Utiliser le programme d'installation programmable

Table des matières

Introduction

Installateur Programmable

Résumé

Comment lire ce document ?

1. Proposition de valeur

    1.1 Problème de livraison

    1.2 Approche de l'installateur programmable

    1.3 Signification de « programmable » dans ce contexte

2. Contexte système

    2.1 Acteurs et environnements

    2.2 Limites de confiance

3. Principes architecturaux

4. Architecture logique

    4.1 Couches

    4.2 Flux de données de bout en bout

    4.3 Produits pris en charge (étendue de l'installateur)

5. Spécification et modèle d'intention

    5.1 Spécifications utilisateur

    5.2 Fragments courants

    5.3 Génération d'intentions

6. Mise en oeuvre

    6.1 Orchestrator de déploiement (cx_deploy_orchestrator.py)

    6.2 Conditions préalables et outils d'emballage (setup_cxinstaller_prereqs)

    6.3 Plan d'automatisation analytique

    6.4 Cadre des contrôles de validation (validation_controls/)

    6.5 Gestionnaire des secrets de coffre-fort (scripts/vault_secrets_manager.py)

7. Modèles de déploiement et durées d'exécution

8. Sécurité, validation et observabilité

    8.1 Posture de sécurité (intention de conception)

    8.2 Validation en tant que porte opérationnelle

    8.3 Discipline relative à la libération

9. Avantages et résultats

10. Extensibilité et entretien

    10.1 Ajout de types d'artefacts

    10.2 Ajout d'un comportement répréhensible

    10.3 Évolution du générateur d'intentions

10.4 Considérations relatives à la feuille de route (illustration)

11. Conclusion

Références et carte de documentation

Annexe A — Présentation du référentiel (résumé)

Annexe B - CLI d'Orchestrator (récapitulatif)

Annexe C — Glossaire

Historique des révisions du document

Introduction

Ce document décrit le programme d'installation programmable, une plate-forme d'automatisation pilotée par des spécifications pour le déploiement de piles logicielles Cisco telles que NSO et CNC.

Installateur Programmable

Champ	Valeur
Product (produit)	Installateur Programmable
Type de document	Livre blanc technique : architecture, mise en oeuvre et résultats
Public principal	Architectes de solutions, ingénieurs de plate-forme, DevOps / SRE, responsables de prestation
Public secondaire	Gestion de l'ingénierie, Réviseurs de sécurité, Responsables de programme

Résumé

Le programme d'installation programmable est une plate-forme d'automatisation pilotée par des spécifications pour le déploiement et l'exploitation de piles logicielles Cisco, y compris Network Services Orchestrator (NSO), Crosswork Network Controller (CNC), Crosswork Data Gateway (CDG) et Business Process Automation (BPA), sur Linux d'entreprise (gamme RHEL) et l'infrastructure associée le cas échéant (VMware vCenter, OpenShift, KVM, Kubernetes à entrefer). Le système sépare l'intention déclarative (spécifications YAML et génération facultative d'intention guidée) de l'exécution (rôles et guides Ansible), avec un plan de contrôle Python qui empaquette les artefacts, vérifie les paquets avant les installations à long terme, prépare les secrets chiffrés et orchestre les portes de validation.

Ce livre blanc explique les couches architecturales, les flux de données principaux, les modèles d'implémentation (y compris un modèle hybride de vérification d'artefact piloté par les données), les modes de déploiement (natif, en conteneur, en ligne, à vide) et les cadres de validation et de journalisation. Pour les organisations de livraison et de plate-forme, la plate-forme vise à réduire

les tâches manuelles, la mauvaise configuration des surfaces et les binaires manquants au début, et à standardiser l'automatisation sur les produits et les topologies tout en préservant le paramétrage spécifique à l'environnement.

Mots-clés : automatisation de l'infrastructure, déploiement déclaratif, Ansible, spécification YAML, conditionnement de l'air-gap, vérification des artefacts, Crosswork, NSO, CNC, CDG, BPA, politique de validation, DevOps.

Comment lire ce document ?

