Programma di installazione programmabile

Sommario

Programma di installazione programmabile

Campo	Valore
Prodotto	Programma di installazione programmabile
Tipo di documento	White paper tecnico: architettura, implementazione e risultati
Destinatari principali	Architetti di soluzioni, ingegneri di piattaforme, DevOps / SRE, responsabili della fornitura
Destinatari secondari	Gestione tecnica, revisori della sicurezza, responsabili di programma

Riassunto

Programmable Installer è una piattaforma di automazione guidata da specifiche per l'installazione e il funzionamento di stack di software Cisco, tra cui Network Services Orchestrator (NSO), Crosswork Network Controller (CNC), Crosswork Data Gateway (CDG) e Business Process Automation (BPA), su Enterprise Linux (famiglia RHEL) e sulle relative infrastrutture ove applicabile (VMware vCenter, OpenShift, KVM, Kubernetes con interruzioni di aria). Il sistema separa l'intento dichiarativo (specifiche YAML e generazione di intento guidato opzionale) dall'esecuzione (ruoli Ansible e playbook), con un piano Pythoncontrol che raccoglie gli artefatti, verifica i bundle prima di installazioni di lunga durata, prepara i segreti crittografati e organizza i gate di convalida.

Questo white paper spiega i livelli dell'architettura, i flussi di dati primari, i modelli di implementazione (incluso un modello di verifica degli artefatti basato su dati ibridi), le modalità di distribuzione (native, containerizzate, online, air-gapped) e i framework di convalida e registrazione. Per le organizzazioni di distribuzione e piattaforme, la piattaforma mira a ridurre gli sforzi manuali, la configurazione errata delle superfici e la mancanza di binari in anticipo, nonché a standardizzare l'automazione tra prodotti e topologie, preservando al contempo la parametrizzazione specifica dell'ambiente.

Parole chiave:automazione dell'infrastruttura, distribuzione dichiarativa, Ansible, specifica YAML, imballaggio air-gap, verifica degli artefatti, Crosswork, NSO, CNC, CDG, BPA, policy di convalida, DevOps.

Come leggere questo documento

Ruolo	Stato attivo consigliato
Responsabili delle decisioni/responsabili	Riassunto; §1 Proposta di valore; §8 benefici e posizione di rischio; §10 Conclusione
Architetti di soluzioni	§3-§6 (architettura, modello di specifica, implementazione, modelli di distribuzione)
DevOps / SRE / tecnici di consegna	§ 5, § 7; Appendice B Allegati del Libro bianco interno di accompagnamento
Revisori della sicurezza	§7 Sicurezza e conformità; limiti del trust di cui al paragrafo 3.2

---

1. Proposta di valore

1.1 Il problema della consegna

L'installazione aziendale dei prodotti di automazione e orchestrazione della rete multi-tier è tradizionalmente di tipo high-touch: runbook lunghi, molti passaggi manuali, incongruenze di versione tra i siti e guasti che rendono possibile un processo (terminazioni mancanti, percorsi degli OAV errati, insiemi di immagini con spazio vuoto incompleto). Questo modello aumenta i costi, allunga le finestre di modifica e rende più difficili le verifiche.

1.2 L'approccio programmabile dell'installatore

Il programma di installazione programmabile considera un'installazione come programmabile parametrizzata in base a una specifica: topologia, versioni, scelta della piattaforma (vCenter vs OpenShift vs VM vanilla), percorsi di file e autorizzazioni. Ove possibile, l'automazione è ripetibile tra i clienti e caricata in anticipo con controlli che consentono di ottenere un risultato rapido ed esplicito prima dell'installazione del cluster o del prodotto.

1.3 Cosa si intende per "programmabile" in questo contesto

- Dichiarativo:operatori che descrivono gli elementi da distribuire; i playbook implementano il thow.
- Verifica basata su dati: gli artifact previsti derivano da tabelle e regole anziché da script ad hoc per versione, quando i pattern sono stabili.
- Qualità basata su criteri: la convalida pre e post-distribuzione viene eseguita in base a criteri gerarchici, con report strutturati e integrazione opzionale di ticketing.
- Funzionamento multimodale:menu interattivi per i nuovi utenti; CLI e binari bloccati per la ripetizione di CI/CD; Flussi basati su Docker per immagini di runtime standardizzate.

