プログラマブルインストーラ

内容

プログラマブルインストーラ

概要

本書の読み方

1. 価値提案

    1.1配送に関する問題

    1.2プログラマブルインストーラのアプローチ

    1.3この文脈での「プログラム可能」の意味

2. システムコンテキスト

    2.1関係主体と環境

    2.2信頼境界

3. アーキテクチャの原則

4. 論理アーキテクチャ

    4.1層

    4.2エンドツーエンドのデータフロー

    4.3サポート対象製品（インストーラの範囲）

5. 仕様とインテント・モデル

    5.1ユーザ仕様

    5.2一般的なフラグメント

    5.3インテント生成

6. 実装の詳細

    6.1導入オーケストレータ(cx_deploy_orchestrator.py)

    6.2前提条件およびパッケージングツール(setup_cxinstaller_prereqs)

    6.3 Ansibleオートメーションプレーン

    6.4検証チェックフレームワーク(validation_checks/)

    6.5 Vault Secrets Manager(scripts/vault_secrets_manager.py)

7. 導入パターンとランタイム

8. セキュリティ、検証、可観測性

    8.1セキュリティポスチャ（設計意図）

    8.2運用ゲートとしての検証

    8.3リリースの規律

9. メリットと成果

10. 拡張性とメンテナンス

    10.1加工品タイプの追加

    10.2 Ansibleの動作の追加

    10.3インテントジェネレータの進化

    10.4ロードマップの考慮事項（説明用）

11. 終わりに

参考資料とドキュメントのマップ

付録A：リポジトリのレイアウト：https://wwwin-github.cisco.com/CX-SAO-TOOLS/cx-installer（まとめ）

付録B：オーケストレータのCLI（概要）

付録C：用語集

ドキュメントの変更履歴

プログラマブルインストーラ

フィールド	値
製品	プログラマブルインストーラ
ドキュメントタイプ	テクニカルホワイトペーパー：アーキテクチャ、実装、成果
主な対象者	ソリューションアーキテクト、プラットフォームエンジニア、DevOps/SRE、デリバリリーダー
二次対象者	エンジニアリング管理、セキュリティレビューア、プログラムマネージャ

概要

Programmable Installerは、Ciscoソフトウェアスタック(Network Services Orchestrator(NSO)、Crosswork Network Controller(CNC)、Crosswork Data Gateway(CDG)、Business Process Automation(BPA)など)を、Enterprise Linux（RHELファミリ）および該当する関連インフラストラクチャ（VMware vCenter、OpenShift、KVM、エアギャップ対応のKubernetes）に導入して運用するための仕様主導型の自動化プラットフォームです。 システムは、宣言型インテント（YAML仕様およびオプションのガイド付きインテント生成）を実行（Ansibleロールおよびプレイブック）から分離し、アーティファクトのパッケージ化、長時間実行するインストール前のバンドルの検証、暗号化されたシークレットの準備、検証ゲートのオーケストレーションを行うPythoncontrolプランを提供します。

このホワイトペーパーでは、アーキテクチャ層、主要なデータフロー、実装パターン（ハイブリッドのデータ駆動型アーティファクト検証モデルを含む）、導入モード（ネイティブ、コンテナ化、オンライン、エアギャップ）、および検証とロギングのフレームワークについて説明します。デリバリおよびプラットフォーム組織にとって、このプラットフォームの目的は、手動による作業を減らし、設定ミスやバイナリの不足を早期に解消し、環境固有のパラメータ設定を維持しながら、製品とトポロジ全体で自動化を標準化することです。

キーワード：インフラストラクチャ自動化、宣言型導入、Ansible、YAML仕様、エアギャップパッケージ、アーティファクト検証、Crosswork、NSO、CNC、CDG、BPA、検証ポリシー、DevOps。

本書の読み方

ロール	推奨されるフォーカス
意思決定者/リーダー	概要、§1価値提案、§8メリットとリスクポスチャ、§10まとめ
ソリューションアーキテクト	§3-§6（アーキテクチャ、仕様モデル、実装、導入パターン）
DevOps/SRE/デリバリエンジニア	§5-§7、付録B、コンパニオン内部ホワイトペーパー付属書
セキュリティ確認者	§7セキュリティとコンプライアンスポスチャ、§3.2の信頼境界

