Verwenden des programmierbaren Installationsprogramms

Inhalt

Einleitung

In diesem Dokument wird der programmierbare Installer beschrieben, eine spezielle Automatisierungsplattform für die Bereitstellung von Cisco Software-Stacks wie NSO und CNC.

Programmierbarer Installer

Feld	Wert
Produkt	Programmierbarer Installer
Dokumenttyp	Technisches Whitepaper: Architektur, Implementierung und Ergebnisse
Primäre Zielgruppe	Lösungsarchitekten, Plattformtechniker, DevOps/SRE, Leads bei der Bereitstellung
Sekundäre Zielgruppe	Engineering-Management, Sicherheitsberater, Programm-Manager

Zusammenfassung

Der programmierbare Installer ist eine spezielle Automatisierungsplattform für die Bereitstellung und den Betrieb von Cisco Software-Stacks, einschließlich Network Services Orchestrator (NSO), Crosswork Network Controller (CNC), Crosswork Data Gateway (CDG) und Business Process Automation (BPA), unter Enterprise Linux (RHEL-Produktfamilie) und ggf. der zugehörigen Infrastruktur (VMware vCenter, OpenShift, KVM, air-gapped Kubernetes). Das System trennt die deklarative Absicht (YAML-Spezifikationen und optionale gesteuerte Absichtserstellung) von der Ausführung (Ansible-Rollen und Playbooks) mit einer Python-Kontrollebene, die Artefakte verpackt, Pakete vor langwierigen Installationen überprüft, verschlüsselte Geheimnisse vorbereitet und Validierungs-Gates orchestriert.

In diesem Whitepaper werden die Architekturschichten, die primären Datenflüsse, die Implementierungsmuster (einschließlich eines hybriden datengesteuerten Artefaktverifizierungsmodells), die Bereitstellungsmodi (nativ, containerisiert, online, Air-Gap) sowie die Validierungs- und Protokollierungs-Frameworks erläutert. Für Bereitstellungs- und Plattformorganisationen zielt die Plattform darauf ab, manuelle Eingriffe, falsche Oberflächenkonfigurationen und fehlende Binärdateien frühzeitig zu reduzieren, die

Automatisierung über Produkte und Topologien hinweg zu standardisieren und dabei die umgebungsspezifische Parametrierung zu bewahren.

Schlüsselwörter: Infrastruktur-Automatisierung, deklarative Bereitstellung, Ansible, YAML-Spezifikation, Air-Gap Packaging, Artefaktverifizierung, Crosswork, NSO, CNC, CDG, BPA, Validierungsrichtlinie, DevOps.

Lesen dieses Dokuments

Rolle	Empfohlener Schwerpunkt
Entscheidungsträger/Leads	Abstrakt; §1 Wertangebot; §8 Nutzen und Risiko; §10 Zusammenfassung
Lösungsarchitekten	§3-§6 (Architektur, Spezifikationsmodell, Implementierung, Bereitstellungsmuster)
DevOps/SRE/Delivery Engineers	§ 5-§ 7 Anhang B; interne White Paper-Anhänge
Sicherheitsüberprüfer	§7 Sicherheit und Compliance Vertrauensgrenzen in §3.2

---

1. Wertangebot

1.1 Das Bereitstellungsproblem

Die Installation von Multi-Tier-Produkten zur Netzwerkautomatisierung und -orchestrierung erfolgt traditionell per High-Touch: lange Runbooks, viele manuelle Schritte, Versions-Skew zwischen Standorten und Ausfälle, die Stunden in einen Prozess überführen (fehlende NEDs, falsche OVA-Pfade, unvollständige Luftspalt-Image-Sets). Dieses Muster erhöht die Kosten, verlängert die Änderungsfenster und erschwert Audits.

1.2 Das Konzept des programmierbaren Installers

Der programmierbare Installer behandelt eine Installation als ein Programm, das durch eine Spezifikation parametrisiert wird: Topologie, Versionen, Plattformauswahl (vCenter vs OpenShift vs Vanille VMs), Dateipfade und Berechtigungen. Die Automatisierung ist, wo möglich, wirkungslos, bei allen Kunden wiederholbar und mit Prüfungen versehen, sodass "nicht bereit" ein schnelles, explizites Ergebnis vor der Cluster- oder Produktinstallation ist.

