Utilizar el instalador programable

Contenido

Introducción

Este documento describe el Instalador programable, una plataforma de automatización basada en especificaciones para implementar pilas de software de Cisco como NSO y CNC.

Instalador programable

Campo	Valor
Producto	Instalador programable
Tipo de documento	Informe técnico: arquitectura, implementación y resultados
Público principal	Arquitectos de soluciones, ingenieros de plataformas, DevOps/SRE, clientes potenciales
Audiencia secundaria	Gestión de ingeniería, revisores de seguridad, directores de programas

Abstracto

El instalador programable es una plataforma de automatización basada en especificaciones para implementar y utilizar pilas de software de Cisco, incluidos Network Services Orchestrator (NSO), Crosswork Network Controller (CNC), Crosswork Data Gateway (CDG) y Business Process Automation (BPA), en Linux empresarial (familia RHEL) y la infraestructura asociada donde corresponda (VMware vCenter, OpenShift, KVM, Air-Gap Kubernetes). El sistema separa la intención declarativa (especificaciones YAML y generación de intención guiada opcional) de la ejecución (roles y cuadernos Ansible), con un plano de control Python que empaqueta artefactos, verifica paquetes antes de instalaciones de larga duración, prepara secretos cifrados y organiza puertas de validación.

En este informe técnico se explican las capas arquitectónicas, los flujos de datos primarios, los patrones de implementación (incluido un modelo híbrido de verificación de artefactos impulsado por datos), los modos de implementación (nativos, en contenedores, online y con espacios entre nodos) y los marcos de validación y registro. En el caso de las organizaciones de plataforma y de distribución, la plataforma pretende reducir el trabajo manual, la configuración incorrecta de la superficie y la ausencia temprana de binarios, así como estandarizar la automatización en los productos y las topologías al tiempo que se conserva la parametrización específica del entorno.

Palabras clave: automatización de infraestructuras, implementación declarativa, Ansible, especificación YAML, empaquetado Air-Gap, verificación de artefactos, Crosswork, NSO, CNC, CDG, BPA, política de validación, DevOps.

Cómo leer este documento

Función	Objetivo sugerido
Responsables de la toma de decisiones/clientes potenciales	Abstracto; §1 Propuesta de valor; §8 Beneficios y posición de riesgo; §10 Conclusión
Arquitectos de soluciones	§3-§6 (arquitectura, modelo de especificación, implementación, patrones de implementación)
DevOps/SRE/ingenieros de entregas	§5-§7; Apéndice B; anexos internos complementarios del informe técnico
Revisores de seguridad	§7 Condición de seguridad y cumplimiento; límites de confianza en §3.2

---

1. Propuesta de valor

1.1 El problema de la entrega

Tradicionalmente, la instalación empresarial de productos de automatización y orquestación de redes multinivel ha sido muy sencilla: runbooks largos, muchos pasos manuales, desviación de versiones entre sitios y fallos que superan horas en un proceso (falta de NED, rutas de OVA erróneas, conjuntos de imágenes con huecos de aire incompletos). Ese patrón aumenta los costos, alarga las ventanas de cambio y dificulta las auditorías.

1.2 El enfoque del instalador programable

El Instalador programable trata una instalación como un programa parametrizado por una especificación: topología, versiones, elección de plataforma (vCenter frente a OpenShift frente a VM virtuales), rutas de archivo y derechos. La automatización es idempotente siempre que sea posible, se puede repetir entre los clientes y se carga de forma frontal con comprobaciones, de modo que "no preparado" es un resultado rápido y explícito antes de la instalación del clúster o del producto.

1.3 Qué significa programable en este contexto

- Declarativo: los operadores describen lo que debe implementarse; los cuadernos de

campaña implementan cómo.
- Verificación basada en datos: los artefactos esperados se derivan de tablas y reglas en lugar de scripts ad-hoc por versión, cuando los patrones son estables.
- Calidad controlada por políticas: la validación previa y posterior a la implementación se ejecuta bajo políticas jerárquicas, con informes estructurados e integración de notificaciones opcional.
- Funcionamiento multimodal: menús interactivos para usuarios principiantes; CLI y binarios congelados para la repetición estilo CI/CD; Flujos basados en Docker para imágenes de tiempo de ejecución estandarizadas.

