Instalador programável

Contents

Instalador programável

Campo	Valor
Produto	Instalador programável
Tipo de documento	White paper técnico — arquitetura, implementação e resultados
Público principal	Arquitetos de soluções, engenheiros de plataforma, DevOps / SRE, líderes de entrega
Público secundário	Gerenciamento de engenharia, revisores de segurança, gerentes de programa

Resumo

O instalador programável é uma plataforma de automação orientada por especificações para implantar e operar pilhas de software da Cisco — incluindo Network Services Orchestrator (NSO), Crosswork Network Controller (CNC), Crosswork Data Gateway (CDG) e Business Process Automation (BPA) — em Linux corporativo (família RHEL) e infraestrutura associada, onde aplicável (VMware vCenter, OpenShift, KVM, Kubernetes totalmente isolados). O sistema separa a intenção declarativa (especificações YAML e geração de intenção guiada opcional) da execução (funções e manuais de atividades Ansible), com um plano Pythoncontrol que empacota artefatos, verifica pacotes antes de instalações de longa execução, prepara segredos criptografados e orquestra portas de validação.

Este white paper explica as camadas de arquitetura, os fluxos de dados primários, os padrões de implementação (inclusive um modelo híbrido de verificação de artefatos orientados por dados), os modos de implantação (nativo, em contêineres, on-line, com isolamento de ar) e as estruturas de validação e registro. Para organizações de plataforma e fornecimento, a plataforma tem como objetivo reduzir o trabalho manual, a configuração incorreta da superfície e a ausência de binários no início, além de padronizar a automação em produtos e topologias, preservando a parametrização específica do ambiente.

Palavras-chave:automação de infraestrutura, implantação declarativa, Ansible, especificação YAML, embalagem de air-gap, verificação de artefatos, Crosswork, NSO, CNC, CDG, BPA, política de validação, DevOps.

Como ler este documento

Função	Foco sugerido
Tomadores de decisão / leads	Resumo; §1 Proposta de valor; §8 Benefícios e postura de risco; §10 Conclusão
Arquitetos de soluções	§3-§6 (arquitetura, modelo de especificação, implementação, padrões de implantação)
DevOps / SRE / engenheiros de entrega	§5-§7; Apêndice B; anexos complementares do white paper interno
Revisores de segurança	§7 Procedimentos de segurança e conformidade; limites de confiança em §3.2

---

1. Proposta de valor

1.1 O problema da entrega

A instalação corporativa de produtos de automação e orquestração de rede de vários níveis é tradicionalmente de alto envolvimento: runbooks longos, muitas etapas manuais, desvio de versão entre sites e falhas que surgem em um processo (NEDs ausentes, caminhos OVA errados, conjuntos de imagens de air-gap incompletos). Esse padrão aumenta os custos, alonga as janelas de mudança e torna as auditorias mais difíceis.

1.2 A abordagem do instalador programável

O Programmable Installer trata uma instalação como um parametrizado por umaespecificação: topologia, versões, escolha de plataforma (vCenter vs OpenShift vs VMs baunilha), caminhos de arquivos e direitos. A automação é potencialonde possível,repetível entre os clientes ecarregada na frente com verificações para que o "não pronto" seja um resultado rápido e explícito antes da instalação do cluster ou do produto.

1.3 O que significa "programável" neste contexto

- Declarativo:Os operadores descrevem o que deve ser utilizado; playbooks implementamentow.
- Verificação orientada por dados:Os artefatos esperados são derivados de tabelas e regras, em vez de scripts ad-hoc por versão, quando os padrões são estáveis.
- Qualidade por políticas:a validação pré e pós-implantação é executada sob políticas hierárquicas, com relatórios estruturados e integração de tíquetes opcional.
- Operação multimodal:menus interativos para usuários iniciantes; Binários congelados e CLI

para repetição do estilo CI/CD; Fluxos baseados em docker para imagens de tempo de execução padronizadas.