1.3 Was bedeutet "programmierbar" in diesem Zusammenhang?

- Deklarativ: Operatoren beschreiben, was bereitgestellt werden muss. Strategische Leitfäden erläutern, wie.
- Datengesteuerte Überprüfung: Bei stabilen Mustern werden erwartete Artefakte aus Tabellen und Regeln abgeleitet und nicht aus Ad-hoc-Skripts pro Version.
- Richtliniengesteuerte Qualität: Die Validierung vor und nach der Bereitstellung erfolgt anhand hierarchischer Richtlinien mit strukturierten Berichten und optionaler Integration von Tickets.
- Multimodaler Betrieb: Interaktive Menüs für erstmalige Benutzer; CLI und eingefrorene Binärdateien für CI/CD-ähnliche Wiederholungen; Docker-basierte Flows für standardisierte Laufzeitbilder.

---

2. Systemkontext

2.1 Akteure und Umgebungen

Akteur/System	Rolle
Bereitstellungstechniker	Verfasst Spezifikationen oder generiert diese, führt Verpackungen aus, bereitet Tresore vor, validiert, orchestriert und erkennt.
Installationshost	Linux-Kontrollknoten (nativ oder Container) mit Python, Ansible Konfiguration, Festplatte für Artefakte
Zielinfrastruktur	VMs mit vCenter, OpenShift/KubeVirt oder Vanille nach Spezifikation
Artefaktquellen	Interne Spiegelungen, Berechtigungs-Layouts, Softwareverteilung - umgebungsspezifisch
Downstream-Systeme	Überwachung, Change-Management, optionale JIRA-Workflows

2.2 Vertrauensgrenzen

1. Spezifikationen ausdrücklich Absicht; können sie auf Pfade und nicht geheime Parameter verweisen. Geheimnisse müssen durch Ansible Vault-Workflows fließen und nicht durch Felder mit Klartext-Spezifikationen, wo dies vermeidbar ist.
2. Die Speicherung von Artefakten muss integritätsgeschützt sein. Bei der Verifizierung liegt der Schwerpunkt auf der Präsenz und der Namensanpassung an die Spezifikation. Dies gilt auch für die organisatorischen Kontrollen (Prüfsummen, signierte Pakete), für die Richtlinien erforderlich sind.
3. Installer-to-Target-SSH ist ein Pfad mit hohen Rechten. Die Kompromittierung des Installationshosts hat große Auswirkungen. Absicherung und Zugriffskontrolle sind Voraussetzung für den Betrieb.

---

3. Architektonische Grundsätze

1. Deklarativ zuerst: Benutzerabsicht in YAML; wird von der Automatisierung konsistent interpretiert.
2. Trennung von Planung und Ausführung: Python plant, verifiziert und orchestriert die Gates; Ansible führt Infrastruktur- und Produktschritte aus.
3. Automatisierung der Zusammenstellung: Playbooks auf Standortebene importieren zielgerichtete Playbooks (Vorinstallation, Kubernetes-Tracks, Produktinstallationen).
4. Progressive Offenlegung: Interaktive Einrichtung für das Onboarding; Flags und Skripts für eine erweiterte Automatisierung.
5. Gleiche Codebasis, mehrere Laufzeiten: Native AlmaLinux/RHEL-Pfade und Docker-basierte Pfade teilen sich ein Repository-Layout.
6. Explizite Luftspaltunterstützung: Verpackung auf einem angeschlossenen Gerät; Bündel übertragen; Sie müssen erforderliche Komponenten installieren und ohne Laufzeitabhängigkeit von öffentlichen Netzwerken bereitstellen.