---

2. Contesto del sistema

2.1 Attori e ambienti

Attore/sistema	Ruolo
Responsabile consegna	Crea o genera le specifiche, esegue il packaging, la preparazione degli archivi, la convalida, l'orchestrator e Ansible
Host del programma di installazione	Nodo di controllo Linux (nativo o contenitore) con Python, configurazione Ansible, disco per artefatti
Infrastruttura di destinazione	vCenter, OpenShift/KubeVirt o VM vanilla in base alle specifiche
Origini artifact	Mirroring interno, layout dei diritti, distribuzione del software, specifico per l'ambiente
Sistemi a valle	Monitoraggio, gestione delle modifiche, flussi di lavoro JIRA opzionali

2.2 Limiti dell'attendibilità

1. Specificheintenti di stampa; possono fare riferimento a percorsi e parametri non segreti. I segreti devono passare attraverso i workflow degli archivi ansibili e non campi di specifiche in formato testo normale dove evitabile.
2. La conservazione degli artefatti deve essere protetta dall'integrità; la verifica è incentrata sulla presenza e sull'allineamento dei nomi con la specifica, estendendosi ai controlli organizzativi (checksum, bundle firmati) laddove richiesto dalla policy.
3. SSH da installatore a destinazione: percorso con privilegi elevati; la compromissione dell'host del programma di installazione ha un impatto notevole. La protezione avanzata e il controllo degli accessi sono prerequisiti operativi.

---

3. Principi di architettura

1. Prima dichiarativa:intento dell'utente in YAML; l'automazione lo interpreta in modo coerente.
2. Separazione della pianificazione e dell'esecuzione:Python pianifica, verifica e organizza le porte; Ansible esegue fasi di infrastruttura e prodotto.
3. Automazione componibile: i playbook a livello di sito importano playbook mirati (preinstallazione, tracce Kubernetes, installazioni di prodotti).
4. Divulgazione progressiva:Impostazione interattiva per l'onboarding; flag e script per l'automazione avanzata.
5. Stessa codebase, più runtime:i percorsi AlmaLinux/RHEL nativi e i percorsi basati su Docker condividono un layout repository.
6. supporto esplicito di air-gap:pacchetto su una macchina collegata; trasferire un fascio; installare i prerequisiti e distribuire senza dipendenza di runtime dalle reti pubbliche.

---

4. Architettura logica

4.1 Livelli

Strato	Responsabilità
Specifiche	Topologia, versioni, piattaforme, percorsi, diritti
Piano di controllo	Packaging, verifica dei bundle, helper di vaulting, driver di convalida, CLI di orchestrator
Piano di automazione	Preparazione host, ciclo di vita Kubernetes, installazione del prodotto e configurazione immediata
Artifact	Binari, immagini, grafici, ovali, palle

4.2 Flussi di dati end-to-end

1. Flusso di package:il prerequisito per l'utilizzo degli utensili consente di scaricare o posizionare i file in una posizione specifica, producendo facoltativamente una pallina da terra trasferibile per le installazioni offline.
2. Verifica del flusso: Orchestrator analizza la specifica, risolve gli elementi previsti (inclusi i filtri della piattaforma e gli elenchi dei diritti) e segnala gli elementi pronti o mancanti prima dell'installazione.
3. Distribuire il flusso:Spec plus vault (e pre-convalida opzionale) consente di creare playbook compatibili con gli inventari derivati dal progetto.

4.3 Prodotti supportati (ambito dell'installatore)

Prodotto/pacchetto
NSO
CNC
CDG
BPA
CNC + NSO

---

5. Specifiche e modello intento

5.1 Specifiche utente

Le specifiche sono documenti YAML che descrivono le piattaforme (ad esempio vCenter, OCP, VM, KVM), gli host, le applicazioni con versioni, la topologia (ad esempio i layout CFS/RFS NSO),i diritti (NED e pacchetti aggiuntivi) e i percorsi di file per gli OVI, le immagini qws2 e le tralla di livello applicazione.

