GUIDE D'INTÉGRATION ET D'OBSERVABILITÉ

SYNOVEX

Guide d'intégration et d'observabilité

Matériel : Raspberry Pi 5

Éditions : Essential, Professional, Enterprise

Public : Architectes IT, DevOps, SRE, responsables de plateformes, intégrateurs



SYNOVEX — SIREN 884 713 009

Table des matières

Informations sur le document 3

Chapitre 1 — Architecture d'observabilité 6
1.1 Publication et interopérabilité des résultats 6
1.2 Vue d'ensemble 7

Chapitre 2 — Intégration aux plateformes d'observabilité 10
2.1 Une approche fondée sur les standards ouverts 10
2.2 Ingestion n'est pas intégration 13
2.3 Conventions de télémétrie 13
2.4 Configurer une destination d'export 20
2.5 Prometheus 23
2.6 Grafana 31
2.7 Datadog 33
2.8 Dynatrace 35
2.9 OpenTelemetry Collector du client 37
2.10 Nagios et Centreon 39
2.11 Journaux d'audit vers un SIEM 44
2.12 Comportement en cas d'indisponibilité d'une destination 44
2.13 Sécurité des intégrations 46
2.14 Choisir une stratégie d'intégration 47
2.15 Exploitation des alertes 48
2.16 Dépannage des exports 63

Chapitre 3 — Santé de la sonde 65
3.1 État de santé de la sonde 65
3.2 Surveillance de la santé de la sonde 68

Chapitre 4 — Maintenance, redondance et alertes 70
4.1 Notification, drainage et état de maintenance 70
4.2 Éviter les fausses alertes pendant une maintenance 74
4.3 Continuité de service et redondance 75

Chapitre 5 — Dépannage des intégrations 81
5.1 Destinations d'export et intégrations 81

Chapitre 6 — Références 84
6.1 Tableau des flux réseau 84
6.2 États et codes retour Nagios 85

SYNOVEX

Guide d'intégration et d'observabilité

Sonde de supervision applicative autonome pour Raspberry Pi 5

Information	Valeur
Document	Guide d'intégration et d'observabilité — identifiant SYN-DOC-05
Révision documentaire	B
Public	Architectes IT, DevOps, SRE, responsables de plateformes, intégrateurs
Produit	SYNOVEX
Matériel pris en charge	Raspberry Pi 5
Éditions couvertes	SYNOVEX Essential, SYNOVEX Professional, SYNOVEX Enterprise
Éditeur	SYNOVEX — SIREN 884 713 009
Documentation en ligne	`https://docs-synovex.incitypark.fr` (après l'annonce de sa mise en service)
Langue	Français

Mentions légales

Éditeur

Cette documentation est éditée par SYNOVEX, SIREN 884 713 009. SYNOVEX est aussi le nom commercial de la gamme de sondes de supervision décrite.

Propriété intellectuelle et diffusion

© SYNOVEX. Tous droits réservés. Les textes, schémas, illustrations et exemples de cette documentation sont protégés par le droit d'auteur.

Cette documentation est réservée aux clients titulaires d'un droit d'utilisation du produit SYNOVEX. Le client peut la reproduire et la diffuser à l'intérieur de son organisation, et la communiquer aux prestataires qui interviennent pour son compte, pour les seuls besoins de l'installation, de l'exploitation et de la maintenance de ses sondes. Toute autre reproduction, diffusion ou mise à disposition, totale ou partielle, requiert l'accord écrit préalable de SYNOVEX.

Les fichiers d'exemple fournis avec la documentation (scénarios, règles d'alerte, tableau de bord, schémas JSON) peuvent être copiés et adaptés par le client pour l'exploitation de ses sondes.

Limitation de responsabilité

Les informations de cette documentation peuvent être modifiées sans préavis. L'éditeur s'efforce d'en assurer l'exactitude, mais ne peut être tenu responsable des erreurs ou omissions, ni des conséquences de l'utilisation des exemples fournis. Les exemples de configuration et de scénarios sont **fictifs** et doivent être adaptés à votre environnement, puis validés, avant tout usage. Cette documentation ne constitue pas un engagement contractuel : les engagements de l'éditeur, notamment de garantie, de support et de maintenance, résultent exclusivement de votre contrat.

Marques

SYNOVEX est un nom commercial de l'éditeur. Raspberry Pi est une marque de Raspberry Pi Ltd. Orange Pi est une marque de Shenzhen Xunlong Software Co., Ltd. Google Chrome est une marque de Google LLC ; Chromium et Puppeteer sont des projets open source de Google LLC. Java, OpenJDK, Oracle et MySQL sont des marques d'Oracle et/ou de ses filiales. Node.js est une marque de l'OpenJS Foundation. Prometheus et OpenTelemetry sont des marques de The Linux Foundation. Grafana est une marque de Grafana Labs. Datadog est une marque de Datadog, Inc. Dynatrace est une marque de Dynatrace LLC. Nagios est une marque de Nagios Enterprises, LLC. Centreon est une marque de Centreon. Microsoft et SQL Server sont des marques du groupe Microsoft. PostgreSQL est une marque de PostgreSQL Community Association of Canada. MariaDB est une marque de MariaDB plc. Les autres noms de produits et de sociétés cités peuvent être des marques de leurs propriétaires respectifs. Leur mention dans cette documentation n'implique aucun partenariat ni aucune approbation.

Logiciels tiers

La sonde intègre des logiciels libres et des composants tiers, distribués selon leurs propres licences. Les principaux composants et leurs licences sont listés au § 11.1 de la référence technique et API ; l'inventaire complet des composants accompagne chaque version du logiciel.

Matériel et fin de vie

Le Raspberry Pi 5 et ses accessoires officiels sont fabriqués par Raspberry Pi Ltd, qui en publie les déclarations de conformité. Avec l'offre microSD préconfigurée, le matériel est choisi et assemblé par vos soins. Avec l'offre boîtier complet, les informations de conformité applicables figurent dans les documents joints au produit. En fin de vie, la sonde relève de la collecte des déchets d'équipements électriques et électroniques : ne la jetez pas avec les déchets ordinaires et confiez-la à une filière de collecte agréée, après avoir effacé ses données (§ 8.13 du guide d'installation et d'administration).

À propos de ce guide

Ce guide explique comment intégrer les résultats de SYNOVEX à vos plateformes d'observabilité et d'alerte : architecture d'observabilité, conventions de télémétrie OpenTelemetry (métriques, traces, journaux, attributs), protocoles OTLP/HTTP et OTLP/gRPC, destinations d'export, Prometheus et Grafana, Datadog, Dynatrace, Nagios et Centreon, journaux d'audit vers un SIEM, comportement en cas d'indisponibilité d'une destination, maintenance des sondes et mises en sourdine, supervision depuis plusieurs sondes et plusieurs sites.

Les conventions de télémétrie et le protocole OTLP relèvent du standard OpenTelemetry ; les procédures propres à chaque plateforme sont présentées séparément, plateforme par plateforme.

La bibliothèque documentaire SYNOVEX

La documentation SYNOVEX se compose d'un manuel technique complet et de six guides spécialisés. Tous sont produits à partir des mêmes sources : un passage repris dans plusieurs documents y est rigoureusement identique.

Document	Identifiant	Public
Manuel technique complet	SYN-DOC-00	Tous publics : référence intégrale
Guide de démarrage rapide	SYN-DOC-01	Techniciens d'installation, nouveaux clients
Guide d'installation et d'administration	SYN-DOC-02	Administrateurs système et réseau, équipes de sécurité
Guide d'utilisation et de supervision	SYN-DOC-03	Équipes d'exploitation et de supervision
Guide de développement des scénarios	SYN-DOC-04	Développeurs, intégrateurs, équipes de qualité
Guide d'intégration et d'observabilité	SYN-DOC-05	Architectes, DevOps, SRE
Référence technique et API	SYN-DOC-06	Intégrateurs avancés, architectes, développeurs

Renvois. Dans un guide, un renvoi sans autre mention désigne une section, un chapitre ou une figure du même guide, avec la numérotation de ce guide. Lorsque le sujet est traité dans un autre guide, le renvoi nomme ce guide, par exemple « § 6.3 du guide d'installation et d'administration ». La numérotation est alors celle du guide cité. Chaque guide se lit ainsi sans le manuel technique complet, qui réunit l'intégralité de la documentation sous sa propre numérotation.

Services en ligne. L'ensemble de la bibliothèque est également publié sur `https://docs-synovex.incitypark.fr`, avec une recherche et une navigation par profil. Ce site, le portail de support et le service de distribution des mises à jour de l'éditeur sont accessibles après l'annonce de leur mise en service. Avant cette annonce, ou si l'un de ces services ne répond pas, utilisez le canal de contact prévu par votre contrat. Les adresses citées dans ce guide s'entendent sous cette réserve.

Chapitre 1 — Architecture d'observabilité

1.1 Publication et interopérabilité des résultats

La sonde ne se contente pas de conserver ses résultats : elle les publie selon des formats standard, pour que vous les exploitiez dans vos outils. Les mêmes résultats alimentent trois canaux, qui peuvent être utilisés simultanément.

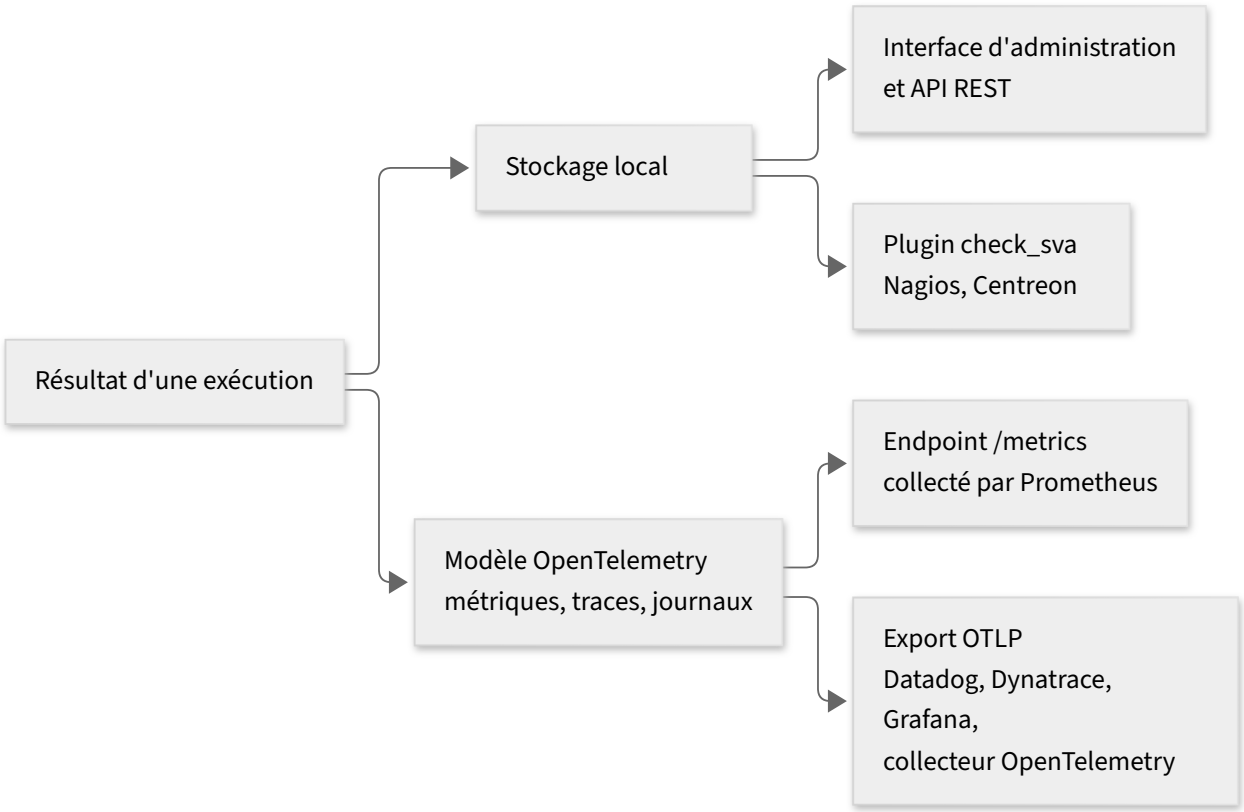


Schéma 1-1 — Publication des résultats : stockage local, API et exports

Canal	Principe	Plateformes
Export OTLP (envoi)	La sonde envoie métriques, traces et journaux selon le protocole OpenTelemetry (principalement OTLP/HTTP)	Collecteur OpenTelemetry, Datadog (via son agent ou en direct), Dynatrace, Grafana Cloud et toute plateforme compatible OTLP
Endpoint Prometheus (collecte)	La sonde expose ses métriques au format Prometheus ; votre serveur Prometheus vient les collecter	Prometheus, et par son intermédiaire Grafana
Plugin de contrôle d'état (interrogation)	Un plugin installé sur votre serveur Nagios ou Centreon interroge l'API de la sonde et renvoie l'état d'un contrôle selon les conventions Nagios	Nagios, Centreon et outils compatibles

OpenTelemetry est un standard ouvert, soutenu par l'ensemble des acteurs de l'observabilité, qui définit comment décrire et transmettre trois types de données : les **métriques** (valeurs numériques mesurées dans le temps, par exemple la durée d'un parcours), les **traces** (déroulement détaillé d'une opération, ici une exécution et ses étapes) et

les **journaux** (événements horodatés, par exemple l'échec d'une étape avec son message). Le choix de ce standard évite de développer un connecteur spécifique à chaque plateforme.

> **Attention** — Le transport standardisé garantit que vos plateformes **reçoivent** les données. Il ne crée pas automatiquement les tableaux de bord ni les règles d'alerte. Ces éléments se configurent dans chaque plateforme ; le chapitre 2 décrit la démarche et les modèles fournis.

La publication est découplée de l'exécution : si une destination d'export est indisponible, la sonde poursuit ses contrôles, conserve les résultats localement et réessaie l'envoi dans des limites définies. Les conséquences d'une indisponibilité prolongée sur chaque type de données sont décrites au chapitre 2.

1.2 Vue d'ensemble

La sonde est installée au point d'observation choisi : dans le réseau de votre organisation, à proximité des utilisateurs ou des applications dont vous souhaitez vérifier le service, ou à l'extérieur, sur un site distant, chez un client, un partenaire ou un sous-traitant, ou dans une autre région géographique (§ 1.2.1). Le schéma ci-dessous présente le cas d'une sonde installée dans le réseau de l'entreprise ; les mêmes catégories d'échanges s'appliquent aux autres emplacements. Elle échange avec quatre catégories d'interlocuteurs :

- les **ressources supervisées** : sites web, API, bases de données, serveurs, équipements, sur lesquels elle exécute ses contrôles ;
- les **administrateurs**, qui la configurent et consultent ses résultats depuis un navigateur ou par l'API ;
- les **plateformes de supervision et d'observabilité** (Prometheus, Grafana, Datadog, Dynatrace, Nagios, Centreon…), qui reçoivent ou collectent ses résultats ;
- les **services d'infrastructure** indispensables : DNS, heure (NTP) et, si vous les activez, dépôt de mises à jour de l'éditeur et proxy sortant.

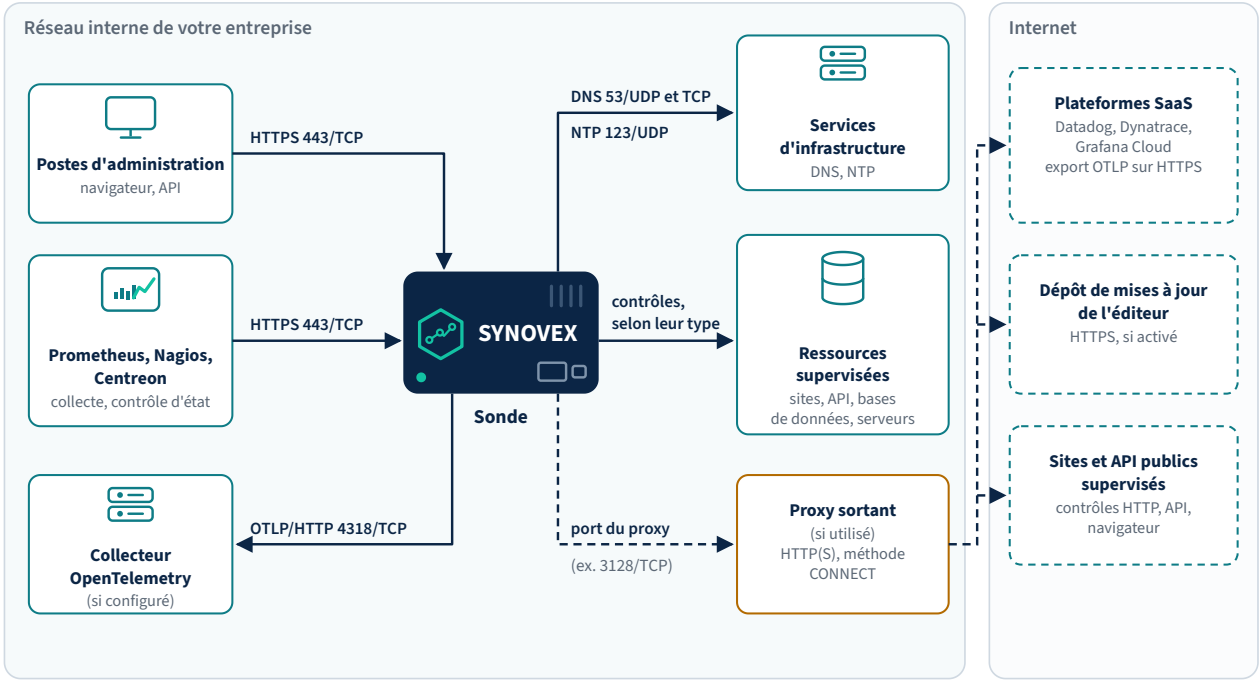


Figure 1-1 — Schéma d'implantation type : sonde dans le réseau interne, plateformes de supervision, accès Internet sortant via proxy

Les flèches indiquent le sens d'établissement des connexions. Les flux en pointillés sont facultatifs et dépendent de votre configuration. La sonde n'envoie aucune donnée de télémétrie à l'éditeur : ses seuls échanges avec lui sont la vérification et le téléchargement des mises à jour, si vous les activez.

1.2.1 Où placer la sonde ?

L'emplacement de la sonde détermine ce qu'elle mesure. Une sonde voit le service tel qu'il est perçu depuis son point d'observation : à travers les mêmes résolveurs DNS, pare-feu, proxys, opérateurs et liens que les utilisateurs ou les applications de ce réseau. Elle peut être installée dans tout environnement qui dispose des accès réseau nécessaires aux contrôles configurés, à l'intérieur comme à l'extérieur de votre organisation.

Emplacement	Ce que la sonde mesure	Usage typique
Réseau des utilisateurs d'un site	Le service tel que le perçoivent les collaborateurs de ce site, y compris les liens intersites et le proxy	Applications métier internes, services SaaS utilisés par l'entreprise
Centre de données, à proximité des serveurs	Le service sans les effets du réseau étendu	Distinguer un problème applicatif d'un problème de réseau
Zone d'accès externe ou réseau isolé vers Internet	Le service tel qu'il est vu depuis l'extérieur, en passant par l'accès Internet	Sites et API publics de l'entreprise
Site distant ou agence : filiale, entrepôt, magasin	Le service tel que le perçoivent les utilisateurs de ce site, à travers son réseau local, son accès opérateur et ses liaisons vers vos centres de données	Vérifier la disponibilité et les performances des applications centrales depuis chaque site
Réseau d'un client, avec son accord	Le service tel que le perçoivent les utilisateurs du client, à travers leur réseau, leur proxy et leur pare-feu	Vérifier que les services proposés par votre organisation sont réellement accessibles et fonctionnels depuis l'environnement du client
Réseau d'un partenaire ou d'un sous-traitant, avec son accord	Les échanges et les services tels qu'ils sont accessibles depuis cette organisation	Superviser un portail partenaire, une interconnexion ou des échanges entre organisations
Différentes régions géographiques	Le service tel qu'il est perçu dans chaque pays ou région : opérateur, liens internationaux, résolution DNS locale, éventuel réseau de diffusion de contenu (CDN)	Mesurer la latence et les temps de réponse par localisation, repérer une dégradation propre à une région ou à un opérateur

Plusieurs sondes placées à des endroits différents fournissent des **indices de localisation** d'un défaut. Si le même contrôle échoue depuis un site et réussit depuis le centre de données, l'écart oriente l'analyse vers ce qui distingue les deux points d'observation, souvent le chemin réseau du site. De même, un contrôle qui réussit depuis Paris mais échoue ou ralentit depuis Hô Chi Minh-Ville oriente l'analyse vers le chemin international, l'opérateur ou les services régionaux. Ces indices ne suffisent pas à conclure : un composant réseau commun peut affecter tous les sites, et une réponse qui dépend de la région, de l'identité du compte de test, du répartiteur de charge ou de la résolution DNS peut n'affecter qu'un seul point d'observation alors que la cause est applicative. Avant d'attribuer l'origine à l'application ou au réseau, comparez les chemins, les cibles effectivement jointes, les résolutions DNS, les politiques appliquées (proxy, filtrage) et les résultats détaillés des sondes. Chaque sonde rend compte de ce qu'elle observe depuis son emplacement ; elle n'identifie pas elle-même la cause d'une dégradation réseau, et chaque sonde fonctionne indépendamment des autres. Chaque sonde se voit attribuer un **nom de site** lors de sa mise en service (chapitre 4 du guide d'installation et d'administration), repris dans tous ses résultats exportés : c'est lui qui permet de comparer les sondes entre elles dans vos tableaux de bord et vos alertes.

Les prérequis d'une sonde installée hors de vos locaux dépendent des services à superviser et de leur accessibilité depuis ce réseau. Prévoyez en particulier les points suivants :

- **accord du propriétaire du réseau** : installez-la uniquement avec l'autorisation écrite de l'organisation qui exploite le réseau, et convenez avec elle des cibles contrôlées, des flux ouverts et de la personne qui intervient physiquement sur l'appareil ;
- **transmission des résultats** : privilégiez l'envoi par OTLP (chapitre 2), qui n'utilise que des connexions sortantes, vers une plateforme joignable depuis ce site. La collecte par Prometheus ou par le plugin Nagios suppose au contraire que votre plateforme puisse joindre la sonde ;
- **administration** : l'interface et l'API ne doivent pas être exposées sur Internet (§ 1.7 de la référence technique et API). Administrez la sonde depuis le réseau où elle est installée ou au travers d'un moyen d'accès sécurisé convenu avec l'exploitant de ce réseau. La sonde ne requiert aucune liaison permanente avec votre organisation pour exécuter ses contrôles ;
- **coupure de liaison** : la sonde continue d'exécuter ses contrôles et de conserver leur historique localement lorsque la liaison vers vos plateformes est interrompue. Les résultats exportés pendant la coupure ne sont pas renvoyés automatiquement ; l'historique local reste consultable (§ 2.12) ;
- **heure et réglementation locales** : vérifiez qu'un serveur NTP est joignable depuis le site et que l'installation respecte les règles locales applicables à l'équipement et aux données collectées.

Chapitre 2 — Intégration aux plateformes d'observabilité

La sonde fonctionne de manière autonome : elle exécute ses contrôles, conserve ses résultats et les présente dans son interface d'administration, sans dépendre d'aucun service extérieur. Dans une organisation qui dispose déjà d'outils de supervision, il est toutefois souhaitable que les résultats de la sonde y apparaissent. Les équipes d'exploitation retrouvent ainsi les parcours applicatifs à côté des métriques d'infrastructure, reçoivent les alertes par leurs canaux habituels et conservent l'historique dans leur outil de référence.

Ce chapitre explique comment raccorder la sonde aux plateformes suivantes :

- Prometheus et l'écosystème Grafana ;
- Datadog ;
- Dynatrace ;
- un OpenTelemetry Collector déployé chez vous ;
- Nagios et Centreon ;
- un SIEM, pour les journaux d'audit.

Il présente d'abord les principes communs (§ 2.1 à § 2.4), puis une procédure par plateforme (§ 2.5 à § 2.11). Il se termine par l'exploitation des alertes, notamment pendant la maintenance des sondes (§ 2.15), et par le dépannage des exports (§ 2.16).

> **Information** — Les adresses, identifiants d'environnement et clés figurant dans ce chapitre sont fictifs. `abc12345` désigne un identifiant d'environnement Dynatrace fictif. Les noms de domaine en `example.com` et `example.internal` sont réservés à la documentation. Les fonctions des plateformes tierces citées ici reflètent leur documentation officielle à la date de rédaction. Ces offres évoluent : vérifiez les points signalés dans la documentation de l'éditeur concerné avant toute mise en production.

2.1 Une approche fondée sur les standards ouverts

2.1.1 Pourquoi OpenTelemetry

OpenTelemetry est un standard ouvert, porté par la Cloud Native Computing Foundation, qui définit comment produire et transmettre des données de supervision. Il est pris en charge par la quasi-totalité des plateformes d'observabilité du marché. **OTLP** (*OpenTelemetry Protocol*) est le protocole de transmission associé.

La sonde produit toutes ses données de supervision au format OpenTelemetry et les transmet en OTLP. Ce choix évite un connecteur propriétaire par plateforme : se raccorder à Datadog, à Dynatrace ou à Grafana revient à **paramétrer une destination d'export** (adresse, authentification, quelques options), et non à installer un module spécifique.

En complément, la sonde expose ses métriques au **format Prometheus**, standard de fait de la collecte de métriques par interrogation. Pour Nagios et Centreon, elle fournit un plugin conforme à la convention des plugins Nagios.

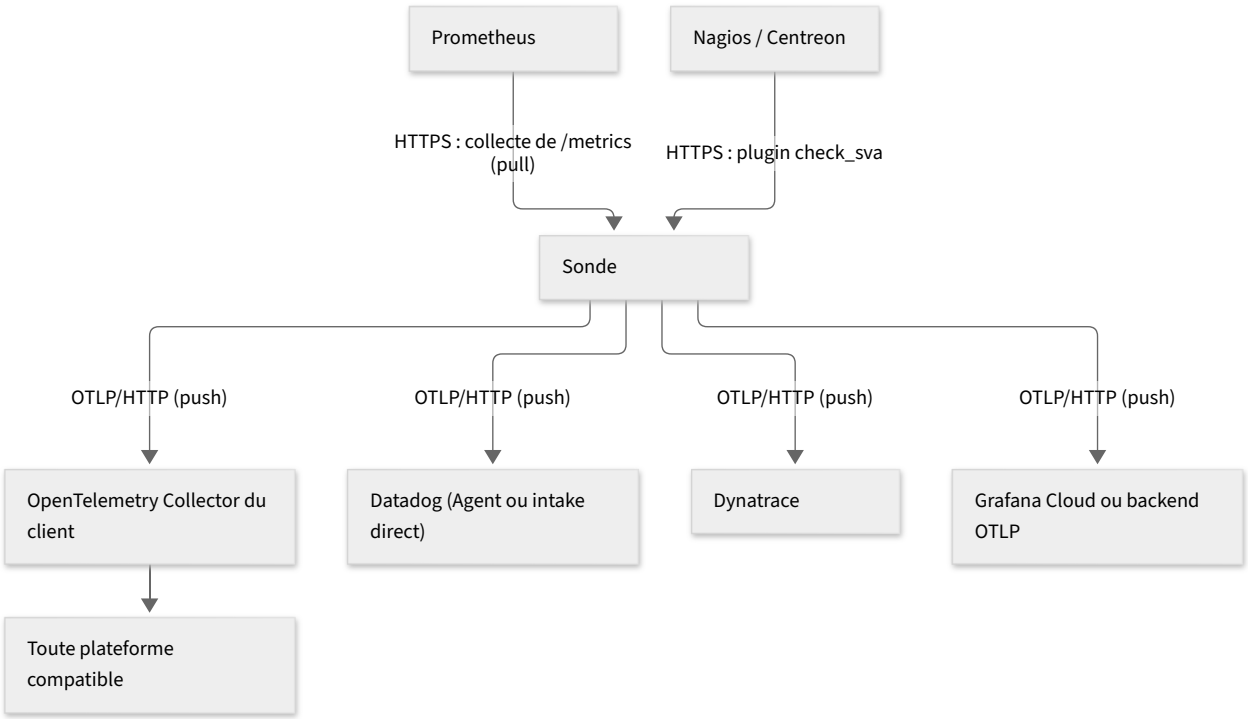


Schéma 2-1 — Envoi des données vers les plateformes d'observabilité

2.1.2 Les trois types de données

OpenTelemetry distingue trois types de données, appelés **signaux**. La sonde produit les trois.

Signal	Nature	Ce que la sonde y place	Question à laquelle il répond
Métriques	Valeurs numériques agrégées dans le temps	États des contrôles, nombre d'exécutions, durées, mesures réseau, indicateurs métier, santé de la sonde	« Combien ? À quelle fréquence ? Quelle tendance ? »
Traces	Déroulement détaillé d'une opération, découpé en **spans** (segments horodatés imbriqués)	Une trace par exécution ; un span par étape d'un scénario	« Où, dans ce parcours précis, le temps a-t-il été passé et où a-t-il échoué ? »
Logs	Événements horodatés, structurés	Fin d'exécution avec erreur, changement d'état, erreurs JavaScript des pages, sortie des plugins, opérations d'administration	« Que s'est-il passé exactement ? »

Les trois signaux portent les mêmes identifiants : identifiant du contrôle, identifiant d'exécution, identifiant de trace. Dans une plateforme qui les met en relation, on passe d'une alerte (métrique) à l'exécution concernée (trace), puis au message d'erreur (log).