Rôle	Thème suggéré
Décideurs/responsables	Résumé ; §1 Proposition de valeur ; §8 Avantages et position de risque ; §10 Conclusion
Architectes de solutions	§3-§6 (architecture, modèle de spécification, implémentation, modèles de déploiement)
DevOps / SRE / Ingénieurs de livraison	§5-§7 ; Appendice B ; annexes du livre blanc interne
Réviseurs de sécurité	§7 Position en matière de sécurité et de conformité ; limites de confiance dans §3.2

---

1. Proposition de valeur

1.1 Problème de livraison

L'installation en entreprise de produits d'orchestration et d'automatisation de réseau multiniveau est traditionnellement très exigeante : de longs runbooks, de nombreuses étapes manuelles, une différence de version entre les sites et des pannes qui surviennent pendant des heures dans un processus (NED manquants, chemins OVA incorrects, ensembles d'images d'entrefer incomplets). Ce modèle augmente les coûts, allonge les fenêtres de modification et rend les audits plus difficiles.

1.2 Approche de l'installateur programmable

L'installateur programmable traite une installation comme un programme paramétré par une spécification : topologie, versions, choix de plate-forme (vCenter ou OpenShift ou machines virtuelles classiques), chemins d'accès aux fichiers et droits. Dans la mesure du possible, l'automatisation est omnipotente, reproductible chez les clients et soumise à des contrôles préalables, de sorte que le terme « non prêt » soit un résultat rapide et explicite avant l'installation

du cluster ou du produit.

1.3 Signification de « programmable » dans ce contexte

- Déclaratif : les opérateurs décrivent ce qui doit être déployé ; les guides implémentent comment.
- Vérification basée sur les données : les artefacts attendus sont dérivés de tables et de règles plutôt que de scripts ad hoc par version, lorsque les modèles sont stables.
- Qualité liée aux politiques : la validation avant et après le déploiement s'exécute sous des politiques hiérarchiques, avec des rapports structurés et l'intégration facultative de tickets.
- Fonctionnement multimodal : menus interactifs pour les nouveaux utilisateurs ; CLI et binaires figés pour la répétition de type CI/CD ; Flux Docker pour les images d'exécution standardisées.

---

2. Contexte système

2.1 Acteurs et environnements

Acteur/système	Rôle
Ingénieur de livraison	Rédige ou génère des spécifications, exécute le packaging, la préparation du coffre-fort, la validation, l'orchestrateur et Ansible
Hôte du programme	Noeud de contrôle Linux (natif ou conteneur) avec Python, configuration Ansible, disque pour les artefacts
Infrastructure cible	VM vCenter, OpenShift/KubeVirt ou vanilla selon les spécifications
Sources Artifact	Miroirs internes, dispositions de droits, distribution de logiciels - spécifiques à l'environnement
Systèmes en aval	Surveillance, gestion des changements, workflows JIRA en option

2.2 Limites de confiance

1. Les spécifications expriment l'intention; ils peuvent référencer des chemins et des paramètres non secrets. Les secrets doivent transiter par les workflows Ansible Vault et non par les champs de spécification en texte brut où ils sont évitables.
2. Le stockage des artefacts doit être protégé en termes d'intégrité ; la vérification est axée sur l'alignement de la présence et de l'attribution de noms avec les spécifications, et s'étend aux contrôles organisationnels (checksum, bundles signés) lorsque la politique l'exige.
3. Installer-to-target SSH est un chemin à haut privilège ; compromission de l'hôte de l'installateur a un impact important. Le renforcement et le contrôle d'accès sont des préalables opérationnels.

---

3. Principes architecturaux

1. Déclaratif d'abord : intention de l'utilisateur dans YAML ; l'automatisation l'interprète de manière cohérente.
2. Séparation de la planification et de l'exécution : Python planifie, vérifie et orchestre les portes ; Ansible exécute les étapes de l'infrastructure et du produit.
3. Automatisation composable : les playbooks au niveau du site importent des playbooks ciblés (préinstallation, pistes Kubernetes, installations de produits).
4. Divulgation progressive : configuration interactive pour l'intégration ; des indicateurs et des scripts pour une automatisation avancée.
5. Même base de code, plusieurs runtimes : les chemins AlmaLinux/RHEL natifs et les chemins basés sur Docker partagent une disposition de référentiel.
6. Prise en charge explicite de l'entrefer : emballage sur une machine connectée ; transférer une offre groupée ; installer les composants requis et effectuer le déploiement sans dépendre du runtime sur les réseaux publics.