---

2. Contexto del sistema

2.1 Actores y entornos

Actor/sistema	Función
Ingeniero de entregas	Crea o genera especificaciones, ejecuta empaquetado, preparación de cajas fuertes, validación, orquestador y Ansible
Host del instalador	Nodo de control Linux (nativo o contenedor) con Python, configuración Ansible, disco para artefactos
Infraestructura objetivo	VM de vCenter, OpenShift/KubeVirt o vanilla según la especificación
Fuentes de artefactos	Duplicaciones internas, diseños de derechos, distribución de software (específico del entorno)
Sistemas descendentes	Supervisión, gestión de cambios y flujos de trabajo JIRA opcionales

2.2 Límites de confianza

1. Especificaciones, intención expresa; pueden hacer referencia a rutas y parámetros no secretos. Los secretos deben fluir por los flujos de trabajo de Ansible Vault, no por campos de texto sin formato donde se puedan evitar.
2. El almacenamiento de artefactos debe estar protegido contra la integridad; la verificación se centra en la alineación de la presencia y los nombres con las especificaciones (amplíe con los controles organizativos (sumas de comprobación, paquetes firmados) donde la política lo requiera.
3. El SSH del instalador al destino es una trayectoria de alto privilegio; el riesgo del host del instalador es de gran impacto. La consolidación y el control de acceso son requisitos previos operativos.

---

3. Principios arquitectónicos

1. Declarativo primero: intención del usuario en YAML; la automatización lo interpreta de forma coherente.
2. Separación de la planificación y la ejecución: Python planifica, verifica y organiza las puertas; Ansible ejecuta pasos de productos e infraestructura.
3. Automatización componible: los cuadernos de campaña a nivel de sitio importan cuadernos centrados (preinstalación, pistas de Kubernetes, instalaciones de productos).
4. Divulgación progresiva: configuración interactiva para la incorporación; indicadores y scripts para automatización avanzada.
5. Misma base de código, varios tiempos de ejecución: las rutas nativas de AlmaLinux/RHEL y las rutas basadas en Docker comparten un diseño de repositorio.
6. Soporte explícito del vacío de aire: paquete en una máquina conectada; transferir un paquete; instalar requisitos previos e implementar sin dependencia de tiempo de ejecución en redes públicas.

---

4. Arquitectura lógica

4.1 Capas

Capa	Responsabilidad
Especificación	Topología, versiones, plataformas, rutas, derechos
Plano de Control	Paquetes, verificación de paquetes, ayudantes de almacén, controlador de validación, CLI de orquestador
Plano de automatización	Preparación del host, ciclo de vida de Kubernetes, instalación del producto y configuración desde el primer día
Artefactos	Binarios, imágenes, gráficos, OVAs, tarballs

4.2 Flujos de datos de extremo a extremo

1. Flujo de paquetes: la herramienta de requisitos previos descarga o almacena en zona intermedia los archivos en una ubicación específica, produciendo opcionalmente un archivo comprimido transferible para las instalaciones sin conexión.
2. Verificación del flujo: el orquestador analiza las especificaciones, resuelve los artefactos esperados (incluidos los filtros de la plataforma y las listas de derechos) e informa de que están listos o no se han instalado antes de la instalación.
3. Implemente el flujo: la spec. más la caja fuerte (y la prevalidación opcional) impulsan los cuadernos de campaña Ansible frente a los inventarios derivados del compromiso.

4.3 Productos admitidos (ámbito del instalador)

Producto/paquete
NSO
CNC
CDG
BPA
CNC + NSO

---

5. Especificación y modelo de intención

5.1 Especificaciones del usuario

Las especificaciones son documentos YAML que describen plataformas (por ejemplo, vCenter, OCP, VM, KVM), hosts, aplicaciones con versiones, topología (por ejemplo, diseños CFS/RFS de NSO), derechos (paquetes NED y complementarios) y rutas de archivos para OVA, imágenes qcow2 y tarballs de capa de aplicación.