2.1.3 Envoi (push) et interrogation (pull)

Deux modes de transmission coexistent :

- **Envoi (push)** : la sonde transmet périodiquement ses données à une destination. C'est le mode OTLP. La connexion est **sortante**, initiée par la sonde : aucune ouverture de port vers la sonde n'est nécessaire, ce qui

convient aux plateformes SaaS.

- **Interrogation (pull)** : un outil vient lire les données sur la sonde. C'est le mode de Prometheus (lecture de `/metrics`) et de Nagios (plugin `check_sva`). La connexion est **entrante** vers la sonde, sur le port HTTPS 443, depuis le réseau d'administration.

2.1.4 OTLP/HTTP et OTLP/gRPC

OTLP existe en deux variantes de transport :

Variante	Principe	Prise en charge par la sonde	Remarques
OTLP/HTTP (encodage protobuf)	Requêtes HTTP POST vers `/v1/metrics`, `/v1/traces`, `/v1/logs`	**Mode par défaut**	Compatible avec les proxys HTTP et les pare-feu applicatifs ; seul mode accepté par Dynatrace, par l'intake direct de Datadog et par le récepteur OTLP de Prometheus
OTLP/HTTP (encodage JSON)	Idem, en JSON	Disponible	Plus volumineux ; utile surtout pour le débogage
OTLP/gRPC	Protocole gRPC sur HTTP/2	Disponible	Accepté par l'OpenTelemetry Collector et l'Agent Datadog ; plus délicat à faire traverser certains proxys

> ✓ **Conseil** — Conservez OTLP/HTTP avec l'encodage protobuf, sauf raison particulière. C'est la variante la plus largement acceptée, et elle traverse sans difficulté les proxys sortants (chapitre 5 du guide d'installation et d'administration).

2.1.5 Temporalité des métriques : cumulative ou delta

Les compteurs et les histogrammes peuvent être transmis de deux manières, appelées **temporalités**. La distinction est importante, car chaque plateforme impose la sienne.

- **Cumulative** : la sonde transmet le total depuis son démarrage. Exemple : « 1 250 exécutions depuis le démarrage », puis « 1 262 exécutions » une minute plus tard. La plateforme calcule elle-même la différence.
- **Delta** : la sonde transmet uniquement l'accroissement depuis le dernier envoi. Exemple : « 12 exécutions pendant la dernière minute ».

Plateforme	Temporalité exigée
Prometheus (collecte `/metrics` ou récepteur OTLP)	Cumulative
OpenTelemetry Collector, Grafana Cloud	Cumulative (le Collector peut convertir)
Agent Datadog (récepteur OTLP)	Cumulative
Intake OTLP direct de Datadog	**Delta uniquement**
Dynatrace (API OTLP)	**Delta uniquement**

La sonde gère la temporalité **destination par destination**. Elle peut donc alimenter simultanément Prometheus en cumulatif et Dynatrace en delta, sans composant intermédiaire. Les préréglages de plateforme (§ 2.4.2) choisissent automatiquement la bonne valeur.

2.2 Ingestion n'est pas intégration

> **Attention** — Raccorder la sonde à une plateforme garantit que les données y **arrivent** (ingestion). Cela ne crée **pas automatiquement** de tableaux de bord, d'alertes ni de configuration propre à la plateforme (intégration).

Concrètement, après la configuration d'une destination :

- les métriques, traces et logs de la sonde sont disponibles dans la plateforme et interrogeables par ses outils habituels : explorateurs, langages de requête, tableaux de bord personnalisés ;
- les résultats de la sonde **n'apparaissent pas** dans les modules natifs de supervision synthétique de ces plateformes (par exemple les modules « Synthetic Monitoring » de Datadog ou de Dynatrace). Ces modules ne gèrent que les tests exécutés par leurs propres robots ;
- **aucune alerte n'est créée** : il vous appartient de définir les alertes dans la plateforme, à partir des exemples de ce chapitre ;
- **aucun tableau de bord n'est créé** : vous pouvez en construire un, ou importer les modèles fournis par l'éditeur.

Modèle fourni par l'éditeur	Format	Disponibilité
Tableau de bord Grafana	JSON d'import Grafana, source de données Prometheus	Exemple fourni avec la documentation, prêt à importer : `exemples/grafana/sva-tableau-de-bord.json` (§ 2.6.3)
Règles d'alerte Prometheus	Fichier de règles YAML (`groups:`)	Exemple fourni avec la documentation, prêt à charger : `exemples/prometheus/sva-regles-alerte.yml` (§ 2.5.5)
Mise en sourdine des fenêtres de maintenance	Fichier de configuration Alertmanager (`time_intervals`, `mute_time_intervals`)	Exemple fourni avec la documentation, à adapter à vos sondes et à leurs fenêtres : `exemples/prometheus/alertmanager-maintenance.yml` (§ 2.15.2)
Requêtes de moniteurs Datadog	Requêtes à saisir dans Datadog	Exemples de ce chapitre (§ 2.7.3) ; aucun modèle à importer
Requêtes Dynatrace	Requêtes DQL à saisir dans Dynatrace	Exemples de ce chapitre (§ 2.8.3) ; aucun modèle à importer

Les fichiers d'exemple sont livrés avec la documentation, accompagnés chacun d'une notice (`README.md`). Ils utilisent uniquement les métriques décrites au § 2.3. Leurs seuils sont indicatifs : adaptez-les à vos engagements de service avant la mise en production.

2.3 Conventions de télémétrie

Cette section décrit les données produites par la sonde. Elle constitue la référence pour construire requêtes, tableaux de bord et alertes.

> **Information** — Le préfixe `sva.` des noms de métriques et d'attributs (`sva_` au format Prometheus) est **définitif et stable** : il ne dépend pas du nom commercial du produit et ne changera pas lors d'une évolution de la marque. Vos tableaux de bord et règles d'alerte restent donc valables d'une version à l'autre. Il se distingue du préfixe `probe_` du *blackbox exporter* de Prometheus.

2.3.1 Identité de la sonde (attributs de ressource)

Chaque envoi est accompagné d'attributs qui identifient la sonde émettrice. Ces attributs ne figurent pas sur chaque point de mesure : ils décrivent l'émetteur.

Attribut	Exemple	Origine
`service.name`	`sva-agent`	Fixe
`service.version`	`4.2.1`	Version installée
`service.instance.id`	`sva-7f3c9a`	Identifiant unique généré au premier démarrage
`host.name`	`sonde-paris-01`	Nom d'hôte configuré
`host.arch`	`arm64`	Détecté
`deployment.environment.name`	`production`	Paramètre de la sonde
`sva.site`	`paris-dc1`	Paramètre de la sonde : emplacement physique ou logique
`sva.edition`	`enterprise`	Licence

✓ **Conseil** — Renseignez soigneusement le paramètre `site` de chaque sonde (chapitre 4 du guide d'installation et d'administration). C'est lui qui permet, dans une plateforme multi-sondes, de distinguer une panne applicative (toutes les sondes en échec) d'un problème réseau local (une seule sonde en échec).

2.3.2 Catalogue des métriques

Les noms ci-dessous sont les noms OpenTelemetry. Les unités suivent la convention UCUM utilisée par OpenTelemetry : `s` pour seconde, `By` pour octet, `1` pour une grandeur sans dimension, et des annotations entre accolades comme `{run}`.

Résultats des contrôles

Métrique	Instrument	Unité	Attributs	Usage
`sva.check.runs`	Compteur	`{run}`	`sva.check.id`, `sva.check.type`, `sva.check.status`, `error.type`	Nombre d'exécutions par état ; taux d'échec ; disponibilité
`sva.check.duration`	Histogramme	`s`	`sva.check.id`, `sva.check.type`, `sva.check.status`	Temps de réponse ; percentiles
`sva.check.status`	Jauge	`{state}`	`sva.check.id`, `sva.check.type`	Dernier état : 0 = `OK`, 1 = `WARNING`, 2 = `CRITICAL`, 3 = `UNKNOWN`
`sva.check.up`	Jauge	`{state}`	`sva.check.id`, `sva.check.type`	1 si `OK` ou `WARNING`, 0 sinon
`sva.step.duration`	Histogramme	`s`	`sva.check.id`, `sva.step.name`, `sva.step.status`	Temps par étape d'un parcours
`sva.step.failures`	Compteur	`{run}`	`sva.check.id`, `sva.step.name`, `error.type`	Étape à laquelle les parcours échouent

Mesures techniques

Métrique	Instrument	Unité	Attributs	Usage
`sva.http.phase.duration`	Histogramme	`s`	`sva.check.id`, `sva.http.phase` (`dns`, `connect`, `tls`, `ttfb`, `transfer`)	Localiser une lenteur HTTP
`sva.tls.certificate.expiry`	Jauge	`s`	`sva.check.id`	Durée restante avant l'expiration du certificat
`sva.icmp.rtt`	Jauge	`s`	`sva.check.id`	Temps aller-retour moyen de la dernière série
`sva.icmp.packet_loss`	Jauge	`1`	`sva.check.id`	Taux de perte, de 0 à 1
`sva.dns.lookup.duration`	Histogramme	`s`	`sva.check.id`	Durée de résolution DNS
`sva.indicator.value`	Jauge	`{value}`	`sva.check.id`, `sva.indicator.name`	Indicateurs métier issus des contrôles de base de données (chapitre 5 du guide d'utilisation et de supervision)
`sva.plugin.perfdata`	Jauge	`{value}`	`sva.check.id`, `sva.perfdata.label`	Données de performance de plugins Nagios, déclarées exportables

Santé de la sonde

Métrique	Instrument	Unité	Attributs	Usage
`sva.agent.up`	Jauge	`{state}`	—	Battement de cœur : son absence signale une sonde arrêtée ou injoignable
`sva.agent.scheduler.lag`	Histogramme	`s`	`sva.exec.class`	Retard de démarrage des exécutions (saturation)
`sva.agent.queue.depth`	Jauge	`{job}`	`sva.exec.class`	Exécutions en attente
`sva.export.failures`	Compteur	`{batch}`	`sva.exporter.id`, `sva.signal`	Échecs d'envoi vers une destination
`sva.export.dropped`	Compteur	`{item}`	`sva.exporter.id`, `sva.signal`	Données abandonnées après échecs répétés
`sva.storage.usage`	Jauge	`By`	`sva.storage.area` (`db`, `artifacts`)	Occupation du stockage
`sva.storage.recoveries`	Compteur	`{recovery}`	—	Récupérations automatiques de la base au démarrage (chapitre 4)
`sva.browser.launches`	Compteur	`{launch}`	`sva.outcome` (`success` ou `failure`)	Fiabilité du lancement du navigateur
`sva.hw.temperature`	Jauge	`Cel`	—	Température du processeur
`sva.hw.throttled`	Jauge	`{state}`	`sva.hw.reason` (`undervoltage`, `thermal`)	Bridage pour alimentation insuffisante ou surchauffe

Métrique	Instrument	Unité	Attributs	Usage
`sva.dot1x.certificate.expiry`	Jauge	`s`	—	Durée restante avant l'expiration du certificat client 802.1X de la sonde, si l'authentification 802.1X est activée (chapitre 5 du guide d'installation et d'administration). La sonde émet elle-même un avertissement 30 jours avant l'échéance
`system.cpu.utilization`, `system.memory.usage`, `system.filesystem.usage`	Conventions OpenTelemetry	—	Selon les conventions sémantiques	Ressources système

Mises à jour et maintenance

Ces métriques annoncent les activations de mises à jour et signalent l'état de maintenance de la sonde (§ 8.4 du guide d'installation et d'administration). Leur exploitation dans les plateformes d'alerte est décrite au § 2.15.

Métrique	Instrument	Unité	Attributs	Usage
`sva.update.pending`	Jauge	`{state}`	`sva.update.channel` (`security` ou `feature`), `sva.update.critical` (`true` ou `false`)	1 lorsqu'une mise à jour, installée sur l'emplacement système inactif, attend son activation. Un correctif critique (vulnérabilité activement exploitée) porte `sva.update.critical` = `true`
`sva.update.last_result`	Jauge	`{state}`	`sva.update.result` (`in_progress`, `succeeded`, `rolled_back` ou `failed`)	Résultat de la dernière tentative d'activation d'une mise à jour : la série dont l'attribut `sva.update.result` correspond à ce résultat vaut 1, une seule série valant 1 à la fois ; même valeur que `lastResult.outcome` de `GET /api/v1/updates`. Elle ne porte ni la version ni l'identifiant d'activation : elle ne suffit pas à valider la reprise d'une sonde après une activation (§ 4.3)
`sva.maintenance.next_start`	Jauge	`s`	—	Début de la prochaine activation planifiée, en horodatage Unix (secondes écoulées depuis le 1er janvier 1970 UTC). Valeur en l'absence de fenêtre et d'activation planifiée : 0
`sva.maintenance.next_end`	Jauge	`s`	—	Fin de la fenêtre correspondante, en horodatage Unix (secondes). Valeur en l'absence de fenêtre et d'activation planifiée : 0. Ce zéro signifie « aucune activation planifiée » ; une série absente traduit une perte de collecte
`sva.agent.maintenance`	Jauge	`{state}`	—	1 lorsque la sonde est dans l'état de maintenance (drainage puis activation), 0 sinon

2.3.3 Noms des métriques dans Prometheus

La convention de nommage de Prometheus n'utilise pas de points dans les noms (Prometheus 3 accepte des noms UTF-8, mais la traduction par défaut conserve la forme classique). Les noms OpenTelemetry y sont donc traduits : les points deviennent des tirets bas, et un suffixe d'unité ou de type est ajouté. Les principaux exemples :

Nom OpenTelemetry	Séries Prometheus correspondantes
`sva.check.runs`	`sva_check_runs_total`
`sva.check.duration`	`sva_check_duration_seconds_bucket`, `_sum`, `_count`
`sva.step.failures`	`sva_step_failures_total`
`sva.step.duration`	`sva_step_duration_seconds_bucket`, `_sum`, `_count`
`sva.http.phase.duration`	`sva_http_phase_duration_seconds_bucket`, `_sum`, `_count`
`sva.tls.certificate.expiry`	`sva_tls_certificate_expiry_seconds`
`sva.check.status`	`sva_check_status`
`sva.check.up`	`sva_check_up`
`sva.agent.up`	`sva_agent_up`
`sva.icmp.rtt`	`sva_icmp_rtt_seconds`
`sva.icmp.packet_loss`	`sva_icmp_packet_loss_ratio`
`sva.dns.lookup.duration`	`sva_dns_lookup_duration_seconds_bucket`, `_sum`, `_count`
`sva.indicator.value`	`sva_indicator_value`
`sva.plugin.perfdata`	`sva_plugin_perfdata`
`sva.agent.scheduler.lag`	`sva_agent_scheduler_lag_seconds_bucket`, `_sum`, `_count`
`sva.agent.queue.depth`	`sva_agent_queue_depth`
`sva.export.failures`	`sva_export_failures_total`
`sva.export.dropped`	`sva_export_dropped_total`
`sva.storage.usage`	`sva_storage_usage_bytes`
`sva.storage.recoveries`	`sva_storage_recoveries_total`
`sva.browser.launches`	`sva_browser_launches_total`
`sva.hw.temperature`	`sva_hw_temperature_celsius`
`sva.hw.throttled`	`sva_hw_throttled`
`sva.dot1x.certificate.expiry`	`sva_dot1x_certificate_expiry_seconds`
`sva.update.pending`	`sva_update_pending`
`sva.update.last_result`	`sva_update_last_result`
`sva.maintenance.next_start`	`sva_maintenance_next_start_seconds`
`sva.maintenance.next_end`	`sva_maintenance_next_end_seconds`
`sva.agent.maintenance`	`sva_agent_maintenance`

Les attributs deviennent des labels selon la même règle : `sva.check.id` devient `sva_check_id`, `sva.update.channel` devient `sva_update_channel` et `sva.update.critical` devient `sva_update_critical`.

> **Information** — Pour une jauge d'unité `1`, la traduction Prometheus ajoute le suffixe `_ratio` (d'où `sva_icmp_packet_loss_ratio`). Les jauges d'état de la sonde sont déclarées avec l'unité annotée `{state}`, qui n'ajoute aucun suffixe : leurs noms `sva_check_status`, `sva_check_up`, `sva_agent_up`, `sva_update_pending`, `sva_update_last_result` et `sva_agent_maintenance` sont stables. De même, les indicateurs métier et les données de performance des plugins sont déclarés avec l'unité annotée `{value}`, car ils portent des valeurs de nature quelconque (un nombre de commandes, par exemple) : leurs noms `sva_indicator_value` et `sva_plugin_perfdata` ne reçoivent pas de suffixe `_ratio`. Une jauge d'unité `s` reçoit le suffixe `_seconds`, comme `sva_tls_certificate_expiry_seconds` : les horodatages de maintenance deviennent donc `sva_maintenance_next_start_seconds` et `sva_maintenance_next_end_seconds`. Leur valeur est un horodatage Unix en secondes, directement comparable à la fonction PromQL `time()`. Si votre Prometheus utilise une stratégie de traduction non standard (`translation_strategy`), vérifiez les noms sur `/metrics` (§ 2.5.2) avant d'écrire vos règles d'alerte.

2.3.4 Maîtrise de la cardinalité

La **cardinalité** est le nombre de séries temporelles distinctes produites, c'est-à-dire le nombre de combinaisons de valeurs d'attributs. Une cardinalité excessive ralentit les plateformes et, chez certains éditeurs, augmente la facturation. La sonde est conçue pour la maîtriser.

Attribut	Nombre de valeurs	Garde-fou
`sva.check.id`	Nombre de contrôles configurés	Aucun quota technique : le nombre de valeurs dépend de la seule configuration
`sva.check.type`	8	Liste fermée
`sva.check.status`	4	Liste fermée
`error.type`	22	Liste fermée (§ 7.5.2 du guide d'utilisation et de supervision), dont `maintenance` pour une exécution interrompue par le drainage d'une maintenance (§ 2.15.1)
`sva.step.name`	50 au maximum par scénario	Noms d'étapes fixes, imposés par le contrat des scénarios (chapitre 4 du guide de développement des scénarios)
`sva.indicator.name`, `sva.perfdata.label`	10 au maximum par contrôle	Liste explicite dans la configuration du contrôle

Ne figurent **jamais** dans les attributs de métriques : les adresses complètes avec paramètres, les identifiants d'utilisateurs, les adresses IP résolues, les messages d'erreur, les horodatages. Ces informations, à forte cardinalité, sont portées par les traces et les logs.

Ordre de grandeur. Une sonde avec 50 contrôles, dont 10 scénarios de 8 étapes, produit environ 6 000 séries, l'essentiel provenant des histogrammes. C'est modeste pour Prometheus. Pour une plateforme qui facture les métriques personnalisées, ce volume est à estimer avec votre contrat. La sonde peut désactiver les histogrammes de durée par étape et ne conserver que les compteurs d'échecs par étape : le paramètre `metrics.stepHistograms` de la destination (§ 2.4.3), à `true` par défaut, est alors positionné à `false`.

2.3.5 Traces

Chaque exécution produit une trace :

- un **span racine** `check <identifiant du contrôle>`, qui porte l'identifiant du contrôle, son type, l'identifiant d'exécution, la révision de la configuration, le numéro de tentative et l'origine du lancement ;
- un **span par étape** `step <nom de l'étape>`, pour les scénarios et les suites de requêtes API ;
- pour les contrôles `http` et `api`, un span par requête HTTP, selon les conventions sémantiques HTTP d'OpenTelemetry. L'attribut `user_agent.synthetic.type` y vaut `test`, ce qui identifie le trafic comme synthétique.

Un span en échec porte le statut `ERROR` et un événement `exception`, qui contient le type et le message d'erreur (secrets masqués). L'identifiant de trace est également enregistré dans le résultat de l'exécution (champ `trace.traceId`, chapitre 7 du guide d'utilisation et de supervision). On passe ainsi de la sonde à la plateforme, et réciproquement.

Les requêtes réseau individuelles d'un parcours navigateur ne sont pas tracées une à une, car le volume serait excessif. Leur synthèse est portée par le span de l'étape, et leur détail par l'artefact réseau conservé sur la sonde.

> **Information** — La sonde peut, en option, transmettre le contexte de trace (en-tête W3C `traceparent`) aux applications supervisées. Si celles-ci sont elles-mêmes instrumentées avec OpenTelemetry ou un agent APM, la trace synthétique se raccorde alors à la trace côté serveur. Cette option est désactivée par défaut. Elle s'active contrôle par contrôle et ne s'applique qu'à une liste d'origines autorisées, déclarée dans le contrôle : l'ajout d'un en-tête personnalisé à des requêtes inter-origines provoque des requêtes de contrôle CORS (*preflight*) qui peuvent perturber le fonctionnement d'une application web.

2.3.6 Logs

Événement	Sévérité	Contenu
`run.finished`	INFO si `OK` ; WARN ; ERROR	Résumé : état, durée, étape en échec, type d'erreur, message masqué, liens vers les artefacts sur la sonde
`check.status.changed`	WARN ou INFO	Transition d'état, par exemple `OK` vers `CRITICAL`
`browser.console.error`	WARN	Erreurs JavaScript émises par la page, 20 au maximum par exécution
`plugin.output`	INFO	Sortie d'un plugin Nagios, tronquée et masquée
`audit.*`	INFO	Opérations d'administration, si l'export de l'audit est activé (§ 2.11)
`agent.*`	Variable	Démarrage, récupération de base, saturation, erreurs d'export

Événement	Sévérité	Contenu
`update.scheduled`	INFO	Annonce ou réannonce d'une activation de mise à jour. Le délai entre l'annonce et l'activation dépend du cas d'activation (matrice du § 8.4.2 du guide d'installation et d'administration) : au moins `updates.notice` pour une activation automatique ordinaire, au moins 15 minutes pour un correctif critique, le délai choisi pour une activation manuelle programmée, **aucun** pour une activation manuelle immédiate (émis avec `maintenance.started`). Attributs : `sva.update.activation.id` (identifiant de l'activation), `sva.update.announcement.sequence` (numéro d'annonce, strictement croissant pour la sonde), `sva.update.activation.kind` (cas d'activation : `manual`, `manual-scheduled`, `automatic`, `critical-next-window` ou `critical-asap`), `sva.update.channel` (`security` ou `feature`), `sva.update.critical` (`true` ou `false`), `sva.update.version` (version à activer), `sva.update.notice_reduced` (préavis inférieur à `updates.notice`), `sva.maintenance.start` (début prévu, ISO 8601) et, seulement s'il existe une borne de fenêtre, `sva.maintenance.end` (fin prévue, ISO 8601)
`update.cancelled`	INFO	Activation annoncée qui n'aura pas lieu sous cet identifiant : report sans instant déterminable, image remplacée, canal passé en mode `manual` ou fenêtres supprimées, image retirée. Attributs : `sva.update.activation.id`, `sva.update.announcement.sequence`, `sva.update.version`, `sva.update.cancel.reason` (`postponed`, `superseded`, `policy-changed` ou `withdrawn`)
`maintenance.started`	WARN	Entrée de la sonde dans l'état de maintenance : début du drainage, aucune nouvelle exécution ne démarre. Attributs : `sva.update.activation.id`, `sva.update.version` (version à activer)
`maintenance.ended`	INFO	Sortie de l'état de maintenance : reprise des exécutions. Attributs : `sva.update.activation.id`, `sva.update.result` (`succeeded`, `rolled_back` ou `failed`), `sva.update.version` (version active)

Ce tableau fait référence pour les attributs des événements de mise à jour et de maintenance ; leur place dans la chronologie d'une activation est décrite au § 4.1.

Les logs liés à une exécution portent les identifiants `trace_id` et `span_id`, ce qui permet de passer d'un log à la trace correspondante. Les captures d'écran ne sont pas transmises : elles restent sur la sonde, et le log contient l'adresse de l'artefact dans l'API de la sonde.

2.4 Configurer une destination d'export

2.4.1 Principes

Une **destination d'export** est un objet de configuration de la sonde (ressource `exporters` de l'API) qui décrit où et comment transmettre la télémétrie. La sonde peut avoir plusieurs destinations actives simultanément, par exemple Prometheus pour l'équipe infrastructure et Dynatrace pour l'équipe applicative. Chaque destination fonctionne indépendamment : une destination en panne n'affecte ni les autres, ni l'exécution des contrôles.

La configuration d'une destination requiert le rôle `admin`. Elle peut s'effectuer :

- dans l'interface d'administration, à la rubrique de gestion des destinations d'export ;
- par l'API, ressource `POST /api/v1/exporters` ;
- lors de la mise en service, dans l'assistant (chapitre 4 du guide d'installation et d'administration).

2.4.2 Préréglages

Un **préréglage** (`preset`) remplit automatiquement les valeurs adaptées à une plateforme : protocole, temporalité, type d'histogramme. Les valeurs saisies explicitement priment sur celles du préréglage.

Préréglage	Usage	Protocole	Temporalité
`otlp-generic`	Tout backend OTLP, OpenTelemetry Collector	`http/protobuf` (gRPC possible)	Cumulative
`datadog-agent`	Agent Datadog présent sur votre réseau	`http/protobuf`	Cumulative
`datadog-agentless`	Intake OTLP direct de Datadog	`http/protobuf`	Delta
`dynatrace`	API OTLP de Dynatrace	`http/protobuf`	Delta
`grafana-cloud`	Passerelle OTLP de Grafana Cloud	`http/protobuf`	Cumulative
`prometheus-otlp`	Récepteur OTLP de Prometheus	`http/protobuf`, métriques uniquement	Cumulative

2.4.3 Paramètres d'une destination OTLP

Paramètre	Description	Valeur par défaut
`id`	Identifiant de la destination	—
`enabled`	Destination active	`true`
`signals`	Signaux transmis : `metrics`, `traces`, `logs`	—
`endpoint`	Adresse de base ; la sonde ajoute `/v1/metrics`, `/v1/traces`, `/v1/logs`	—
`signalEndpoints`	Adresses complètes par signal, utilisées telles quelles (sans ajout de suffixe)	—
`protocol`	`http/protobuf`, `http/json` ou `grpc`	`http/protobuf`
`headers`	En-têtes HTTP, notamment d'authentification. Les valeurs sensibles **doivent** référencer un secret : `${secret:NOM}`	—
`compression`	`gzip` ou `none`	`gzip`
`timeout`	Délai maximal d'un envoi	`10s`
`tls.caRef`	Autorité de certification interne à utiliser pour vérifier le serveur	Magasin de certificats système
`tls.clientCertRef`	Certificat client, pour une authentification mutuelle	—
`tls.insecureSkipVerify`	Désactive la vérification du certificat	`false` (à ne pas modifier en production)
`metrics.temporality`	`cumulative` ou `delta`	Selon le préréglage
`metrics.exportInterval`	Intervalle d'envoi des métriques	`60s`
`metrics.histogram`	`explicit` ou `exponential`	`explicit`
`metrics.stepHistograms`	Exporte les histogrammes de durée par étape ; à `false`, seuls les compteurs d'échecs par étape sont conservés (§ 2.3.4)	`true`
`retry.maxElapsed`	Durée maximale des nouvelles tentatives pour un envoi	`5m`

Paramètre	Description	Valeur par défaut
`queue.maxBatches`	Nombre maximal de lots en attente en mémoire ; un lot compte au plus 1 000 points	`100`
`logs.audit`	Transmet aussi les événements du journal d'audit (§ 2.11) ; nécessite le signal `logs`	`false`

Le schéma complet de l'objet est donné au § 6.1 de la référence technique et API (§ 6.1.4 de la référence technique et API).

2.4.4 Procédure générique

Objectif : créer une destination d'export OTLP et vérifier qu'elle reçoit les données.

Prérequis :

- un jeton d'API de rôle `admin` (chapitre 6 du guide d'installation et d'administration) ;
- l'adresse de la destination et ses informations d'authentification ;
- l'ouverture du flux sortant de la sonde vers la destination, directement ou via le proxy (chapitre 5 du guide d'installation et d'administration).

Dans les exemples de cette procédure, le jeton d'API de la sonde n'apparaît jamais dans la ligne de commande : il est lu dans le fichier d'en-tête protégé `$HOME/.sva/entete-auth`, créé une fois selon le § 6.3.3 du guide d'installation et d'administration. Le certificat de la sonde, récupéré au format PEM et dont l'empreinte a été vérifiée selon le § 4.4.4 du guide d'installation et d'administration, est enregistré dans `sonde-ca.pem`, et la variable `SONDE` contient l'adresse de la sonde (par exemple `https://sonde-paris-01.example.internal`, nom fictif).

Étapes :

1. Enregistrez la clé ou le jeton d'authentification de la plateforme comme **secret** de la sonde. Le secret n'est plus jamais lisible ensuite, seulement remplaçable. Indiquez dans `allowedHosts` l'hôte de la destination : la sonde n'injectera ce secret que dans les requêtes adressées à cet hôte (§ 6.8 du guide d'installation et d'administration). Exemple pour l'environnement Dynatrace fictif du § 2.8.2, la clé étant saisie au clavier sans affichage :

   ```bash
   read -rs VAL && printf '%s' "$VAL" \
     | jq -Rs '{value: ., allowedHosts: ["abc12345.live.dynatrace.com"]}' \
     | curl --cacert "$CA" -X PUT \
            -H @"$HOME/.sva/entete-auth" \
            -H "Content-Type: application/json" \
            -H 'If-None-Match: *' \
            --data @- \
            "$SONDE/api/v1/secrets/DT_TOKEN"; unset VAL
   ```

   > **Information** — Le corps de la requête contient la valeur du secret, dans le champ `value`, et la liste obligatoire des hôtes autorisés, dans le champ `allowedHosts` (au moins un hôte), soit `{"value": "…", "allowedHosts": ["abc12345.live.dynatrace.com"]}`. La valeur est transmise à `curl` par son entrée standard (`--data @-`) : elle n'apparaît ni dans les arguments d'un processus, ni dans l'historique du shell. Les commandes `read` et `printf` sont internes au shell ; `jq` met la valeur au format JSON.