---

1. 価値提案

1.1配送に関する問題

企業による複数階層のネットワーク自動化およびオーケストレーション製品のインストールは、従来はハイタッチで行われていました。長いランブック、多くの手動ステップ、サイト間のバージョンのずれ、プロセスに侵入する表面的な障害（NEDの欠落、誤ったOVAパス、不完全なエアギャップイメージセット）です。 このパターンでは、コストが増加し、変更ウィンドウが長くなり、監査が困難になります。

1.2プログラマブルインストーラのアプローチ

プログラマブルインストーラは、インストールを、トポロジ、バージョン、プラットフォームの選択（vCenterとOpenShiftとVanilla VM）、ファイルパス、および権限の特定によってパラメータ化されたプログラム形式で扱います。自動化可能な場合はisidempotentwhere、お客様を越えて繰り返し実行でき、「not ready」がクラスタまたは製品のインストール前に迅速かつ明確な結果になるように事前ロードとチェックが行われます。

1.3この文脈での「プログラム可能」の意味

- 宣言型：演算子で、導入する内容を説明します。プレイブックの実装手順を参照してください。
- データ駆動型の検証：パターンが安定している場合、リリースごとのアドホックスクリプトではなく、テーブルとルールから予想されるアーティファクトが生成されます。
- ポリシーに基づく品質：階層型ポリシーに基づいて、導入前および導入後の検証が実行され、構造化されたレポートとオプションのチケット作成機能が統合されます。
- マルチモーダル操作：初めてのユーザ用のインタラクティブメニュー、CI/CDスタイルの反復用のCLIおよびフローズンバイナリ、標準化されたランタイムイメージ用のDockerベースのフロー

---

2. システムコンテキスト

2.1関係主体と環境

アクター/システム	ロール
デリバリーエンジニア	仕様書の作成または生成、パッケージの実行、Vaultの準備、検証、オーケストレータ、およびAnsible
インストーラホスト	Linuxコントロールノード（ネイティブまたはコンテナ）、Python、Ansible構成、アーティファクト用ディスク
ターゲットインフラストラクチャ	仕様ごとのvCenter、OpenShift/KubeVirt、またはvanilla VM
加工品のソース	内部ミラー、権限付与レイアウト、ソフトウェア配布：環境固有
ダウンストリームシステム	モニタリング、変更管理、オプションのJIRAワークフロー

2.2信頼境界

1. 仕様は意図を表します。これらはパスと非シークレットパラメータを参照します。秘密は、必要に応じてプレーンテキスト仕様のフィールドではなく、Ansible Vaultworkflowsを通過する必要があります。
2. アーティファクト・ストレージは整合性が保護されている必要があります。検証では、仕様に基づくプレゼンスとnamingalignmentに重点を置き、ポリシーで必要とされる場合には組織の管理（チェックサム、署名付きバンドル）を拡張します。
3. Installer-to-target SSHは特権の高いパスであり、インストーラホストの侵害は大きな影響を与えます。強化とアクセス制御は運用上の前提条件です。

---

3. アーキテクチャの原則

1. 最初に宣言型：YAMLのユーザーの意図。自動化により一貫して解釈されます。
2. 計画と実行の分離：Pythonはゲートを計画、検証、調整します。Ansibleはインフラストラクチャと製品ステップを実行します。
3. 構成可能な自動化：サイトレベルのプレイブックは、焦点を絞ったプレイブック（プレインストール、Kubernetesトラック、製品インストール）をインポートします。
4. 先進的な情報開示：オンボーディングのためのインタラクティブなセットアップ、高度な自動化のためのフラグとスクリプト。
5. 同じコードベース、複数のランタイム：ネイティブのAlmaLinux/RHELパスとDockerベースのパスは、1つのリポジトリレイアウトを共有します。
6. 明示的なエアギャップサポート：接続されたマシン上のパッケージ、バンドルの転送、前提条件のインストール、パブリックネットワークへの実行時の依存のない展開。

---

4. 論理アーキテクチャ

4.1層

レイヤ	担当
仕様	トポロジ、バージョン、プラットフォーム、パス、資格
コントロール プレーン	パッケージ、バンドル検証、Vaultヘルパー、検証ドライバ、オーケストレータCLI
自動化プレーン	ホストの準備、Kubernetesのライフサイクル、製品のインストール、および初期構成
成果物	バイナリ、イメージ、チャート、OVA、tarballs