---

4. Logische Architektur

4.1 Schichten

Schicht	Verantwortung
Spezifikation	Topologie, Versionen, Plattformen, Pfade, Berechtigungen
Steuern Sie Fläche	Verpackung, Paketverifizierung, Vault Helper, Validierungstreiber, Orchestrator-CLI
Automatisierungsebene	Host-Vorbereitung, Kubernetes-Lebenszyklus, Produktinstallation und Day-one-Konfiguration
Artefakte	Binärdateien, Bilder, Diagramme, OVAs, Tarballs

4.2 End-to-End-Datenströme

1. Paketfluss: Das erforderliche Tool lädt Dateien herunter oder stuft sie an einen bestimmten Ort, um optional einen übertragbaren Tarball für Offline-Installationen zu erzeugen.
2. Workflow-Überprüfung: Der Orchestrator analysiert die Spezifikation, löst erwartete Artefakte (einschließlich Plattformfilter und Berechtigungslisten) auf und meldet vor der Installation die Bereitschaft bzw. das Fehlen.
3. Flow bereitstellen: Spez. plus Tresor (und optionale Vorabvalidierung) sorgen für ansprechende Playbooks für Bestände, die aus dem Projekt abgeleitet wurden.

4.3 Unterstützte Produkte (Umfang des Installers)

Produkt/Paket
NSO

Produkt/Paket
CNC
CDG
BPA
CNC + NSO

---

5. Spezifikation und Absichtsmodell

5.1 Benutzerspezifikationen

Die Spezifikationen sind YAML-Dokumente, die Plattformen (z. B. vCenter, OCP, VM, KVM), Hosts, Anwendungen mit Versionen, Topologie (z. B. CFS/RFS-Layouts von NSO), Berechtigungen (NEDs und Add-on-Pakete) und Dateipfade für OVAs, qcow2-Images und Tarballs auf Anwendungsebene beschreiben.

5.2 Häufige Fragmente

Eine spezielle Anwendung user_spec liefert Standardwerte und Pfad-Fallbacks für CNC/CDG. Der Parser des Orchesters behandelt die Benutzerspezifikation als Quelle der Wahrheit und verwendet gemeinsame Spezifikationseinträge, wenn keine Benutzerschlüssel vorhanden sind.

5.3 Absichtserstellung

Der Intent Generator unterstützt die gezielte Erhebung von Anforderungen per Fragebogen, einer Regel-Engine (datumsgesteuerte Logik) und die schemagestützte Zuordnung zu intent.yaml.

---

6. Details zur Implementierung

6.1 Deployment Orchestrator (cx_deploy_orchestrator.py)

Der Orchestrator ist der einzige Eintrag für die Koordinierung skriptgesteuerter oder interaktiver Generierungs-, Verifizierungs- und Installationspakete. Sein Design ist explizit hybrid:

- ARTIFACT_DEFS: Deklariert Artefakttypen und Benennungsmuster pro Anwendung (NSO-Installationsprogramm, NED-signierte Pakete, optionale Pakete; CNC OVA/qcow2/Tier Tarballs; CDG-Bilder; BPA-Diagramme und Air-Gap-Bild-Tarballs).

- APP_CONFIG: Ordnet CLI —app-Werte (nso, cross-worksuite, bpa) den Namen der Spezifikationsordner und den Namen der Standard-Absichtsdateien zu.
- Parser/Handler: Auflösen von CNC/CDG-Pfaden mithilfe der Benutzerspezifikation und der allgemeinen Spezifikation; benutzerdefinierte Handler decken die ungleichmäßige Benennung (z. B. TSDN/DLM) und die BPA-Versionsformatierung für Diagramm- und Tarball-Pfade ab.

Bereitschaft:AReportaggregiert erkannte Artefakte; is_readyist true, wenn nach der Spec-Analyse keine erforderlichen Dateien fehlen. Das Modul unterstützt PyInstaller-gefrorene Binärdateien über die sys.freepath-Auflösung.

6.2 Voraussetzungen und Paket-Tooling (setup_cxinstaller_prereqs)

Diese Komponente stellt interaktive Menüs und CLI-Modi für das Packen von Artefakten und die Installation der erforderlichen Host-Komponenten bereit: online, Air-Gap oder automatische Erkennung; Verpackung für mehrere Anwendungen, einschließlich `kombinierterCNC_NSO`. Er füllt den Artefaktbaum auf, der von Ansible und von der Verifizierungs-Paketlogik erwartet wird.