2. Créez la destination en référençant le secret dans l'en-tête d'authentification. Les exemples propres à chaque

plateforme figurent dans les sections suivantes.

```bash
curl --cacert sonde-ca.pem -X POST \
     -H @"$HOME/.sva/entete-auth" \
     -H "Content-Type: application/json" \
     -d @destination.json \
     "$SONDE/api/v1/exporters"
```

3. Testez la destination. La sonde envoie un point de test et restitue le diagnostic : code HTTP obtenu, message d'erreur renvoyé par la plateforme, latence.

```bash
curl --cacert sonde-ca.pem -X POST \
     -H @"$HOME/.sva/entete-auth" \
     "$SONDE/api/v1/exporters/dynatrace/test"
```

   La sonde répond `202 Accepted`, avec un en-tête `Location` vers le résultat du test. Ce résultat contient les champs `status`, `httpStatus` (code HTTP renvoyé par la plateforme), `durationMs` (latence) et `message` (message d'erreur éventuel).
4. Attendez au moins deux intervalles d'export (2 minutes avec les valeurs par défaut), puis recherchez les métriques `sva.agent.up` et `sva.check.runs` dans la plateforme.

Résultat attendu : le test renvoie un succès ; la destination apparaît à l'état `OK` dans la santé de la sonde (§ 3.1) ; les métriques sont visibles dans la plateforme.

En cas de problème : voir le § 2.16, puis le chapitre 9 du guide d'installation et d'administration (§ 5.1). Les causes les plus fréquentes sont un flux sortant bloqué, un proxy non déclaré, une clé invalide (401 ou 403), une adresse ne correspondant pas à la région ou au site du compte, ou une temporalité refusée par la plateforme.

> **Avertissement** — N'écrivez jamais une clé d'API en clair dans l'objet de destination. Utilisez toujours la référence `${secret:NOM}`. Les exports de configuration excluent ou chiffrent les secrets, mais pas une clé saisie en clair dans un en-tête.

2.5 Prometheus

Deux modes d'intégration sont possibles. Le mode par **collecte** (pull) est recommandé : c'est le fonctionnement natif de Prometheus, et la sonde n'a rien à configurer en dehors du jeton.

2.5.1 Mode collecte (pull) : principes

Prometheus interroge périodiquement l'adresse `https://<sonde>/metrics`. La sonde y expose ses métriques au format texte Prometheus, en temporalité cumulative. L'accès est protégé :

- par HTTPS ;
- par un jeton d'API de rôle `metrics`, le rôle le plus restreint, qui ne donne accès qu'à `/metrics` et à l'état des contrôles.

> **Information** — En mode collecte, la sonde n'envoie rien. Si Prometheus ne peut pas joindre la sonde pendant un certain temps, les mesures de cette période ne sont pas récupérables dans Prometheus. Elles restent consultables sur la sonde pendant la durée de rétention (§ 7.8 du guide d'utilisation et de supervision).

2.5.2 Mode collecte : procédure

Objectif : faire collecter les métriques de la sonde par un serveur Prometheus.

Prérequis :

- flux HTTPS (TCP 443) autorisé du serveur Prometheus vers la sonde ;
- certificat de la sonde, ou de l'autorité qui l'a émis, disponible sur le serveur Prometheus (chapitre 5 du guide d'installation et d'administration) ;
- droits d'administration sur la configuration de Prometheus.

Étapes :

1. Créez un jeton d'API de rôle `metrics` dans l'interface d'administration de la sonde, ou par l'API (chapitre 6 du guide d'installation et d'administration). Copiez-le immédiatement : il ne sera plus affiché.
2. Sur le serveur Prometheus, enregistrez le jeton dans un fichier lisible uniquement par le compte de service Prometheus, par exemple `/etc/prometheus/secrets/sonde-token`.

   ```bash
   sudo install -m 0400 -o prometheus -g prometheus /dev/null /etc/prometheus/secrets/sonde-token
   # Éditez ensuite ce fichier pour y coller le jeton, sans saut de ligne final
   ```

   Les chemins de cet exemple correspondent à une installation courante de Prometheus sous Linux ; adaptez-les à votre installation.
3. Vérifiez manuellement l'accès depuis le serveur Prometheus. Pour que le jeton n'apparaisse pas dans les arguments de `curl`, placez l'en-tête dans un fichier temporaire **unique**, créé par `mktemp` avec des droits réservés à votre compte avant toute écriture, et effacé à la fin du test **même en cas d'échec** :

   ```bash
   (
     set -eu
     umask 077
     entete=$(mktemp)                  # nom unique, fichier créé en mode 600
     trap 'rm -f "$entete"' EXIT       # effacement en fin de test, succès ou échec
     trap 'exit 130' INT TERM          # interruption : sortie, donc effacement
     { printf 'Authorization: Bearer '
       sudo cat /etc/prometheus/secrets/sonde-token
       printf '\n'; } > "$entete"
     curl -sS --cacert /etc/prometheus/certs/sonde-ca.pem \
          -H @"$entete" \
          https://sonde-paris-01.example.internal/metrics | head -n 20
   )
   ```

   Le jeton passe directement du fichier de Prometheus au fichier d'en-tête, créé vide en mode `600` par `mktemp` avant d'être rempli : il ne figure dans les arguments d'aucun processus, `sudo` et `cat` ne recevant que le chemin du fichier. Un nom fixe est évité, car un fichier préexistant du même nom conserverait ses propres droits. Prometheus lui-même lit le jeton dans le fichier `credentials_file` (étape 4) et n'a pas besoin de ce fichier d'en-tête.

   Le résultat doit contenir des lignes commençant par `# HELP`, `# TYPE` et des séries `sva_…`. Notez les noms exacts des métriques (§ 2.3.3).

4. Ajoutez un travail de collecte (*scrape job*) dans le fichier de configuration de Prometheus :

```yaml
scrape_configs:
  - job_name: sondes
    scheme: https
    metrics_path: /metrics
    scrape_interval: 60s
    authorization:
      type: Bearer
      credentials_file: /etc/prometheus/secrets/sonde-token
    tls_config:
      ca_file: /etc/prometheus/certs/sonde-ca.pem
    static_configs:
      - targets:
          - sonde-paris-01.example.internal:443
        labels:
          site: paris-dc1
      - targets:
          - sonde-lyon-01.example.internal:443
        labels:
          site: lyon-dc1
```

5. Validez la configuration, puis rechargez Prometheus :

```bash
promtool check config /etc/prometheus/prometheus.yml
# Rechargement selon votre installation, par exemple :
sudo systemctl reload prometheus
```

Résultat attendu : dans l'interface de Prometheus, la cible du travail `sondes` apparaît à l'état `UP`, et la requête `sva_check_runs_total` renvoie des séries.

Vérification : la requête `up{job="sondes"}` vaut 1 pour chaque sonde.

> ✓ **Conseil** — Un intervalle de collecte de 60 secondes suffit : les contrôles eux-mêmes s'exécutent rarement plus d'une fois par minute. Un intervalle plus court n'apporte pas d'information supplémentaire et augmente la charge de la sonde.

> ℹ **Information** — Si plusieurs sondes partagent le même jeton, sa révocation les coupe toutes de Prometheus. Préférez un jeton par sonde, ou au minimum par site, et utilisez un fichier `credentials_file` par travail de collecte.

2.5.3 Mode envoi via le récepteur OTLP de Prometheus

Prometheus peut aussi recevoir des métriques au format OTLP. Ce mode convient lorsque la sonde n'est pas joignable par Prometheus, par exemple derrière un pare-feu qui interdit les connexions entrantes.

Contraintes à connaître (documentation officielle de Prometheus) :

- le récepteur est **désactivé par défaut** et s'active par l'option de lancement `--web.enable-otlp-receiver` ;
- il n'accepte qu'**OTLP/HTTP**, et uniquement des **métriques** ;
- l'adresse de réception est `/api/v1/otlp/v1/metrics` ;
- Prometheus n'authentifie pas **par défaut** les écritures reçues. L'authentification Basic et le TLS, y compris l'authentification des clients par certificat, se configurent dans le fichier de configuration web désigné par l'option de lancement `--web.config.file` ; ce fichier s'applique à l'ensemble de l'interface HTTP de

Prometheus, et non au seul récepteur. À défaut, placez Prometheus derrière un proxy inverse qui exige une authentification, et limitez dans tous les cas l'accès réseau au récepteur.

Configuration de la destination dans la sonde. L'adresse complète est déclarée dans `signalEndpoints`, car elle ne suit pas la convention `/v1/metrics` :

```json
{
  "id": "prometheus-otlp",
  "kind": "otlp",
  "preset": "prometheus-otlp",
  "enabled": true,
  "signals": ["metrics"],
  "signalEndpoints": {
    "metrics": "https://prometheus.example.internal/api/v1/otlp/v1/metrics"
  },
  "protocol": "http/protobuf",
  "headers": { "Authorization": "Bearer ${secret:PROM_OTLP_TOKEN}" },
  "metrics": { "temporality": "cumulative", "exportInterval": "60s" }
}
```

Dans cet exemple, l'en-tête d'authentification est vérifié par le proxy inverse placé devant Prometheus, et non par Prometheus lui-même. Si vous utilisez plutôt l'authentification Basic de Prometheus (`--web.config.file`), l'en-tête prend la forme `Basic ${secret:…}`, comme pour Grafana Cloud (§ 2.6.2).

> **Attention** — Dans ce mode, les attributs de ressource qui identifient la sonde (§ 2.3.1) ne figurent pas sur chaque série : ils sont rattachés à une série d'information `target_info`. Seuls les labels `job` et `instance` sont présents sur toutes les séries, dérivés de `service.name` et de `service.instance.id`.

Identifier le site en mode envoi : promotion d'un attribut de ressource. En mode collecte, le label `site` est ajouté par la configuration de collecte (§ 2.5.2). En mode envoi, il n'existe pas : l'information correspondante est l'attribut de ressource `sva.site`. Pour l'utiliser dans vos requêtes et vos règles d'alerte, demandez à Prometheus de **promouvoir** cet attribut en label de chaque série. Dans Prometheus 3, la promotion se déclare dans la section `otlp` du fichier de configuration de Prometheus, par le paramètre `promote_resource_attributes` :

```yaml
# Extrait de prometheus.yml (Prometheus 3) : promotion d'attributs de ressource OTLP en labels
otlp:
  promote_resource_attributes:
    - sva.site
```

Avec la traduction par défaut des noms (§ 2.3.3), l'attribut `sva.site` devient le label `sva_site`, présent sur toutes les séries de la sonde reçues après le rechargement de la configuration. Remplacez alors `site` par `sva_site` dans les règles d'alerte de site (§ 2.15.6). Vérifiez le nom exact du paramètre et du label dans la documentation de votre version de Prometheus, puis sur les séries reçues, avant de charger vos règles. À défaut de promotion, l'attribut reste lisible par une jointure sur la série `target_info`, ce qui alourdit chaque requête.

2.5.4 Requêtes PromQL utiles

Les requêtes ci-dessous utilisent les noms de séries du § 2.3.3, à adapter si nécessaire. Les vues **par sonde** regroupent les séries par `job`, `instance` et `sva_check_id` : deux sondes peuvent porter un contrôle de même identifiant, et un regroupement sur le seul identifiant mélangerait leurs résultats. L'**agrégation de parc**, qui additionne les exécutions de toutes les sondes, fait l'objet d'une requête distincte.

```promql
# Contrôles actuellement en CRITICAL (2) ou UNKNOWN (3), sonde par sonde
sva_check_status >= 2
```

```
# Taux d'échec de chaque contrôle sur la dernière heure, par sonde
# (exécutions évaluables : OK, WARNING ou CRITICAL ; UNKNOWN exclu)
(
    sum by (job, instance, sva_check_id) (increase(sva_check_runs_total{sva_check_status="CRITICAL"}
    [1h]))
  or
    sum by (job, instance, sva_check_id)
    (increase(sva_check_runs_total{sva_check_status=~"OK|WARNING|CRITICAL"}[1h])) * 0
)
  /
(
  sum by (job, instance, sva_check_id)
    (increase(sva_check_runs_total{sva_check_status=~"OK|WARNING|CRITICAL"}[1h])) > 0
)

# Disponibilité sur 30 jours, par sonde (UNKNOWN exclu, WARNING compté comme disponible)
(
    sum by (job, instance, sva_check_id)
    (increase(sva_check_runs_total{sva_check_status=~"OK|WARNING"}[30d]))
  or
    sum by (job, instance, sva_check_id)
    (increase(sva_check_runs_total{sva_check_status=~"OK|WARNING|CRITICAL"}[30d])) * 0
)
  /
(
  sum by (job, instance, sva_check_id)
    (increase(sva_check_runs_total{sva_check_status=~"OK|WARNING|CRITICAL"}[30d])) > 0
)

# Agrégation de parc : taux d'échec d'un contrôle sur l'ensemble des sondes, sur la dernière heure
# (n'a de sens que si les sondes exécutent la même définition de contrôle)
(
    sum by (sva_check_id) (increase(sva_check_runs_total{sva_check_status="CRITICAL"}[1h]))
  or
    sum by (sva_check_id) (increase(sva_check_runs_total{sva_check_status=~"OK|WARNING|CRITICAL"}
    [1h])) * 0
)
  /
(
  sum by (sva_check_id) (increase(sva_check_runs_total{sva_check_status=~"OK|WARNING|CRITICAL"}
    [1h])) > 0
)

# 95e percentile du temps de réponse par contrôle et par sonde, sur 15 minutes
histogram_quantile(0.95,
  sum by (job, instance, sva_check_id, le) (rate(sva_check_duration_seconds_bucket[15m])))

# Les 5 étapes qui échouent le plus sur 24 heures, par sonde (au plus 5 lignes par couple
    job/instance)
topk by (job, instance) (5, sum by (job, instance, sva_check_id, sva_step_name)
    (increase(sva_step_failures_total[24h])))

# Certificats expirant dans moins de 21 jours
sva_tls_certificate_expiry_seconds < 21 * 86400
```

La formule de disponibilité et le traitement de l'état `UNKNOWN` sont justifiés au § 7.7.3 du guide d'utilisation et de supervision.

Ces requêtes distinguent trois situations, qui ne doivent pas être confondues :

Situation	Résultat de la requête	Mécanisme
Exécutions évaluables, sans aucun échec	`0` (taux d'échec) ou `1` (disponibilité)	Le terme `or … * 0` fournit un numérateur nul lorsque la série `CRITICAL` (ou `OK` / `WARNING`) n'a jamais été émise, par exemple pour un contrôle récent qui n'a jamais échoué : sans lui, la division ne renverrait rien
Aucune exécution évaluable sur la période (dénominateur nul : contrôle uniquement en `UNKNOWN`, ou sans exécution)	Aucun résultat pour ce contrôle	Le filtre `> 0` sur le dénominateur écarte la division par zéro ; l'absence de résultat signifie « pas de mesure », et non « 0 % » ou « 100 % »
Aucune série reçue (sonde injoignable, export interrompu)	Aucun résultat	La perte de collecte se surveille par les alertes `SondeInjoignable` ou `SondeSilencieuse` (§ 2.5.5)

Les états sont désignés explicitement (`sva_check_status=~"OK|WARNING|CRITICAL"`) plutôt que par exclusion (`!="UNKNOWN"`) : un filtre d'exclusion sélectionnerait aussi une série dépourvue du label d'état. Ces requêtes ont été vérifiées avec `promtool test rules` sur des séries simulées couvrant deux sondes homonymes, un contrôle sans échec, un contrôle uniquement en `UNKNOWN` et une série sans label d'état ; vérifiez-les sur vos propres séries avant de les intégrer à un tableau de bord.

2.5.5 Règles d'alerte

Exemple de fichier de règles d'alerte Prometheus. Les seuils sont indicatifs. Ces règles, complétées des principales alertes de santé recommandées au § 3.2.1 (l'alerte « Mise à jour en échec » s'ajoute sur le modèle des autres, à partir de `sva_update_last_result`) et des alertes de mises à jour et de maintenance (groupe `sondes-mises-a-jour`, § 2.15.3), sont fournies prêtes à charger dans le fichier d'exemple `exemples/prometheus/sva-regles-alerte.yml` (§ 2.2).

```yaml
groups:
  - name: sondes-supervision
    rules:
      - alert: ParcoursEnEchec
        expr: sva_check_status == 2
        for: 5m
        labels:
          severity: critical
        annotations:
          summary: "Contrôle {{ $labels.sva_check_id }} en CRITICAL"
          description: "Le contrôle {{ $labels.sva_check_id }} est en échec depuis au moins 5
    minutes. Consultez le détail de la dernière exécution sur la sonde {{ $labels.instance }}."

      # Pendant le drainage d'une maintenance (documentation SYNOVEX, « Maintenance de la sonde et
      # fausses alertes »), les exécutions interrompues
      # sont closes en UNKNOWN avec le type d'erreur maintenance : la clause unless neutralise
      # l'alerte tant que la sonde signale l'état de maintenance.
      - alert: ControleNonEvaluable
        expr: sva_check_status == 3 unless on(instance) sva_agent_maintenance == 1
        for: 15m
        labels:
          severity: warning
        annotations:
          summary: "Contrôle {{ $labels.sva_check_id }} en UNKNOWN"
          description: "La sonde {{ $labels.instance }} ne parvient pas à évaluer ce contrôle.
    Vérifiez l'état de la sonde."

      # Alerte « Sonde muette » (documentation SYNOVEX, « Surveillance de la santé de la sonde »),
      # mode collecte.
```

```
      # Adaptez la valeur du label job au nom de votre travail de collecte.
      - alert: SondeInjoignable
        expr: up{job="sondes"} == 0
        for: 5m
        labels:
          severity: critical
        annotations:
          summary: "Sonde {{ $labels.instance }} injoignable"
          description: "Prometheus ne parvient plus à collecter la sonde. Les contrôles qu'elle
    exécute ne sont plus visibles."

      # Alerte « Sonde muette » (documentation SYNOVEX, « Surveillance de la santé de la sonde »),
      # mode envoi (OTLP).
      # Règle par sonde : une sonde qui a émis sva_agent_up au cours des dernières 24 heures,
      # mais plus depuis 5 minutes, est signalée. Au-delà de 24 heures de silence, l'alerte
      # se résout d'elle-même : la règle SondeInjoignable (mode collecte) n'a pas cette limite.
      # LIMITE : la résolution après 24 heures ne signifie pas que la sonde est revenue.
      # Tenez un inventaire externe des sondes attendues (CMDB, liste de déploiement) et
      # comparez-le aux sondes qui émettent sva_agent_up (documentation SYNOVEX, « Règles d'alerte
    »).
      # La clause unless neutralise l'alerte tant que la sonde signale l'état de maintenance
      # (drainage) ; le redémarrage qui suit est couvert par la mise en sourdine d'Alertmanager
      # (alertmanager-maintenance.yml).
      - alert: SondeSilencieuse
        # Mode envoi (OTLP) : le label job est dérivé de service.name (sva-agent)
        expr: |
          (
            max by (job, instance) (max_over_time(sva_agent_up{job="sva-agent"}[1d]))
              unless on(job, instance)
            max by (job, instance) (max_over_time(sva_agent_up{job="sva-agent"}[5m]))
          )
          unless on(instance) sva_agent_maintenance == 1
        labels:
          severity: critical
        annotations:
          summary: "Sonde {{ $labels.instance }} silencieuse"
          description: "Aucune série sva_agent_up reçue de la sonde {{ $labels.instance }} depuis 5
    minutes (équivalent de l'état STALE pour la sonde)."

      # Alerte « Sous-tension » (documentation SYNOVEX, « Surveillance de la santé de la sonde »).
      - alert: SondeSousTension
        expr: sva_hw_throttled{sva_hw_reason="undervoltage"} == 1
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: "Alimentation insuffisante sur {{ $labels.instance }}"
          description: "Risque d'instabilité et de corruption de la carte microSD. Remplacez
    l'alimentation."

      - alert: CertificatBientotExpire
        expr: sva_tls_certificate_expiry_seconds < 14 * 86400
        labels:
          severity: warning
        annotations:
          summary: "Certificat bientôt expiré ({{ $labels.sva_check_id }})"
          description: "Le certificat présenté par la cible expire dans moins de 14 jours."

      - alert: Certificat8021XBientotExpire
        expr: sva_dot1x_certificate_expiry_seconds < 30 * 86400
        labels:
          severity: warning
        annotations:
          summary: "Certificat 802.1X de la sonde {{ $labels.instance }} bientôt expiré"
          description: "Le certificat client 802.1X de la sonde expire dans moins de 30 jours.
    Renouvelez-le avant l'échéance, faute de quoi la sonde perdra l'accès au réseau."
```

> **Information** — En mode collecte, la règle `SondeInjoignable`, fondée sur la série `up` que Prometheus produit pour chaque cible, est le moyen le plus fiable de détecter une sonde arrêtée. La règle `SondeSilencieuse` convient au mode envoi, où la série `up` n'existe pas. Elle est évaluée **sonde par sonde** : une sonde qui a émis `sva_agent_up` au cours des dernières 24 heures, mais plus depuis 5 minutes, est signalée. Contrairement à `absent_over_time`, qui ne détecte que l'absence de **toutes** les séries correspondant au filtre, cette forme distingue les sondes ; en contrepartie, l'alerte se résout d'elle-même après 24 heures de silence. Allongez la plage `[1d]` si votre politique d'astreinte l'exige.

> **Attention** — En mode envoi, la disparition de l'alerte `SondeSilencieuse` après 24 heures **ne signifie pas que la sonde est revenue** : Prometheus a simplement oublié qu'elle existait. Tenez donc un **inventaire externe des sondes attendues** et comparez-le régulièrement aux sondes qui émettent `sva_agent_up`.

Une sonde arrêtée pendant un week-end prolongé, ou dont l'alerte n'a pas été traitée dans la journée, cesse en effet d'être signalée par `SondeSilencieuse`. L'inventaire des sondes attendues peut être une base de gestion des configurations, une liste de déploiement ou un fichier versionné. Dans Prometheus, il peut prendre la forme d'une série d'enregistrement par sonde, comparée aux séries reçues. Exemple fictif, à adapter, de règles à ajouter dans un groupe de votre fichier de règles :

```yaml
# Une règle par sonde attendue (nom fictif) ; instance = service.instance.id
- record: sva:sonde_attendue
  expr: vector(1)
  labels:
    instance: sonde-paris-01
- alert: SondeAttendueAbsente
  expr: sva:sonde_attendue unless on (instance) sva_agent_up
  for: 10m
  labels:
    severity: critical
  annotations:
    summary: "Sonde attendue {{ $labels.instance }} sans signal"
```

Cette alerte ne se résout pas d'elle-même tant que la sonde reste absente. Elle se déclenche aussi pendant le redémarrage d'une activation : ajoutez son nom aux alertes mises en sourdine pendant les fenêtres de la sonde (§ 2.15.2).

> **Information** — La règle d'export du groupe `sondes-sante` se nomme `ExportEchecsRepetes` (anciennement `ExportEnEchec`). Elle se déclenche lorsqu'au moins un échec d'envoi survient dans chaque fenêtre glissante de 15 minutes, pendant 15 minutes consécutives : un échec isolé ne la déclenche pas, mais des échecs intermittents rapprochés la déclenchent, même si la plupart des envois réussissent. Elle ne prouve donc pas une indisponibilité continue de la destination. Les données réellement abandonnées sont signalées par `ExportDonneesPerdues` (§ 2.12).

> **Information** — Les règles `ControleNonEvaluable` et `SondeSilencieuse` comportent la clause `unless on(instance) sva_agent_maintenance == 1`, qui les neutralise tant que la sonde signale l'état de maintenance (phase de drainage). Cette clause ne couvre pas le redémarrage qui suit, pendant lequel la sonde ne produit plus aucune série. Elle n'est pas ajoutée à `SondeInjoignable` : lorsqu'une collecte échoue, Prometheus marque aussitôt comme périmées toutes les séries de la cible, y compris `sva_agent_maintenance`, et la clause resterait sans effet. Les alertes émises pendant le redémarrage se traitent dans Alertmanager (§ 2.15.2).

Validez le fichier de règles avec `promtool check rules <fichier>` avant de le charger.

2.6 Grafana

Grafana est un outil de visualisation qui s'appuie sur des sources de données. Avec la sonde, deux architectures sont courantes.

2.6.1 Pile auto-hébergée

Composant	Rôle	Alimentation par la sonde
Prometheus ou Mimir	Stockage des métriques	Collecte `/metrics`, ou OTLP (§ 2.5)
Tempo	Stockage des traces	OTLP, généralement via un collecteur
Loki	Stockage des logs	OTLP (ingestion OTLP native dans les versions récentes de Loki), généralement via un collecteur
Grafana Alloy ou OpenTelemetry Collector	Collecteur : reçoit l'OTLP de la sonde et le répartit vers Mimir, Tempo et Loki	Destination OTLP unique de la sonde
Grafana	Tableaux de bord, alertes, navigation entre métriques, traces et logs	—

Dans cette architecture, configurez dans la sonde une seule destination `otlp-generic` qui pointe vers le collecteur (Alloy ou OpenTelemetry Collector). Le collecteur se charge de la répartition par signal. Un exemple de configuration de collecteur est donné au § 2.9.

2.6.2 Grafana Cloud

Grafana Cloud propose une passerelle OTLP qui reçoit les trois signaux. Son adresse et l'identifiant d'instance sont indiqués dans le portail Grafana Cloud, sur la fiche de configuration OpenTelemetry de votre pile. L'adresse est de la forme `https://otlp-gateway-<zone>.grafana.net/otlp`.

L'authentification est de type **Basic** : la valeur de l'en-tête est l'encodage Base64 de la chaîne `<identifiant d'instance>:<jeton>`. Le jeton est un jeton de politique d'accès disposant des droits d'écriture sur les signaux transmis.

Étapes :

1. Dans le portail Grafana Cloud, relevez l'adresse de la passerelle OTLP et l'identifiant d'instance, puis créez un jeton disposant des droits d'écriture sur les métriques, traces et logs.
2. Sur votre poste, calculez la valeur Base64. N'ajoutez pas de saut de ligne :

   ```bash
   printf '%s' '123456:<jeton Grafana Cloud>' | base64 | tr -d '\n'
   ```

3. Enregistrez le résultat comme secret `GRAFANA_OTLP_BASIC` sur la sonde (§ 2.4.4, étape 1).

4. Créez la destination :

```json
{
  "id": "grafana-cloud",
  "kind": "otlp",
  "preset": "grafana-cloud",
  "enabled": true,
  "signals": ["metrics", "traces", "logs"],
  "endpoint": "https://otlp-gateway-prod-eu-west-2.grafana.net/otlp",
  "protocol": "http/protobuf",
  "headers": { "Authorization": "Basic ${secret:GRAFANA_OTLP_BASIC}" },
  "metrics": { "temporality": "cumulative" }
}
```

   L'adresse ci-dessus est un exemple : utilisez celle qui est affichée dans votre portail.

5. Testez la destination (§ 2.4.4, étape 3).

> **Information** — Une erreur 401 provient le plus souvent d'un en-tête mal construit : jeton seul au lieu de `identifiant:jeton`, ou saut de ligne inclus dans l'encodage Base64.

2.6.3 Tableaux de bord et alertes dans Grafana

Les requêtes PromQL du § 2.5.4 s'utilisent telles quelles dans les panneaux Grafana dont la source est Prometheus ou Mimir. Organisation suggérée pour un tableau de bord « Supervision applicative » :

Rangée	Panneaux	Requête de base
Synthèse	Nombre de contrôles par état ; liste des contrôles en échec	`sva_check_status`
Disponibilité	Disponibilité par contrôle sur la période affichée (24 heures, 7 jours ou 30 jours selon le choix de période)	§ 2.5.4
Performance	p50 et p95 par contrôle ; durée par étape des parcours	`sva_check_duration_seconds_bucket`, `sva_step_duration_seconds_bucket`
Diagnostic	Étapes en échec, types d'erreur, phases HTTP	`sva_step_failures_total`, `sva_check_runs_total`, `sva_http_phase_duration_seconds_bucket`
Santé des sondes	Signal de vie, retard de planification, température, stockage, exports	Métriques `sva_agent_*`, `sva_hw_*`, `sva_export_*`

Utilisez une variable de tableau de bord sur le label `instance` pour comparer les sondes entre elles, et sur le label de site pour les regrouper par site.