4.2エンドツーエンドのデータフロー

1. パッケージフロー：前提条件のツールは、ファイルを特定の場所にダウンロードまたはステージングし、オプションでオフラインインストール用に転送可能なtarballを作成します。
2. フローの確認：オーケストレータが仕様を解析し、予期されるアーティファクト（プラットフォームフィルタとエンタイトルメントリストを含む）を解決し、インストールの前に準備完了状態または確認不足を報告します。
3. 導入フロー：SpecとVault（およびオプションの事前検証）により、Ansibleプレイブックをエンゲージメントから導出されたインベントリと比較して活用できます。

4.3サポート対象製品（インストーラの範囲）

製品/バンドル
nso
CNC
CDG
BPA
CNC + NSO

---

5. 仕様とインテント・モデル

5.1ユーザ仕様

仕様は、プラットフォーム（vCenter、OCP、VM、KVMなど）、ホスト、アプリケーションとバ

ージョン、トポロジ（NSO CFS/RFSレイアウトなど）、エンタイトルメント（NEDおよびアドオンパッケージ）、およびファイルパス（OVA、qcow2イメージ、アプリケーション層のボールなど）を説明するYAMLドキュメントです。

5.2一般的なフラグメント

特定のアプリケーションuser_specは、CNC/CDGのデフォルトとパスフォールバックを提供します。オーケストレータのパーサーは、ユーザの仕様を真実のソースとして扱い、ユーザのキーが存在しないときに共通の仕様エントリを使用します。

5.3インテント生成

インテント・ジェネレータは、一連の質問表、ルール・エンジン(日付駆動型ロジック)、およびintent.yamlへのスキーマバックマッピング。

---

6. 実装の詳細

6.1導入オーケストレータ(`cx_deploy_orchestrator.py`)

オーケストレータは、スクリプトまたはinteractivegenerate-intent、verify-bundle、およびinstallcoordinationの単一のエントリです。この設計は明確にハイブリッドです。

- **`ARTIFACT_DEFS`**:アプリケーションごとにアーティファクトのタイプと命名パターンを宣言します（NSOインストーラ、NED署名付きパッケージ、オプションパッケージ、CNC OVA/qcow2/tier tarballs、CDGイメージ、BPAチャート、エアギャップイメージtarballs）。
- **`APP_CONFIG`**:CLI:appvalues(`nso`、`crossworksuite`、`bpa`)を仕様のフォルダ名とデフォルトのインテントファイル名にマッピングします。
- パーサ/ハンドラ:ユーザ仕様および共通仕様を使用してCNC/CDGパスを解決します。カスタムハンドラ**TSDN/DLM**などの不統一な命名や、チャートおよび**tarball**パスの**BPA**バージョン形式をカバーします。

Readiness:AReportaggregates discovered artifacts;`is`_readyis trueは、仕様の解析後に必要なファイルが欠落していない場合に使用します。このモジュールは、PyInstallerでフリーズされたバイナリの`viasys.frozenpath`解決をサポートしています。

6.2前提条件およびパッケージングツール(`setup_cxinstaller_prereqs`)

このコンポーネントは、アーティファクトパッケージングとホスト前提条件インストール（オン

ライン、エアギャップ、自動検出）のためのインタラクティブメニューとCLI モデル、combinedCNC_NSOを含むマルチアプリケーションパッケージングを提供します。Ansibleおよびverify-bundleロジックで期待されるアーティファクトツリーを生成します。

6.3 Ansibleオートメーションプレーン

構成：これはスタックのマルチトラック性を示しています。異なるKubernetesのブートストラップパスをインポートします。これは、異なる製品が異なるKubernetesのブートストラップパスをターゲットにしていることを反映しています。

ロール（代表ファミリ）:preinstall（SELinux，ファイアウォール， SSH，レジストリヘルパー）,k8s_install/rke2,deploy_nso,deploy_cnc,deploy_cdg,deploy_bpa,postinstall，アンインストールおよび証明書更新ヘルパー。所有権とテストは最適に管理されたアトロールバウンダリです。ネストされたタスクフォルダはサブワークフローを実装します（NSO L3 HAなど）。

6.4検証チェックフレームワーク(`validation_checks/`)

このフレームワークは、階層型ポリシー制御(global → app → stage → individual check)、チェックの自動検出、構造化されたログへの拡張されたレポート、およびオプションのJIRA統合を提供します。オペレータは導入前または導入後のフェーズをインストールに使用したものと同じ仕様に基づいて実行し、自動化を企業変更の規律に適した品質のゲートウェイに合わせます。

