JENKINS

Le cockpit industriel du CI/CD

Automation Server | Pipeline as Code | Agents | Plugins | DevOps at scale

C'EST QUOI JENKINS ?

Serveur d'automatisation

Jenkins orchestre les tâches répétitives du delivery logiciel : build, tests, packaging, analyse qualité, publication et déploiement.

CI/CD extensible

Il peut servir de simple serveur d'intégration continue ou devenir un hub complet de livraison continue grâce à son écosystème de plugins.

Pipeline as Code

Le Jenkinsfile versionne le processus de build et de déploiement avec le code applicatif, comme une documentation exécutable.

POURQUOI JENKINS RESTE CENTRAL

20+
ans de maturité DevOps
Un outil historique, toujours vivant

100s
de plugins
SCM, cloud, test, sécurité, artefacts

1
orchestrateur flexible
Un moteur adaptable aux contextes complexes

Jenkins n'est pas seulement un outil de build. C'est une plateforme d'automatisation programmable, capable de relier des mondes hétérogènes : Git, Docker, Kubernetes, Maven, npm, Python, Ansible, Terraform, SonarQube, Nexus, Artifactory, AWS, Azure, GCP.

LE MODÈLE CI/CD EN UNE IMAGE

Idée clé

Jenkins transforme un changement de code en une chaîne reproductible de preuves : compilation, tests, sécurité, artefacts et déploiement contrôlé.

ARCHITECTURE GÉNÉRALE

Le contrôleur décide. Les agents exécutent. Le stockage Jenkins Home garde l’état vital du serveur.

CONTRÔLEUR VS AGENTS

Contrôleur

- Interface utilisateur
- Planification des jobs
- Gestion des plugins
- File d'attente d'exécution
- Configuration globale

A protéger fortement : il contient le coeur de Jenkins.

Agents

- Exécution des builds
- Isolation des outils
- Environnements spécialisés
- Scalabilité horizontale
- Nettoyage workspace

Bonne pratique : exécuter les builds hors du contrôleur.

JOBS, PROJETS ET DOSSIERS

Freestyle

Approche historique : configuration par écrans. Simple, mais moins versionnable et moins réutilisable.

Pipeline

Approche moderne : le processus est codé dans un Jenkinsfile, versionné et relu par l'équipe.

Multibranch

Jenkins détecte automatiquement branches et pull requests, puis applique le Jenkinsfile correspondant.

La montée en maturité naturelle : Freestyle -> Pipeline -> Multibranch Pipeline -> Shared Libraries.

PIPELINE AS CODE

```
pipeline {
  agent any
  stages {
    stage('Build') { steps { sh 'mvn -B package' } }
    stage('Test')  { steps { sh 'mvn test' } }
    stage('Deploy') { steps { sh './deploy.sh' } }
  }
}
```

Pourquoi c'est décisif

- Versionné avec le code
- Reproductible
- Relu en pull request
- Traçable
- Factorisable
- Portable entre équipes

Le pipeline devient un contrat technique entre dev, QA, sécurité et production.

DECLARATIVE VS SCRIPTED PIPELINE

Declarative

Syntaxe plus structurée et lisible. Idéale pour la majorité des équipes : stages, agent, environment, options, post, when, parallel.

Scripted

Plus proche du Groovy pur. Très flexible, mais plus difficile à gouverner. Utile pour les cas complexes et les bibliothèques internes.

Recommandation : déclaratif par défaut, scripted seulement quand nécessaire

ANATOMIE D'UN JENKINSFILE

agent

où exécuter

environment

variables

options

règles globales

stages

chaîne métier

steps

actions

post

nettoyage et notifications

Un bon Jenkinsfile raconte une histoire claire : préparer, construire, vérifier, publier, déployer, notifier.

LES STAGES : LA NARRATION DU DELIVERY

- Checkout : récupérer le code et identifier le commit exact.
- Build : produire un binaire, une image ou un paquet.
- Test : prouver que le changement ne casse pas le produit.
- Analyse : qualité, couverture, sécurité, dépendances.
- Publish : stocker des artefacts immuables.
- Deploy : promouvoir vers dev, staging ou production.

L'ÉCOSYSTÈME DE PLUGINS

SCM

Git, GitHub, GitLab, Bitbucket, webhooks.

Build tools

Maven, Gradle, npm, Python, Make, Docker.

Qualité

JUnit, cobertura, SonarQube, lint, reports.

Déploiement

SSH, Kubernetes, Helm, Ansible, Terraform.