5.2 Fragmentos comunes

Una aplicación específica user_spec suministra valores por defecto y rutas de reserva para CNC/CDG. El analizador del orquestador trata las especificaciones del usuario como fuente de veracidad y utiliza entradas de especificaciones comunes cuando no hay claves de usuario.

5.3 Generación de iniciativas

El generador de intents admite la recopilación guiada de requisitos mediante un conjunto de cuestionarios, un motor de reglas (lógica controlada por fecha) y asignación respaldada por esquema a intent.yaml.

---

6. Implementación en profundidad

6.1 Orquestador de implementación (cx_deploy_orchestrator.py)

El orquestador es la única entrada para la coordinación de la instalación, la verificación de agrupamiento y la intención de generar con scripts o interactiva. Su diseño es explícitamente

híbrido:

- ARTIFACT_DEFS: declara tipos de artefactos y patrones de denominación por aplicación (instalador NSO, paquetes firmados NED, paquetes opcionales; CNC OVA/qcow2/tier tarballs; imágenes CDG; gráficos BPA y tarballs de imágenes Air-Gap).
- APP_CONFIG: Asigna los valores de CLI —app (nso, crossworksuite, bpa) a los nombres de carpeta de especificación y a los nombres de archivo de intent predeterminados.
- Analizador/Manejadores: Resuelva las rutas CNC/CDG utilizando especificaciones de usuario y especificaciones comunes; los controladores personalizados cubren la nomenclatura no uniforme (por ejemplo, TSDN/DLM) y el formato de versión BPA para las rutas de gráficos y tarball.

Readiness:AReportaggregates detected artifacts; is_readyis true cuando no faltan archivos requeridos después del análisis de especificaciones. El módulo admite binarios congelados por PyInstaller mediante la resolución de ruta sys.locked.

6.2 Requisitos previos y herramientas de empaquetado (setup_cxinstaller_prereqs)

Este componente proporciona menús interactivos y modos CLI para empaquetar artefactos e instalar requisitos previos del host: online, air-gap o auto-detect; empaquetado multiaplicación, incluido `combinadoCNC_NSO`. Rellena el árbol de artefactos esperado por Ansible y por la lógica verify-bundle.

6.3 Plano de automatización de Ansible

- Composición: ilustra la naturaleza multipista de la pila: importa diferentes rutas de arranque de Kubernetes, lo que refleja que los diferentes productos se dirigen a diferentes rutas de arranque de Kubernetes.
- Funciones (familias representativas): preinstalación (SELinux, firewall, SSH, ayudantes de registro), k8s_install / rke2, deploy_nso, deploy_cnc, deploy_cdg, deploy_bpa, postinstall, uninstall y ayudantes de renovación de certificados. La propiedad y las pruebas se administran mejor en los límites de las funciones; las carpetas de tareas anidadas implementan subflujos de trabajo (por ejemplo, NSO L3 HA).

6.4 Marco de comprobaciones de validación (validation_checks/)

El marco proporciona un control jerárquico de políticas (global → app → stage → individual check), detección automática de comprobaciones, informes mejorados a registros estructurados e integración opcional de JIRA. Los operadores ejecutan las fases anteriores y posteriores a la implementación con las mismas especificaciones utilizadas para la instalación, alineando la automatización con puertas de calidad adecuadas para la disciplina de cambio empresarial.

Escala indicativa en una sucursal típica: en el orden de treinta comprobaciones a través de BPA y NSO (recuentos que se confirmarán confeccionar comprobaciones de validación de listas en su pago).

6.5 Gestor de secretos de almacén (scripts/vault_secrets_manager.py)

Deriva las variables de depósito necesarias de las especificaciones, solicita o acepta contraseñas en virtud de la política y emite group_vars/all_secrets.yaml cifrado más un archivo de contraseña de depósito para Ansible, lo que reduce la incrustación de secretos ad-hoc en los cuadernos.