---

2. Contexto do sistema

2.1 Atores e ambientes

Ator / sistema	Função
Engenheiro de entrega	Autora ou gera especificações, executa pacotes, prepara compartimentos, valida, orquestrador e Ansible
Host do instalador	Nó de controle do Linux (nativo ou contêiner) com Python, configuração Ansible, disco para artefatos
Infraestrutura de destino	VMs vCenter, OpenShift/KubeVirt ou vanilla por especificação
Fontes de artefato	Espelhos internos, layouts de qualificação, distribuição de software — específico para o ambiente
Sistemas downstream	Monitoramento, gerenciamento de mudanças, fluxos de trabalho JIRA opcionais

2.2 Limites de confiança

1. As especificaçõesexpressam a intenção; eles podem fazer referência a caminhos e parâmetros não secretos. Segredos devem fluir através de Vaultworkflows Ansible e não campos de especificação de texto simples onde for possível evitar.
2. O armazenamento de artefatos deve ser protegido contra a integridade; a verificação se concentra na presença e no alinhamento de nomes com as especificações — amplie com controles organizacionais (checksums, pacotes assinados) onde a política exigir.
3. O SSH do instalador para o destino é um caminho de alto privilégio; o comprometimento do host do instalador é de alto impacto. Fortalecer e controlar o acesso são pré-requisitos operacionais.

---

3. Princípios arquitetônicos

1. Primeiro declarativo:Intenção do usuário em YAML; a automação a interpreta de forma consistente.
2. Separação de planejamento e execução:Python planeja, verifica e orquestra portões; O Ansible executa etapas de infraestrutura e produto.
3. Automação combinável:Playbooks em nível de site importam playbooks focados (pré-instalação, trilhas Kubernetes, instalações de produtos).

4. Divulgação progressiva:Configuração interativa para integração; sinalizadores e scripts para automação avançada.
5. Mesma base de código, vários tempos de execução:Caminhos nativos AlmaLinux/RHEL e caminhos baseados em Docker compartilham um layout de repositório.
6. Suporte explícito a ar-gap:pacote em uma máquina conectada; transferir um pacote; instalar pré-requisitos e implantar sem dependência de tempo de execução em redes públicas.

---

4. Arquitetura lógica

4.1 Camadas

Camada	Responsabilidade
Especificação	Topologia, versões, plataformas, caminhos, direitos
Controle o plano	Pacote, verificação de pacote, auxiliares de compartimento, driver de validação, CLI do orquestrador
Plano de automação	Preparação do host, ciclo de vida do Kubernetes, instalação do produto e configuração de primeiro dia
Artefatos	Binários, imagens, gráficos, OVAs, tarballs

4.2 Fluxos de dados fim-a-fim

1. Fluxo do pacote:As ferramentas de pré-requisito fazem o download ou transferem os arquivos para um local específico, produzindo opcionalmente um tarball transferível para instalações off-line.
2. Fluxo de verificação:o orquestrador analisa a especificação, resolve os artefatos esperados (incluindo filtros de plataforma e listas de direitos) e relata pronto / ausente antes da instalação.
3. Fluxo de implantação:Spec plus vault (e pré-validação opcional) direcionam os manuais de atividades Ansible em relação aos inventários derivados do compromisso.

4.3 Produtos suportados (escopo do instalador)

Produto / pacote
NSO
CNC
CDG
BPA
CNC + NSO

---

5. Especificação e modelo de intenção

5.1 Especificações do usuário

As especificações são documentos YAML que descrevem plataformas (por exemplo, vCenter, OCP, VM, KVM),hosts,aplicativos com versões,topologia(por exemplo, layouts NSO CFS/RFS),direitos (NEDs e pacotes complementares), e caminhos de arquivo para OVAs, imagens qcow2 e tarballs de camada de aplicativo.

5.2 Fragmentos comuns

Um aplicativo específico user_spec fornece padrões e fallbacks de caminho para CNC/CDG. O analisador do orquestrador trata a especificação do usuário como fonte de verdade e usa entradas de especificação comuns quando as chaves do usuário estão ausentes.