Sécurité

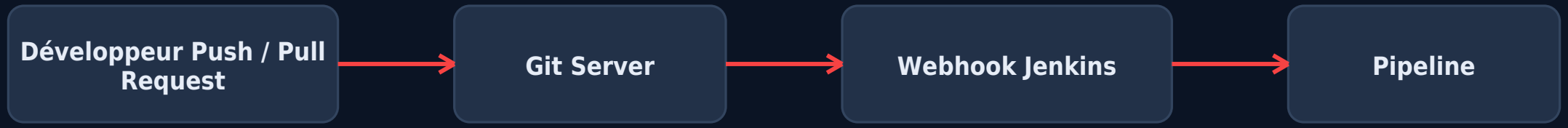
Credentials, RBAC, scans, signatures.

Notifications

Mail, Slack, Teams, dashboards.

Puissance : tout connecter. Risque : trop de plugins non gouvernés.

INTÉGRATION GIT ET WEBHOOKS



Effet recherché

A chaque changement significatif, Jenkins déclenche une preuve automatisée. Le feedback doit être rapide, lisible et actionnable.

CREDENTIALS & SECRETS

Ce que Jenkins sait gérer

- Token Git
- Clé SSH
- Mot de passe
- Certificat
- Secret text
- Identifiants cloud

Les secrets sont injectés au moment du build, jamais codés en dur dans le Jenkinsfile.

Règles d'ingénierie

- Moindre privilège
- Rotation régulière
- Séparation par environnement
- Pas de secrets dans les logs
- Pas de secrets dans les images Docker
- Audit des permissions Credentials.

SÉCURITÉ : LE TRIANGLE CRITIQUE

Jenkins doit savoir qui agit, ce qu’il a le droit de faire, et où le code non fiable peut s’exécuter sans menacer le contrôleur.

ISOLATION DU CONTRÔLEUR

Danger

Exécuter des builds sur le contrôleur expose Jenkins Home, les plugins, la configuration, les credentials et la stabilité globale.

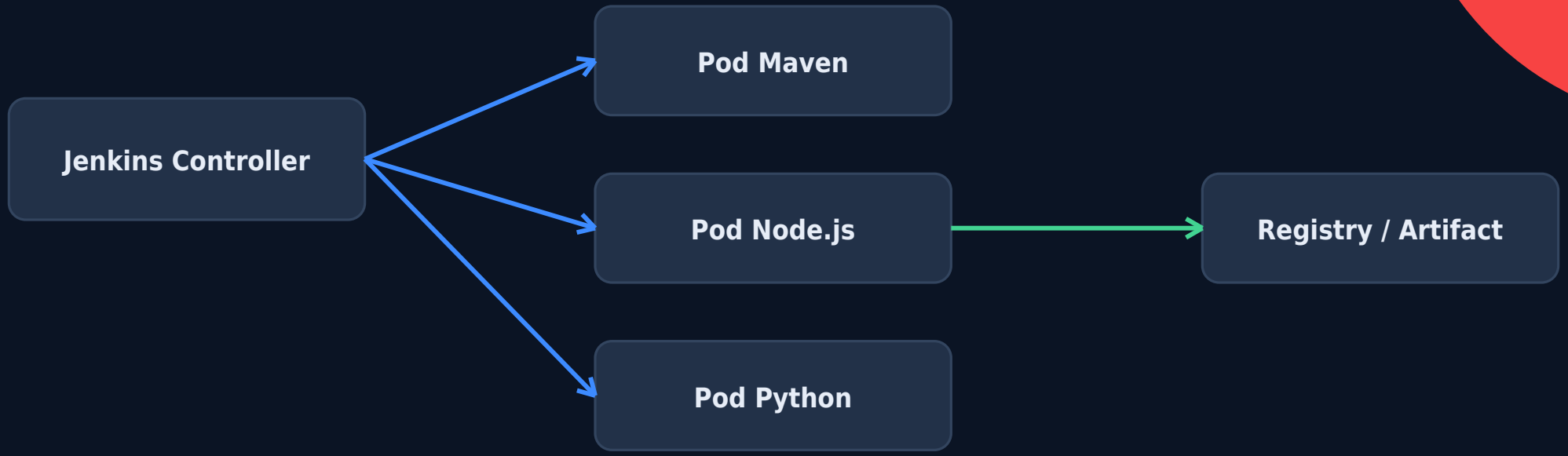
Réponse

Utiliser des agents dédiés. Le contrôleur orchestre, les agents construisent, testent et déploient.

Bénéfice

Scalabilité, sécurité, nettoyage plus simple, environnements spécialisés et meilleur contrôle opérationnel.