---

7. Patrones de implementación y tiempos de ejecución

Patrón	Summary
Nativo (AlmaLinux / RHEL)	Set PYTHONPATH y ANSIBLE_CONFIG; ejecute packaging, vault, validation, orchestrator y playbooks por guía de productos
Instalador basado en Docker	scripts/setup_installer.sh y scripts/start_installer.sh con montajes host para artefactos grandes;
Intervalo de aire	Paquete en una máquina conectada; paquete de transferencia; extracto sobre objetivo; instalar con: airgap
creación del paquete macOS	Usepython3 ./setup_cxinstaller_prereqs.py en Mac para preparar paquetes; la implementación objetivo sigue orientada a Linux por documento de proyecto

---

8. Seguridad, validación y observabilidad

8.1 Condición de seguridad (intención de diseño)

- Secretos:Prefiera las variables de grupo cifradas de Ansible Vault; utilice los modos estrictos de vault manager cuando corresponda.
- Host del instalador:tratar como un plano de control de alta confianza; restringir el acceso y supervisar.
- Artefactos:Proteger los canales de adquisición; los procesos organizativos pueden aumentar verify-bundle con la verificación criptográfica.
- Registro:Registros de aplicaciones y de Ansible subimplementados/registros/

8.2 Validación como puerta operativa

Ejemplo de invocación previa a la implementación:

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

Con indicadores opcionales:

- -p <policy_file>: utilice una directiva de validación personalizada (el valor predeterminado es validation_checks/validation_policies/default.yaml)
- -a <app> : limita las comprobaciones a una aplicación específica (en minúsculas, como cnc, nso, bpa, cdg)
- —report-file <path> — escribe un informe de comprobación previa JSON independiente

8.3 Disciplina de liberación

Se trata de un modelo de abastecimiento interno en el que para más instalaciones de aplicaciones de CISCO se puede seguir el mismo principio bifurcando el repositorio.

---

9. Beneficios y resultados

Tema	Resultado
Tiempo y trabajo	Menos pasos manuales; fallos detectados en las fases de verificación de paquetes y validación en lugar de en los instaladores de productos o Ansible
Uniformidad	Los esquemas compartidos, las funciones y el diseño de artefactos en los compromisos reducen las diferencias de "copo de nieve"
Operaciones desconectadas	La transferencia de paquetes documentada admite redes reguladas sin descargas en tiempo de ejecución
Administración	Los informes de validación estructurados y los enlaces JIRA opcionales admiten registros de cambios y seguimiento
Extensibilidad	Puntos de extensión claros: ARTIFACT_DEFS, controladores, nuevos roles/cuadernos, esquemas por intención

Las métricas cuantitativas (duración de la instalación, tasas de defectos) son específicas de cada organización; los equipos deben comparar los runbooks heredados con topologías comparables.

---

10. Extensibilidad y mantenimiento

10.1 Adición de tipos de artefactos

1. ExtendARTIFACT_DEFS(y etiquetas si es necesario) incx_deploy_orchestrator.py.
2. Agregue un controlador personalizado cuando los patrones solos no puedan capturar la nomenclatura.
3. Actualice la lógica de empaquetado insetup_cxinstaller_prereqscuando las descargas estén automatizadas.

10.2 Adición de un comportamiento ansible

Preferir nuevas tareas dentro de roles cohesionados; introducir nuevos roles cuando los límites estén claros. Cuadernos de hilo viaimport_playbooko cuadernos de entrada documentados. Mantenga los valores predeterminados seguros en group_vars / vars.

10.3 Evolución del generador de intención

Actualizar esquemas YAML con entradas de generación/esquema/subintencional y chatbot; asegúrese de que los archivos generados coincidan con los nombres de archivo esperados por APP_CONFIG.

10.4 Consideraciones sobre la hoja de ruta (ilustración)

- SBOM más profunda o integración de verificación de firma de imagen.
- Cobertura de validación ampliada para escenarios CNC/CDG.
- Referencias de CI para linting de especificaciones y comprobaciones de sintaxis de Ansible.

