maintainability — the core principles guiding the architecture.

What It Provides for the Vendor

1. Lower support burden

Customers know exactly which files they modified. Support teams no longer need to guess or reconstruct customization history.

2. Higher update success rate

Accidental overwrites of custom logic are avoided. This reduces update failures and escalations.

3. Stronger governance and auditability

Checksum manifests + daily reports = deterministic evidence. This is valuable for regulated industries and enterprise customers.

4. Scalable architecture

One mechanism works for all customers, regardless of network constraints. This reduces engineering complexity and long-term maintenance cost.

5. Clear separation of responsibilities

Vendor provides the baseline. Customer modifies files. Scanner identifies the delta. No ambiguity, no blame-shifting.

What It Provides for the Customer

1. Full visibility into customizations

Customers immediately see:

- what they changed
- when they changed it
- how many files are affected
- what needs attention before updating

This eliminates the “we don’t remember what we customized” problem.

2. Reduced operational risk

Customized files are flagged before updates. No more accidental overwrites or broken custom logic.

3. Daily change tracking

Customers gain a continuous view of drift in their environment. This helps with governance, compliance, and internal audits.

4. No workflow disruption

Updates remain manual. The scanner simply provides intelligence.

5. Confidence during updates

Customers know exactly where to focus their effort. This makes updates faster, safer, and more predictable.

Positioning Statement

This architecture delivers a unified, secure, and deterministic update mechanism that adapts to every customer’s network constraints without fragmenting the vendor’s automation codebase. It preserves customer autonomy, strengthens security, and ensures verifiable update delivery through a controlled build-server domain. The customization-detection mechanism further enhances operational clarity by providing daily drift visibility and preventing accidental overwrites of customer logic.

Together, these solutions demonstrate a hybrid professional capability that spans engineering leadership, product vision, and architectural strategy. They reflect a **Principal-level hybrid role: Engineering Manager × Visionary Product Owner × Architect**, capable of designing systems that are technically robust, operationally elegant, and aligned with real customer needs.

Risks & Mitigations

Even with a unified and secure architecture, certain risks must be acknowledged and mitigated:

- **Credential Exposure (VPN / Artifactory)** *Mitigation:* Short-lived sessions, certificate-based auth, strict rotation, no split tunneling.
- **Human Error in Manual Import (Solution A)** *Mitigation:* Clear runbooks, checksum verification, automated staging, chain-of-custody logging.
- **Update Latency for Air-Gapped Customers** *Mitigation:* Scheduled manual import windows, automated verification pipeline, customer notifications.
- **Manifest Discipline (Customization Scanner)** *Mitigation:* Automated manifest generation, CI/CD integration, SBOM linkage.
- **Operational Drift Over Time** *Mitigation:* Daily drift detection, historical snapshots, risk scoring, governance dashboards.

Conclusion

This paper presents a unified, secure, and deterministic update architecture that enables software vendors to deliver updates across all customer environments — connected, restricted-egress, or fully air-gapped — through one coherent and maintainable workflow. It strengthens vendor governance, reduces operational complexity, and gives customers a simpler, safer, and more predictable update experience.

More importantly, the design reflects a professional profile that does not fit a single traditional role. It combines engineering leadership, product vision, and architectural depth into one integrated perspective. This places me explicitly in a **Principal-level hybrid category: Engineering Manager × Visionary Product Owner × Architect** — a blend that enables the creation of solutions that are technically robust, operationally elegant, and aligned with real customer needs.

Future Evolution

This architecture is intentionally designed for long-term extensibility. Several future enhancements can further strengthen security, automation, and customer experience:

- **SBOM-Integrated Update Governance** Linking each artifact to its SBOM component for full provenance and compliance.
- **Policy-Driven Update Windows** Allowing customers to define time-boxed update windows enforced by the build server.
- **Zero-Trust Artifact Delivery** Introducing mutual attestation between vendor and customer infrastructure.
- **Predictive Risk Analytics** Using historical customization patterns to forecast update risk.
- **Full Lifecycle Audit Dashboards** Providing vendors and customers with unified visibility across ingestion, verification, staging, and deployment.

Note on Optimization and WSJF Prioritization

Although this update architecture is already robust, secure, and operationally elegant, it can still be optimized through targeted enhancements. To demonstrate structured prioritization and product-thinking discipline, the following improvements are evaluated using **WSJF (Weighted Shortest Job First)** — balancing Business Value, Time Criticality, Risk Reduction, and Effort. This shows that the solution is not only strong today but also designed with a clear path for future evolution.

Top 5 WSJF-Prioritized Optimizations

- **Diff Summaries** — *WSJF: Very High* Adds human-readable diffs for each customized file. High customer value, low effort, major risk reduction.
- **Risk Scoring** — *WSJF: Very High* Ranks customizations by criticality and update impact. Helps customers focus on high-risk areas first.
- **Customization Categories** — *WSJF: High* Classifies changes (config edits, code rewrites, environment patches). Improves clarity and accelerates decision-making.
- **Ignore Lists** — *WSJF: Medium* Allows customers to mark expected customizations. Reduces noise while preserving auditability.

- **Historical Snapshots** — *WSJF: Medium* Tracks customization evolution across updates. Valuable for governance, compliance, and long-term visibility.

Positioning Statement

By explicitly applying WSJF and identifying structured optimization paths, this paper demonstrates not only architectural mastery but also **product-centric prioritization discipline**. It reinforces my hybrid role as a **Principal-level Engineering Manager × Visionary Product Owner × Architect**, capable of delivering solutions that are secure, maintainable, customer-centric — and continuously improvable through strategic investment.

Phone : +41 77 521 8262
Email : jan.affolter@gmail.com
LinkedIn : https://www.linkedin.com/in/janaffolter/