Le fichier d'exemple `exemples/grafana/sva-tableau-de-bord.json` (§ 2.2) met en œuvre cette organisation. Il s'importe dans Grafana (menu **Dashboards**, **New**, **Import**) ; la source de données Prometheus est choisie à l'importation. Ses variables `job`, `instance` et `check` filtrent l'affichage par travail de collecte, par sonde et par contrôle. Le tableau de bord ne filtre pas par site, et ce choix est délibéré : les labels `job` et `instance` existent dans les deux modes d'intégration (en mode collecte, nom du travail et adresse de la cible ; en mode envoi, `service.name` et `service.instance.id`), alors que le label de site dépend de votre configuration. Il se nomme `site` lorsqu'il est ajouté par la configuration de collecte (§ 2.5.2), `sva_site` lorsqu'il résulte de la promotion de l'attribut `sva.site` en mode envoi (§ 2.5.3), et il est absent sans l'une de ces deux configurations. Pour filtrer par site, ajoutez au tableau de bord importé une variable sur le label que vous utilisez, puis ajoutez-la au filtre des requêtes. Une rangée supplémentaire suit l'expiration des certificats des cibles et du certificat client 802.1X des sondes. La rangée « Mises à

jour et maintenance » affiche, pour chaque sonde, les mises à jour en attente d'activation (`sva_update_pending`), l'état de maintenance (`sva_agent_maintenance`) et la prochaine fenêtre planifiée (`sva_maintenance_next_start_seconds` et `sva_maintenance_next_end_seconds`).

Les alertes se définissent dans le module d'alerte de Grafana, avec les mêmes expressions que les règles du § 2.5.5. Lorsque Tempo et Loki sont alimentés, les logs `run.finished` et les traces des exécutions sont accessibles depuis les panneaux grâce à l'identifiant de trace.

2.7 Datadog

Datadog peut recevoir les données de la sonde de deux manières. Le choix dépend de la présence d'un Agent Datadog sur votre réseau.

Critère	Via l'Agent Datadog	Intake OTLP direct
Prérequis	Un Agent Datadog joignable par la sonde	Accès Internet sortant de la sonde vers Datadog
Signaux	Métriques, traces, logs	Métriques et logs ; traces selon la documentation de Datadog en vigueur (§ 2.7.2)
Temporalité	Cumulative	**Delta uniquement**
Clé d'API Datadog	Détenue par l'Agent	Stockée sur la sonde
Recommandation	**Recommandé** si un Agent existe	Lorsqu'aucun Agent n'est disponible

2.7.1 Via l'Agent Datadog (recommandé)

L'Agent Datadog intègre un récepteur OTLP. Ce récepteur est **désactivé par défaut** : l'Agent ne reçoit aucune donnée OTLP tant que vous ne l'avez pas activé dans sa configuration (étapes ci-dessous). Une fois activé, il écoute sur les adresses que vous déclarez, habituellement sur les ports standard d'OTLP, 4317 (gRPC) et 4318 (HTTP). La sonde envoie ses données à l'Agent, qui les transmet à Datadog avec sa propre clé d'API. Aucune clé Datadog n'est alors stockée sur la sonde.

Étapes côté Agent Datadog (d'après la documentation de Datadog) :

1. Activez le récepteur OTLP dans la configuration de l'Agent (`datadog.yaml`), section `otlp_config.receiver.protocols`, en écoutant sur une adresse joignable par la sonde. Les variables d'environnement `DD_OTLP_CONFIG_RECEIVER_PROTOCOLS_HTTP_ENDPOINT` et `DD_OTLP_CONFIG_RECEIVER_PROTOCOLS_GRPC_ENDPOINT` jouent le même rôle pour un Agent en conteneur.
2. Pour recevoir les logs de la sonde, activez explicitement la collecte des logs (`logs_enabled: true`) **et** l'ingestion OTLP des logs (`otlp_config.logs.enabled: true`). L'ingestion OTLP des logs est **désactivée par défaut** dans l'Agent, afin d'éviter une facturation inattendue.
3. Redémarrez l'Agent et vérifiez que le port 4318 est ouvert depuis le réseau de la sonde.

Exemple d'extrait de configuration de l'Agent :

```yaml
# Extrait de datadog.yaml (Agent Datadog)
logs_enabled: true
otlp_config:
  receiver:
    protocols:
      http:
        endpoint: 0.0.0.0:4318
  logs:
    enabled: true
```

Avertissement — Le récepteur OTLP de l'Agent n'authentifie pas les émetteurs. Limitez par pare-feu l'accès au port 4318 aux seules adresses des sondes et des applications autorisées.

Étape côté sonde : créez la destination.

```json
{
  "id": "datadog-agent",
  "kind": "otlp",
  "preset": "datadog-agent",
  "enabled": true,
  "signals": ["metrics", "traces", "logs"],
  "endpoint": "http://datadog-agent.example.internal:4318",
  "protocol": "http/protobuf",
  "metrics": { "temporality": "cumulative" }
}
```

Information — La liaison entre la sonde et l'Agent utilise ici HTTP sans chiffrement, ce qui est la configuration par défaut du récepteur OTLP de l'Agent. Elle reste acceptable sur un réseau interne maîtrisé. Pour la chiffrer, configurez TLS sur le récepteur de l'Agent selon la documentation de Datadog, puis utilisez une adresse `https://`.

2.7.2 Intake OTLP direct

L'intake OTLP de Datadog reçoit directement les données, sans Agent. Contraintes publiées par Datadog :

- authentification par l'en-tête `dd-api-key` contenant une clé d'API Datadog ;
- protocole **OTLP/HTTP protobuf** ;
- métriques en temporalité **delta uniquement** ;
- l'adresse dépend du **site Datadog** de votre organisation. Elle est de la forme `https://otlp.<site>`, à laquelle la sonde ajoute `/v1/metrics`, `/v1/traces` et `/v1/logs` : `https://otlp.datadoghq.com` (US1), `https://otlp.us3.datadoghq.com` (US3), `https://otlp.us5.datadoghq.com` (US5), `https://otlp.datadoghq.eu` (EU1), `https://otlp.ap1.datadoghq.com` (AP1), `https://otlp.ap2.datadoghq.com` (AP2), `https://otlp.uk1.datadoghq.com` (UK1), `https://otlp.ddog-gov.com` (US1-FED). Vérifiez l'adresse de votre site dans la documentation de Datadog, qui fait foi ;
- une erreur 403 signifie en général que l'adresse ne correspond pas au site du compte, ou que l'organisation n'a pas accès à l'intake.

Étapes :

1. Dans Datadog, créez une clé d'API dédiée aux sondes.
2. Enregistrez-la comme secret `DD_API_KEY` sur la sonde (§ 2.4.4).

3. Créez la destination. Exemple pour le site US1 :

```json
{
  "id": "datadog-direct",
  "kind": "otlp",
  "preset": "datadog-agentless",
  "enabled": true,
  "signals": ["metrics", "logs"],
  "endpoint": "https://otlp.datadoghq.com",
  "protocol": "http/protobuf",
  "headers": { "dd-api-key": "${secret:DD_API_KEY}" },
  "metrics": { "temporality": "delta", "histogram": "explicit" }
}
```

4. Testez la destination (§ 2.4.4, étape 3).

L'exemple ne transmet que les métriques et les logs (`"signals": ["metrics", "logs"]`). La sonde sait produire des traces, mais leur réception par l'intake OTLP direct dépend de l'offre de Datadog pour votre site et votre organisation : n'ajoutez `traces` à la liste `signals` qu'après avoir vérifié, dans la documentation de Datadog, que l'intake OTLP direct accepte les traces pour votre compte, puis contrôlé leur arrivée dans l'APM. Pour transmettre les traces sans cette vérification, passez par l'Agent Datadog (§ 2.7.1), qui accepte les trois signaux.

> **Attention** — Datadog impose des limites de taille par requête (5,1 Mio non compressés pour les métriques et les logs, 15 Mio pour les traces, selon sa documentation). La sonde découpe ses envois en lots adaptés. Si des erreurs 413 apparaissent dans les journaux d'export, signalez-les au support (§ 10.5 du guide d'installation et d'administration, chapitre 6).

2.7.3 Exploitation dans Datadog

Dans Datadog, les métriques conservent leurs noms OpenTelemetry (avec des points), et leurs attributs deviennent des **tags**. Les modalités précises de conversion des attributs et des histogrammes dépendent de la configuration de Datadog (voir sa documentation sur le mappage des métriques OTLP).

Exemples de requêtes de moniteur, à adapter :

```
# Moniteur de métrique : parcours navigateur en échec
max(last_5m):max:sva.check.status{sva.check.type:browser} by {sva.check.id} >= 2

# Moniteur de métrique : absence de signal d'une sonde (option "notify if data is missing")
max(last_10m):max:sva.agent.up{*} by {host} < 1
```

Pour l'absence de signal, activez dans le moniteur l'option de notification en cas de données manquantes. C'est elle qui détecte une sonde arrêtée.

Les traces des exécutions sont visibles dans l'APM sous le service `sva-agent` (si les traces sont transmises), et les logs dans le Log Explorer, filtrables par `service:sva-agent`.

2.8 Dynatrace

Dynatrace reçoit les données de la sonde par son **API OTLP**. Contraintes publiées par Dynatrace :

- adresse de la forme `https://<identifiant-environnement>.live.dynatrace.com/api/v2/otlp` pour un environnement SaaS ; les chemins `/v1/metrics`, `/v1/traces` et `/v1/logs` y sont ajoutés. Pour un environnement Managed ou un ActiveGate, l'adresse diffère : consultez la documentation de Dynatrace ;

- **OTLP/HTTP protobuf uniquement** : gRPC n'est pas accepté ;
- métriques en temporalité **delta uniquement** : une requête ne contenant que des métriques cumulatives est rejetée (code 400), et une requête mixte est partiellement acceptée ;
- les histogrammes exponentiels perdent leurs intervalles (seuls minimum, maximum, somme et compte sont conservés), et les métriques de type *summary* ne sont pas prises en charge. La sonde utilise donc des **histogrammes explicites**, comme le fixe le préréglage `dynatrace`.

2.8.1 Jeton d'accès

Dynatrace propose deux types de jetons, selon le type d'environnement. Leurs étendues (*scopes*) diffèrent.

Type de jeton	En-tête	Étendues
Jeton d'accès classique (jeton d'API)	`Authorization: Api-Token <jeton>`	Étendues d'ingestion de l'API OTLP, par exemple `metrics.ingest` (métriques), `openTelemetryTrace.ingest` (traces) et `logs.ingest` (logs)
Jeton de plateforme	`Authorization: Bearer <jeton>`	Étendues d'ingestion propres aux jetons de plateforme, dont les noms diffèrent de ceux des jetons classiques : relevez-les dans la documentation de Dynatrace pour l'API OTLP de votre environnement

Ne cochez que les étendues correspondant aux signaux transmis. Pour un jeton de plateforme, l'étendue n'est effective que si le compte propriétaire du jeton dispose lui-même des droits correspondants. Vérifiez dans la documentation de Dynatrace le type de jeton recommandé pour votre environnement et les noms exacts de ses étendues : ils évoluent avec les versions de la plateforme.

2.8.2 Procédure

Objectif : transmettre métriques, traces et logs de la sonde à un environnement Dynatrace SaaS.

Prérequis : flux HTTPS sortant de la sonde vers `*.live.dynatrace.com`, directement ou via le proxy ; droits de création de jetons dans Dynatrace.

Étapes :

1. Dans Dynatrace, créez un jeton avec les étendues du § 2.8.1.
2. Enregistrez-le comme secret `DT_TOKEN` sur la sonde (§ 2.4.4, étape 1).
3. Créez la destination. Exemple avec un jeton classique :

   ```json
   {
     "id": "dynatrace",
     "kind": "otlp",
     "preset": "dynatrace",
     "enabled": true,
     "signals": ["metrics", "traces", "logs"],
     "endpoint": "https://abc12345.live.dynatrace.com/api/v2/otlp",
     "protocol": "http/protobuf",
     "headers": { "Authorization": "Api-Token ${secret:DT_TOKEN}" },
     "metrics": { "temporality": "delta", "histogram": "explicit", "exportInterval": "60s" }
   }
   ```

   Avec un jeton de plateforme, remplacez la valeur de l'en-tête par `Bearer ${secret:DT_TOKEN}`.
4. Testez la destination (§ 2.4.4, étape 3).
5. Dans Dynatrace, recherchez la métrique `sva.agent.up` dans l'explorateur de données ou un notebook.

Résultat attendu : les métriques `sva.*` apparaissent dans Dynatrace, les traces sont visibles dans l'explorateur de traces, et les logs `run.finished` dans la visionneuse de logs.

> **Information** — Dynatrace peut rejeter partiellement un envoi tout en renvoyant un succès (réponse de succès partiel). La sonde journalise ces rejets partiels dans ses journaux d'export (§ 2.16).

2.8.3 Exploitation dans Dynatrace

Les métriques conservent leurs noms OpenTelemetry, et leurs attributs deviennent des dimensions. Les alertes se définissent à partir de ces métriques, au moyen des mécanismes d'alerte de Dynatrace (événements métriques, règles d'anomalie). La syntaxe exacte des requêtes dépend du type d'environnement (Grail et DQL, ou sélecteurs de métriques de l'environnement classique).

Exemple de requête DQL indicative, à valider sur votre environnement :

```texte
timeseries statut = max(sva.check.status), by: { sva.check.id }
| filter arrayMax(statut) >= 2
```

Pour détecter l'arrêt d'une sonde, configurez une alerte sur l'absence de la métrique `sva.agent.up`.

2.9 OpenTelemetry Collector du client

Si votre organisation exploite déjà un **OpenTelemetry Collector** (le composant standard de réception, de transformation et de routage de la télémétrie), c'est la destination à privilégier. La sonde n'envoie qu'à un seul point, et c'est le Collector qui :

- route les données vers une ou plusieurs plateformes ;
- centralise les clés d'API des plateformes, qui ne sont alors pas stockées sur les sondes ;
- convertit les temporalités, par exemple en delta pour Dynatrace, grâce au processeur `cumulativetodelta` ;
- assure une **file d'attente persistante** sur disque, qui réduit les pertes de données en cas d'indisponibilité prolongée d'une plateforme, dans la limite de sa capacité et des données qu'il a effectivement reçues (§ 2.12).

Configuration de la destination dans la sonde :

```json
{
  "id": "collecteur",
  "kind": "otlp",
  "preset": "otlp-generic",
  "enabled": true,
  "signals": ["metrics", "traces", "logs"],
  "endpoint": "https://otel-collector.example.internal:4318",
  "protocol": "http/protobuf",
  "tls": { "caRef": "ac-interne" },
  "metrics": { "temporality": "cumulative" }
}
```

L'identifiant `ac-interne` désigne une autorité de certification interne importée dans la sonde (chapitre 5 du guide d'installation et d'administration).

Exemple de configuration du Collector : réception OTLP, conversion delta, envoi à Dynatrace avec file persistante et copie des métriques vers Prometheus. Cet exemple utilise des composants de la distribution `otelcol-contrib` du Collector.

```yaml
receivers:
  otlp:
    protocols:
      http:
        endpoint: 0.0.0.0:4318
        tls:
          cert_file: /etc/otelcol/tls/collector.crt
          key_file: /etc/otelcol/tls/collector.key

processors:
  memory_limiter:
    check_interval: 1s
    limit_percentage: 80
    spike_limit_percentage: 20
  cumulativetodelta: {}
  batch: {}

extensions:
  file_storage:
    directory: /var/lib/otelcol/file_storage

exporters:
  otlphttp/dynatrace:
    endpoint: https://abc12345.live.dynatrace.com/api/v2/otlp
    headers:
      Authorization: "Api-Token ${env:DT_TOKEN}"
    sending_queue:
      enabled: true
      storage: file_storage
  prometheusremotewrite:
    endpoint: https://prometheus.example.internal/api/v1/write

service:
  extensions: [file_storage]
  pipelines:
    metrics/dynatrace:
      receivers: [otlp]
      processors: [memory_limiter, cumulativetodelta, batch]
      exporters: [otlphttp/dynatrace]
    metrics/prometheus:
      receivers: [otlp]
      processors: [memory_limiter, batch]
      exporters: [prometheusremotewrite]
    traces:
      receivers: [otlp]
      processors: [memory_limiter, batch]
      exporters: [otlphttp/dynatrace]
    logs:
      receivers: [otlp]
      processors: [memory_limiter, batch]
      exporters: [otlphttp/dynatrace]
```

Points d'attention :

- les composants disponibles et leurs paramètres dépendent de la distribution et de la version du Collector : référez-vous à sa documentation ;
- le processeur `cumulativetodelta` n'est appliqué qu'au pipeline vers Dynatrace, car Prometheus attend des données cumulatives ;

- l'écriture vers Prometheus par *remote write* exige que le récepteur correspondant soit activé sur Prometheus ;
- le répertoire de la file persistante doit exister et appartenir au compte de service du Collector.

> ✓ **Conseil** — Lorsque vous disposez d'un Collector, faites-lui porter les clés d'API des plateformes SaaS et ne configurez sur les sondes que l'adresse du Collector. La rotation des clés se fait alors en un seul endroit, et une sonde compromise ou volée ne livre aucune clé de plateforme.

2.10 Nagios et Centreon

Pour les organisations dont l'outil de supervision principal est Nagios ou un outil compatible comme Centreon, la sonde fournit un **plugin** conforme à la convention des plugins Nagios, nommé `check_sva`. Nagios (ou Centreon) exécute ce plugin à intervalles réguliers. Le plugin interroge la sonde en HTTPS et renvoie l'état d'un contrôle sous la forme attendue : un code retour et une ligne de texte avec des données de performance.

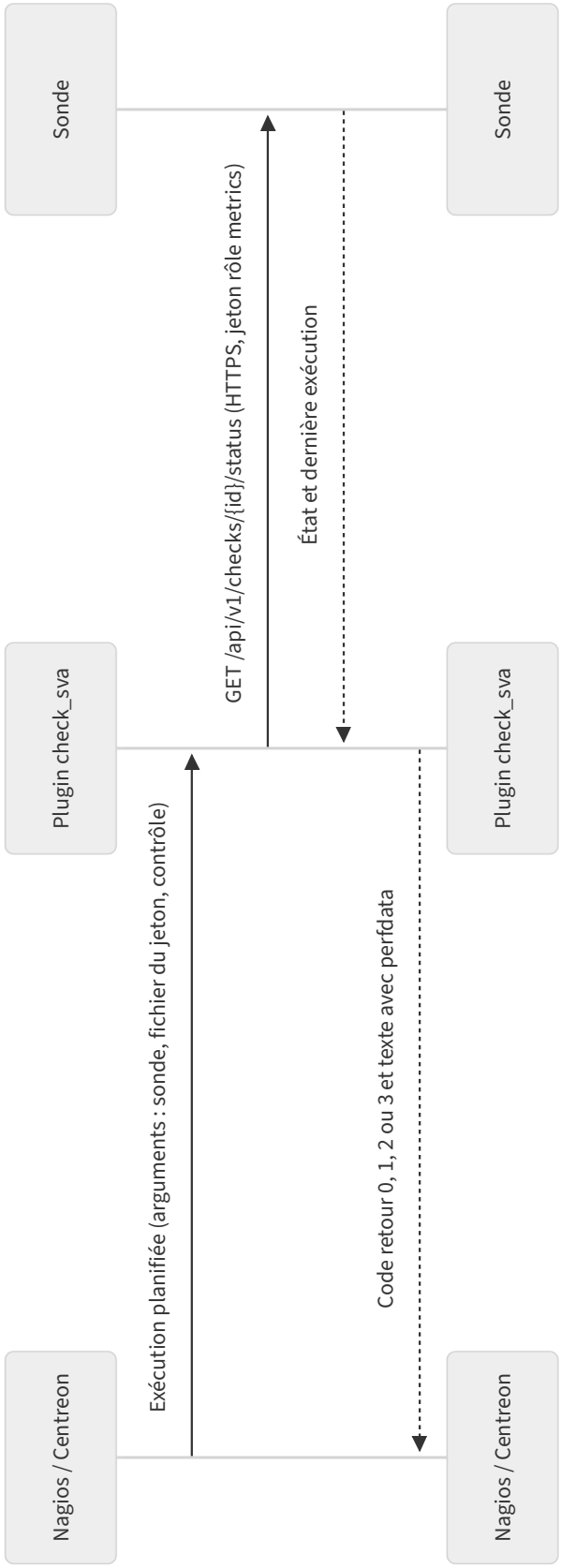


Schéma 2-2 — Interrogation d'une sonde par le plugin check_sva

Schéma présenté en orientation paysage : tournez le document d'un quart de tour pour le lire.

Ce mode fonctionne en **interrogation active** : Nagios interroge la sonde, qui n'envoie rien vers Nagios.

2.10.1 Correspondance des états

État du contrôle sur la sonde	Code retour du plugin	État Nagios
`OK`	0	OK
`WARNING`	1	WARNING
`CRITICAL`	2	CRITICAL
`UNKNOWN`	3	UNKNOWN
`STALE`	3	UNKNOWN
Sonde dans l'état de maintenance (§ 2.15.1)	3, avec une sortie de la forme `UNKNOWN - MAINTENANCE : boutique-connexion non évalué pendant l'activation d'une mise à jour`	UNKNOWN
Sonde injoignable, jeton refusé, contrôle inexistant	3	UNKNOWN

Exemple de sortie du plugin :

```
CRITICAL - boutique-connexion : étape 'Vérifier l'accès à l'espace client' (assertion_failed) |
    duration=14.65s;8;;
```

La partie qui suit le caractère `|` contient les données de performance : la durée de la dernière exécution et son seuil d'avertissement. Nagios et Centreon peuvent en tracer l'évolution.

2.10.2 Installation du plugin

Le plugin est un script Python 3 qui n'utilise que la bibliothèque standard : il ne requiert que Python 3 sur le serveur Nagios ou sur le collecteur Centreon (*poller*). Son mode de distribution est indiqué dans les notes de version. Le plugin doit être placé dans le répertoire des plugins, désigné par la macro `$USER1$` (par exemple `/usr/lib/nagios/plugins` sur une installation Debian ; adaptez à votre installation), et être exécutable par le compte de service de Nagios.

2.10.3 Syntaxe

```
check_sva -H <sonde> --token-file <fichier du jeton> -C <identifiant du contrôle> --cacert <fichier
    PEM> [-t <secondes>]
```

Option	Description
`-H`	Nom ou adresse de la sonde
`--token-file`	Fichier protégé contenant uniquement le jeton d'API de rôle `metrics`, lisible seulement par le compte qui exécute le plugin. C'est le seul mode de transmission du jeton : le plugin n'accepte pas de jeton en argument
`-C`, `--check`	Identifiant du contrôle à interroger
`--cacert`	Fichier PEM utilisé pour vérifier le certificat de la sonde : certificat autosigné de la sonde, dont l'empreinte a été vérifiée selon le § 4.4.4 du guide d'installation et d'administration, ou certificat de l'AC interne qui a émis le certificat de la sonde (§ 5.7 du guide d'installation et d'administration). À toujours renseigner

Option	Description
`-t`, `--timeout`	Délai d'exécution maximal du plugin, c'est-à-dire délai d'attente de la réponse de la sonde, en secondes (10 par défaut)

Les noms d'options suivent la convention des plugins Nagios : `-t` est réservé au délai d'exécution, et `-w` et `-c` aux seuils d'avertissement et critique. `check_sva` n'utilise pas `-w` ni `-c` : l'état qu'il renvoie est celui du contrôle, évalué par la sonde avec ses propres seuils.

> **Avertissement** — Un jeton passé en argument de commande apparaîtrait dans la liste des processus du serveur de supervision, même s'il provenait d'une macro masquée dans l'interface (macro de type mot de passe de Centreon, par exemple) : la macro est remplacée par sa valeur au moment de l'exécution. C'est pourquoi le plugin ne lit le jeton que dans le fichier désigné par `--token-file` ; les macros ne contiennent que le **chemin** de ce fichier. Utilisez en outre un jeton de rôle `metrics`, qui ne permet aucune modification, et restreignez l'accès au serveur de supervision.

2.10.4 Configuration dans Nagios

Étapes :

1. Créez sur la sonde un jeton d'API de rôle `metrics` dédié à Nagios (chapitre 6 du guide d'installation et d'administration).
2. Enregistrez le jeton dans un fichier lisible uniquement par le compte de Nagios, puis déclarez le chemin de ce fichier dans le fichier de ressources de Nagios (`resource.cfg`). Le chemin ci-dessous est un exemple, à adapter à votre serveur :

```
# resource.cfg : chemin du fichier contenant le jeton de la sonde (exemple)
# Le fichier contient uniquement le jeton ; droits recommandés : 0400, propriétaire nagios
$USER5$=/usr/local/nagios/etc/sonde-token
```

3. Déposez sur le serveur Nagios le certificat de la sonde, dont l'empreinte a été vérifiée selon le § 4.4.4 du guide d'installation et d'administration, ou celui de l'AC interne qui a émis son certificat (§ 5.7 du guide d'installation et d'administration). Déclarez son chemin dans `resource.cfg` (exemple, à adapter) :

```
# resource.cfg : certificat utilisé par check_sva pour vérifier la sonde (exemple)
$USER6$=/usr/local/nagios/etc/sonde-paris-01.pem
```

4. Définissez la commande :

```
define command {
    command_name    check_sva
    command_line    $USER1$/check_sva -H $HOSTADDRESS$ --token-file $USER5$ --cacert $USER6$ --
    check $ARG1$
}
```

5. Déclarez la sonde comme hôte, puis un service par contrôle à suivre :

```
define host {
    use                 generic-host
    host_name           sonde-paris-01
    address             sonde-paris-01.example.internal
}

define service {
    use                 generic-service
    host_name           sonde-paris-01
    service_description Boutique - connexion
    check_command       check_sva!boutique-connexion
    check_interval      5
    retry_interval      1
    max_check_attempts  2
}
```

6. Vérifiez la configuration (`nagios -v <fichier de configuration principal>`), puis rechargez Nagios.

Résultat attendu : le service « Boutique - connexion » prend l'état du contrôle correspondant sur la sonde.

> ✓ **Conseil** — Alignez l'intervalle de vérification de Nagios sur la fréquence du contrôle dans la sonde. Interroger la sonde toutes les minutes pour un contrôle exécuté toutes les 5 minutes n'apporte aucune information nouvelle. Les nouvelles tentatives de la sonde (§ 7.6 du guide d'utilisation et de supervision) confirment déjà les échecs : limitez `max_check_attempts` pour ne pas retarder inutilement l'alerte.

> ℹ **Information** — Le modèle `generic-host` utilise par défaut une vérification par ping de l'hôte. Si le ping vers la sonde est filtré sur votre réseau, adaptez la commande de vérification de l'hôte, par exemple avec un test HTTPS de l'adresse `/health` de la sonde, qui ne nécessite pas d'authentification.

2.10.5 Configuration dans Centreon

Centreon s'appuie sur les mêmes principes. La configuration s'effectue dans son interface web :

1. Copiez le plugin sur chaque collecteur (*poller*) Centreon concerné (§ 2.10.2). Déposez aussi sur chaque collecteur le certificat de la sonde, dont l'empreinte a été vérifiée selon le § 4.4.4 du guide d'installation et d'administration, ou celui de l'AC interne qui a émis son certificat (§ 5.7 du guide d'installation et d'administration), et déclarez son chemin dans une macro de ressource, ici `$USER6$`.
2. Créez sur la sonde un jeton d'API de rôle `metrics` dédié à Centreon (chapitre 6 du guide d'installation et d'administration). Sur chaque collecteur concerné, enregistrez-le dans un fichier qui ne contient que le jeton, lisible uniquement par le compte qui exécute les plugins (droits recommandés : 0400). Le chemin de ce fichier est propre à votre installation.
3. Créez une **commande** de type vérification, avec la ligne de commande :

```
$USER1$/check_sva -H $HOSTADDRESS$ --token-file $_HOSTSVATOKENFILE$ --cacert $USER6$ --check
    $_SERVICECHECKID$
```

4. Sur l'hôte représentant la sonde, définissez la **macro personnalisée** `SVATOKENFILE` avec le **chemin** du fichier créé à l'étape 2. Cette macro ne contient jamais la valeur du jeton.
5. Créez un **modèle de service** utilisant cette commande, avec une macro de service `CHECKID`.
6. Créez un service par contrôle à suivre, en renseignant la macro `CHECKID` (par exemple `boutique-connexion`).
7. Exportez la configuration vers les collecteurs.

> **Avertissement** — Ne placez jamais le jeton lui-même dans une macro, même de type mot de passe : une macro masquée dans l'interface de Centreon est remplacée par sa valeur au moment de l'exécution et apparaîtrait dans la liste des processus des collecteurs (§ 2.10.3). Le plugin n'accepte d'ailleurs le jeton que par `--token-file`. Si d'anciennes configurations passent le jeton en argument, remplacez-les par la commande ci-dessus.

Les libellés exacts des menus dépendent de votre version de Centreon : référez-vous à sa documentation.

2.10.6 Quels contrôles suivre dans Nagios ou Centreon

Il n'est pas nécessaire de déclarer un service Nagios pour chaque contrôle de la sonde. Une pratique courante consiste à ne déclarer que les contrôles qui doivent déclencher une alerte dans l'outil de supervision principal (parcours critiques, API métier), et à consulter les autres dans l'interface de la sonde ou dans une plateforme d'observabilité.

> **Information** — Une fonction de découverte automatique des contrôles de la sonde par Centreon ou Nagios n'est pas prévue.

2.11 Journaux d'audit vers un SIEM

Les opérations d'administration de la sonde sont enregistrées dans son journal d'audit (chapitre 6 du guide d'installation et d'administration) : connexions, modifications de configuration, changements de secrets (sans leur valeur), téléversements de scénarios, mises à jour. Pour les centraliser dans un **SIEM** (système de gestion des informations et des événements de sécurité), la sonde peut transmettre ces événements sous forme de logs OTLP, de type `audit.*`.