---

4. Architecture logique

4.1 Couches

Couche	Responsabilité
Spécification	Topologie, versions, plates-formes, chemins, droits
Plan de contrôle	Emballage, vérification des offres groupées, assistants de stockage en chambre forte, pilote de validation, CLI d'orchestrateur
Plan Automation	Préparation de l'hôte, cycle de vie Kubernetes, installation du produit et configuration au premier jour
Artefacts	Binaires, images, graphiques, OVA, tarballs

4.2 Flux de données de bout en bout

1. Flux de package : l'outil prérequis télécharge ou transfère les fichiers dans un emplacement spécifique, produisant éventuellement une archive tar transférable pour les installations hors ligne.
2. Vérifier le flux : l'orchestrateur analyse la spécification, résout les artefacts attendus (y compris les filtres de plate-forme et les listes d'autorisation) et génère des rapports prêts/manquants avant l'installation.
3. Flux de déploiement : Spec plus vault (et pré-validation facultative) permet de comparer les guides Ansible aux inventaires issus de l'engagement.

4.3 Produits pris en charge (étendue de l'installateur)

Produit/offre groupée
NSO
CNC
CDG
BPA
CNC + NSO

---

5. Spécification et modèle d'intention

5.1 Spécifications utilisateur

Les spécifications sont des documents YAML décrivant les plates-formes (par exemple vCenter, OCP, VM, KVM), les hôtes, les applications avec des versions, la topologie (par exemple les dispositions NSO CFS/RFS), les droits (NED et modules complémentaires) et les chemins d'accès aux fichiers pour les OVA, les images qcow2 et les archives de couche application.

5.2 Fragments courants

Une application spécifique user_spec fournit des valeurs par défaut et des chemins de secours pour CNC/CDG. L'analyseur syntaxique de l'orchestrateur traite la spécification utilisateur comme une source de vérité et utilise les entrées de spécification communes lorsque les clés utilisateur sont absentes.

5.3 Génération d'intentions

Le générateur d'intention prend en charge la collecte guidée des exigences via un ensemble de questionnaires, un moteur de règles (logique pilotée par la date) et mappage basé sur un schéma vers intent.yaml.

---

6. Mise en oeuvre

6.1 Orchestrator de déploiement (cx_deploy_orchestrator.py)

L'orchestrateur est l'entrée unique pour la coordination de génération d'intention, de vérification d'offre groupée et d'installation scriptée ou interactive. Sa conception est explicitement hybride :

- ARTIFACT_DEFS : déclare les types d'artefacts et les modèles d'attribution de noms par application (programme d'installation NSO, packages signés NED, packages facultatifs ; CNC OVA/qcow2/tier tarballs ; images CDG ; des graphiques BPA et des billes d'image d'entrefer).
- APP_CONFIG : Mappe les valeurs CLI —app (nso, crossworksuite, bpa) aux noms de dossiers de spécification et aux noms de fichiers d'intention par défaut.
- Analyseur / gestionnaires : résolvez les chemins CNC/CDG en utilisant les spécifications utilisateur et les spécifications communes ; les gestionnaires personnalisés couvrent le formatage de noms non uniformes (par exemple TSDN/DLM) et de versions BPA pour les chemins de graphique et d'archive.

Readiness:AReportaggregates a découvert des artefacts ; is_ready est vrai lorsqu'aucun fichier requis n'est manquant après l'analyse des spécifications. Le module prend en charge les binaires figés PyInstaller via la résolution de chemin sys.gelé.

6.2 Conditions préalables et outils d'emballage (setup_cxinstaller_prereqs)

Ce composant fournit des menus interactifs et des modes CLI pour l'empaquetage des artefacts et l'installation préalable de l'hôte : en ligne, à air-gap ou à détection automatique ; emballage multi-applications incluant `combineCNC_NSO`. Il remplit l'arborescence d'artefacts attendue par Ansible et par la logique verify-bundle.

6.3 Plan d'automatisation analytique

- Composition : ceci illustre la nature multipiste de la pile : Elle importe différents chemins d'amorçage Kubernetes, ce qui reflète le fait que différents produits ciblent différents chemins d'amorçage Kubernetes.
- Rôles (familles représentatives) : préinstallation (SELinux, pare-feu, SSH, assistants de registre), k8s_install / rke2, deploy_nso, deploy_cnc, deploy_cdg, deploy_bpa, postinstall, uninstall et certificate renew helpers. La propriété et les tests sont mieux gérés aux limites des rôles ; Les dossiers de tâches imbriquées implémentent des sous-workflows (par exemple NSO L3 HA).

6.4 Cadre des contrôles de validation (validation_controls/)

L'infrastructure fournit un contrôle hiérarchique des politiques (global → app → stage → contrôle individuel), la détection automatique des contrôles, des rapports améliorés vers les journaux structurés et l'intégration JIRA facultative. Les opérateurs exécutent des phases de

prédéploiement ou de post-déploiement selon les mêmes spécifications que celles utilisées pour l'installation, en alignant l'automatisation sur des critères de qualité adaptés à la discipline de changement de l'entreprise.