一般的な支店の指標スケール：BPAとNSO全体の30チェックの順序で(カウントはチェックアウトのlist-validation-checksonで確認する必要があります)。

6.5 Vault Secrets Manager(`scripts/vault_secrets_manager.py`)

仕様から必要なVault変数を導出し、ポリシーに基づいてパスワードのプロンプトや受け入れを行い、emitsencryptedgroup_vars/all_secrets.yamlplusにAnsibleのVaultパスワードファイルを追加することで、プレイブックへのアドホックな秘密の埋め込みを削減します。

---

7. 導入パターンとランタイム

パターン	要約
ネイティブ (AlmaLinux/RHEL)	SetPYTHONPATHandANSIBLE_CONFIG：製品ガイドごとにパッケージング、ヴォールト、検証、オーケストレータ、プレイブックを実行
Dockerベースのインストーラ	scripts/setup_installer.shandscripts/start_installer.shwith大規模なアーティファクト用のホストマウント
エアギャップ	接続されたマシン上のパッケージ、バンドルの転送、ターゲットへの抽出、

パターン	要約
	インストール–airgap
macOSバンドルの作成	Usepython3 ./setup_cxinstaller_prereqs.pyon Macを使用してバンドルを準備します。ターゲットの配備は、プロジェクト文書ごとにLinux指向のままです

---

8. セキュリティ、検証、可観測性

8.1セキュリティポスチャ（設計意図）

- シークレット：Ansible Vaultで暗号化されたグループ変数を使用します。必要に応じてVault Managerのストリクトモードを使用します。
- インストーラホスト：高信頼制御プレーンとして扱い、アクセスを制限し、監視します。
- アーティファクト：取得チャネルの保護。組織のプロセスにより、検証バンドルと暗号化検証が強化される可能性があります。
- ロギング：Application and Ansible logs `underdeploy/logs/`

8.2運用ゲートとしての検証

配置前の呼び出しの例：

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

オプションのフラグ付き：

- `-p <policy_file>` – カスタムの検証ポリシーを使用します(デフォルトは `validation_checks/validation_policies/`default.yaml)。
- `-a <app>` – チェック対象を特定のアプリケーションに限定します(小文字、例：`cnc`、`nso`、`bpa`、`cdg`)。
- `–report-file <path>` – スタンドアロンJSONプレチェックレポートを作成します。

8.3リリースの規律

これは内部ソーシングのモデルであり、より多くのシスコアプリケーションをインストールする場合は、同じ原則に従ってリポジトリを分岐できます。

---

9. メリットと成果

テーマ	成果
時間と労力	手動の手順が少なくて済む。Ansibleまたは製品インストーラで遅れるのではなく、検証バンドルおよび検証フェーズで障害が検出される
一貫性	契約間で共有されるスキーマ、ロール、およびアーティファクトのレイアウトにより、「スノーフレーク」の違いが軽減されます
切断された操作	文書化されたバンドル転送は、ランタイムダウンロードなしで規制されたネットワークをサポートします
ガバナンス	構造化検証レポートとオプションのJIRAフックにより、変更記録とフォローアップをサポート
拡張性	拡張ポイントのクリア：`ARTIFACT_DEFS`、ハンドラ、新しいロール/プレイブック、ダイナミックスキーマ

定量的なメトリック(インストール期間、不具合率)は組織に固有です。比較可能なトポロジの従来のランブックを基準にする必要があります。

---

10. 拡張性とメンテナンス

10.1加工品タイプの追加

1. `ExtendARTIFACT_DEFS`（および必要に応じてラベル） `incx_deploy_orchestrator.py`。
2. パターンだけでは名前を取得できない場合に、精度ハンドラを追加します。
3. ダウンロードが自動化されると、パッケージロジック`insetup_cxinstaller_`prereqswhenが更新されます。

10.2 Ansibleの動作の追加

統合的な役割の内部の新しいタスクを好む。境界が明確な場合は新しいタスクを導入する。Wireプレイブック`viaimport_playbookor`ドキュメント化されたエントリプレイブックKeepsafe `defaultsingroup_vars/vars`コマンドを発行します。