Mise en œuvre :

1. Activez l'export du journal d'audit sur la destination OTLP qui le recevra, avec le paramètre `logs.audit: true` (§ 2.4.3). Il est désactivé par défaut.
2. Configurez cette destination pour qu'elle transmette le signal `logs`, de préférence vers l'OpenTelemetry Collector de votre organisation. Celui-ci se charge de la transmission au SIEM, dans le format attendu par ce dernier.
3. Dans le SIEM, filtrez les événements sur `service.name = sva-agent` et les noms d'événements `audit.*`.

Le journal d'audit reste par ailleurs consultable sur la sonde (ressource `GET /api/v1/audit-events`, rôle `admin`), avec une rétention de 365 jours par défaut.

2.12 Comportement en cas d'indisponibilité d'une destination

L'indisponibilité d'une plateforme d'observabilité n'interrompt jamais la supervision. La sonde continue d'exécuter ses contrôles, d'enregistrer ses résultats et de les présenter localement.

Situation	Comportement de la sonde
Erreur temporaire : 429, 502, 503, 504, coupure réseau	Nouvelles tentatives avec un délai croissant, en respectant l'en-tête `Retry-After` s'il est fourni
Erreur définitive : 400, 401, 403, 404, 413	Abandon du lot concerné, journalisation de l'erreur, destination signalée en erreur dans la santé de la sonde

Situation	Comportement de la sonde
Indisponibilité au-delà de `retry.maxElapsed` (5 minutes par défaut)	Abandon des lots les plus anciens ; incrément du compteur `sva.export.dropped`
File d'attente pleine (`queue.maxBatches`)	Suppression des lots les plus anciens au profit des plus récents
Retour de la destination	Reprise de l'envoi des nouvelles données

Conséquences :

- **Métriques** : les points produits pendant une indisponibilité prolongée sont perdus pour la plateforme externe. Une courbe présentera un trou sur cette période. Les résultats restent intégralement consultables sur la sonde pendant la durée de rétention (chapitre 7 du guide d'utilisation et de supervision).
- **Traces et logs** : ils suivent la même règle que les métriques.
- **Collecte Prometheus** : en mode pull, Prometheus ne récupère pas a posteriori les mesures d'une période où il n'a pas pu joindre la sonde.

→ **Évolution prévue** — Une version ultérieure permettra de rejouer, après le retour d'une destination, les traces et logs des exécutions survenues pendant l'indisponibilité, à partir de l'historique local de la sonde et dans la limite des fenêtres d'acceptation des plateformes.

✓ **Conseil** — Si votre organisation exige de **réduire les pertes** de télémétrie lors des indisponibilités de ses plateformes, faites transiter les données par un OpenTelemetry Collector disposant d'une file d'attente persistante (§ 2.9). Placez ce Collector au plus près des sondes, sur un serveur plus robuste qu'une carte microSD. Ce dispositif réduit les pertes ; il ne garantit pas la livraison, pour les raisons exposées au paragraphe « Portée d'un Collector à file persistante ».

Portée d'un Collector à file persistante. La chaîne comporte deux segments, qui ne sont pas protégés de la même manière :

Segment	Ce qui protège les données	Ce qui reste perdu
Sonde → Collector	Les nouvelles tentatives et la file en mémoire de la sonde (`retry.maxElapsed` , `queue.maxBatches` , tableau ci-dessus). Le Collector étant proche et disponible, ce segment échoue rarement	Les lots que la sonde abandonne au terme de ses reprises, ou qu'elle supprime lorsque sa file est pleine ; les lots en mémoire au moment d'un redémarrage de la sonde. Ces abandons sont comptés par `sva.export.dropped` (série `sva_export_dropped_total` , alerte `ExportDonneesPerdues`)
Collector → destination	La file persistante du Collector (`sending_queue` adossée à une extension de stockage), qui conserve les données reçues pendant l'indisponibilité de la destination	Les données au-delà de la capacité de la file ou de sa durée de rétention, celles que la destination refuse (erreur définitive, données trop anciennes pour sa fenêtre d'acceptation) et celles perdues en cas de panne du disque du Collector. Surveillez les métriques internes du Collector (taille de la file, données refusées ou abandonnées)

Le Collector ne peut préserver que ce qu'il a reçu. L'**historique local** de la sonde reste la référence en cas de doute : il conserve les résultats pendant la durée de rétention (chapitre 7 du guide d'utilisation et de supervision), mais n'est pas renvoyé automatiquement vers les plateformes. Dimensionnez la file du Collector d'après la durée d'indisponibilité de destination que vous voulez absorber, et alertez sur `sva_export_dropped_total` comme sur les abandons du Collector.

Pour diagnostiquer une destination en erreur, utilisez le test de destination (§ 2.4.4), puis le tableau de dépannage des exports (§ 2.16).

2.12.1 Maintenance de la sonde et alertes

L'activation d'une mise à jour rend la sonde momentanément injoignable : ses données cessent d'arriver dans les plateformes, qui déclenchent alors des alertes d'absence de données alors qu'il s'agit d'une opération planifiée. La prévention de ces fausses alertes (mise en sourdine des fenêtres de maintenance, silences ponctuels, alertes propres aux mises à jour) et les alertes fondées sur l'ensemble des sondes d'un site sont décrites au § 2.15, « Exploitation des alertes ».

2.13 Sécurité des intégrations

Mesure	Détail
Secrets	Toute clé d'API ou tout jeton de plateforme est stocké comme secret de la sonde et référencé par `${secret:NOM}`. Il n'est jamais saisi en clair dans une configuration
Moindre privilège	Jetons de plateforme limités à l'ingestion des seuls signaux transmis ; jeton de rôle `metrics` pour Prometheus, Nagios et Centreon
Un jeton par usage	Un jeton distinct par outil consommateur (Prometheus, Nagios, etc.) et, si possible, par sonde, pour révoquer finement
Chiffrement	HTTPS vers les plateformes SaaS ; HTTPS pour `/metrics` et `check_sva` ; TLS recommandé vers un Collector ou un Agent interne
Vérification des certificats	Ne désactivez jamais la vérification (`insecureSkipVerify`) en production ; importez plutôt l'autorité de certification interne (chapitre 5 du guide d'installation et d'administration)
Flux réseau	Seuls les flux nécessaires sont ouverts : sortant vers les destinations configurées, entrant 443 depuis les postes d'administration et, le cas échéant, Prometheus, Nagios ou Centreon (§ 6.1, chapitre 6)
Données transmises	Aucune valeur de secret, aucune adresse complète avec paramètres, aucun identifiant d'utilisateur dans les attributs de métriques. Les messages d'erreur transmis dans les logs et les traces sont masqués, mais peuvent contenir des informations techniques sur vos applications : choisissez des destinations dont l'hébergement et la politique de conservation sont conformes à vos exigences
Rotation	Remplacez les secrets périodiquement par l'API (`PUT /api/v1/secrets/{name}`), sans interruption des exports
Concentration des clés	Avec un Collector, les clés des plateformes SaaS restent sur le Collector, et non sur les sondes (§ 2.9)

2.14 Choisir une stratégie d'intégration

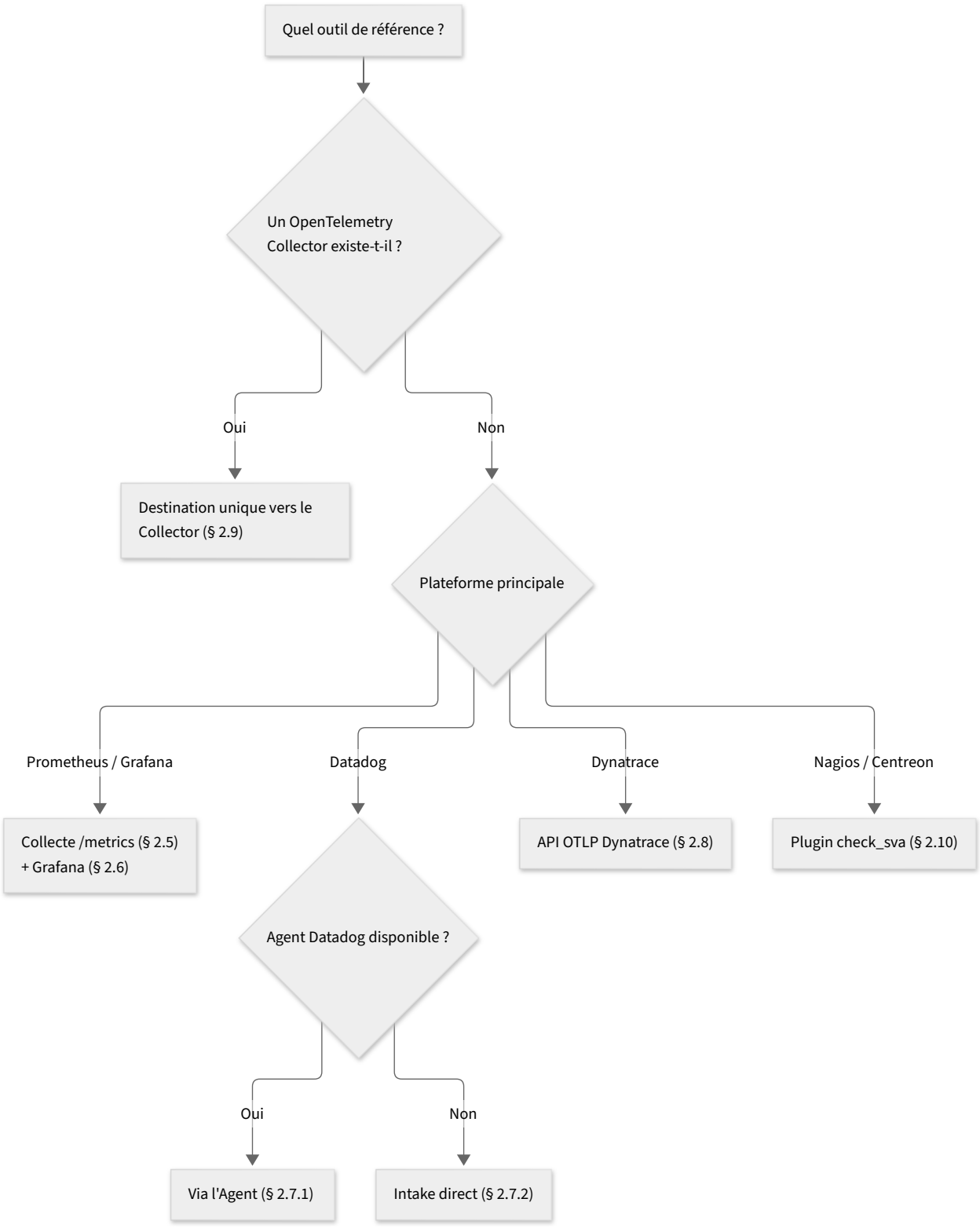


Schéma 2-3 — Choisir une stratégie d'intégration

Les modes ne sont pas exclusifs. Une configuration fréquente associe :

- le plugin `check_sva` pour les alertes dans l'outil historique de supervision ;
- l'export OTLP vers la plateforme d'observabilité, pour l'analyse, l'historique et les tableaux de bord ;

- la consultation locale de la sonde pour le diagnostic détaillé d'un échec (captures d'écran, console).

2.15 Exploitation des alertes

Cette section regroupe ce qu'il faut configurer dans vos plateformes d'alerte, une fois les intégrations en place, pour exploiter les alertes de la sonde dans la durée : éviter les fausses alertes pendant l'activation des mises à jour, traiter les activations hors fenêtre, suivre les alertes propres aux mises à jour et, sur un site équipé de plusieurs sondes, fonder les alertes sur l'ensemble des sondes du site.

2.15.1 Maintenance de la sonde et fausses alertes

L'activation d'une mise à jour, c'est-à-dire le redémarrage de la sonde sur son nouvel emplacement système, a lieu dans une **fenêtre de maintenance** définie par le client (`updates.windows`) pour une activation automatique ordinaire ou un correctif critique en mode `next-window` (réglage par défaut), à l'heure choisie par un administrateur (`POST /api/v1/updates/apply`, éventuellement avec `scheduledAt`), ou, pour un correctif critique en mode `asap` choisi explicitement, 15 minutes après son annonce, augmentées du décalage de la sonde, même hors fenêtre. La sonde n'a **aucune fenêtre par défaut** : sans fenêtre, une mise à jour ordinaire ou un correctif critique en mode `next-window` reste en attente d'activation. La matrice complète figure au § 8.4.2 du guide d'installation et d'administration. Le choix des fenêtres et le déroulement d'une mise à jour sont décrits au § 8.4 du guide d'installation et d'administration. Cette section explique comment éviter que ces activations planifiées ne déclenchent de fausses alertes dans vos plateformes, sans masquer les incidents réels.

Déroulement d'une activation vu des plateformes

Phase	Comportement de la sonde	Ce que voient les plateformes
Préavis	Selon le cas d'activation (§ 8.4.2 du guide d'installation et d'administration) : au moins `updates.notice` (24 heures par défaut) pour une activation automatique ordinaire ; au moins 15 minutes pour un correctif critique ; aucun pour une activation manuelle immédiate (`POST /api/v1/updates/apply` sans `scheduledAt`). Un préavis inférieur à `updates.notice` est signalé par `noticeReduced: true`	Événement de log `update.scheduled` ; métriques `sva.update.pending`, `sva.maintenance.next_start` et `sva.maintenance.next_end` ; champ `scheduledActivation` de `GET /api/v1/updates`
Drainage	État de maintenance : aucune exécution nouvelle ne démarre ; celles en cours se terminent dans la limite de `updates.drainTimeout` (5 minutes par défaut, 30 minutes au maximum), puis sont interrompues en `UNKNOWN` avec le type d'erreur `maintenance`	`sva_agent_maintenance` = 1 ; événement `maintenance.started` ; `/health` répond `{"status": "MAINTENANCE"}` avec le code HTTP 503 ; `check_sva` renvoie 3 (`UNKNOWN`) avec le texte `MAINTENANCE`
Redémarrage	La sonde redémarre sur le nouvel emplacement	**Plus aucune donnée** : collecte `/metrics` en échec (`up` = 0), envois OTLP interrompus, `/health` et `check_sva` sans réponse de la sonde
Validation	Autotest de 10 minutes au plus ; en cas d'échec, retour automatique à l'emplacement précédent, avec un nouveau redémarrage	Reprise progressive des données ; éventuelle nouvelle interruption en cas de retour arrière
Reprise	Sortie de l'état de maintenance	Événement `maintenance.ended` ; `sva_agent_maintenance` = 0 ; reprise des exécutions

La durée d'indisponibilité d'une activation s'observe sur vos sondes entre les événements `maintenance.started` et `maintenance.ended` (§ 8.4.5 du guide d'installation et d'administration) ; elle dépend notamment du délai de drainage (`updates.drainTimeout`) et de l'issue de l'autotest.

Pourquoi l'état de maintenance ne suffit pas

L'état de maintenance est exposé par la sonde elle-même. Il est donc visible pendant le drainage, mais **plus du tout pendant le redémarrage** : une sonde qui redémarre ne répond plus. Conséquences :

- en mode collecte, Prometheus constate l'échec de la collecte (`up` = 0) et marque aussitôt comme périmées toutes les séries de la sonde, y compris `sva_agent_maintenance`. Une règle qui s'appuie sur cette série ne peut plus en tenir compte ;
- en mode envoi, les séries cessent simplement d'arriver ; les règles d'absence (`SondeSilencieuse`) se déclenchent après quelques minutes ;
- Nagios et Centreon reçoivent un `UNKNOWN` « sonde injoignable », et la vérification de l'hôte fondée sur `/health` échoue ;
- les moniteurs Datadog et Dynatrace d'absence de données (`sva.agent.up`) se déclenchent.

La sonde suspend son propre calcul de l'état `STALE` pendant la maintenance, puis pendant un intervalle de planification de chaque contrôle après l'événement `maintenance.ended`, mais cette suspension n'a pas d'effet sur les plateformes externes. Il faut donc **déclarer les fenêtres de maintenance dans chaque plateforme d'alerte**, en complément des règles qui exploitent l'état de maintenance pendant le drainage.

Règles communes à toutes les plateformes

1. **Reproduisez chaque fenêtre de la sonde.** Pour chaque entrée de `updates.windows` (`days`, `start`, `duration`), déclarez une période récurrente équivalente dans la plateforme, limitée à la sonde concernée (hôte, tag ou label).
2. **Alignez les fuseaux horaires.** Les fenêtres de la sonde s'interprètent dans le fuseau horaire configuré sur la sonde (§ 6.9 du guide d'installation et d'administration). Déclarez la période de la plateforme dans ce même fuseau, sous la forme d'un identifiant IANA (`Europe/Paris`, par exemple), et non en UTC : sinon, la période se décale d'une ou deux heures selon la saison.
3. **Prévoyez une marge après la fin de la fenêtre.** Une activation peut débuter jusqu'à la fin de la fenêtre. La période de la plateforme doit donc couvrir, au-delà de la fin de la fenêtre : le drainage (`updates.drainTimeout`), le redémarrage, l'autotest de 10 minutes et, le cas échéant, le retour arrière, qui comporte un second redémarrage. Les exemples de cette section retiennent une marge indicative de 45 minutes, à ajuster d'après la durée mesurée (§ 8.4 du guide d'installation et d'administration) et votre valeur de `drainTimeout`.
4. **Couvrez aussi les activations hors fenêtre.** Une activation demandée par un administrateur (`POST /api/v1/updates/apply`, avec ou sans `scheduledAt`) ou un correctif critique en mode `asap` (`updates.security.criticalPolicy: asap`) peut avoir lieu en dehors des fenêtres récurrentes. Elle est annoncée par l'événement `update.scheduled` et par `sva.maintenance.next_start`, mais **sans fin prévue** et, pour une activation manuelle immédiate, **au moment même où elle commence**. Créez alors une suspension ponctuelle (silence Alertmanager, temps d'arrêt ponctuel) d'une durée maximale que vous fixez d'après la durée observée d'une activation, et levez-la après la reprise vérifiée de la sonde : procédure manuelle ou automatisation pour Alertmanager au § 2.15.4.
5. **Ne masquez que la sonde concernée.** Ne déclarez jamais une période qui suspend les alertes de toutes les sondes ou de tout l'environnement.

> ! **Attention** — Pendant la période de maintenance déclarée, les alertes de la sonde concernée sont suspendues, y compris celles qui signaleraient une panne réelle des applications supervisées. C'est la raison pour laquelle les environnements critiques doivent disposer de plusieurs sondes par site (§ 2.15.6).

Cas des correctifs de sécurité critiques. Le traitement des correctifs critiques est détaillé au § 8.4.11 du guide d'installation et d'administration ; en voici l'essentiel pour vos plateformes. Un correctif est dit critique lorsqu'il corrige une vulnérabilité activement exploitée. Il est installé dès sa publication sur l'emplacement inactif, sans

interrompre les contrôles ; l'état de santé `updates` passe alors en `CRITICAL` et la métrique `sva.update.pending` porte `sva.update.critical` = `true` (alerte `CorrectifCritiqueEnAttente`, § 2.15.3). Son activation suit le paramètre `security.criticalPolicy` : en mode `next-window` (par défaut), elle a lieu au début de la première fenêtre qui commence au moins 15 minutes après l'annonce, et la mise en sourdine récurrente la couvre ; sans fenêtre définie, le correctif attend qu'un administrateur l'active. En mode `asap`, choisi explicitement, elle a lieu 15 minutes après l'annonce, augmentées du décalage `security.criticalStagger` de la sonde (de `0` à `4h`), même hors fenêtre : elle relève alors de la suspension ponctuelle de la règle 4. Lorsque le canal `security` est en mode `manual`, aucun correctif n'est activé automatiquement. Le report est limité à 7 jours, et un administrateur peut toujours avancer l'activation. Sur un site équipé de plusieurs sondes, des décalages `security.criticalStagger` différents réduisent le risque d'activations simultanées sans l'exclure (§ 2.15.6).

2.15.2 Mise en sourdine dans Prometheus et Alertmanager

Objectif : ne pas notifier les alertes d'une sonde pendant ses fenêtres de maintenance, et neutraliser les alertes de contrôle pendant le drainage.

Prérequis : règles d'alerte des sondes chargées dans Prometheus (§ 2.5.5) ; droits d'administration sur la configuration d'Alertmanager ; fenêtres de maintenance de chaque sonde (`GET /api/v1/settings/updates`) et fuseau horaire de chaque sonde.

Étapes :

1. Chargez la version à jour du fichier `exemples/prometheus/sva-regles-alerte.yml`. Ses règles `ControleNonEvaluable` et `SondeSilencieuse` comportent la clause `unless on(instance) sva_agent_maintenance == 1`, qui couvre la phase de drainage (§ 2.5.5).
2. Dans la configuration d'Alertmanager, déclarez un **intervalle de temps** (`time_intervals`) par sonde, qui reproduit ses fenêtres augmentées de la marge, dans le fuseau horaire de la sonde (champ `location`).
3. Ajoutez, sous la route racine, une **route par sonde** qui sélectionne ses alertes de **disponibilité et d'état** (label `instance` et nom d'alerte `alertname`) et qui porte `mute_time_intervals` avec le nom de l'intervalle. La route racine ne peut pas porter `mute_time_intervals` : utilisez toujours une route enfant.
4. Placez **avant** les routes par sonde une route **sans sourdine** pour les alertes de mises à jour et de sécurité (`CorrectifCritiqueEnAttente`, `MaintenancePrologee`, alertes de certificats et d'alimentation) : ces alertes doivent être notifiées même pendant une fenêtre.
5. Validez la configuration avec `amtool check-config`, vérifiez l'acheminement avec `amtool config routes test`, puis rechargez Alertmanager.

> **Attention** — Une route en sourdine suspend **toutes** les alertes qu'elle sélectionne, pendant toute la durée de l'intervalle, **même si aucune activation n'a lieu** ce jour-là. Une route sélectionnée par le seul label `instance` suspendrait donc aussi un incident indépendant survenu pendant la fenêtre, un correctif critique en attente ou une maintenance qui se prolonge. Limitez la sourdine aux alertes qu'une activation déclenche normalement (`SondeInjoignable`, `SondeSilencieuse`, `ControleNonEvaluable`, `ParcoursEnEchec`, et `SondeAttendueAbsente` si vous l'utilisez), comme dans l'exemple ci-dessous. Les alertes de site (`ParcoursEnEchecSurLeSite`, `CouvertureInsuffisanteSurLeSite` et, si vous l'utilisez, `ControleAttenduNonCouvertSurLeSite`) ne portent pas de label `instance` et ne sont pas concernées. Pour une sourdine encore plus étroite, renoncez aux intervalles récurrents et créez un silence ponctuel à chaque annonce d'activation (automatisation du § 2.15.4, déclenchée par l'événement `update.scheduled`) : la sourdine ne s'applique alors qu'aux jours où une activation est réellement planifiée.

Exemple pour deux sondes d'un même site dont les fenêtres ne se chevauchent pas (fenêtres **fictives** : mardi et jeudi à 04:00 pendant 1 heure, marge de 45 minutes). Le fichier complet, avec ses commentaires, est fourni dans `exemples/prometheus/alertmanager-maintenance.yml` :

```yaml
route:
  receiver: equipe-supervision
  group_by: ['alertname', 'instance']
  routes:
    # Alertes de mises à jour et de sécurité : jamais mises en sourdine.
    # Placée en premier, cette route capte ces alertes avant les routes par sonde.
    - matchers:
        -
    alertname=~"CorrectifCritiqueEnAttente|MaintenanceProlongee|CertificatBientotExpire|Certificat802
    1XBientotExpire|SondeSousTension"
      receiver: equipe-supervision

    # Alertes de disponibilité et d'état de la sonde sonde-paris-01
    - matchers:
        - instance="sonde-paris-01.example.internal:443"
        - alertname=~"SondeInjoignable|SondeSilencieuse|ControleNonEvaluable|ParcoursEnEchec"
      receiver: equipe-supervision
      mute_time_intervals:
        - maintenance-sonde-paris-01

    # Alertes de disponibilité et d'état de la sonde sonde-paris-02
    - matchers:
        - instance="sonde-paris-02.example.internal:443"
        - alertname=~"SondeInjoignable|SondeSilencieuse|ControleNonEvaluable|ParcoursEnEchec"
      receiver: equipe-supervision
      mute_time_intervals:
        - maintenance-sonde-paris-02

    # Toutes les autres alertes suivent la route racine, sans sourdine.

time_intervals:
  - name: maintenance-sonde-paris-01
    time_intervals:
      - weekdays: ['tuesday']
        times:
          - start_time: '04:00'
            end_time: '05:45'
        # Fuseau horaire de la sonde (identifiant IANA). Sans location, Alertmanager utilise UTC.
        location: 'Europe/Paris'

  - name: maintenance-sonde-paris-02
    time_intervals:
      - weekdays: ['thursday']
        times:
          - start_time: '04:00'
            end_time: '05:45'
        location: 'Europe/Paris'

receivers:
  # Destinataire fictif : ajoutez votre intégration (courriel, messagerie, outil d'astreinte).
  - name: equipe-supervision
```

Points de syntaxe à connaître :

- dans `times`, l'heure de début est incluse et l'heure de fin exclue ; l'heure de début doit être strictement antérieure à l'heure de fin. Une fenêtre qui franchit minuit se déclare donc en deux entrées (par exemple `23:00`-`24:00` le premier jour et `00:00`-`00:45` le jour suivant) ;
- sans `location`, l'intervalle est interprété en UTC ;

- `mute_time_intervals` n'est pas hérité par les routes enfants ;
- une route en sourdine n'envoie aucune notification, mais l'alerte reste active dans Prometheus et visible dans Alertmanager ;
- les routes sont examinées dans l'ordre et la première qui correspond l'emporte : la route des alertes de mises à jour et de sécurité doit donc précéder les routes par sonde. Les alertes qui ne correspondent à aucune route enfant suivent la route racine, sans sourdine.

Le paramètre `active_time_intervals` fonctionne en sens inverse : la route ne notifie **que** pendant les intervalles listés. Il convient à une route dédiée aux alertes de maintenance que vous souhaitez recevoir uniquement pendant les fenêtres, et non à la suppression des alertes.

Résultat attendu : pendant chaque fenêtre (marge comprise), les alertes de disponibilité et d'état de la sonde concernée ne sont pas notifiées ; ses alertes de mises à jour et de sécurité, ainsi que les alertes des autres sondes et les alertes de site, le sont normalement.

Vérification :

- `amtool config routes test --config.file=<fichier> alertname=SondeInjoignable instance=<instance de la sonde>` indique le destinataire choisi pour une alerte de disponibilité de cette sonde ; répétez le test avec `alertname=CorrectifCritiqueEnAttente` pour vérifier que cette alerte suit la route sans sourdine. Avec un destinataire unique, comme dans l'exemple, attribuez temporairement un destinataire distinct à chaque route pour distinguer les routes empruntées ;
- lors d'une première activation, vérifiez dans l'interface d'Alertmanager que les alertes `SondeInjoignable` ou `SondeSilencieuse` apparaissent sans être notifiées, puis disparaissent après la reprise.

Erreurs possibles : fuseau horaire omis (intervalle en UTC) ; label `instance` différent de celui des alertes (en mode envoi, il vaut `service.instance.id`) ; fenêtre franchissant minuit déclarée en une seule entrée (refusée par `amtool check-config`) ; marge trop courte, qui laisse passer une alerte pendant l'autotest ou un retour arrière.

Précautions : une fenêtre modifiée sur la sonde doit être reportée dans Alertmanager ; vérifiez la concordance à chaque modification de `updates.windows`.