---

11. Conclusión

El instalador programable CX combina especificaciones declarativas, un plano de control Python para empaquetado y verificación y un plano de automatización Ansible para implementaciones escalables y repetibles de productos relacionados con Crosswork en diversos modelos de infraestructura y conectividad. Su arquitectura separa intencionadamente la intención de la ejecución, aplica expectativas de artefactos impulsados por datos cuando es práctico e integra flujos de trabajo de validación y depósito adecuados para la entrega empresarial. Para ver los anexos operativos completos (tablas del cuaderno de campaña, matrices de resolución de problemas, matrices de conectividad y referencias de comandos ampliadas), consulte el informe técnico interno complementario.

Mapa de referencias y documentación

Documento	Trayecto:
Guía del operador	README.md
Libere el cuaderno	RELEASE_GUIDE.md
Anexos de arquitectura interna	docs/CX_INSTALLER_TECHNICAL_WHITE_PAPER_INTERNAL.md
Docker (online / air-gap / uso)	SETUP_ONLINE_DOCKER.md, SETUP_AIRGAPPED_DOCKER.md, USAGE_DOCKER.md
Marco de validación	docs/validation_checks/README.md
Gestor de depósitos	docs/scripts/VAULT_SECRETS_MANAGER.md
Guías de productos	docs/nso.md, docs/bpa.md, docs/CNC_VCENTER_DEPLOYMENT_GUIDE.md, docs/CNC_OCP_DEPLOYMENT_GUIDE.md, docs/CNC_NSO_DEPLOYMENT_GUIDE.md
Generador de intención	intent-generator/README.md
Descripción general del flujo de reglas/Chatbot	docs/HowItWorks.md

---

Apéndice A: Diseño del repositorio (Resumen)

```
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
```

Puntos focales de artefactos de configuración posterior (típico):ansible_playbooks/files/artifacts/, files/bin/, files/charts/, files/images/.

Apéndice B — Orchestrator CLI (Resumen)

Script:cx_deploy_orchestrator.py

Argumento	Descripción
—app / -a	nso | crossworksuite | bpa
—spec / -s	Ruta a especificación YAML
--paso	generate-intent | verify-bundle | install
—verify-only	Verifique el paquete; salir distinto de cero si no está listo
—dry-run	Ejecución en seco donde se admite
—list-specs	Enumerar especificaciones conocidas

Entorno (sesión típica):

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

Apéndice C: Glosario

Término	Definición
Especificación	Especificación de usuario YAML: plataformas, aplicaciones, topología, rutas y derechos
Intención	YAML normalizado a partir del generador de intents o equivalente de autor manual
Paquete	Árbol de instalación empaquetado (a menudo tarball) para la transferencia de Air-Gap
Orquestador	cx_deploy_orchestrator.py: verificación, intento y coordinación de la instalación
Verificación de artefactos	El sistema de archivos comprueba que existen binarios/imágenes requeridos por especificación
Bóveda	Archivo de variable cifrado de Ansible Vault para secretos
FIN	Paquete de controladores de elementos de red (NSO)
CFS/RFS	Conceptos de topología de reenviador de clústeres/reenviador redundante de NSO
Intervalo de aire	Entorno sin acceso en tiempo de instalación a los terminales de descarga de paquetes

---

Historial de Revisión del Documento

Versión	Fecha	Notas
1.0	2026-03-27	Informe técnico inicial preparado para publicación (marco de instalador programable)

Acerca de esta traducción
Cisco ha traducido este documento combinando la traducción automática y los recursos humanos a fin de ofrecer a nuestros usuarios en todo el mundo contenido en su propio idioma.

Tenga en cuenta que incluso la mejor traducción automática podría no ser tan precisa como la proporcionada por un traductor profesional.

Cisco Systems, Inc. no asume ninguna responsabilidad por la precisión de estas traducciones y recomienda remitirse siempre al documento original escrito en inglés (insertar vínculo URL).