5.2 Frammenti comuni

Un'applicazione specifica user_spec fornisce valori predefiniti e fallback di percorso per CNC/CDG. Il parser dell'orchestrator tratta la specifica dell'utente come origine della verità e utilizza le voci delle specifiche comuni quando le chiavi utente sono assenti.

5.3 Generazione intento

Il generatore di intento supporta la raccolta guidata dei requisiti tramite un insieme di questionari, un motore delle regole (la logica basata sulla data) e il mapping basato su schema a intent.yaml.

---

6. Profonda implementazione

6.1 Deployment orchestrator (`cx_deploy_orchestrator.py`)

L'orchestrator è la voce singola per la generazione con script o interattiva con intento di generazione,verifica-bundle, e il coordinamento dell'installazione. Il suo design è esplicitamente ibrido:

- **`ARTIFACT_DEFS`**:Dichiara tipi di artifact e modelli di denominazione per applicazione (programma di installazione NSO, pacchetti firmati END, pacchetti facoltativi; CNC OVULI/qvacs2/palle di articolazione; immagini CDG; Grafici BPA e tarball immagine air-gap).
- **`APP_CONFIG`:**esegue il mapping di CLI–applica i valori (`nso`,`crossworksuite`,`bpa`) ai nomi di cartella di specifica e ai nomi di file intento predefiniti.
- **`Parser`/`Handler`**:Risolvere i percorsi CNC/CDG utilizzando le specifiche utente e le specifiche comuni;handler personalizzati che individuano la denominazione non uniforme (ad esempio TSDN/DLM) e la formattazione della versione BPA per i tracciati grafico e tallonamento.

Preparazione:AReportaggregates ha individuato elementi;`is_`readyis true quando non mancano file necessari dopo l'analisi delle specifiche. Il modulo supporta la risoluzione `viasys`.frozenpath dei file binari congelati di PyInstaller.

6.2 Prerequisiti e strumenti per la creazione del package (`setup_cxinstaller_prereqs`)

Questo componente fornisce menu interattivi e modalità CLI per il packaging degli artifact e l'installazione dei prerequisiti dell'host: online, air-gap o auto-detect; packaging di più applicazioni, incluso `CombinedCNC_NSO`. Popola l'albero degli artefatti previsto dalla logica Ansible e verify-bundle.

6.3 Piano di automazione andibile

Composizione:illustra la natura a più tracce dello stack: importa percorsi di bootstrap Kubernetes diversi, a indicare che i diversi prodotti sono destinati a percorsi di bootstrap Kubernetes diversi.

Ruoli(famiglie rappresentative):`preinstall(SELinux, firewall, SSH, helper del Registro di sistema),k8s_install/rke2,deploy_nso,deploy_cnc,deploy_cdg,deploy_bpa,postinstall`, disinstallazione e helper per il rinnovo dei certificati. La proprietà e la sperimentazione sono limiti gestiti al meglio; le cartelle di attività nidificate implementano flussi di lavoro secondari (ad esempio NSO L3 HA).

6.4 Quadro dei controlli di validità (**`validation_checks/`**)

Il framework fornisce il controllo gerarchico delle regole (globale → app → stadio → controllo individuale), auto-discovery dei controlli,enhanced reporting to structures logs, e opzionale integrazione JIRA. Gli operatori eseguono le fasi di pre o post-distribuzione sulla base della stessa specifica utilizzata per l'installazione, allineando l'automazione con i gati di qualità adatti al cambiamento della disciplina aziendale.

Scala indicativa su un ramo tipico: nell'ordine di trenta controlli tra BPA e NSO (i conteggi devono essere confermati `con l'esecuzione di list-validation-`checkout all'estrazione).

6.5 Gestione segreti Vault (**`scripts/vault_secrets_manager.py`**)

Deriva le variabili di archivio richieste dalle specifiche, richiede o accetta password in base ai criteri ed `emette encryptedgroup_vars/all_secrets.`yamlplus un file di password di archivio per Ansible, riducendo l'incorporamento di segreti ad hoc nei playbook.