5.3 Propósito da geração

O gerador de intenção suporta a coleta guiada de requisitos através de um conjunto de questionário, um mecanismo de regra (lógica orientada por data) e mapeamento apoiado por esquema para intent.yaml.

---

6. Mergulho profundo na implementação

6.1 Orquestrador de implantação (`cx_deploy_orchestrator.py`)

O orquestrador é a única entrada para script ou interativegenerate-intent,verify-bundle, andinstallCoordination. Seu design é explicitamente híbrido:

- **`ARTIFACT_DEFS`**:Declara tipos de artefatos e padrões de nomenclatura por aplicativo (instalador NSO, pacotes assinados NED, pacotes opcionais; Alcatrões CNC OVA/qcow2/tier; Imagens CDG; gráficos BPA e tarballs para imagens de air-gap).
- **`APP_CONFIG`**:Mapeia CLI – appvalues (`nso`,**`crossworksuite`**,**`bpa`**) para especificar nomes de pastas e nomes de arquivos de intenção padrão.
- **`Analisador`**/**`Manipuladores`**:**`Resolver caminhos CNC/CDG usando especificação do usuário e especificação comum;`**personalizado **`manipuladores abordam nomeação não uniforme (por exemplo, TSDN/DLM) e formatação de versão BPA para caminhos de gráfico e tarball.`**

Preparação:AReportaggregates artefatos descobertos;`is_`readyé verdadeiro quando nenhum arquivo necessário está faltando após a análise das especificações. O módulo suporta binários congelados do PyInstaller `viasys.`frozenpath resolution.

6.2 Pré-requisito e ferramentas de empacotamento (`setup_cxinstaller_prereqs`)

Este componente fornece menus interativos emodos CLI para empacotamento de artefatos e instalação de pré-requisitos de host: on-line, air-gap ou detecção automática; embalagem multiaplicação, incluindo `combineCNC_NSO`. Ele preenche a árvore de artefatos esperada pelo Ansible e pela lógica verify-bundle.

6.3 Plano de automação visível

Composição:Isso ilustra a natureza de várias trilhas da pilha: ele importa caminhos de bootstrap diferentes do Kubernetes — refletindo que diferentes produtos têm como alvo caminhos de bootstrap diferentes do Kubernetes.

Funções (famílias representativas):`preinstall(SELinux, firewall, SSH, auxiliares de registro),k8s_install/rke2,deploy_nso,deploy_cnc,`deploy_cdg,`deploy_bpa,`postinstall, `auxiliares de desinstalação e renovação de certificado.` A propriedade e os testes são limites mais bem gerenciados; pastas de tarefas aninhadas implementam subfluxos de trabalho (por exemplo, HA NSO L3).

6.4 Estrutura dos controlos de validação (`validation_checks/`)

A estrutura fornece controle de política hierárquico(aplicativo → global → etapa → verificação individual),autodescoberta de cheques,aprimorado relatando a logs estruturados, e integração de JIRApcional. Os operadores executam fases de pré ou pós-implantação na mesma especificação usada para a instalação, alinhando a automação com gateways de qualidade adequados para a disciplina de alteração da empresa.

Escala indicativa em uma filial típica: na ordem detrinta verificações em BPA e NSO (as contagens devem ser confirmadas `comfazer verificações de validação de lista` no seu caixa).

6.5 Gerenciador de segredos do Vault (`scripts/vault_secrets_manager.py`)

Deriva as variáveis de cofre necessárias de especificações, prompts ou aceitações de senhas sob política e `emitsencryptedgroup_vars/all_secrets.`yamlplus um arquivo de senha de cofre para Ansible—reduzindo a incorporação de segredos ad-hoc nos manuais de atividades.