2.15.3 Alertes propres aux mises à jour

Le fichier de règles comprend un groupe `sondes-mises-a-jour`, identique à l'extrait ci-dessous :

```yaml
groups:
  - name: sondes-mises-a-jour
    rules:
      # Mise à jour installée sur l'emplacement inactif, mais aucune activation planifiée :
      # la sonde n'a pas de fenêtre de maintenance (updates.windows vide) et aucun administrateur
      # n'a programmé l'activation (POST /api/v1/updates/apply). L'état de santé updates est en
    WARNING.
      # Sans fenêtre ni activation planifiée, sva_maintenance_next_start_seconds vaut 0
    (documentation SYNOVEX,
      # « Notification, drainage et état de maintenance ») :
      # ce zéro est contractuel, et la règle l'exige explicitement (« and … == 0 »). Une série
      # absente traduit au contraire une perte de collecte, qui peut ne toucher que cette série :
      # la règle ne l'interprète jamais comme « aucune activation planifiée » et ne se déclenche
      # pas. La perte de toutes les séries de la sonde relève de SondeInjoignable ou de
      # SondeSilencieuse ; la disparition de la seule série d'horodatage n'est signalée par
      # aucune règle de ce fichier.
      - alert: MiseAJourEnAttenteSansFenetre
        expr: |
          max by (job, instance, sva_update_channel) (sva_update_pending == 1)
            and on(job, instance)
          (sva_maintenance_next_start_seconds == 0)
        for: 24h
```

```
        labels:
          severity: warning
        annotations:
          summary: "Mise à jour en attente sans activation planifiée sur {{ $labels.instance }}"
          description: "Une mise à jour du canal {{ $labels.sva_update_channel }} attend son
    activation depuis au moins 24 heures, sans fenêtre de maintenance ni activation programmée.
    Définissez une fenêtre (updates.windows) ou activez la mise à jour (POST /api/v1/updates/apply,
    éventuellement avec scheduledAt)."

      # Correctif de sécurité critique (vulnérabilité activement exploitée) en attente d'activation.
      # L'état de santé updates est en CRITICAL. Report possible de 7 jours au plus.
      - alert: CorrectifCritiqueEnAttente
        expr: max by (job, instance, sva_update_channel)
    (sva_update_pending{sva_update_critical="true"}) == 1
        for: 5m
        labels:
          severity: critical
        annotations:
          summary: "Correctif de sécurité critique en attente sur {{ $labels.instance }}"
          description: "Un correctif critique est installé sur l'emplacement inactif et attend son
    activation. Vérifiez la prochaine fenêtre de maintenance (sva_maintenance_next_start_seconds) ou
    activez-le (POST /api/v1/updates/apply). Consultez l'avis de sécurité correspondant."

      # Sonde joignable, mais dans l'état de maintenance depuis plus longtemps que prévu.
      # Seuil indicatif : drainTimeout maximal (30 minutes) et une marge de 10 minutes.
      # Pendant le redémarrage, la série disparaît : le décompte de « for » repart alors de zéro.
      - alert: MaintenanceProlongee
        expr: max by (job, instance) (sva_agent_maintenance) == 1
        for: 40m
        labels:
          severity: warning
        annotations:
          summary: "Maintenance prolongée sur {{ $labels.instance }}"
          description: "La sonde {{ $labels.instance }} signale l'état de maintenance depuis plus de
    40 minutes. Aucune nouvelle exécution ne démarre pendant cet état. Consultez GET /api/v1/updates
    et le journal d'audit de la sonde."
```

Règle	Déclenchement	Action attendue
`MiseAJourEnAttenteSansFenetre`	Une mise à jour attend son activation depuis 24 heures sans fenêtre ni activation programmée	Définir une fenêtre (`updates.windows`) ou activer la mise à jour (`POST /api/v1/updates/apply`)
`CorrectifCritiqueEnAttente`	Un correctif critique attend son activation	Vérifier la date de la prochaine activation ; l'avancer si nécessaire. Le report d'un correctif critique est limité à 7 jours
`MaintenanceProlongee`	La sonde est joignable mais signale l'état de maintenance depuis plus de 40 minutes	Consulter `GET /api/v1/updates` et le journal d'audit ; en cas de blocage, chapitre 9 du guide d'installation et d'administration

2.15.4 Silences ponctuels pour les activations hors fenêtre

Procédure manuelle de référence. Pour une activation hors fenêtre (règle commune 4 du § 2.15.1), la procédure suivante reste sûre quelles que soient les données reçues. Elle convient seule aux activations manuelles immédiates, dont l'annonce arrive trop tard pour un automatisme.

1. Avant une activation manuelle, ou dès la réception de l'annonce dans les autres cas, créez un silence limité à la

sonde et aux alertes de disponibilité et d'état :

```bash
# Silence de 90 minutes au plus (durée indicative, à fixer d'après la durée observée
# d'une activation, retour arrière compris). Adresse et labels fictifs.
amtool silence add --alertmanager.url=https://alertmanager.example.internal \
  --duration=90m --comment="Activation manuelle sonde-paris-01" \
  job="sondes" instance="sonde-paris-01.example.internal:443" \
  alertname=~"SondeInjoignable|SondeSilencieuse|ControleNonEvaluable|ParcoursEnEchec"
```

2. Lancez ou laissez se dérouler l'activation, et notez l'identifiant d'activation (réponse de `POST /api/v1/updates/apply` ou attribut de l'annonce).
3. Après l'événement `maintenance.ended`, vérifiez la reprise comme au § 4.3 : identifiant et résultat de `lastResult`, version active, `/health`, arrivée de nouveaux résultats.
4. Expirez alors le silence (`amtool silence expire <identifiant>`). Si la reprise n'est pas vérifiée, ne prolongez pas le silence : laissez-le expirer pour que les alertes reprennent, et traitez l'incident (§ 9.7 du guide d'installation et d'administration).

Option : automatisation par les événements de mise à jour. Un automatisme de votre chaîne de traitement des logs (par exemple déclenché par les événements reçus par votre OpenTelemetry Collector ou votre SIEM) peut appliquer la même procédure. Il doit respecter les règles suivantes, faute de quoi un silence peut masquer un incident sur un mauvais créneau :

Règle	Mise en œuvre
Identifier l'activation	Chaque silence porte l'identifiant `sva.update.activation.id` et le numéro `sva.update.announcement.sequence` de l'annonce qui l'a créé
Ordonner et dédupliquer	Pour une sonde, seule l'annonce de plus grand numéro compte : une annonce ou une annulation dont le numéro ne dépasse pas le dernier traité est ignorée (doublon ou événement reçu en retard)
Sérialiser	Les événements d'une même sonde sont traités l'un après l'autre (verrou par sonde) ; deux sondes différentes sont traitées indépendamment
Remplacer sans régresser	Un nouveau silence n'expire que les silences créés pour une annonce **antérieure** de la même sonde ; il n'en touche aucun autre
Annuler	`update.cancelled` expire les silences des annonces antérieures de la sonde, sans en créer
Borner	La fin du silence est la fin annoncée (`sva.maintenance.end`) augmentée d'une marge ; sans fin annoncée, une durée maximale fixée par vous. Aucun silence n'est illimité
Lever après reprise	À `maintenance.ended`, le silence de **cette** activation n'est expiré qu'une fois la reprise constatée (sonde joignable et sortie de l'état de maintenance) ; sinon il s'achève à sa borne

Les métriques `sva_maintenance_next_start_seconds` et `sva_maintenance_next_end_seconds` ne conviennent pas à cet usage : interrogées séparément, elles peuvent refléter une autre annonce ou manquer. Prometheus sert seulement à vérifier que la sonde émettrice est connue de façon unique, puis qu'elle a repris. Exemple à adapter, pour un serveur Linux disposant de Bash, `curl`, `jq`, `flock` (paquet util-linux), `amtool` et de la commande `date` de GNU coreutils ; le fichier est fourni dans `exemples/prometheus/silence-activation.sh` :

```bash
#!/bin/bash
# Exemple : silences Alertmanager pilotés par les événements de mise à jour de la sonde.
# Appel par votre chaîne de traitement des logs, une fois par événement reçu :
#   silence-activation.sh annonce    <job> <instance> <activation.id> <sequence> <version> <début>
    [<fin>]
#   silence-activation.sh annulation <job> <instance> <activation.id> <sequence>
#   silence-activation.sh fin        <job> <instance> <activation.id> <résultat>
# annonce : update.scheduled ; annulation : update.cancelled ; fin : maintenance.ended.
# <activation.id>, <sequence>, <début>, <fin>, <résultat> : attributs sva.update.activation.id,
```

```
# sva.update.announcement.sequence, sva.maintenance.start, sva.maintenance.end (absent hors
# fenêtre : laisser vide) et sva.update.result (documentation SYNOVEX, « Logs » des
# « Conventions de télémétrie »).
# <job> et <instance> : labels Prometheus de la sonde émettrice (mode envoi : job « sva-agent »,
# instance = service.instance.id ; mode collecte : votre table de correspondance).
# Adresses fictives ; durées indicatives, à fixer d'après la durée observée d'une activation.
# Codes de sortie : 0 traité ou ignoré ; 1 arguments invalides ; 2 erreur Prometheus ;
# 3 sonde inconnue ou ambiguë ; 4 bornes incohérentes ; 5 erreur Alertmanager ;
# 6 reprise non vérifiée (réessayer plus tard) ; 7 verrou indisponible.
set -u -o pipefail

PROM='https://prometheus.example.internal'
AM='https://alertmanager.example.internal'
ETAT='/var/lib/silences-sva'   # dernier numéro d'annonce traité, par sonde
MARGE=2700                     # marge après une fin de fenêtre annoncée, en secondes
DUREE_SANS_FIN=5400            # durée maximale d'un silence sans fin annoncée, en secondes
DUREE_MAX=43200                # durée maximale plausible d'une fenêtre annoncée, en secondes
ALERTES='SondeInjoignable|SondeSilencieuse|ControleNonEvaluable|ParcoursEnEchec'

erreur() { echo "$2" >&2; exit "$1"; }
iso() { date -u -d "@$1" +%Y-%m-%dT%H:%M:%SZ; }

[ $# -ge 5 ] || erreur 1 "Usage : voir l'en-tête du script"
ACTION=$1; JOB=$2; INSTANCE=$3; ID=$4
for v in "$JOB" "$INSTANCE"; do
  [[ $v =~ ^[A-Za-z0-9._:/-]+$ ]] || erreur 1 "Valeur refusée : $v"
done
[[ $ID =~ ^[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}$ ]] \
  || erreur 1 "Identifiant d'activation refusé : $ID"
SONDE="$JOB/$INSTANCE"
ETIQUETTE="sva-maintenance $SONDE"

# Sérialisation par sonde : deux événements de la même sonde ne sont jamais traités en même temps.
mkdir -p "$ETAT" || erreur 7 "Dossier d'état inaccessible : $ETAT"
FICHIER="$ETAT/$(printf '%s' "$SONDE" | tr -c 'A-Za-z0-9._-' '_')"
exec 9>"$FICHIER.verrou" || erreur 7 "Verrou inaccessible"
flock -w 60 9 || erreur 7 "Verrou de $SONDE indisponible"
DERNIER=$(cat "$FICHIER.sequence" 2>/dev/null || echo 0)

# Silences actifs ou à venir créés par ce script pour cette sonde : « id sequence activation ».
silences() {
  curl -sS --fail "$AM/api/v2/silences" | jq -r --arg e "$ETIQUETTE " '.[]
    | select(.status.state != "expired" and (.comment | startswith($e)))
    | select(.comment | test("activation=[0-9a-f-]+ sequence=[0-9]+"))
    | (.comment | capture("activation=(?<a>[0-9a-f-]+) sequence=(?<s>[0-9]+)")) as $c
    | "\(.id) \($c.s) \($c.a)"'
}
expirer() {  # expirer <id>… ; un échec est signalé, jamais ignoré
  [ $# -eq 0 ] || amtool silence expire --alertmanager.url="$AM" "$@" \
    || erreur 5 "Expiration en échec : $*"
}
# Une annonce ou une annulation n'est traitée que si son numéro dépasse le dernier traité :
# les doublons et les événements reçus en retard sont ignorés.
plus_recent() {
  [[ $1 =~ ^[0-9]+$ ]] || erreur 1 "Numéro d'annonce refusé : $1"
  if [ "$1" -le "$DERNIER" ]; then
    echo "Ignoré : numéro $1 déjà dépassé pour $SONDE ($DERNIER)"; exit 0
  fi
}
# Les listes sont lues dans une variable avant usage : une erreur de lecture arrête le script.
lire() { LISTE=$(silences) || erreur 5 "Lecture des silences en échec"; }

case $ACTION in
annonce)
  [ $# -ge 7 ] || erreur 1 "annonce : <sequence> <version> <début> [<fin>] attendus"
  SEQ=$5; VERSION=$6; DEBUT_ISO=$7; FIN_ISO=${8:-}
  [[ $VERSION =~ ^[A-Za-z0-9._+-]+$ ]] || erreur 1 "Version refusée : $VERSION"
```

```
  plus_recent "$SEQ"
  MAINTENANT=$(date -u +%s)
  DEBUT=$(date -u -d "$DEBUT_ISO" +%s 2>/dev/null) || erreur 4 "Début illisible : $DEBUT_ISO"
  [ "$DEBUT" -ge "$MAINTENANT" ] || DEBUT=$MAINTENANT   # activation déjà commencée
  if [ -n "$FIN_ISO" ]; then                             # borne de fenêtre annoncée
    FIN=$(date -u -d "$FIN_ISO" +%s 2>/dev/null) || erreur 4 "Fin illisible : $FIN_ISO"
    [ "$FIN" -gt "$DEBUT" ] || erreur 4 "Fin annoncée antérieure au début"
    [ $((FIN - DEBUT)) -le "$DUREE_MAX" ] || erreur 4 "Durée annoncée invraisemblable"
    FIN=$((FIN + MARGE))
  else                                                   # aucune fin annoncée
    FIN=$((DEBUT + DUREE_SANS_FIN))
  fi
  # La sonde doit correspondre à exactement une série sva_agent_up dans Prometheus.
  R=$(curl -sS --fail --get \
    --data-urlencode "query=sva_agent_up{job=\"$JOB\",instance=\"$INSTANCE\"}" \
    "$PROM/api/v1/query") || erreur 2 "Erreur d'interrogation de Prometheus"
  [ "$(jq -r '.status' <<<"$R")" = success ] || erreur 2 "Réponse de Prometheus en erreur"
  [ "$(jq '.data.result | length' <<<"$R")" -eq 1 ] || erreur 3 "Sonde $SONDE inconnue ou ambiguë"
  lire
  # Reprise d'un traitement interrompu : un silence de ce numéro existe déjà, il n'est pas recréé.
  if ! awk -v n="$SEQ" '$2 == n { t = 1 } END { exit !t }' <<<"$LISTE"; then
    amtool silence add --alertmanager.url="$AM" \
      --start="$(iso "$DEBUT")" --end="$(iso "$FIN")" \
      --comment="$ETIQUETTE activation=$ID sequence=$SEQ version=$VERSION" \
      job="$JOB" instance="$INSTANCE" alertname=~"$ALERTES" >/dev/null \
      || erreur 5 "Création en échec"
  fi
  # N'expirer que les silences d'annonces antérieures de la même sonde, puis retenir le numéro.
  expirer $(awk -v n="$SEQ" '$2 < n { print $1 }' <<<"$LISTE")
  echo "$SEQ" >"$FICHIER.sequence"
  echo "Silence de $SONDE : activation $ID, annonce $SEQ, jusqu'à $(iso "$FIN")" ;;
annulation)
  SEQ=$5
  plus_recent "$SEQ"
  lire
  expirer $(awk -v n="$SEQ" '$2 < n { print $1 }' <<<"$LISTE")
  echo "$SEQ" >"$FICHIER.sequence"
  echo "Activation $ID annulée : silences antérieurs de $SONDE expirés" ;;
fin)
  RESULTAT=$5
  [[ $RESULTAT =~ ^(succeeded|rolled_back|failed)$ ]] || erreur 1 "Résultat refusé : $RESULTAT"
  # Reprise vérifiée : sonde joignable et sortie de l'état de maintenance.
  R=$(curl -sS --fail --get --data-urlencode \
    "query=sva_agent_up{job=\"$JOB\",instance=\"$INSTANCE\"} == 1 and on(job, instance)
   sva_agent_maintenance{job=\"$JOB\",instance=\"$INSTANCE\"} == 0" \
    "$PROM/api/v1/query") || erreur 2 "Erreur d'interrogation de Prometheus"
  [ "$(jq '.data.result | length' <<<"$R")" -eq 1 ] \
    || erreur 6 "Reprise de $SONDE non vérifiée : silence conservé jusqu'à sa fin"
  lire
  expirer $(awk -v a="$ID" '$3 == a { print $1 }' <<<"$LISTE")
  echo "Maintenance $ID terminée ($RESULTAT) : silences de cette activation expirés" ;;
*) erreur 1 "Action inconnue : $ACTION" ;;
esac
```

`amtool silence add` accepte des dates de début et de fin au format RFC 3339 (`--start`, `--end`) et des filtres sur les labels. Le script lit les silences existants par l'API d'Alertmanager (`GET /api/v2/silences`) et ne considère que ceux qu'il a créés, reconnus à leur commentaire. Le filtre sur `alertname` limite le silence aux alertes de disponibilité et d'état : les alertes de mises à jour et de sécurité de la sonde restent notifiées. Si votre Alertmanager exige une authentification, utilisez l'option `--http.config.file` d'`amtool` et un fichier d'en-tête protégé pour `curl` (`-H @fichier`), jamais un jeton en argument ; procédez de même pour Prometheus.

Limites de l'automatisation. Elle réduit le travail manuel sans le supprimer :

- **activation manuelle immédiate** : l'annonce est émise au moment où l'activation commence et peut arriver après le début du redémarrage. Appliquez la procédure manuelle de référence **avant** la commande ; l'annonce reçue ensuite est traitée normalement par l'automatisme, dont le silence s'ajoute au silence manuel. Expirez ce dernier après la reprise vérifiée, comme à l'étape 4 ;
- **événement perdu** : si une annonce n'est jamais reçue, aucun silence n'est créé et les alertes se déclenchent ; si une annulation ou une fin de maintenance est perdue, le silence s'achève à sa borne. Dans les deux cas, l'erreur va dans le sens de la notification ;
- **ordre des annonces** : il repose sur le numéro d'annonce de la sonde. Après le remplacement d'une sonde qui reprend l'identité de la précédente, ce numéro n'est croissant que si l'heure de la nouvelle sonde est fiable (§ 4.1) ; videz alors l'état de l'automatisme pour cette sonde ;
- **reprise** : le script vérifie que la sonde répond et a quitté l'état de maintenance ; il ne vérifie pas l'arrivée de nouveaux résultats de contrôle. Les alertes de contrôle, rétablies après la levée du silence, prennent le relais ;
- **validation** : comme tout exemple de cette documentation, adaptez ce script à votre chaîne de traitement des logs et validez-le dans un environnement de recette avant de l'utiliser en production.

2.15.5 Mise en sourdine dans Datadog, Dynatrace, Centreon et Nagios

Les règles communes du § 2.15.1 s'appliquent à chaque plateforme : une période récurrente par fenêtre de la sonde, dans son fuseau horaire, prolongée de la marge et limitée à la sonde concernée. Les sections suivantes indiquent le mécanisme propre à chaque plateforme.

Datadog

Datadog suspend les notifications de moniteurs au moyen de **temps d'arrêt** (*downtimes*), qui peuvent être récurrents.

1. Ouvrez la page **Manage Downtime**, puis **Schedule Downtime**.
2. Sélectionnez les moniteurs de la sonde **par tags de moniteur** (*monitor tags*), par exemple un tag que vous avez posé sur tous les moniteurs `sva.*`. Un temps d'arrêt par tags ne s'applique qu'aux moniteurs qui portent **tous** les tags sélectionnés.
3. Restreignez la **portée de groupe** (*group scope*) à la sonde concernée, par exemple `host:sonde-paris-01` pour des moniteurs groupés par `host` (§ 2.7.3). Sans portée de groupe, toutes les sondes sont suspendues.
4. Choisissez une planification **récurrente** : date et heure de début, **fuseau horaire** de la sonde, répétition hebdomadaire sur les jours de la fenêtre, et durée égale à celle de la fenêtre augmentée de la marge. Une règle de récurrence au format RRULE est également acceptée.
5. Enregistrez, puis vérifiez dans l'aperçu la liste des moniteurs concernés.

Par défaut, Datadog envoie une nouvelle notification si un moniteur est toujours en `ALERT`, `WARNING` ou `NO DATA` à l'expiration du temps d'arrêt : une sonde qui ne serait pas revenue après la fenêtre est donc signalée. Tout moniteur créé ou modifié ultérieurement et correspondant à la portée est automatiquement inclus. Les temps d'arrêt peuvent aussi être créés par l'API de Datadog, ce qui permet de les dériver de la configuration des sondes.

Dynatrace

Dynatrace suspend l'alerte au moyen de **fenêtres de maintenance** (*maintenance windows*).

1. Ouvrez **Settings** > **Maintenance windows** > **Monitoring, alerting, and availability**, puis **Create maintenance window**.
2. Nommez la fenêtre (par exemple « Activation sonde-paris-01 ») et choisissez le type **Planned**.
3. Choisissez une récurrence **hebdomadaire**, les jours et l'horaire de la fenêtre augmentés de la marge, et le **fuseau horaire** de la sonde.

4. Choisissez le comportement de détection : **Detect problems but don't alert** (les problèmes restent enregistrés, sans notification ; recommandé pour conserver une trace) ou **Disable problem detection** (aucun problème n'est détecté). L'option **Detect problems and alert** ne supprime rien et ne convient pas.
5. Limitez la **portée** à la sonde par un filtre (entité, tags d'entité ou zone de gestion). Sans filtre, la fenêtre s'applique à **tout l'environnement**.

> **Attention** — Les données de la sonde arrivent dans Dynatrace par OTLP (§ 2.8). Vérifiez dans votre environnement que les alertes définies sur les métriques `sva.*` (événements métriques, détecteurs d'anomalies) sont bien rattachées aux entités ou aux tags retenus dans la portée de la fenêtre. À défaut, la fenêtre n'aurait aucun effet sur ces alertes. Les libellés ci-dessus sont ceux de la documentation de Dynatrace en vigueur à la date de rédaction ; ils peuvent varier selon votre version.

Centreon

Centreon suspend les notifications au moyen de **temps d'arrêt récurrents** (*recurrent downtimes*).

1. Ouvrez **Monitoring** > **Downtimes** > **Recurrent Downtimes**, puis **Add**.
2. Renseignez le nom et la description, et cochez l'activation.
3. Ajoutez une **période** (*Add new period*) de type hebdomadaire (**Weekly**), avec les jours de la fenêtre, la plage horaire `HH:MM - HH:MM` (fenêtre augmentée de la marge) et le type **fixe** (*fixed*). Le type flexible ne démarre qu'à l'apparition d'un état non `OK` et ne couvre pas toute la fenêtre. Pour une fenêtre qui franchit minuit, déclarez deux périodes.
4. Dans les relations, liez le temps d'arrêt à l'**hôte** de la sonde (**Linked with Hosts**) **et** à ses **services** (**Linked with Services** ou **Linked with Service Groups**). Liez explicitement les services : la documentation de Centreon ne précise pas qu'un temps d'arrêt récurrent posé sur un hôte s'étend à ses services.
5. Enregistrez, puis vérifiez dans **Monitoring** > **Downtimes** que les temps d'arrêt apparaissent aux dates attendues.

Vérifiez que le fuseau horaire pris en compte par Centreon pour ces périodes correspond à celui de la sonde. Pendant le drainage, les services `check_sva` passent en `UNKNOWN` avec le texte `MAINTENANCE`, et la vérification de l'hôte fondée sur `/health` (§ 2.10.4) échoue, car `/health` répond alors avec le code HTTP 503 : ces deux effets sont couverts par le temps d'arrêt.

Nagios

Nagios Core suspend les notifications au moyen de **temps d'arrêt programmés** (*scheduled downtime*), créés depuis l'interface web (lien « Schedule downtime » de la fiche de l'hôte ou du service) ou par **commande externe**. Sa documentation ne décrit pas de temps d'arrêt récurrent natif : pour reproduire les fenêtres de la sonde, programmez chaque occurrence par une tâche planifiée (*cron*) qui écrit dans le fichier de commandes de Nagios.

Les deux commandes externes utiles sont :

Commande	Effet
`SCHEDULE_HOST_DOWNTIME;<hôte>;<début>;<fin>;<fixe>;<id déclencheur>;<durée>;<auteur>;<commentaire>`	Temps d'arrêt de l'hôte
`SCHEDULE_HOST_SVC_DOWNTIME;<hôte>;<début>;<fin>;<fixe>;<id déclencheur>;<durée>;<auteur>;<commentaire>`	Temps d'arrêt de tous les services de l'hôte

Les dates sont des horodatages Unix ; `<fixe>` vaut 1 pour un temps d'arrêt qui commence et finit exactement aux dates indiquées.

Exemple de script, exécuté par *cron* chaque mardi avant 04:00 pour la fenêtre fictive de `sonde-paris-01` (mardi à 04:00 pendant 1 heure, marge de 45 minutes). Le chemin du fichier de commandes est celui d'une installation de Nagios Core depuis les sources : adaptez-le à votre installation.

```shell
#!/bin/sh
# Temps d'arrêt de la sonde sonde-paris-01 pour sa fenêtre du jour (exemple fictif).
CMD=/usr/local/nagios/var/rw/nagios.cmd
HOTE=sonde-paris-01
MAINTENANT=$(date +%s)
# Début de la fenêtre dans le fuseau horaire de la sonde (GNU date)
DEBUT=$(TZ=Europe/Paris date -d 'today 04:00' +%s)
FIN=$((DEBUT + 6300))   # 1 h de fenêtre + 45 min de marge

printf '[%s] SCHEDULE_HOST_DOWNTIME;%s;%s;%s;1;0;0;supervision;Fenêtre de maintenance de la sonde\n'
    \
  "$MAINTENANT" "$HOTE" "$DEBUT" "$FIN" > "$CMD"
printf '[%s] SCHEDULE_HOST_SVC_DOWNTIME;%s;%s;%s;1;0;0;supervision;Fenêtre de maintenance de la
    sonde\n' \
  "$MAINTENANT" "$HOTE" "$DEBUT" "$FIN" > "$CMD"
```

Nagios envoie une notification de début (`DOWNTIMESTART`) et de fin (`DOWNTIMEEND`) de temps d'arrêt aux contacts qui les ont activées. Vérifiez le résultat dans la page **Downtime** de l'interface. Pour une activation hors fenêtre, programmez un temps d'arrêt ponctuel de la même manière, à partir des dates annoncées par la sonde.

2.15.6 Redondance : alertes de site

Une sonde unique ne supervise pas pendant son propre redémarrage. Pour les environnements critiques, le § 4.3, qui en décrit la mise en place, recommande au moins **deux sondes par site** exécutant les mêmes contrôles, avec des fenêtres de maintenance **qui ne se chevauchent pas**. Les alertes applicatives doivent alors porter sur **l'ensemble des sondes du site**, et non sur une sonde isolée. Deux alertes distinctes sont nécessaires, car un échec applicatif et une perte de couverture appellent des réactions différentes :

Alerte	Se déclenche lorsque	Signification
`ParcoursEnEchecSurLeSite`	Toutes les sondes **évaluables** du site voient le contrôle en `CRITICAL`, au moins une sonde étant évaluable	Incident probable de l'application ou du parcours supervisé
`CouvertureInsuffisanteSurLeSite`	Le nombre de sondes à résultat évaluable est inférieur à l'**effectif attendu** du site, y compris en l'absence totale de séries	La supervision du site repose sur moins de sondes que prévu, ou n'est plus assurée

Une sonde est **évaluable** pour un contrôle lorsqu'elle en fournit un résultat `OK`, `WARNING` ou `CRITICAL` et qu'elle n'est pas en maintenance. Les règles s'appuient sur la série `sva_check_status` (§ 2.3.3), qui vaut `0` pour `OK`, `1` pour `WARNING`, `2` pour `CRITICAL` et `3` pour `UNKNOWN` ; l'état `STALE` n'y est pas reporté : la série conserve le dernier état produit par une exécution. L'effectif attendu est un paramètre propre à chaque site, déclaré par une règle d'enregistrement `sva:site_effectif_attendu`, une par site : c'est le seul paramètre à adapter. Le groupe suivant, avec le label `site` ajouté par la configuration de collecte (§ 2.5.2), est identique au groupe `sondes-site` du fichier `exemples/prometheus/sva-regles-alerte.yml` :

```yaml
groups:
  - name: sondes-site
    rules:
      # PARAMÈTRE À ADAPTER : effectif attendu de chaque site, c'est-à-dire le nombre de
      # sondes du site qui exécutent les mêmes contrôles. Une règle par site (valeurs fictives).
      - record: sva:site_effectif_attendu
        expr: vector(2)
```

```
      labels:
        site: paris-dc1

    - record: sva:site_effectif_attendu
      expr: vector(2)
      labels:
        site: lyon-dc1

    # Toutes les sondes évaluables du site voient le contrôle en CRITICAL, au moins une
    # sonde étant évaluable. Le filtre <= 2 écarte UNKNOWN ; la clause unless écarte les
    # sondes en maintenance. Si aucune sonde n'est évaluable, l'expression ne renvoie rien :
    # cette situation relève de CouvertureInsuffisanteSurLeSite.
    - alert: ParcoursEnEchecSurLeSite
      expr: |
        min by (site, sva_check_id) (
          sva_check_status <= 2
            unless on (job, instance) sva_agent_maintenance == 1
        ) == 2
      for: 5m
      labels:
        severity: critical
      annotations:
        summary: "Contrôle {{ $labels.sva_check_id }} en CRITICAL sur toutes les sondes évaluables
du site {{ $labels.site }}"
        description: "Toutes les sondes du site {{ $labels.site }} qui fournissent un résultat
évaluable pour ce contrôle le voient en échec. Les sondes non évaluables (UNKNOWN, maintenance,
injoignables) sont signalées par CouvertureInsuffisanteSurLeSite."

    # Nombre de sondes à résultat évaluable inférieur à l'effectif attendu du site.
    # Première partie : par contrôle ; une sonde est comptée une seule fois par son label
    # instance, même si elle est collectée par plusieurs travaux (count by … instance) ; un
    # contrôle dont toutes les séries sont UNKNOWN ou en maintenance compte 0 sonde évaluable
    # (terme « or … * 0 »).
    # Seconde partie : absence totale de séries de contrôle pour un site attendu (toutes
    # les sondes du site injoignables) ; renvoie la valeur 0, sans limite de durée.
    # Une activation planifiée réduit la couverture pendant sa durée : le délai « for »
    # doit dépasser la durée totale mesurée d'une activation (documentation SYNOVEX, « Mises à
jour »),
    # 45 minutes ici à titre indicatif.
    # LIMITES : un contrôle qui disparaît de toutes les sondes du site, alors que d'autres
    # contrôles y restent présents, ne produit plus aucune série et n'est pas signalé ; une
    # série présente ne prouve pas une exécution récente (sva_check_status conserve le
    # dernier état, l'état STALE n'y est pas reporté). Le groupe facultatif
    # sondes-site-controles couvre ces deux cas pour les contrôles déclarés attendus.
    - alert: CouvertureInsuffisanteSurLeSite
      expr: |
        (
          (
              count by (site, sva_check_id) (
                count by (site, sva_check_id, instance) (
                  sva_check_status <= 2
                    unless on (job, instance) sva_agent_maintenance == 1
                )
              )
            or
              count by (site, sva_check_id) (sva_check_status) * 0
          )
            < on (site) group_left ()
          sva:site_effectif_attendu
        )
        or on (site)
        (
          sva:site_effectif_attendu
            unless on (site)
          count by (site) (sva_check_status)
        ) * 0
      for: 45m
```

```
        labels:
          severity: warning
        annotations:
          summary: "Couverture insuffisante sur le site {{ $labels.site }}"
          description: "{{ $value }} sonde(s) du site {{ $labels.site }} fournissent un résultat
    évaluable (contrôle : {{ $labels.sva_check_id }} ; vide si aucune série du site n'est reçue).
    Vérifiez l'état des sondes du site et la valeur de sva:site_effectif_attendu."
```

Lecture des expressions :

- **`ParcoursEnEchecSurLeSite`** : le filtre `sva_check_status <= 2` écarte les séries `UNKNOWN`, qui ne disent rien de l'état de l'application. La clause `unless` écarte les sondes en maintenance : leur dernière valeur de `sva_agent_maintenance` est `1` ; pendant leur redémarrage, leurs séries cessent d'être actualisées, puis ne participent plus au calcul. Le minimum vaut `2` lorsque toutes les sondes restantes voient le contrôle en `CRITICAL`. Si aucune sonde n'est évaluable, l'expression ne renvoie rien : cette situation relève de l'alerte de couverture ;
- **`CouvertureInsuffisanteSurLeSite`** : le regroupement intermédiaire par `instance` compte chaque sonde une seule fois, même si elle est collectée par plusieurs travaux de collecte sous la même adresse. Le terme `or … * 0` compte zéro sonde évaluable lorsque toutes les séries d'un contrôle sont `UNKNOWN` ou en maintenance. Le second terme, après `or on (site)`, signale avec la valeur `0` l'absence totale de séries pour un site attendu (toutes ses sondes injoignables), sans limite de durée.