---

7. Modelli di distribuzione e runtime

Motivo	Riepilogo
Nativo (AlmaLinux / RHEL)	`SetPYTHONPATHandANSIBLE_CONFIG`; esecuzione di packaging, vault, validation, orchestrator e playbook per guida prodotto
Programma di installazione	`scripts/setup_installer.shandscripts/start_installer.`shwith installazione host per artifact di grandi dimensioni;

Motivo	Riepilogo
basato su Docker	
Intervallo d'aria	Pacchetto su una macchina collegata; pacchetto di trasferimento; estrazione a destinazione; installa con: `airgap`
creazione bundle macOS	`Usepython3 ./setup_cxinstaller_prereqs.`pyon Mac per preparare i pacchetti; l'implementazione di destinazione rimane orientata a Linux per i documenti del progetto

---

8. Sicurezza, convalida e osservabilità

8.1 Postura di sicurezza (finalità di progettazione)

- Secrets:Preferire le variabili di gruppo crittografate di Ansible Vault; utilizzare le modalità rigorose di vault manager, se necessario.
- Host installatore:da trattare come piano di controllo con attendibilità elevata; limitare l'accesso e il monitoraggio.
- Artifact:Proteggi canali di acquisizione; i processi organizzativi possono aumentare verify-bundle con la verifica crittografica.
- Registrazione:log delle applicazioni e ansibili `sottodistribuzione/log/`

8.2 Convalida come gate operativo

Esempio di chiamata pre-distribuzione:

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

Con flag opzionali:

- `-p <file_criteri>`— utilizza un criterio di convalida personalizzato (default `tovalidation_checks/validation_policies/default.yaml`)
- `-a <app>`— limita i controlli a un'app specifica (lettere minuscole, ad esempio `cnc`,`nso`,`bpa`,`cdg`)
- `–report-file <percorso>`— scrive un report di verifica preliminare JSON autonomo

8.3 Disciplina della release

Si tratta di un modello per l'origine interna in cui per un numero maggiore di applicazioni CISCO è possibile seguire lo stesso principio biforcando il repository.

---

9. Vantaggi e risultati

Tema	Risultato
Tempo e fatica	Meno passaggi manuali; errori rilevati nelle fasi di verifica del bundle e di convalida anziché in ritardo nei programmi di installazione Ansible o del prodotto
Coerenza	Gli schemi, i ruoli e il layout degli artifact condivisi tra più progetti riducono le differenze di tipo "snowflake"
Operazioni disconnesse	Il trasferimento documentato dei bundle supporta le reti regolamentate senza download di runtime
Governance	Report di convalida strutturati e hook JIRA facoltativi per il supporto dei record di modifica e per il follow-up
Estendibilità	Cancella punti di estensione:`ARTIFACT_DEFS`, gestori, nuovi ruoli/playbook, schemi intento

Le metriche quantitative (durata dell'installazione, percentuali di difetti) sono specifiche dell'organizzazione; i team devono basarsi su runbook legacy su topologie confrontabili.

---

10. Estendibilità e manutenzione

10.1 Aggiunta di tipi di artifact

1. `ExtendARTIFACT_DEFS`(e le etichette se necessarie) `incx_deploy_orchestrator.py`.
2. Aggiungere un gestore personalizzato quando la denominazione non può essere acquisita dai soli modelli.
3. Aggiornare la logica di creazione pacchetti `installetup_cxinstaller_`prereqsquando i download sono automatizzati.

10.2 Aggiunta di un comportamento ansibile

Preferire le nuove operazioninei ruoli coesivi; introdurre nuovi ruoli quando i limiti sono chiari. Playbook su filo `viaimport_`playbook o playbook in entrata documentati. Keepsafe `defaultsingroup_vars/vars`.

10.3 Evoluzione del generatore di intento

Aggiornare gli schemi YAML `underintent-generator/schema/`e gli input chatbot; verificare che i file generati corrispondano ai nomi file previsti `da APP_CONFIG`.

10.4 Considerazioni sulla roadmap (illustrative)

- Integrazione di DeeperSBOM o di verifica della firma dell'immagine.
- Ampliamento della copertura di convalida per gli scenari CNC/CDG.
- Riferimenti CI per il plinting delle specifiche e i controlli della sintassi Ansible.