Échelle indicative sur une succursale type : de l'ordre de trente contrôles sur BPA et NSO (comptes à confirmer avec make list-validation-checkout).

6.5 Gestionnaire des secrets de coffre-fort (scripts/vault_secrets_manager.py)

Dérive les variables de coffre-fort requises des spécifications, invite ou accepte les mots de passe dans le cadre de la stratégie, et émet des group_vars/all_secrets.yaml chiffrés ainsi qu'un fichier de mot de passe de coffre-fort pour Ansible, réduisant ainsi l'intégration de secrets ad hoc dans les guides.

---

7. Modèles de déploiement et durées d'exécution

Modèle	Résumé
Native (AlmaLinux / RHEL)	Définissez PYTHONPATH et ANSIBLE_CONFIG ; exécuter le packaging, le coffre-fort, la validation, l'orchestrateur et les guides par guide produit
Installer basé sur Docker	scripts/setup_installer.sh et scripts/start_installer.sh avec montures d'hôtes pour les artefacts volumineux ;
Espace d'air	Emballage sur une machine connectée ; offre groupée de transfert ; extrait sur la cible ; installer avec—airgap
création d'offres groupées macOS	Utilisez ython3 ./setup_cxinstaller_prereqs.py sur Mac pour préparer des bundles ; Le déploiement cible reste orienté Linux par projet

---

8. Sécurité, validation et observabilité

8.1 Posture de sécurité (intention de conception)

- Secrets : Préférer les variables de groupe chiffrées Ansible Vault ; utilisez le cas échéant les modes strict de vault manager.
- Hôte de l'installateur :traiter comme un plan de contrôle de confiance élevée ; restreindre l'accès et surveiller.
- Artefacts :protéger les canaux d'acquisition ; les processus organisationnels peuvent compléter verify-bundle par une vérification cryptographique.
- Journalisation:Journaux d'applications et d'analyse sous-déployés/journaux/

8.2 Validation en tant que porte opérationnelle

Exemple d'appel préalable au déploiement :

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

Avec des indicateurs facultatifs :

- -p <policy_file>— utiliser une stratégie de validation personnalisée (par défaut, validation_checks/validation_policies/default.yaml)
- -a <app> — limiter les vérifications à une application spécifique (minuscules, telles que cnc, nso, bpa, cdg)
- —report-file <chemin> — écrire un rapport de prévérification JSON autonome

8.3 Discipline relative à la libération

Il s'agit d'un modèle d'approvisionnement interne dans lequel, pour une installation plus poussée des applications CISCO, le même principe peut être suivi par le transfert du référentiel.

---

9. Avantages et résultats

Thème	Résultat
Temps et labeur	Moins d'étapes manuelles ; défaillances détectées lors des phases de vérification du bundle et de validation plutôt qu'à un moment tardif dans Ansible ou les installateurs de produits
Cohérence	Les schémas, les rôles et la disposition des artefacts partagés entre les missions réduisent les différences de « flocons de neige »
Opérations déconnectées	Transfert de bundle documenté prenant en charge les réseaux réglementés sans téléchargements d'exécution
Gouvernance	Les rapports de validation structurés et les hooks JIRA facultatifs prennent en charge les enregistrements des modifications et le suivi
Extensibilité	Effacer les points d'extension : ARTIFACT_DEFS, gestionnaires, nouveaux rôles/guides, schémas d'intention

Les mesures quantitatives (durée d'installation, taux de défauts) sont propres à l'organisation ; les équipes doivent établir une référence par rapport aux runbooks existants sur des topologies comparables.

---

10. Extensibilité et entretien

10.1 Ajout de types d'artefacts

1. ExtendARTIFACT_DEFS (et les étiquettes si nécessaire) incx_deploy_orchestrator.py.
2. Ajoutez un gestionnaire personnalisé lorsque l'attribution de noms ne peut pas être capturée par les modèles seuls.
3. Mettez à jour la logique de packaging insetup_cxinstaller_preREQUIS lorsque les téléchargements sont automatisés.

10.2 Ajout d'un comportement répréhensible

Préfèrent les nouvelles tâches dans des rôles cohérents ; introduisez de nouveaux rôles lorsque les limites sont claires. Classeurs filaires viaimport_playbook ou des guides documentés d'entrée. Conservez les valeurs par défaut sûres dans group_vars / vars.