Avec cette forme, l'activation d'une sonde n'interrompt pas la détection des pannes applicatives : l'autre sonde du site continue de les signaler. Les alertes propres à chaque sonde (`SondeInjoignable`, `SondeSilencieuse`) restent mises en sourdine pendant ses seules fenêtres.

Avec ces règles :

- un `CRITICAL` vu par une sonde alors que l'autre voit le contrôle en `OK` ne déclenche pas `ParcoursEnEchecSurLeSite` : il signale plutôt un problème local, suivi par les alertes de la sonde ;
- un `CRITICAL` sur une sonde et un `UNKNOWN` sur l'autre déclenchent les deux alertes : l'échec est observé par la seule sonde évaluable, et la couverture est réduite ;
- l'absence totale de séries pour un site attendu déclenche `CouvertureInsuffisanteSurLeSite` avec la valeur `0`, sans limite de durée ;
- une sonde collectée par deux travaux de collecte n'est comptée qu'une fois, à condition que son label `instance` soit le même dans les deux travaux. Collectée sous deux adresses différentes (nom et adresse IP, par exemple), elle serait comptée deux fois : collectez chaque sonde sous une seule adresse ;
- une activation planifiée réduit la couverture pendant sa durée : le délai `for` de l'alerte de couverture (45 minutes, indicatif) doit dépasser la durée totale mesurée d'une activation (§ 8.4.5 du guide d'installation et d'administration) ; à défaut, neutralisez cette seule alerte pendant les fenêtres des sondes du site (§ 2.15.1). Ne mettez pas `ParcoursEnEchecSurLeSite` en sourdine pendant les fenêtres : c'est elle qui détecte un incident pendant la maintenance d'une sonde.

> ! **Attention** — Ces deux règles raisonnent sur les séries présentes. Elles ne signalent pas un contrôle qui disparaît de **toutes** les sondes du site (contrôle supprimé ou désactivé, configuration divergente) tant que d'autres contrôles du site restent présents : aucune série ne le représente plus. Elles ne vérifient pas non plus l'âge du résultat : `sva_check_status` conserve le dernier état d'un contrôle, et l'état `STALE` n'y est pas reporté. Une sonde dont un contrôle ne s'exécute plus reste donc comptée comme évaluable avec son dernier résultat. Le catalogue des métriques ne comporte pas d'horodatage de dernière exécution. L'extension facultative ci-dessous couvre ces deux cas pour les contrôles que vous déclarez attendus.

Extension facultative : couverture de chaque contrôle attendu. Le groupe `sondes-site-controles` du même fichier déclare l'inventaire des contrôles attendus sur chaque site, par une règle d'enregistrement `sva:site_controle_attendu` par couple site/contrôle, sur le modèle de `sva:site_effectif_attendu`. Son alerte `ControleAttenduNonCouvertSurLeSite` compte, pour chaque contrôle de l'inventaire, les sondes **fraîchement évaluables** : évaluables au sens précédent, et ayant terminé au moins une exécution de ce contrôle pendant une fenêtre de fraîcheur, mesurée par le compteur `sva_check_runs_total`. Elle se déclenche lorsque ce nombre est inférieur à l'effectif attendu du site, y compris lorsque le contrôle a disparu de toutes les sondes (valeur `0`). Le groupe suivant est identique à celui du fichier :

YAML

```yaml
groups:
  - name: sondes-site-controles
    rules:
      # PARAMÈTRE À ADAPTER : inventaire des contrôles attendus sur chaque site. Une règle
      # par couple site/contrôle (valeurs fictives). Un contrôle absent de cet inventaire
      # n'est pas surveillé par ce groupe.
      - record: sva:site_controle_attendu
        expr: vector(1)
        labels:
          site: paris-dc1
          sva_check_id: boutique-connexion

      - record: sva:site_controle_attendu
        expr: vector(1)
        labels:
          site: lyon-dc1
          sva_check_id: boutique-connexion

      # Nombre de sondes fraîchement évaluables inférieur à l'effectif attendu, pour un
      # contrôle de l'inventaire, y compris lorsque ce contrôle a disparu de toutes les
      # sondes du site (valeur 0). Chaque sonde est comptée une seule fois (label instance).
      # PARAMÈTRE À ADAPTER : fenêtre de fraîcheur [15m], au moins deux intervalles de
      # planification du contrôle le plus espacé, comme l'état STALE (documentation SYNOVEX,
      # « Les états »), et au
      # moins deux intervalles de collecte. Pour des contrôles d'intervalles très différents,
      # dupliquez la règle avec une fenêtre propre et un filtre sur sva_check_id.
      # Le délai « for » doit dépasser la durée totale mesurée d'une activation augmentée de
      # la fenêtre de fraîcheur (45 minutes ici, à titre indicatif).
      - alert: ControleAttenduNonCouvertSurLeSite
        expr: |
          (
              count by (site, sva_check_id) (
                count by (site, sva_check_id, instance) (
                  (
                    sva_check_status <= 2
                      unless on (job, instance) sva_agent_maintenance == 1
                  )
                    and on (job, instance, sva_check_id)
                  (
                    sum by (job, instance, sva_check_id) (increase(sva_check_runs_total[15m])) > 0
                  )
                )
              )
            or
              sva:site_controle_attendu * 0
          )
            < on (site) group_left ()
          sva:site_effectif_attendu
            and on (site, sva_check_id)
          sva:site_controle_attendu
        for: 45m
        labels:
          severity: warning
        annotations:
```

```
        summary: "Contrôle attendu {{ $labels.sva_check_id }} insuffisamment couvert sur le site
    {{ $labels.site }}"
        description: "{{ $value }} sonde(s) du site {{ $labels.site }} ont fourni récemment un
    résultat évaluable pour le contrôle attendu {{ $labels.sva_check_id }}. Vérifiez que le contrôle
    existe et s'exécute sur chaque sonde du site, et la valeur de sva:site_effectif_attendu."
```

Limites de cette extension : l'inventaire est tenu à la main et doit suivre chaque ajout ou suppression de contrôle ; une seule fenêtre de fraîcheur s'applique à tous les contrôles de la règle, ce qui impose de dupliquer la règle pour des contrôles d'intervalles très différents ; l'identité d'une sonde repose sur son label `instance`, avec la réserve indiquée plus haut pour une sonde collectée sous deux adresses ; enfin, `ParcoursEnEchecSurLeSite` n'applique pas cette condition de fraîcheur : un dernier résultat `CRITICAL` ancien peut la maintenir active ; pour un contrôle de l'inventaire, l'alerte de l'extension signale alors que ce résultat n'est plus récent. Si vous n'utilisez pas l'extension, supprimez le groupe `sondes-site-controles` du fichier : ses valeurs fictives déclencheraient sinon des alertes.

En mode envoi, remplacez le label `site` par le label issu de l'attribut de ressource `sva.site` (`sva_site`), après sa promotion en label (§ 2.5.3). Ces règles ont été vérifiées avec `promtool test rules` sur des séries simulées : sondes `OK`, `CRITICAL` sur toutes les sondes, `CRITICAL` et `UNKNOWN`, `UNKNOWN` seul, sonde en maintenance et absence totale de séries ; pour l'extension et le comptage par sonde, contrôle attendu disparu seul, sonde collectée par deux travaux, jauge périmée (exécutions arrêtées) et maintenance suivie d'une reprise.

2.16 Dépannage des exports

Le tableau suivant regroupe les incidents d'export les plus fréquents. Commencez toujours par le test intégré de la destination (§ 2.4.4, étape 3) : il renvoie le code HTTP obtenu, le message d'erreur de la plateforme et la latence. Le chapitre 9 du guide d'installation et d'administration (§ 5.1) complète ce tableau dans la démarche générale de diagnostic de la sonde.

Symptôme	Cause probable	Vérification	Action
HTTP 401	Clé ou jeton d'ingestion absent, invalide ou révoqué ; format d'en-tête erroné	Test de la destination ; secret référencé dans `headers`	Remplacer le secret ; vérifier le format de l'en-tête : `Api-Token` (Dynatrace, jeton classique), `dd-api-key` (Datadog), `Basic` (Grafana Cloud, § 2.6.2)
HTTP 403	Jeton sans les étendues d'ingestion requises (§ 2.8.1) ; pour Datadog, adresse qui ne correspond pas au site du compte (§ 2.7.2)	Étendues du jeton ; site Datadog	Ajouter les étendues nécessaires ; utiliser l'adresse du bon site
HTTP 400 sur les métriques vers Dynatrace ou l'intake direct de Datadog	Temporalité cumulative envoyée à une plateforme qui n'accepte que le delta (§ 2.1.5)	Paramètre `metrics.temporality`	Utiliser le préréglage `dynatrace` ou `datadog-agentless` (temporalité `delta`)
Succès partiel, certaines métriques absentes dans Dynatrace	Requête mêlant données acceptées et refusées (type non pris en charge, temporalité)	Journaux d'export de la sonde ; `sva.export.failures`	Utiliser le préréglage `dynatrace` (delta, histogrammes explicites)
HTTP 404	Chemin d'adresse incorrect, notamment pour le récepteur OTLP de Prometheus	Adresse complète testée	Pour Prometheus, déclarer l'adresse complète `/api/v1/otlp/v1/metrics` dans `signalEndpoints.metrics` (§ 2.5.3)

Symptôme	Cause probable	Vérification	Action
HTTP 413	Lot trop volumineux	Volume des envois	Désactiver `metrics.stepHistograms`, allonger `metrics.exportInterval` ; si l'erreur persiste, contacter le support
HTTP 429	Limitation de débit ou quota d'ingestion atteint	En-tête `Retry-After` ; consommation dans la plateforme	La sonde réessaie d'elle-même ; réduire le volume ou ajuster le quota
HTTP 5xx, erreurs réseau	Plateforme indisponible ; proxy ou pare-feu	Test de la destination ; flux sortant vers l'adresse	Ouvrir le flux sortant (§ 6.1) ; la sonde réessaie pendant `retry.maxElapsed`, puis abandonne le lot (`sva.export.dropped`, § 2.12)
Cible Prometheus à l'état `DOWN` (mode collecte)	Pare-feu ; jeton absent, expiré ou de rôle insuffisant (`metrics` au minimum) ; certificat de la sonde non reconnu	Page des cibles de Prometheus ; `curl` vers `/metrics` (§ 2.5.2, étape 3)	Ouvrir le port 443 depuis Prometheus ; utiliser un jeton `metrics` ; renseigner `tls_config.ca_file`
Données envoyées au récepteur OTLP de Prometheus, mais invisibles	Récepteur non activé ; noms traduits avec suffixes ; labels de site absents faute de promotion	Option `--web.enable-otlp-receiver` ; noms des séries (§ 2.3.3)	Activer le récepteur ; rechercher les noms traduits ; promouvoir `sva.site` (§ 2.5.3)
Métriques présentes, mais aucun tableau de bord ni aucune alerte	Comportement normal : l'ingestion ne crée ni tableau de bord ni alerte (§ 2.2)	—	Importer les modèles fournis (`exemples/grafana/`, `exemples/prometheus/`) ou créer vos alertes
`check_sva` renvoie `UNKNOWN`	Sonde injoignable ; jeton invalide ; identifiant de contrôle erroné ; texte `MAINTENANCE` : activation d'une mise à jour en cours (§ 2.15.1)	Exécuter le plugin en ligne de commande sur le serveur Nagios	Corriger le jeton (rôle `metrics`), l'identifiant (`--check`) ou le flux HTTPS
Trous dans les courbes après une coupure	Comportement documenté : les données ne sont pas rejouées après une indisponibilité de la destination (§ 2.12)	`sva.export.dropped`	Consulter l'historique local de la sonde ; pour réduire les pertes, utiliser un Collector à file persistante (§ 2.9)
Absence de données de la sonde dans la plateforme alors qu'elle fonctionne	Export interrompu (lignes précédentes) ; heure de la sonde décalée ; activation d'une mise à jour en cours	Test de la destination ; synchronisation horaire de la sonde ; événements `maintenance.started` et `maintenance.ended`	Corriger l'export ou la synchronisation NTP ; pendant une activation, l'absence de données est normale (§ 2.15.1)
Alertes reçues pendant la maintenance de la sonde	Périodes de maintenance absentes de la plateforme, ou non alignées sur les fenêtres de la sonde	Heure des alertes comparée aux fenêtres de la sonde	Aligner les périodes de maintenance de la plateforme (§ 2.15)

Chapitre 3 — Santé de la sonde

3.1 État de santé de la sonde

Avant de conclure qu'une application est défaillante, il faut s'assurer que la sonde elle-même fonctionne correctement. La sonde publie son propre état de santé, distinct de celui des contrôles.

3.1.1 Consultation

Dans l'interface d'administration, l'état de santé de la sonde est affiché sur le tableau de bord et détaillé dans une page dédiée. Il comprend les informations suivantes : version, édition, durée de fonctionnement, ressources, état des composants et des destinations d'export.

Par l'API, deux ressources sont disponibles :

- `GET /health` : test de vivacité, **sans authentification**. Il répond `200` si le moteur principal répond, sans aucune information sensible. Pendant l'**état de maintenance** qui précède l'activation d'une mise à jour, il répond `{"status": "MAINTENANCE"}` avec le code HTTP `503` ; la sonde devient ensuite injoignable le temps de son redémarrage (§ 3.14.2 du guide d'utilisation et de supervision). Il convient à un simple test de disponibilité de la sonde par un outil tiers.
- `GET /api/v1/system` (rôle `viewer`) : état détaillé.

```bash
curl --cacert "$CA" \
     -H @"$HOME/.sva/entete-auth" \
     "$SONDE/api/v1/system"
```

Exemple de réponse (le détail des champs figure dans la description OpenAPI de la sonde) :

```json
{
  "version": "4.2.1",
  "edition": "enterprise",
  "hostname": "sonde-paris-01",
  "site": "paris-dc1",
  "uptimeSeconds": 1296000,
  "supportEndsAt": "2030-11-30T23:59:59Z",
  "components": {
    "mainEngine": "OK",
    "scenarioEngine": "OK",
    "storage": "OK",
    "network": "OK",
    "support": "OK",
    "updates": "OK"
  },
  "resources": {
    "memoryUsedPercent": 61,
    "storageUsedPercent": 34,
    "temperatureCelsius": 58,
    "throttled": false
  },
  "exporters": [
    { "id": "dynatrace", "status": "OK", "lastSuccess": "2026-11-19T09:20:00Z" }
  ]
}
```

Chaque composant prend l'un des états `OK`, `WARNING`, `CRITICAL` ou `UNKNOWN`. Quatre d'entre eux appellent une explication particulière :

Composant	Rôle	États notables
`storage`	Volume de données de la sonde : microSD et, le cas échéant, SSD NVMe facultatif	`CRITICAL` si le SSD activé est absent ou défaillant : les contrôles continuent et l'historique local est conservé sur la microSD en mode réduit (§ 7.8 du guide d'utilisation et de supervision)
`network`	Raccordement réseau de la sonde, y compris l'authentification 802.1X lorsqu'elle est activée (chapitre 5 du guide d'installation et d'administration)	Signale un échec de l'authentification 802.1X, ou un certificat client 802.1X proche de l'expiration ; en cas d'échec, la sonde renouvelle ses tentatives et affiche l'état sur la console HDMI
`support`	Période de support de sécurité de la version installée	`WARNING` après la date de fin de support : la sonde continue de fonctionner, mais ne reçoit plus de correctifs
`updates`	Mises à jour installées sur l'emplacement inactif et en attente d'activation (chapitre 4)	`OK` : aucune mise à jour en attente, ou activation prévue dans une fenêtre de maintenance. `WARNING` : mode automatique sans aucune fenêtre définie (configuration incomplète), mise à jour en attente sans fenêtre (aucune activation automatique possible ; un administrateur l'active par `POST /api/v1/updates/apply`), ou activation reportée (`POST /api/v1/updates/postpone`). `CRITICAL` : correctif de sécurité critique (vulnérabilité activement exploitée) en attente d'activation

La réponse indique également deux échéances, sous forme de dates RFC 3339 en UTC :

- `supportEndsAt` (toujours présent) : fin de la **période de support de sécurité** de la version majeure installée. Cette période dure au moins 5 ans après la mise sur le marché de la version majeure ; la date de fin (mois et année) est indiquée à l'achat, et le champ contient le dernier instant de ce mois. Après cette date, la sonde continue de fonctionner sans nouveau correctif ;
- `maintenanceEndsAt` : échéance du **justificatif de maintenance**, ou `null` si aucun justificatif n'a été importé. Le justificatif donne accès aux nouvelles versions fonctionnelles ; son échéance n'affecte aucune fonction acquise, et les correctifs de sécurité restent disponibles pendant la période de support.

Ces deux champs figurent aussi dans la réponse de `GET /api/v1/updates` (chapitre 4). Le champ facultatif `maintenanceEndsAt` est omis de l'exemple ci-dessus, qui reste volontairement minimal.

3.1.2 Indicateurs de santé à surveiller

Indicateur	Métrique exportée	Signal d'alerte
Sonde vivante	`sva.agent.up`	Absence de signal : la sonde est arrêtée ou injoignable (équivalent de l'état `STALE` pour la sonde entière)
Retard de planification	`sva.agent.scheduler.lag`	Retard croissant : la sonde est saturée, trop de contrôles pour sa capacité
Files d'attente	`sva.agent.queue.depth`	File qui ne se vide pas, en particulier pour la classe navigateur
Lancements du navigateur	`sva.browser.launches`	Échecs de lancement : mémoire, mise à jour incomplète
Stockage	`system.filesystem.usage` (taux d'occupation du système de fichiers) ; `sva.storage.usage` (volume par zone : base, artefacts)	Occupation proche de la capacité

Indicateur	Métrique exportée	Signal d'alerte
Température	`sva.hw.temperature`	Température élevée : refroidissement insuffisant
Bridage matériel	`sva.hw.throttled`	`undervoltage` : alimentation insuffisante, risque de corruption de la carte ; `thermal` : surchauffe
Exports	`sva.export.failures`, `sva.export.dropped`	Échecs répétés ou données perdues vers une destination
Certificat 802.1X	`sva.dot1x.certificate.expiry`	Moins de 30 jours avant l'expiration du certificat client 802.1X, si l'authentification 802.1X est activée (la sonde émet elle-même un avertissement à cette échéance)
Ressources système	`system.cpu.utilization`, `system.memory.usage`, `system.filesystem.usage`	Saturation durable
Mises à jour en attente	`sva.update.pending` (attributs `sva.update.channel`, `sva.update.critical`)	Mise à jour en attente d'activation depuis longtemps ; correctif critique en attente (`sva.update.critical="true"`)
Résultat de la dernière mise à jour	`sva.update.last_result` (attribut `sva.update.result` : `in_progress`, `succeeded`, `rolled_back`, `failed` ; Prometheus : `sva_update_last_result`)	Retour arrière (`rolled_back`) ou échec (`failed`) ; sur un site redondant, vérifiez `succeeded` sur une sonde avant d'activer la suivante (§ 4.3)
Prochaine maintenance	`sva.maintenance.next_start`, `sva.maintenance.next_end` (horodatages, en secondes)	Activation annoncée en dehors des créneaux attendus ; absence de prochaine maintenance alors qu'une mise à jour est en attente
État de maintenance	`sva.agent.maintenance` (Prometheus : `sva_agent_maintenance`)	Valeur `1` : la sonde est en état de maintenance, du drainage de ses exécutions jusqu'au retour en service après l'activation d'une mise à jour. Les alertes d'absence de signal (`sva.agent.up`) émises pendant la maintenance sont attendues : leur suspension se configure dans la plateforme (chapitre 2)

> **Avertissement** — Un bridage pour cause de sous-tension (`undervoltage`) ne doit jamais être ignoré. Une alimentation insuffisante provoque des comportements erratiques et peut corrompre la carte microSD. Remplacez l'alimentation par un modèle conforme aux prérequis (chapitre 2 du guide d'installation et d'administration).

Le catalogue complet des métriques et la façon de les exploiter dans une plateforme d'observabilité sont détaillés au chapitre 2. Les symptômes de dysfonctionnement de la sonde et leurs remèdes sont traités au chapitre 9 du guide d'installation et d'administration.

3.2 Surveillance de la santé de la sonde

3.2.1 Indicateurs à suivre

La sonde publie des métriques sur sa propre santé (chapitre 2). Les alertes suivantes sont recommandées dans votre plateforme de supervision.

Alerte	Condition indicative	Signification
Sonde muette	Absence de `sva.agent.up` pendant 5 minutes	Sonde éteinte, injoignable ou export interrompu (équivalent de l'état `STALE` de la sonde)
Sous-tension	`sva.hw.throttled` égal à 1 avec l'attribut `sva.hw.reason` = `undervoltage`	Alimentation insuffisante : risque de corruption
Surchauffe	`sva.hw.throttled` égal à 1 avec `sva.hw.reason` = `thermal`, ou `sva.hw.temperature` durablement élevée	Refroidissement à vérifier
Stockage	Occupation d'un système de fichiers de la sonde (`system.filesystem.usage`, état `used`) au-delà de 80 % de sa capacité pendant 30 minutes ; le détail par zone est donné par `sva.storage.usage`	Rétention à réduire ou support à remplacer (§ 8.6.4 du guide d'installation et d'administration)
Saturation	`sva.agent.scheduler.lag` en hausse durable	Trop de contrôles pour la sonde
Échecs de Chromium	Hausse de `sva.browser.launches` en échec	Problème du moteur de scénarios
Export	`sva.export.failures` ou `sva.export.dropped` en hausse	Destination d'export en erreur
Récupération de base	Incrément de `sva.storage.recoveries`	Coupure ayant corrompu la base : examiner l'alimentation
Mise à jour en attente sans fenêtre	État de santé `updates` en `WARNING`, ou `sva.update.pending` égal à 1 depuis plus de 24 heures sans maintenance planifiée (`sva.maintenance.next_start` égal à `0` ; une série absente traduit une perte de collecte, pas l'absence de planification)	Aucune fenêtre de maintenance : la mise à jour ne sera pas activée sans intervention (§ 8.4.2, § 8.4.8 du guide d'installation et d'administration)
Correctif critique en attente	État de santé `updates` en `CRITICAL`, ou `sva.update.pending` égal à 1 avec l'attribut `sva.update.critical` = `true`	Vulnérabilité activement exploitée : décider de l'activation (§ 8.4.11 du guide d'installation et d'administration)
Mise à jour en échec	`sva.update.last_result` porte `rolled_back` ou `failed` (`lastResult` de `GET /api/v1/updates`)	Retour arrière automatique ou échec avant redémarrage : la version précédente reste active ; consulter le journal d'audit et le § 9.7 du guide d'installation et d'administration
Maintenance anormalement longue	`sva.agent.maintenance` égal à 1 au-delà de la durée de la fenêtre, ou sonde muette après la fin de la fenêtre de maintenance (`sva.maintenance.next_end` dépassé) ou, pour une activation sans fin prévue (`sva.maintenance.next_end` à `0`), après la durée maximale que vous avez retenue	Activation bloquée ou sonde non revenue après son redémarrage : consulter le journal d'audit et le § 9.7 du guide d'installation et d'administration

Les noms des métriques utilisent le préfixe stable `sva.`. Leur traduction au format Prometheus et des exemples de requêtes figurent au chapitre 2.

3.2.2 Vérification manuelle

En complément des alertes, consultez périodiquement la page de santé de l'interface ou `GET /api/v1/system` : version, ressources, état des modules et des destinations d'export. Trois états de santé concernent particulièrement la maintenance : `storage`, qui passe en `CRITICAL` si le SSD activé est absent ou défaillant (§ 8.3.4 du guide d'installation et d'administration) ; `support`, qui passe en `WARNING` à la fin de la période de support (§ 8.4.6 du guide d'installation et d'administration) ; `updates`, qui passe en `WARNING` lorsqu'une mise à jour attend sans activation planifiée ou après un report (§ 8.4.2 du guide d'installation et d'administration) et en `CRITICAL` tant qu'un correctif critique n'est pas actif (§ 8.4.11 du guide d'installation et d'administration).

Chapitre 4 — Maintenance, redondance et alertes

4.1 Notification, drainage et état de maintenance

Une activation suit toujours la même chronologie, qu'elle ait lieu dans une fenêtre de maintenance ou à la demande d'un administrateur.

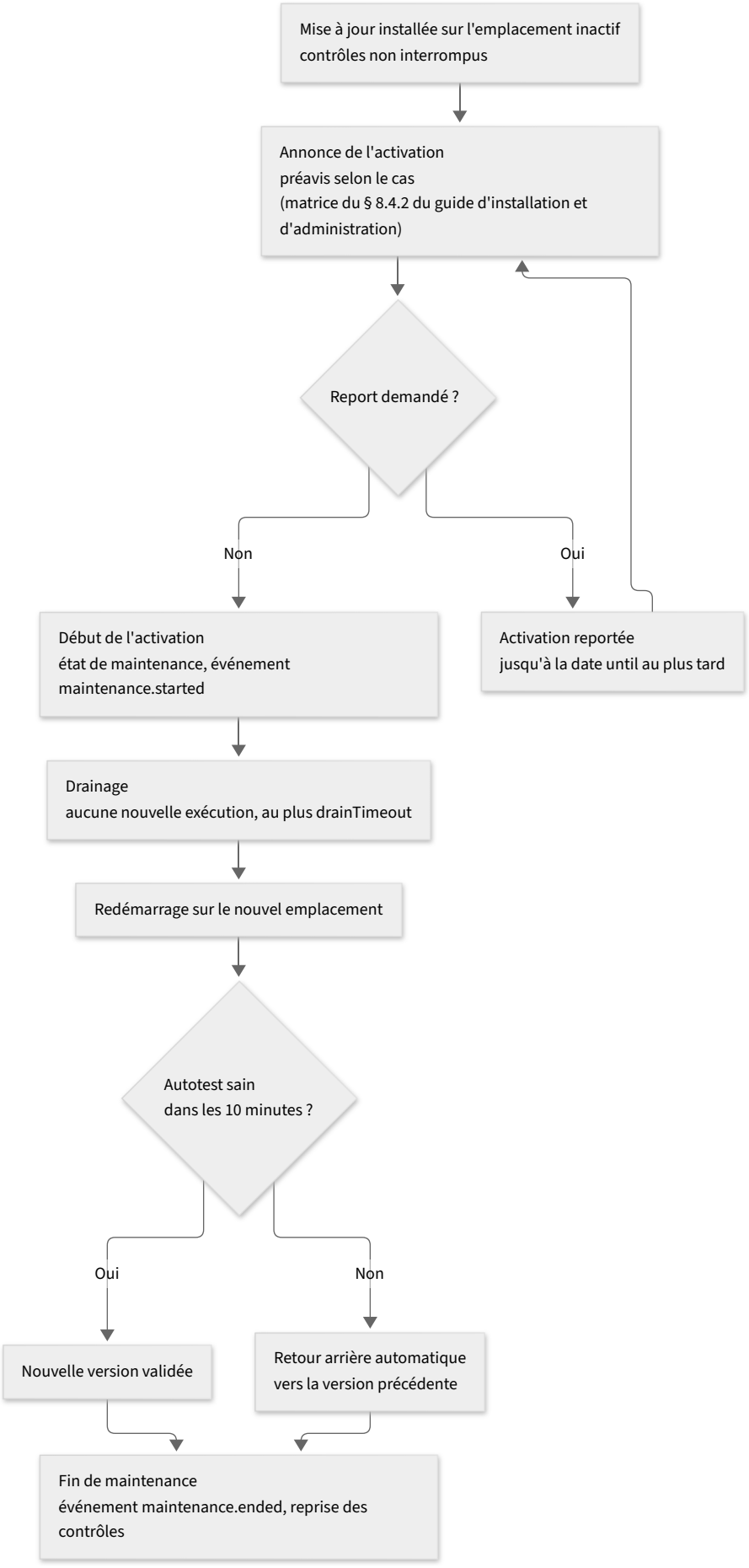


Schéma 4-1 — Chronologie d'une activation : notification, drainage, maintenance

1. Annonce. Dès qu'une activation est planifiée ou replanifiée, la sonde l'annonce par les moyens suivants, avec le préavis que fixe la matrice du § 8.4.2 du guide d'installation et d'administration pour son cas : au moins `notice` pour une activation automatique ordinaire, 15 minutes pour un correctif critique (augmentées du décalage en mode `asap`), le délai choisi pour une activation manuelle programmée, aucun pour une activation manuelle immédiate. Un préavis effectif inférieur à `notice` est signalé par `noticeReduced: true` :

Moyen	Contenu
Interface d'administration	Bandeau indiquant la mise à jour et l'heure prévue de son activation
`GET /api/v1/updates`	Champ `scheduledActivation` : activation planifiée
Journal d'audit	Planification de l'activation, puis report éventuel
Événement OTLP `update.scheduled`	Transmis aux destinations d'export (chapitre 2), à chaque planification et à chaque replanification
Événement OTLP `update.cancelled`	Activation annoncée qui n'aura pas lieu sous cet identifiant : report sans instant déterminable, image remplacée par une autre, canal passé en mode `manual` ou fenêtres supprimées, image retirée
Métrique `sva.update.pending` (unité `{state}`, attributs `sva.update.channel` et `sva.update.critical`)	Mise à jour en attente d'activation, avec son canal et son caractère critique
Métriques `sva.maintenance.next_start` et `sva.maintenance.next_end` (unité `s`, horodatages)	Début et fin **prévue** de la prochaine maintenance planifiée

Ces signaux permettent à vos équipes et à vos plateformes d'anticiper l'activation : vérifier les périodes de maintenance, prévenir les équipes concernées ou reporter l'activation (§ 8.4.2 du guide d'installation et d'administration).