DOCKER & KUBERNETES AGENTS



Principe

Créer des agents éphémères à la demande : un build, un environnement propre, puis destruction. Très efficace pour éviter les workspaces pollués.

ARTEFACTS : PREUVE ET TRAÇABILITÉ

- Un artefact doit être immuable : même commit, même version, même contenu.
- Publier les rapports : tests, couverture, qualité, sécurité.
- Séparer build et déploiement : on déploie ce qui a été testé, pas ce qui vient d’être reconstruit.
- Nommer clairement : version, commit SHA, branche, date, environnement.

Formule simple

Source + Pipeline + Artefact + Logs = chaîne de preuve exploitable.

QUALITY GATES

Décider

Un pipeline mature ne se contente pas d'exécuter. Il décide : continuer, bloquer, notifier, demander validation ou rollback.

STRATÉGIES DE DÉPLOIEMENT

Push

Jenkins pousse le changement vers un serveur, un cluster ou un service cloud.

Promotion

Même artefact promu de dev à staging puis production.

Approval

Gate manuel avant production : validation métier, ops ou sécurité.

Blue/Green

Deux environnements, bascule rapide, rollback simple.

Canary

Déploiement progressif pour limiter l'impact d'un défaut.

OBSERVABILITÉ & LOGS

Ce qu'il faut voir

- Durée des stages
- Taux d’échec
- Agents saturés
- Files d’attente
- Plugins instables
- Tests flakys
- Erreurs réseau

Ce qu'il faut garder

- Console logs
- Build metadata
- Rapports de tests
- Artefacts
- Historique déploiements
- Audit actions utilisateur

Ce qu'il faut alerter

- Build rouge sur branche critique
- Production bloquée
- Credentials expirés
- Agent indisponible
- Espace disque bas

BACKUP : JENKINS HOME EST VITAL

A sauvegarder

- config.xml
- jobs/
- nodes/
- plugins + versions
- credentials
- secrets
- userContent
- fingerprints si utilisés
- historiques utiles

A tester

Un backup non restauré n'est qu'une hypothèse. Il faut tester la restauration sur un serveur isolé, avec plugins cohérents et secrets disponibles.

LTS VS WEEKLY

LTS

Ligne stable, plus conservatrice. Bon choix pour production, avec correctifs importants et sécurité backportés selon la politique Jenkins.

Weekly

Ligne rapide : nouvelles fonctionnalités et correctifs livrés plus vite. Utile pour tests, plugins et équipes capables d'absorber plus de changement.

Production : privilégier LTS + fenêtre de maintenance + rollback

GOUVERNANCE DES PLUGINS

- Installer peu, installer utile, documenter pourquoi.
- Épingler et tracer les versions installées.
- Tester les mises à jour sur staging avant production.
- Supprimer les plugins morts ou inutilisés.
- Surveiller les advisories de sécurité Jenkins.
- Limiter les plugins qui manipulent secrets, SCM, déploiements et permissions.

SCALABILITÉ : COMMENT GRANDIR PROPREMENT

Agents horizontaux

Ajouter des workers spécialisés et éphémères plutôt que gonfler le contrôleur.

Queues maîtrisées

Surveiller la file d’attente, les executors et la durée moyenne des builds.

Multi-controller

Séparer par domaine, criticité ou équipe quand un seul contrôleur devient ingouvernable.

Shared Libraries

Factoriser les patterns sans dupliquer 200 Jenkinsfiles fragiles.

Templates

Créer des standards : build Java, build Node, Docker, Helm, release.