---

7. Padrões de implantação e tempos de execução

Padrão	Summary
Nativo (AlmaLinux / RHEL)	`SetPYTHONPATHeANSIBLE_CONFIG`; execute empacotamento, cofre, validação, orquestrador e manuais de atividades por guia de produto
Instalador baseado em Docker	`scripts/setup_installer.shandscripts/start_installer.`shwith montagens de host para artefatos grandes;
Air-gap	Embalagem em uma máquina conectada; pacote de transferência; Extrato no alvo; instalar com `– airgap`
criação de pacote macOS	`Use o` Mac `./setup_cxinstaller_prereqs.`pyon `python3` para preparar pacotes; a implantação de destino permanece orientada para Linux por documentos de projeto

---

8. Segurança, validação e observabilidade

8.1 Postura de segurança (intenção do projeto)

- Segredos:Preferem variáveis de grupo criptografadas do Ansible Vault; use os modos restritos do gerenciador de cofre, quando apropriado.
- Host do instalador:Tratar como um plano de controle de alta confiabilidade; restringir o acesso e monitorar.
- Artefatos:Proteger canais de aquisição; os processos organizacionais podem aumentar o verify-bundle com verificação criptográfica.
- Registro em log:Logs de aplicativo e Ansible `underdeploy/logs/`

8.2 Validação como porta operacional

Exemplo de chamada pré-implantação:

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

Com indicações facultativas:

- `-p <policy_file>`— use uma política de validação personalizada (o padrão é `validation_checks/validation_policies/default.yaml`)

- `-a <app>` — verificações de limite para um aplicativo específico (por exemplo, `cnc,nso,bpa,cdg`)
- `–report-file <caminho>` — grava um relatório de pré-verificação JSON independente

8.3 Disciplina da versão

Este é um modelo de aprovisionamento interno onde, para mais instalação de aplicativos da CISCO, o mesmo princípio pode ser seguido pela bifurcação do repositório.

---

9. Benefícios e resultados

Tema	Resultado
Tempo e trabalho	Menos etapas manuais; falhas detectadas nas fases de verificação de pacotes e validação, em vez de serem detectadas tardiamente nos instaladores Ansible ou de produtos
Consistência	Esquemas, funções e layout de artefatos compartilhados em todos os engajamentos reduzem as diferenças "flocos de neve"
Operações desconectadas	A transferência de pacotes documentada suporta redes regulamentadas sem downloads em tempo de execução
Governança	Relatórios de validação estruturados e ganchos JIRA opcionais suportam registros de alteração e acompanhamento
Extensibilidade	Limpar pontos de extensão:`ARTIFACT_DEFS`, manipuladores, novas funções/manuais, esquemas de intenção

As métricas quantitativas (duração da instalação, taxas de defeito) são específicas da organização; as equipes devem basear-se em runbooks herdados em topologias comparáveis.

---

10. Extensibilidade e manutenção

10.1 Adicionando tipos de artefatos

1. `ExtendARTIFACT_DEFS` (e rótulos, se necessário) `incx_deploy_orchestrator.py`.
2. Adicionar manipulador personalizado quando a nomeação não pode ser capturada apenas por padrões.
3. Atualize a lógica de empacotamento `insetup_cxinstaller_`prereqsquando os downloads forem automatizados.

10.2 Adicionando comportamento Ansible

Preferem novidades em funções coesas; introduza novasfunções quando os limites estiverem claros. Livros de reprodução com fios `viaimport_`playbook ou livros de jogos de entrada documentados. Keepsafe `defaultsingroup_vars/vars`.

10.3 Evolução intencional do gerador

Atualizar entradas `underintent-generate/schema/`e chatbot de esquemas YAML; verifique se os arquivos gerados correspondem aos nomes de arquivo esperados `porAPP_CONFIG`.

10.4 Considerações sobre o roteiro (ilustrativo)

- SBOM mais profunda ou integração de verificação de assinatura de imagem.
- Cobertura de validação expandida para cenários CNC/CDG.
- Referências de CI para linhas de especificações e verificações de sintaxe Ansible.