10.3 Évolution du générateur d'intentions

Mettre à jour les schémas YAML sous-intention-générateur/schéma/ et les entrées de chatbot ; assurez-vous que les fichiers générés correspondent aux noms de fichiers attendus par APP_CONFIG.

10.4 Considérations relatives à la feuille de route (illustration)

- Intégration plus approfondie de la SBOM ou de la vérification des signatures d'image.
- Couverture de validation étendue pour les scénarios CNC/CDG.
- Références de l'IC pour les contrôles de syntaxe des spécifications et Ansible.

---

11. Conclusion

Le programme d'installation programmable CX combine des spécifications déclaratives, un plan de contrôle Python pour l'emballage et la vérification, et un plan d'automatisation Ansible pour les déploiements évolutifs et reproductibles de produits liés aux réseaux croisés sur divers modèles d'infrastructure et de connectivité. Son architecture sépare intentionnellement l'intention de l'exécution, applique les attentes d'artefact basées sur les données lorsque cela est pratique, et intègre des workflows de validation et de stockage en chambre forte adaptés à la livraison d'entreprise. Pour obtenir des annexes opérationnelles complètes (tables de lecture, matrices de

dépannage, matrices de connectivité et références de commandes étendues), reportez-vous au livre blanc interne.

Références et carte de documentation

Documenter	Chemin
Guide de l'opérateur	README.md
Libérer le guide	GUIDE_VERSION.md
Annexes d'architecture interne	docs/CX_INSTALLER_TECHNICAL_WHITE_PAPER_INTERNAL.md
Docker (en ligne / air-gap / utilisation)	SETUP_ONLINE_DOCKER.md, SETUP_AIRGAPPED_DOCKER.md, USAGE_DOCKER.md
Cadre de validation	docs/validation_checks/README.md
Gestionnaire de coffre-fort	docs/scripts/VAULT_SECRETS_MANAGER.md
Guides des produits	docs/nso.md, docs/bpa.md, docs/CNC_VCENTER_DEPLOYMENT_GUIDE.md, docs/CNC_OCP_DEPLOYMENT_GUIDE.md, docs/CNC_NSO_DEPLOYMENT_GUIDE.md
Générateur d'intention	intent-generator/README.md
Présentation du chatbot/flux de règles	docs/HowItWorks.md

Annexe A — Présentation du référentiel (résumé)

```
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
```

Points focaux des artefacts post-configuration (standard) : ansible_playbooks/files/artifacts/, files/bin/, files/charts/, files/images/.

Annexe B - CLI d'Orchestrator (récapitulatif)

Script : cx_deploy_orchestrator.py

Argument	Description
—app / -a	nso | crossworksuite | bpa
—spec / -s	Chemin vers la spécification YAML
: étape	generate-intent | verify-bundle | install
—verify-only	Vérifier le bundle ; sortie non nulle si non prête
—parcours à sec	Essai à sec si pris en charge
—list-specs	Répertorier les spécifications connues

Environnement (session type) :

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

Annexe C — Glossaire

Terme	Définition
Spécification	Spécification utilisateur YAML : plates-formes, applications, topologie, chemins, droits
Intention	YAML normalisé à partir du générateur d'intention ou équivalent manuscrit
Offre Groupée	Arbre d'installation fourni (souvent tarball) pour le transfert d'air-gap
Orchestrator	cx_deploy_orchestrator.py — vérification / intention / coordination de l'installation
Vérification des artefacts	Le système de fichiers vérifie que les fichiers binaires/images requis existent par spécification
Voûte	Fichier variable chiffré Ansible Vault pour les secrets
EXTRÉMITÉ	Module NSO (Network Element Driver)
CFS/RFS	Concepts de topologie de redirecteur de cluster NSO / redirecteur redondant

Terme	Définition
Espace d'air	Environnement sans accès au moment du programme d'installation aux terminaux de téléchargement de package

Historique des révisions du document

Version	Date	Remarques
1.0	2026-03-27	Livre blanc technique prêt pour la publication initiale (encadrement du programme d'installation programmable)

À propos de cette traduction
Cisco a traduit ce document en traduction automatisée vérifiée par une personne dans le cadre d'un service mondial permettant à nos utilisateurs d'obtenir le contenu d'assistance dans leur propre langue.

Il convient cependant de noter que même la meilleure traduction automatisée ne sera pas aussi précise que celle fournie par un traducteur professionnel.