ANTI-PATTERNS À ÉVITER

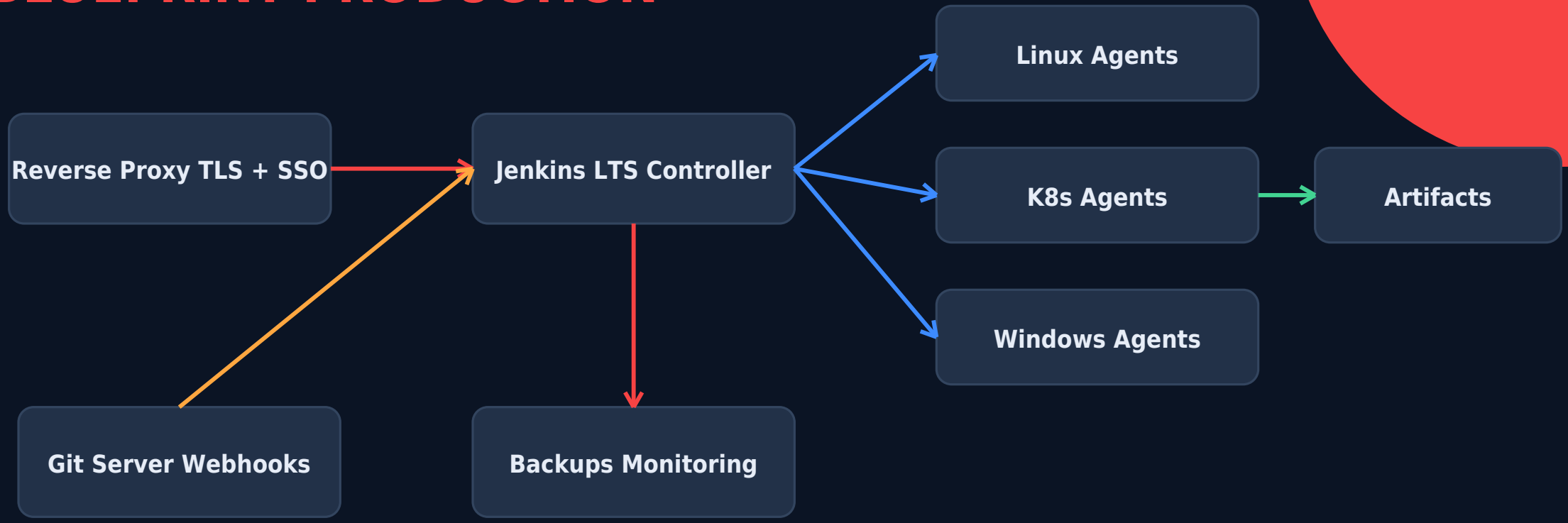
- Tout exécuter sur le contrôleur.
- Mettre les secrets dans le Jenkinsfile ou les logs.
- Installer des plugins sans gouvernance.
- Avoir 300 jobs Freestyle non versionnés.
- Reconstruire en production au lieu de promouvoir un artefact testé.
- Ignorer les backups, les upgrades et le nettoyage des workspaces.
- Masquer les échecs avec des scripts qui retournent toujours 0.

JENKINS FACE AUX CI MODERNES

Critère	Jenkins	GitHub/GitLab CI
Flexibilité	Très forte	Forte mais cadrée
Héritage existant	Excellent	Variable
Plugins	Très large	Marketplace intégrée
Maintenance	A assumer	Plus managée
Cloud-native	Possible via agents	Souvent natif
Contrôle interne	Très élevé	Dépend de la plateforme

Conclusion : Jenkins gagne quand l’environnement est hétérogène, historique, fortement personnalisé ou très contrôlé.

BLUEPRINT PRODUCTION



RUNBOOK OPÉRATIONNEL

Chaque jour

- Surveiller jobs rouges
- Agents offline
- Queue trop longue
- Espace disque

Chaque semaine

- Nettoyer workspaces
- Revoir plugins à risque
- Tester pipelines critiques
- Lire advisories

Chaque mois

- Upgrade staging
- Backup restore test
- Revue permissions
- Supprimer dette CI

Incident

Stopper la propagation, préserver les logs, identifier commit/pipeline/agent, rollback artefact, corriger puis écrire le postmortem.

JENKINS

Ce n'est pas un simple serveur de build.

C'est une usine de preuves, d'automatisation et de livraison logicielle.

Message final

Un Jenkins bien gouverné accélère. Un Jenkins abandonné devient une dette critique. La différence tient à l'architecture, aux agents, à la sécurité, aux backups et à la discipline Pipeline as Code.

Sources principales

- Jenkins official website: https://www.jenkins.io/
- Jenkins User Documentation - What is Jenkins?: https://www.jenkins.io/doc/
- Jenkins Pipeline Handbook: https://www.jenkins.io/doc/book/pipeline/
- Jenkinsfile documentation: https://www.jenkins.io/doc/book/pipeline/jenkinsfile/
- Pipeline syntax reference: https://www.jenkins.io/doc/book/pipeline/syntax/
- Using Jenkins agents: https://www.jenkins.io/doc/book/using/using-agents/
- Controller isolation: https://www.jenkins.io/doc/book/security/controller-isolation/
- Access control: https://www.jenkins.io/doc/book/security/access-control/
- Using credentials: https://www.jenkins.io/doc/book/using/using-credentials/
- Installing Jenkins: https://www.jenkins.io/doc/book/installing/
- Jenkins LTS release line: https://www.jenkins.io/download/lts/
- Jenkins weekly release line: https://www.jenkins.io/download/weekly/

Autres sources Jenkins officielles : backup, scaling, best practices, security.