6.3 Ansible Automation Plane

- Zusammensetzung: Diese illustriert den mehrspurigen Charakter des Stacks: Es importiert verschiedene Kubernetes-Bootstrap-Pfade - dies zeigt, dass verschiedene Produkte unterschiedliche Kubernetes-Bootstrap-Pfade anvisieren.
- Rollen (repräsentative Familien): Vorinstallation (SELinux, Firewall, SSH, Registry Helpers), k8s_install / rke2, deploy_nso, deploy_cnc, deploy_cdg, deploy_bpa, Postinstall, Deinstallation und Zertifikatverlängerung. Verantwortung und Tests lassen sich am besten an Rollengrenzen verwalten. geschachtelte Aufgabenordner implementieren Sub-Workflows (z. B. NSO L3 HA).

6.4 Framework für Validierungsprüfungen (validation_checks/)

Das Framework bietet eine hierarchische Richtlinienkontrolle (globale → app → stage → individuelle Prüfung), automatische Erkennung von Prüfungen, verbesserte Berichterstellung in strukturierten Protokollen und optionale JIRA-Integration. Anwender führen vor oder nach der Bereitstellung Phasen mit derselben Spezifikation aus, die für die Installation verwendet wird, und richten die Automatisierung auf Qualitätsgates aus, die für die Änderungsdisziplin des Unternehmens geeignet sind.

Indikative Skalierung auf einer typischen Außenstelle: in der Größenordnung von dreißig Prüfungen bei BPA und NSO (Zählungen müssen mit "Listen-Validierung-Checkout" bestätigt werden).

6.5 Vault Secrets Manager (scripts/vault_secrets_manager.py)

Ableitung erforderlicher Vault-Variablen von Spezifikationen, Aufforderung oder Annahme von Passwörtern gemäß Richtlinie und Ausgabe von verschlüsseltem group_vars/all_secrets.yaml sowie einer Vault-Passwortdatei für Ansible - dadurch wird die Ad-hoc-Geheimeinbettung in strategischen Büchern reduziert.

---

7. Bereitstellungsmuster und Laufzeiten

Muster	Zusammenfassung
Nativ (AlmaLinux/RHEL)	PYTHONPATH und ANSIBLE_CONFIG festlegen; Ausführung von Packaging, Vault, Validierung, Orchestrierung und strategischen Leitfäden pro Produktleitfaden
Docker-basiertes Installationsprogramm	scripts/setup_installer.sh und scripts/start_installer.sh mit Host-Mounts für große Artefakte;
Luftspalt	Verpackung auf einem angeschlossenen Rechner; Transfer-Paket; Extrakt am Ziel; Installation mit - Luftspalt
Erstellung von MacOS-Paketen	Verwenden Sie python3 ./setup_cxinstaller_prereqs.py auf Mac, um Pakete vorzubereiten. bleibt Linux-orientiert pro Projektdokumentation

---

8. Sicherheit, Validierung und Beobachtbarkeit

8.1 Sicherheitsstatus (Designabsicht)

- Geheimnisse: Ansible Vault-verschlüsselte Gruppenvariablen bevorzugen; verwenden Sie ggf. den strikten Modus des Vault Managers.
- Installationstechniker-Host: Behandlung als stark vertrauenswürdige Kontrollebene; den Zugriff einzuschränken und zu überwachen.
- Artefakte:Schutz der Übernahmekanäle; Unternehmensprozesse können das verify-Bundle durch kryptografische Verifizierung ergänzen.
- Protokollierung:Anwendungs- und Ansible-Protokolle nicht ausreichend bereitgestellt/protokolliert/

8.2 Validierung als operatives Gate

Beispiel für einen Aufruf vor der Bereitstellung:

```
cd /opt/cx-installer
python3 validation_checks/run_validation_checks.py -t pre -s specification/your_spec.yaml
```

Mit optionalen Flags:

- -p <policy_file>— Verwenden Sie eine benutzerdefinierte Validierungsrichtlinie (standardmäßig validation_checks/validation_policies/default.yaml).
- -a <App> — Beschränkung der Prüfungen auf eine bestimmte App (Kleinbuchstaben, z. B. cnc, nso, bpa, cdg)
- —report-file <Pfad> — Schreiben eines eigenständigen JSON-Vorabprüfungsberichts

8.3 Freigabedisziplin

Hierbei handelt es sich um ein Modell für die interne Beschaffung, bei dem bei einer größeren Anzahl von CISCO-Anwendungsinstallationen das gleiche Prinzip durch das Gabeln des Repositorys befolgt werden kann.