---

11. Conclusão

O CX Programmable Installer combina especificações declarativas, plano de controle Python para empacotamento e verificação e plano de automação Ansible para implantações escaláveis e repetíveis de produtos relacionados ao Crosswork em diversos modelos de infraestrutura e conectividade. Sua arquitetura intencionalmente separa intententfromexecution, aplica expectativas de artefato orientadas por dados onde for prático e incorpora fluxos de trabalho de validação e de compartimento adequados para a entrega corporativa. Para obter os anexos operacionais completos (tabelas do manual de atividades, matrizes de solução de problemas, matrizes de conectividade e referências de comandos estendidos), consulte o white paper interno complementar.

Referências e mapa da documentação

Documento	Caminho
Guia do operador	`LEIAME.md`
Liberar manual	`RELEASE_GUIDE.md`
Anexos da arquitetura interna	`docs/CX_INSTALLER_TECHNICAL_WHITE_PAPER_INTERNAL.md`
Docker	`SETUP_ONLINE_DOCKER.md`,`SETUP_AIRGAPPED_DOCKER.md`,`USAGE_DOCKER.md`

Documento	Caminho
(online / air-gap / uso)	
Estrutura de validação	`docs/validation_checks/README.md`
Gerenciador de cofre	`docs/scripts/VAULT_SECRETS_MANAGER.md`
Guias de produtos	`docs/nso.md,docs/bpa.md,docs/CNC_VCENTER_DEPLOYMENT_GUIDE.md,docs/CNC_OCP_DEPLOYMENT_GUIDE.m`
Gerador de intenção	`intent-generator/README.md`
Visão geral do Chatbot / fluxo de regras	`docs/HowItWorks.md`

Apêndice A — Layout do repositório - https://wwwin-github.cisco.com/CX-SAO-TOOLS/cx-installer (resumo)

```
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
```

Pontos focais de artefato pós-configuração (típico):`ansible_playbooks/files/artifacts/,files/bin/,files/charts/,files/images/.`

Apêndice B — CLI do Orchestrator (resumo)

Script:`cx_deploy_orchestrator.py`

Argumento	Descrição
`–app/-a`	`nso|crossworksuite|bpa`
`–spec/-s`	Caminho para a especificação YAML
`--etapa`	`generate-intent|verify-bundle|install`
`– verify- only`	Verificar o pacote; sair diferente de zero se não estiver pronto
`– dry- run`	Execução a seco, quando suportado
`– list- specs`	Listar especificações conhecidas

Ambiente (sessão típica):

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

Apêndice C — Glossário

Termo	Definição
Espec.	Especificação de usuário YAML: plataformas, aplicativos, topologia, caminhos, direitos
Intenção	YAML normalizado do gerador de intenção ou equivalente de autoria manual
Pacote	Árvore de instalação empacotada (geralmente tarball) para transferência de ar-gap
Orquestrador	`cx_deploy_orchestrator.py`— coordenação de verificação / intenção / instalação
Verificação de artefatos	Verificações do sistema de arquivos que os binários/imagens necessários existem por especificação
Cofre	Arquivo de variável criptografado Ansible Vault para segredos
NED	Pacote de driver de elemento de rede (NSO)
CFS/RFS	Conceitos de topologia de encaminhador de cluster NSO / encaminhador redundante
Air-gap	Ambiente sem acesso em tempo de instalação a pontos de extremidade de download de pacotes

---

Histórico de revisão do documento

Versão	Data	Notas
1.0	2026-03-27	White paper técnico inicial pronto para publicação (Enquadramento do instalador programável)

Sobre esta tradução
A Cisco traduziu este documento com a ajuda de tecnologias de tradução automática e humana para oferecer conteúdo de suporte aos seus usuários no seu próprio idioma, independentemente da localização.

Observe que mesmo a melhor tradução automática não será tão precisa quanto as realizadas por um tradutor profissional.

A Cisco Systems, Inc. não se responsabiliza pela precisão destas traduções e recomenda que o documento original em inglês (link fornecido) seja sempre consultado.