---

11. Conclusione

Il programma di installazione programmabile CX combina specifiche dichiarative, un Python control plan per il packaging e la verifica e un Ansible automation plan per implementazioni scalabili e ripetibili di prodotti correlati a Crosswork su infrastrutture e modelli di connettività diversi. La sua architettura separa intenzionalmente l'esecuzione, applica le aspettative relative agli artifact guidati dai dati, quando possibile, e incorpora la convalida e i vaultworkflow adatti alla distribuzione aziendale. Per gli allegati operativi completi (tabelle di playbook, matrici per la risoluzione dei problemi, matrici di connettività e riferimenti a comandi estesi), consultare il white paper interno.

Mappa riferimenti e documentazione

Documento	Percorso
Manuale dell'operatore	`LEGGIMI.md`
Rilascia playbook	`RELEASE_GUIDE.md`
Allegati architettura interna	`docs/CX_INSTALLER_TECHNICAL_WHITE_PAPER_INTERNAL.md`
Docker (online/air-gap/utilizzo)	`SETUP_ONLINE_DOCKER.md`,`SETUP_AIRGAPPED_DOCKER.md`,`USAGE_DOCKER.md`
Framework di convalida	`docs/validation_checks/README.md`
Gestione archivi	`docs/scripts/VAULT_SECRETS_MANAGER.md`
Guide del prodotto	`docs/nso.md`,`docs/bpa.md`,`docs/CNC_VCENTER_DEPLOYMENT_GUIDE.md`,`docs/CNC_OCP_DEPLOYMENT`
Generatore di intento	`intent-generator/README.md`
Chatbot/panoramica sul flusso delle regole	`docs/HowItWorks.md`

Appendice A — Layout del repository - https://wwwin-github.cisco.com/CX-SAO-TOOLS/cx-installer (riepilogo)

```
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
```

Punti focali artefatto post-installazione (tipici):`playbooks_andibili/file/artefatti/`,`file/bin/`,`file/grafici/`,`file/immagini/`.

Appendice B — CLI di Orchestrator (riepilogo)

Script:`cx_deploy_orchestrator.py`

Argomento	Descrizione
`–app/-a`	`nso|crossworksuite|bpa`
`–spec/-s`	Percorso della specifica YAML
`- passo`	`generate-intent|verify-bundle|installa`
`- solo verifica`	Verificare il pacchetto; esci da un valore diverso da zero se non pronto
`–a secco`	Dry run, se supportato
`- list-specs`	Elenca specifiche note

Ambiente (sessione tipica):

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

Appendice C — Glossario

Termine	Definizione
Specifica	Specifica utente YAML: piattaforme, applicazioni, topologia, percorsi, diritti
Intento	YAML normalizzato dal generatore di intento o equivalente creato a mano
Pacchetto	Albero di installazione preconfezionato (spesso tarball) per il trasferimento di air-gap
Orchestrator	`cx_deploy_orchestrator.py`— verifica/intento/installazione coordinamento
Verifica artifact	Il file system verifica che esistano file binari/immagini necessari per specifica
Archivio	Ansible Vault - File variabile crittografato per segreti
FINE	Pacchetto NSO (Network Element Driver Package)
CFS/RFS	Concetti su topologia server d'inoltro cluster NSO/server d'inoltro ridondante
Intervallo d'aria	Ambiente senza accesso in fase di installazione agli endpoint di download dei pacchetti

---

Cronologia revisioni documenti

Version	Data	Note
1.0	2026-03-27	White paper tecnico iniziale pronto per la pubblicazione (frame del programma di installazione)

Informazioni su questa traduzione

Cisco ha tradotto questo documento utilizzando una combinazione di tecnologie automatiche e umane per offrire ai nostri utenti in tutto il mondo contenuti di supporto nella propria lingua. Si noti che anche la migliore traduzione automatica non sarà mai accurata come quella fornita da un traduttore professionista. Cisco Systems, Inc. non si assume alcuna responsabilità per l’accuratezza di queste traduzioni e consiglia di consultare sempre il documento originale in inglese (disponibile al link fornito).