---

9. Vorteile und Ergebnisse

Thema	Ergebnis
Zeit und Mühe	Weniger manuelle Schritte Fehler wurden in der Phase zur Überprüfung des Pakets und der Validierung entdeckt und nicht erst spät in den Installationsprogrammen von Ansible oder Produkten
Konsistenz	Gemeinsame Schemas, Rollen und Artefaktlayouts für verschiedene Projekte reduzieren "Schneeflocke"-Unterschiede
Getrennte Vorgänge	Der dokumentierte Pakettransfer unterstützt regulierte Netzwerke ohne Runtime-Downloads.
Governance	Strukturierte Validierungsberichte und optionale JIRA-Hooks unterstützen Änderungsdatensätze und Folgemaßnahmen
Erweiterbarkeit	Klare Erweiterungspunkte: ARTIFACT_DEFS, Handler, neue Rollen/Playbooks, Intent Schemas

Quantitative Kennzahlen (Installationsdauer, Fehlerquoten) sind organisationsspezifisch; Teams müssen einen Vergleich mit älteren Runbooks in vergleichbaren Topologien erstellen.

---

10. Erweiterbarkeit und Wartung

10.1 Hinzufügen von Artefakttypen

1. ExtendARTIFACT_DEFS(und ggf. Labels) incx_deploy_orchestrator.py.
2. Fügen Sie einen benutzerdefinierten Handler hinzu, wenn die Benennung nicht durch Muster allein erfasst werden kann.
3. Aktualisieren Sie die Paketlogik insetup_cxinstaller_prereqs, wenn Downloads automatisiert werden.

10.2 Hinzufügen von ansible Verhalten

Neue Aufgaben in zusammenhängenden Rollen bevorzugen; neue Rollen einführen, wenn die Grenzen klar sind. Leitende Leitfäden viaimport_playbookoder dokumentierte Leitfäden für den Eintrag. Behalten Sie die sicheren Standardeinstellungen in group_vars / vars bei.

10.3 Entwicklung absichtlicher Generatoren

YAML-Schemas unter Intent-Generator/Schema/ und Chatbot-Eingaben aktualisieren; sicherstellen, dass die generierten Dateien den Dateinamen entsprechen, die von APP_CONFIG erwartet werden.

10.4 Überlegungen zur Roadmap (anschaulich)

- Detailliertere Integration von SBOM oder Bildsignaturverifizierung.
- Erweiterte Validierung für CNC/CDG-Szenarien.
- CI-Verweise für Spec Linting und Ansible-Syntaxprüfungen.

---

11. Schlussfolgerung

Der CX Programmable Installer kombiniert deklarative Spezifikationen, eine Python-Kontrollebene für die Verpackung und Verifizierung und eine Ansible-Automatisierungsebene für skalierbare, wiederholbare Bereitstellungen von Crosswork-bezogenen Produkten über verschiedene Infrastruktur- und Verbindungsmodelle hinweg. Die Architektur trennt Absichten absichtlich von der Ausführung, setzt bei Bedarf datengesteuerte Artefakterwartungen um und integriert Validierungs- und Vault-Workflows, die für die unternehmensweite Bereitstellung geeignet sind. Die vollständigen Anhänge mit dem strategischen Leitfaden, Fehlerbehebungsmatrizen, Konnektivitätsmatrizen und erweiterte Befehlsreferenzen finden Sie im internen Whitepaper.