10.3インテントジェネレータの進化

YAMLスキーマ`underintent-generator/schema/`およびchatbot入力を更新します。生成されたファイルが`APP_CONFIG`で予期されるファイル名と一致することを確認します。

10.4ロードマップの考慮事項（説明用）

- DeeperSBOMまたは画像署名検証の統合。
- CNC/CDGシナリオの検証範囲の拡大
- スペックリンティングおよびAnsible構文チェックのCIリファレンス。

---

11. 終わりに

CX Programmable Installer は、宣言型の仕様、パッケージングと検証のためのaPython制御計画、および多様なインフラストラクチャと接続モデルにわたるCrosswork関連製品のスケーラブルで反復可能な導入のためのanSible自動化計画を組み合わせています。このアーキテクチャは、意図的に意図された実行から分離し、実用的な場合はデータ駆動型アーティファクトの期待を適用し、企業配信に適した埋め込み検証と検証ワークフローを提供します。運用に関する付属書（プレイブックテーブル、トラブルシューティングマトリクス、接続マトリクス、拡張コマンドリファレンス）については、このアドバイザリに関連する社内用ホワイトペーパーを参照してください。

参考資料とドキュメントのマップ

ドキュメント	パス
オペレータガイド	README.md
リリースプレイブック	リリース_ガイド.md
内部アーキテクチャ付属書	docs/CX_INSTALLER_TECHNICAL_WHITE_PAPER_INTERNAL.md
Docker（オンライン/エアギャップ/使用状況）	SETUP_ONLINE_DOCKER.md,SETUP_AIRGAPPED_DOCKER.md,USAGE_DOCKER.md
検証フレームワーク	docs/validation_checks/README.md
Vaultマネージャ	docs/scripts/VAULT_SECRETS_MANAGER.md
製品ガイド	docs/nso.md、docs/bpa.md、docs/CNC_VCENTER_DEPLOYMENT_GUIDE.md、docs/CNC_OCP_DEPLOYMENT_GUIDE.md、docs/CNC_NSO_DEPLOYMENT_GUIDE.md
インテントジェネレータ	intent-generator/README.md
チャットボット/ルールフローの概要	docs/HowItWorks.md

付録A：リポジトリレイアウト – https://wwwin-github.cisco.com/CX-SAO-TOOLS/cx-installer（概要）

```
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
```

セットアップ後のアーティファクトの焦点（標準）:ansible_playbooks/files/artifacts/、files/bin/、files/charts/、files/images/。

付録B：オーケストレータのCLI（概要）

スクリプト：cx_deploy_orchestrator.py

引数	説明
–app/-a	nso|crossworksuite|bpa
–spec/-s	YAML仕様へのパス
--手順	generate-intent|verify-bundle|インストール
– 確認のみ	バンドルを確認します。準備ができていない場合はゼロ以外で終了します。
– ドライ作動	ドライ作動（サポートされている場合）
–list-specs	既知の仕様の一覧表示

環境（標準セッション）:

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

付録C：用語集

用語	定義
仕様	YAMLユーザー仕様：プラットフォーム、アプリケーション、トポロジ、パス、資格
意図	インテントジェネレータまたは手書きの同等のツールから正規化されたYAML
バンドル	パッケージ化されたインストーラツリー（通常はtarball）によるエアギャップ転送
オーケストレータ	`cx_deploy_orchestrator.py`：調整の確認/意図/インストール
成果物の検証	必要なバイナリ/イメージがスペックごとに存在するファイルシステムのチェック
Vault	シークレット用のVaultで暗号化された変数ファイル
NED	ネットワーク要素ドライバパッケージ(NSO)
CFS/RFS	NSOクラスタフォワーダ/冗長フォワーダトポロジの概念
エアギャップ	パッケージのダウンロードエンドポイントへのインストーラ時アクセスがない環境

ドキュメントの変更履歴

バージョン	日付	注意事項
1.0	2026-03-27	初期発行準備テクニカルホワイトペーパー（プログラム可能なインストーラのフレーミング）

翻訳について
シスコは世界中のユーザにそれぞれの言語でサポート コンテンツを提供するために、機械と人による翻訳を組み合わせて、本ドキュメントを翻訳しています。ただし、最高度の機械翻訳であっても、専門家による翻訳のような正確性は確保されません。シスコは、これら翻訳の正確性について法的責任を負いません。原典である英語版（リンクからアクセス可能）もあわせて参照することを推奨します。