Referenz- und Dokumentationsübersicht

Dokument	Pfad
Bedienungsanleitung	README.md
Strategischen Leitfaden veröffentlichen	RELEASE_GUIDE.md
Interne Architekturanhänge	docs/CX_INSTALLER_TECHNICAL_WHITE_PAPER_INTERNAL.md
Docker (online/Luftspalt/Nutzung)	SETUP_ONLINE_DOCKER.md, SETUP_AIRGAPPED_DOCKER.md, USAGE_DOCKER.md
Validierungs-Framework	docs/validation_checks/README.md
Vault-Manager	docs/scripts/VAULT_SECRETS_MANAGER.md
Produktleitfäden	docs/nso.md, docs/bpa.md, docs/CNC_VCENTER_DEPLOYMENT_GUIDE.md, docs/CNC_OCP_DEPLOYMENT_GUIDE.md, docs/CNC_NSO_DEPLOYMENT_GUIDE.md
Absichtsgenerator	intent-generator/README.md
Übersicht über Chatbot/Regelablauf	docs/HowItWorks.md

Anhang A — Repository-Layout (Zusammenfassung)

```
cx-installer/
├── ansible_playbooks/     # ansible.cfg, files/, group_vars/, playbooks/, roles/, vars/
├── apps/                  # App-specific supporting content
├── deploy/                # Python deploy helpers, logging utilities
├── docs/                  # Technical documentation
├── intent-generator/      # Chatbot, rule engine, schemas, output/
├── scripts/               # Docker setup/start, vault_secrets_manager.py, ...
├── specification/         # User specs, samples, common fragments
├── validation_checks/     # Policies, runners, reports
├── cx_deploy_orchestrator.py
├── setup_cxinstaller_prereqs*
├── requirements.txt
└── README.md
```

Brennpunkte nach der Einrichtung (typisch):ansible_playbooks/files/artifacts/, files/bin/, files/charts/, files/images/.

Anhang B - Orchestrator-CLI (Zusammenfassung)

Skript:cx_deploy_orchestrator.py

Argument	Beschreibung
—app / -a	nso | cross-worksuite | bpa
—spec / -s	Pfad zur YAML-Spezifikation
—step	Generierungsabsicht | Verifizierungspaket | Installation
—verify-only	Paket überprüfen; Beenden von ungleich null, wenn nicht bereit
—Trockenlauf	Trockenlauf, sofern unterstützt
—list-specs	Notieren bekannter Spezifikationen

Umgebung (typische Sitzung):

```
export PYTHONPATH=$(pwd)
export ANSIBLE_CONFIG=$(pwd)/ansible_playbooks/ansible.cfg
```

Anhang C — Glossar

Begriff	Definition
Spez	YAML-Benutzerspezifikation: Plattformen, Anwendungen, Topologie, Pfade, Berechtigungen
Absicht	Normalisierte YAML aus dem Absichtsgenerator oder handgeschriebenem Äquivalent
Paket	Verpackter Installationsbaum (oft Tarball) für Luftspaltübertragung
Orchestrator	cx_deploy_orchestrator.py — Überprüfen / Vorsatz / Installationskoordination
Artefaktüberprüfung	Dateisystemprüfungen, ob erforderliche Binärdateien/Images pro Spezifikation vorhanden sind
Tresor	Ansible Vault-verschlüsselte variable Datei für Geheimnisse
BEDARF	Netzwerkelement-Treiberpaket (NSO)
CFS/RFS	Topologiekonzepte für NSO-Cluster-Forwarder/redundante Forwarder
Luftspalt	Umgebung ohne Installationszugriff auf Paket-Download-Endpunkte

---

Dokument-Revisionsverlauf

Version	Datum	Hinweise
1.0	2026-03-27	Erstes veröffentlichungsfähiges technisches Whitepaper (Programmable Installer Framing)

Informationen zu dieser Übersetzung

Cisco hat dieses Dokument maschinell übersetzen und von einem menschlichen Übersetzer editieren und korrigieren lassen, um unseren Benutzern auf der ganzen Welt Support-Inhalte in ihrer eigenen Sprache zu bieten. Bitte beachten Sie, dass selbst die beste maschinelle Übersetzung nicht so genau ist wie eine von einem professionellen Übersetzer angefertigte. Cisco Systems, Inc. übernimmt keine Haftung für die Richtigkeit dieser Übersetzungen und empfiehlt, immer das englische Originaldokument (siehe bereitgestellter Link) heranzuziehen.