Identification et ordre des annonces. Chaque activation planifiée reçoit un **identifiant d'activation** unique (`activationId` dans l'API, attribut `sva.update.activation.id` des événements). Il reste le même tant que la même image doit être activée : report, nouvelle date, activation manuelle qui avance l'activation annoncée, recalcul après un changement de fuseau. Une nouvelle image reçoit un nouvel identifiant, et l'activation de l'image remplacée est annulée par `update.cancelled`. Chaque annonce porte en outre un **numéro d'annonce** (`sva.update.announcement.sequence`, repris par `scheduledActivation.sequence`), strictement croissant pour une même sonde, toutes activations confondues. Ce numéro ne régresse ni après un redémarrage, ni après un retour arrière, ni après une restauration de configuration. Il est dérivé de l'heure de la sonde : après le remplacement d'une sonde qui reprend l'identité de la précédente (§ 8.6 du guide d'installation et d'administration), sa croissance n'est assurée que si l'heure de la nouvelle sonde est fiable (indicateur `clock` à `OK`, § 5.4 du guide d'installation et d'administration).

Pour un automatisme qui reçoit ces événements, la règle est simple : pour une sonde donnée, **seule compte l'annonce de plus grand numéro**. Une annonce reçue en double, ou après une annonce de numéro supérieur, est ignorée. Le chapitre 2 (§ 2.15.4) en donne une mise en œuvre pour Alertmanager.

Les attributs de ces quatre événements (`update.scheduled`, `update.cancelled`, `maintenance.started`, `maintenance.ended`) sont décrits au chapitre 2 (§ 2.3.6), qui fait référence : identifiant d'activation, numéro d'annonce, cas d'activation, canal, caractère critique, version, préavis réduit, bornes de maintenance, motif d'annulation et résultat.

Valeurs selon le mode d'activation. L'instant de l'annonce et la borne de fin dépendent du mode d'activation. Dans le tableau suivant, l'**annonce** est l'événement `update.scheduled` ; le **début annoncé** est la valeur de `scheduledActivation.at` et de `sva.maintenance.next_start` ; la **fin prévue** est celle de `windowEnd` et de `sva.maintenance.next_end`.

Mode d'activation	Annonce	`noticeReduced`	Début annoncé	Fin prévue
Automatique ordinaire	Au moins `notice` avant le début de la fenêtre retenue	`false`	Début de la fenêtre retenue	Fin de cette fenêtre
Manuelle immédiate (`POST /api/v1/updates/apply` sans `scheduledAt`)	Au début de l'activation, avec `maintenance.started`	`true`	Heure de réception de la demande (champ `at` de la réponse)	Aucune fenêtre : `windowEnd` vaut `null`
Manuelle à date fixe (`scheduledAt`)	À l'acceptation de la demande	`true` si `scheduledAt` précède l'expiration du préavis `notice` , `false` sinon	`scheduledAt`	Hors fenêtre : `windowEnd` vaut `null` . Date située dans une fenêtre : fin de cette fenêtre
Correctif critique, `criticalPolicy: asap`	Dès l'installation, au moins 15 minutes avant l'activation (§ 8.4.11 du guide d'installation et d'administration)	`true` si 15 minutes augmentées du décalage restent inférieures à `notice`	Instant annoncé : 15 minutes après l'annonce, augmentées du décalage `security.criticalStagger` de la sonde (§ 8.4.11 du guide d'installation et d'administration)	Aucune fenêtre : `windowEnd` vaut `null`
Correctif critique, `criticalPolicy: next-window`	Dès l'installation, au moins 15 minutes avant le début de la fenêtre retenue	`true` si le délai jusqu'à la fenêtre retenue est inférieur à `notice`	Début de la première fenêtre qui commence au moins 15 minutes après l'annonce	Fin de cette fenêtre

Lorsque `windowEnd` vaut `null` , la métrique `sva.maintenance.next_end` vaut `0` alors que `sva.maintenance.next_start` porte le début annoncé, et l'attribut `sva.maintenance.end` est **absent** de `update.scheduled` : aucune fin n'est annoncée, et la durée de la maintenance se prépare comme indiqué au paragraphe suivant. Sans fenêtre ni activation planifiée, les deux métriques valent `0` .

La **fin prévue** est une borne de planification. La **reprise réelle** est signalée par l'événement `maintenance.ended` (autotest réussi ou retour arrière terminé) : elle peut précéder la fin prévue ou la dépasser, par exemple lorsqu'une activation commencée en fin de fenêtre déborde (§ 8.4.8 du guide d'installation et d'administration). Pour une activation hors fenêtre, dimensionnez la période de maintenance de vos plateformes sur une **durée maximale** que vous fixez d'après la durée totale observée d'une activation sur vos sondes, retour arrière compris (§ 8.4.5 du guide d'installation et d'administration), puis levez-la dès la reprise vérifiée de la sonde (§ 4.3).

2. Début de l'activation et drainage. Au début de l'activation, la sonde passe en **état de maintenance** et émet l'événement `maintenance.started` . Aucune exécution nouvelle ne démarre. Les exécutions en cours disposent du délai `drainTimeout` (5 minutes par défaut, 30 minutes au maximum) pour se terminer normalement. Au-delà, elles sont interrompues et closes avec l'état `UNKNOWN` et le type d'erreur `maintenance` .

3. Redémarrage et autotest. La sonde redémarre sur le nouvel emplacement, puis vérifie son fonctionnement pendant 10 minutes au plus. En cas d'échec, elle revient automatiquement à la version précédente (§ 8.4.5 du guide d'installation et d'administration).

4. Fin de maintenance. La sonde émet l'événement `maintenance.ended` , quitte l'état de maintenance et reprend ses contrôles à leur prochaine échéance. Cet événement est émis lorsque l'autotest a réussi, ou lorsque le retour arrière est terminé ; son attribut `sva.update.result` indique lequel. Il signale la fin de la maintenance, pas à lui seul

le bon fonctionnement des contrôles : la reprise se vérifie comme au § 4.3.

Exposition de l'état de maintenance. Tant que la sonde répond, son état de maintenance est visible par les moyens suivants :

Moyen	Valeur pendant la maintenance
`GET /health`	`{"status": "MAINTENANCE"}`, avec le code HTTP `503`
Métrique `sva.agent.maintenance` (unité `{state}`, Prometheus `sva_agent_maintenance`)	`1`
Événements OTLP	`maintenance.started` au début, `maintenance.ended` à la fin
Plugin `check_sva` (Nagios, Centreon)	Code de retour `3` (`UNKNOWN`), avec le texte `MAINTENANCE` (§ 2.10)

> **Attention** — Pendant le redémarrage proprement dit, la sonde ne répond plus du tout : ni `/health`, ni les métriques, ni `check_sva` ne sont disponibles. L'état de maintenance ne peut donc pas, à lui seul, éviter les alertes de vos plateformes (§ 4.2).

Conséquences pour les contrôles

Situation	Conséquence
Exécution en cours au début de l'activation	Se termine normalement si elle s'achève dans le délai `drainTimeout` ; sinon, close en `UNKNOWN` avec le type d'erreur `maintenance`
Exécution prévue pendant la maintenance	Non lancée ; elle n'est pas rattrapée après la reprise (§ 8.2.3 du guide d'installation et d'administration)
État `STALE`	Son calcul est suspendu pendant la maintenance et pendant un intervalle de planification de chaque contrôle après la reprise : une maintenance ne rend aucun contrôle `STALE`

> **Information** — Le type d'erreur `maintenance` signale une exécution interrompue par la sonde elle-même pour activer une mise à jour. Comme `agent_restart` (§ 8.2.3 du guide d'installation et d'administration), il ne traduit aucun incident de la ressource supervisée. Vous pouvez l'exclure de vos calculs de disponibilité, selon la méthode convenue avec vos parties prenantes (chapitre 7 du guide d'utilisation et de supervision).

4.2 Éviter les fausses alertes pendant une maintenance

L'état de maintenance (§ 4.1) informe vos plateformes que la sonde va redémarrer, mais il **ne suffit pas à supprimer les alertes**. Pendant le redémarrage, la sonde est injoignable : Prometheus constate l'échec de la collecte, les métriques `sva.agent.up` cessent d'arriver, et `check_sva` ne reçoit aucune réponse. Sans précaution, vos plateformes déclenchent alors les alertes « sonde injoignable » ou « sonde muette », alors qu'il s'agit d'une opération planifiée.

Objectif : éviter que l'activation d'une mise à jour ne déclenche de fausses alertes, sans masquer une véritable panne.

Prérequis : fenêtres de maintenance définies (§ 8.4.8 du guide d'installation et d'administration) ; accès à la configuration des alertes de vos plateformes.

Étapes

1. Dans chaque plateforme qui reçoit les données de la sonde, configurez une période de maintenance récurrente **alignée sur chacune des fenêtres** de la sonde (mêmes jours, même heure de début, même durée, même fuseau horaire). Les mécanismes à utiliser sont décrits au chapitre 2 (§ 2.15) : intervalles de temps d'Alertmanager, règles excluant les sondes en maintenance (`sva_agent_maintenance == 1`), temps d'arrêt programmés récurrents de Datadog, fenêtres de maintenance de Dynatrace, temps d'arrêt récurrents de Centreon, temps d'arrêt programmés de Nagios.
2. Limitez cette période aux alertes qui concernent **la sonde en maintenance** et les contrôles qu'elle exécute. Ne neutralisez pas les alertes des autres sondes du site, qui continuent de superviser (§ 4.3).
3. Pour une activation hors fenêtre, c'est-à-dire une activation manuelle (§ 8.4.3 du guide d'installation et d'administration) ou un correctif critique en mode `asap` (§ 8.4.11 du guide d'installation et d'administration), planifiez une période de maintenance ponctuelle limitée à la sonde concernée : **avant** de lancer une activation manuelle immédiate, car son annonce arrive au moment où elle commence ; **dès l'annonce** dans les autres cas, le préavis étant alors de 15 minutes au moins. Fixez sa durée maximale d'après la durée totale observée d'une activation, retour arrière compris, et levez-la après la reprise vérifiée de la sonde (§ 4.3). Une automatisation par événement est décrite au chapitre 2 (§ 2.15.4).
4. Conservez une alerte qui se déclenche si la sonde n'est pas revenue **après la fin de la fenêtre** ou, sans fenêtre, après la durée maximale retenue (§ 3.2.1, alerte « maintenance anormalement longue ») : c'est elle qui révèle une activation qui a mal tourné.

Résultat attendu : une activation dans une fenêtre, ou couverte par une période ponctuelle, ne produit aucune alerte ; une sonde qui ne revient pas après la fin de la fenêtre ou de la période déclenche une alerte.

Vérification : lors de la première activation qui suit la configuration, contrôlez dans vos plateformes qu'aucune alerte n'a été déclenchée pendant la fenêtre, et que les événements `maintenance.started` et `maintenance.ended` ont bien été reçus.

Erreurs possibles

Situation	Cause probable	Action
Alerte « sonde injoignable » pendant la fenêtre	Période de maintenance de la plateforme absente, décalée ou exprimée dans un autre fuseau horaire	Réaligner la période sur la fenêtre de la sonde
Aucune alerte alors que la sonde n'est jamais revenue	Période de maintenance plus longue que la fenêtre, ou alerte de maintenance anormalement longue absente	Limiter la période à la fenêtre ; ajouter l'alerte du § 3.2.1

> **Avertissement** — Une période de maintenance trop large masque les véritables pannes. Ne neutralisez jamais durablement les alertes d'une sonde au motif qu'elle se met à jour.

Les configurations détaillées propres à chaque plateforme figurent au chapitre 2 (§ 2.15.2 à § 2.15.5).

4.3 Continuité de service et redondance

Une sonde unique **ne supervise pas pendant son propre redémarrage**. Pendant l'activation d'une mise à jour (§ 4.1), les contrôles qu'elle exécute sont suspendus : une panne survenant à ce moment ne serait détectée qu'après son retour. Pour la plupart des usages, cette interruption brève et planifiée est acceptable. Pour les environnements critiques, elle ne l'est pas.

Architecture recommandée pour les environnements critiques

Principe	Mise en œuvre
Au moins **deux sondes par site**	Chaque sonde exécute les mêmes contrôles, sur les mêmes cibles
Fenêtres **qui ne se chevauchent pas**	L'écart entre les fenêtres des sondes d'un même site dépasse la durée totale mesurée d'une activation, retour arrière compris (§ 8.4.8 du guide d'installation et d'administration), afin que leurs activations automatiques ne se recouvrent pas ; elles ne sont jamais activées manuellement en même temps (§ 8.4.3 du guide d'installation et d'administration). Cet espacement ne coordonne pas les activations `asap`, qui ont lieu hors fenêtre (§ 8.4.11 du guide d'installation et d'administration)
Activations critiques **échelonnées**	Avec `criticalPolicy: asap`, un décalage `security.criticalStagger` différent sur chaque sonde du site (§ 8.4.11 du guide d'installation et d'administration). Le décalage réduit le risque d'activations simultanées sans le supprimer : il ne vérifie pas l'état des autres sondes
Retour en service **vérifié**	Pour les environnements critiques : fenêtres distinctes avec `criticalPolicy: next-window`, ou orchestration externe qui n'active une sonde qu'après le retour vérifié de la précédente (ci-dessous)
Alertes fondées sur **l'ensemble des sondes** du site	Une alerte ne dépend jamais d'une seule sonde : elle est évaluée à partir des résultats de toutes les sondes qui répondent
Indépendance matérielle	Si possible, alimentations et raccordements réseau distincts (commutateurs différents), pour qu'un même incident local n'affecte pas toutes les sondes

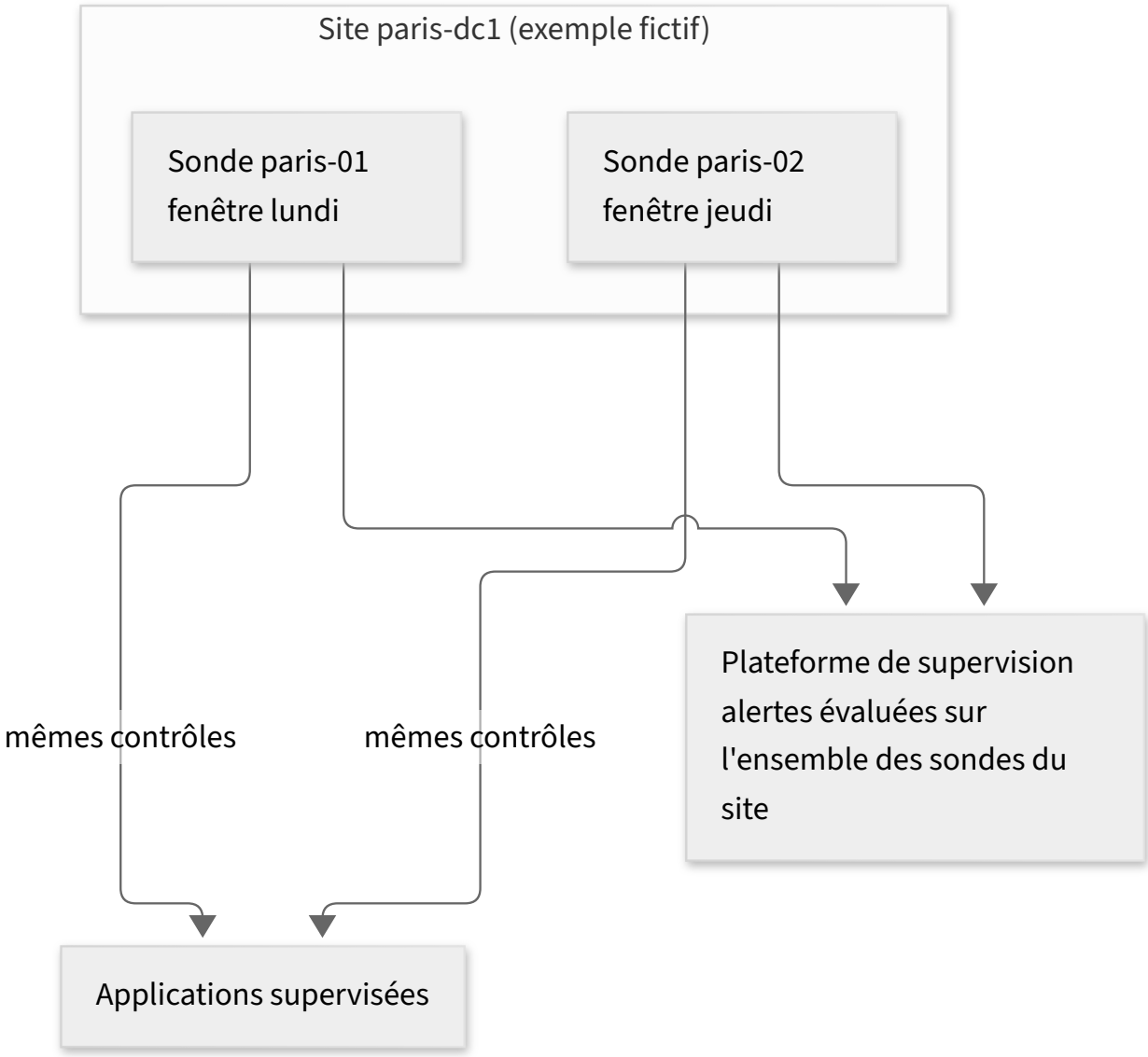


Schéma 4-2 — Redondance : plusieurs sondes sur un même site

Objectif : maintenir la supervision d'un site pendant la maintenance de l'une de ses sondes.

Prérequis

- Au moins deux sondes installées sur le site (chapitres 3 et 4 du guide d'installation et d'administration), chacune avec sa propre carte microSD de l'édition requise, et le même paramètre `site`.
- Une plateforme de supervision capable d'agréger les résultats de plusieurs sondes (chapitre 2).

Étapes

1. Installez et mettez en service la seconde sonde, avec le même paramètre `site` que la première (chapitre 4 du guide d'installation et d'administration).
2. Exportez la configuration de la première sonde (§ 8.5.2 du guide d'installation et d'administration), puis importez-la sur la seconde avec le paramètre `newIdentity=true` (§ 8.5.3 du guide d'installation et d'administration) : les deux sondes exécutent les mêmes contrôles, avec les mêmes identifiants de contrôle, tout en restant distinctes dans vos plateformes.
3. Attribuez à chaque sonde des fenêtres de maintenance qui ne se chevauchent pas (§ 8.4.8 du guide d'installation et d'administration). L'import ayant recopié la section `updates`, modifiez les fenêtres de la seconde sonde.

4. Créez des règles d'alerte fondées sur l'ensemble des sondes du site (règles d'alerte de site ci-dessous ; groupe `sondes-site` du § 2.15.6), en complément des alertes de santé propres à chaque sonde (§ 3.2.1).
5. Configurez les périodes de maintenance de vos plateformes sonde par sonde (§ 4.2).
6. Si vous retenez `criticalPolicy: asap`, attribuez à chaque sonde un décalage `security.criticalStagger` différent (§ 8.4.11 du guide d'installation et d'administration).

Orchestration externe des activations (environnements critiques). Les sondes ne se coordonnent pas entre elles. Lorsque l'interruption simultanée de la supervision d'un site n'est pas acceptable, confiez l'enchaînement des activations à votre propre outil d'automatisation, avec l'API existante :

1. Placez le canal `security` des sondes du site en mode `manual` : aucune activation n'a lieu sans votre outil.
2. Activez la première sonde (`POST /api/v1/updates/apply`, § 8.4.3 du guide d'installation et d'administration).
3. Conservez l'identifiant `activationId` renvoyé par la réponse de l'activation, ainsi que la version attendue (`staged.version` de `GET /api/v1/updates`, relevée avant l'activation).
4. Attendez son retour en service effectif, et non un simple délai. Interrogez `GET /api/v1/updates` à intervalles réguliers, jusqu'à un délai maximal que vous fixez d'après la durée totale observée d'une activation, retour arrière compris. La sonde n'est revenue en service que si **toutes** les conditions suivantes sont réunies :
   - `lastResult` n'est pas `null` et `lastResult.activationId` est égal à l'identifiant conservé à l'étape 3 ;
   - `lastResult.outcome` vaut `succeeded` ;
   - la version de l'emplacement actif (`slots` et `activeSlot`) est la version attendue ;
   - `state` ne vaut pas `maintenance` ;
   - `/health` répond `OK` (§ 4.6.2 du guide d'installation et d'administration) ;
   - votre plateforme de supervision reçoit de cette sonde des résultats d'exécution postérieurs à `lastResult.finishedAt`.
5. Seulement alors, activez la sonde suivante.

Interprétez les autres situations ainsi :

Situation observée	Signification	Action
`lastResult` à `null`, ou `activationId` différent de celui de l'étape 3	Le résultat ne concerne pas votre activation : elle n'a pas encore produit de résultat, ou il s'agit d'une tentative antérieure	Continuer d'attendre, jusqu'au délai maximal
`outcome` à `in_progress`, ou `state` à `maintenance`	Activation en cours (drainage, redémarrage ou autotest)	Continuer d'attendre, jusqu'au délai maximal
`outcome` à `rolled_back`	Retour arrière automatique : l'ancienne version est active	Arrêter l'enchaînement et traiter l'incident (§ 9.7 du guide d'installation et d'administration)
`outcome` à `failed`	Échec avant redémarrage : l'emplacement actif est inchangé	Arrêter l'enchaînement et traiter l'incident
`outcome` à `succeeded`, mais version active différente de la version attendue	Incohérence : une autre image a été activée	Arrêter l'enchaînement et vérifier `GET /api/v1/updates` et le journal d'audit
Conditions réunies, sauf l'arrivée de résultats récents	La sonde a redémarré mais ne supervise pas, ou ses résultats ne parviennent pas à la plateforme	Ne pas poursuivre ; vérifier les destinations d'export et les contrôles de la sonde
Délai maximal dépassé	Sonde injoignable ou activation bloquée	Arrêter l'enchaînement ; intervenir sur la sonde (chapitre 9 du guide d'installation et d'administration)

Un succès antérieur ne prouve jamais la reprise : `lastResult` désigne la **dernière tentative**, quelle que soit l'activation. La jauge `sva.update.last_result` peut servir de signal complémentaire dans vos tableaux de bord, mais elle ne porte ni la version ni l'identifiant d'activation : elle ne suffit pas à valider une étape d'orchestration. De même, le décalage `security.criticalStagger` (§ 8.4.11 du guide d'installation et d'administration) réduit le risque d'activations simultanées, mais ne vérifie pas le retour des autres sondes.

> **Attention** — En mode `manual`, la sonde n'active jamais d'elle-même un correctif critique (§ 8.4.11 du guide d'installation et d'administration). La priorité du correctif repose alors sur votre procédure : l'état de santé `updates` reste en `CRITICAL` et l'alerte « correctif critique en attente » (§ 3.2.1) reste active jusqu'à l'activation. Fixez dans votre procédure un délai maximal de traitement, afin que la coordination ne retarde jamais indéfiniment un correctif de sécurité.

> **Évolution prévue** — La console de gestion centralisée prévue (§ 1.8 de la référence technique et API) doit permettre d'orchestrer **séquentiellement** les mises à jour des sondes d'un même site : chaque activation ne serait engagée qu'après le retour vérifié de la précédente à un état sain, avec un délai maximal au-delà duquel un correctif critique serait activé malgré tout. Elle est prévue dans une version ultérieure ; la sonde ne dépendra jamais de cette console pour fonctionner.

Règles d'alerte de site. Deux alertes distinctes sont nécessaires, car un échec applicatif et une perte de couverture appellent des réactions différentes. Une sonde est dite **évaluable** pour un contrôle lorsqu'elle en fournit un résultat `OK`, `WARNING` ou `CRITICAL` et qu'elle n'est pas en maintenance.

Alerte	Se déclenche lorsque	Signification
`ParcoursEnEchecSurLeSite`	Toutes les sondes évaluables du site voient le contrôle en `CRITICAL`, au moins une sonde étant évaluable	Incident probable de l'application ou du parcours supervisé
`CouvertureInsuffisanteSurLeSite`	Le nombre de sondes évaluables est inférieur à l'effectif attendu du site, y compris lorsqu'aucune sonde ne fournit plus de résultat évaluable	Perte de couverture : la supervision du site repose sur moins de sondes que prévu, ou n'est plus assurée

Ces deux alertes forment le groupe de règles Prometheus `sondes-site`, fourni prêt à charger dans le fichier `exemples/prometheus/sva-regles-alerte.yml`. Leurs expressions PromQL, le seul paramètre à adapter (l'effectif attendu de chaque site, `sva:site_effectif_attendu`), leurs limites, l'extension facultative `sondes-site-controles` et l'adaptation au mode envoi (label `sva_site`) sont décrits au chapitre 2 (§ 2.15.6).

> **Information** — L'alerte de couverture se déclenche aussi pendant la maintenance planifiée d'une sonde, puisque celle-ci n'est plus évaluable. Choisissez pour elle un délai `for` supérieur à la durée totale mesurée d'une activation (§ 8.4.5 du guide d'installation et d'administration), ou neutralisez cette seule alerte pendant les fenêtres des sondes du site (§ 4.2). Ne neutralisez jamais `ParcoursEnEchecSurLeSite` : c'est elle qui détecte un incident pendant la maintenance d'une sonde.

Résultat attendu : pendant la maintenance d'une sonde, la supervision du site reste assurée par les autres, et les alertes restent fondées sur leurs résultats.

Vérification : activez une mise à jour sur l'une des sondes (§ 8.4.3 du guide d'installation et d'administration), ou attendez sa fenêtre, et vérifiez que l'autre continue de publier ses résultats et que la règle reste évaluée.

Erreurs possibles

Situation	Cause probable	Action
Alerte déclenchée pendant la maintenance d'une sonde	Règle fondée sur une seule sonde ; label `site` (ou `sva_site`) absent	Réécrire la règle sur l'ensemble des sondes ; vérifier les labels
Aucune supervision pendant une fenêtre	Fenêtres des sondes qui se chevauchent ; activation qui déborde sur la fenêtre de l'autre sonde ; activations manuelles simultanées	Décaler les fenêtres en tenant compte de la durée totale d'une activation (§ 8.4.8 du guide d'installation et d'administration) ; activer les sondes l'une après l'autre
`CouvertureInsuffisanteSurLeSite` à chaque activation planifiée	Délai `for` inférieur à la durée d'une activation	Allonger le délai `for` ou neutraliser cette seule alerte pendant les fenêtres du site
Résultats divergents entre les sondes	Configurations différentes, ou accès réseau différents	Réimporter la configuration ; comparer les flux réseau des deux sondes (chapitre 5 du guide d'installation et d'administration)

Information — La durée pendant laquelle une sonde ne supervise pas lors d'une activation s'observe entre les événements `maintenance.started` et `maintenance.ended` (§ 8.4.5 du guide d'installation et d'administration, § 4.1). Pour les sites qui ne disposent que d'une sonde, cette durée est la seule interruption planifiée de la supervision. Une activation **automatique** commence dans l'une des fenêtres que vous avez choisies, mais elle peut se prolonger au-delà de la fin de la fenêtre lorsque sa durée totale, retour arrière éventuel compris, dépasse le temps restant dans la fenêtre (§ 8.4.8 du guide d'installation et d'administration). L'activation manuelle par un administrateur (§ 8.4.3 du guide d'installation et d'administration) et l'activation d'un correctif critique avec `criticalPolicy: asap` (§ 8.4.11 du guide d'installation et d'administration) peuvent en outre intervenir hors fenêtre. Sur un site redondant, l'espacement des fenêtres ne coordonne pas les activations `asap` : un même correctif critique publié pour toutes les sondes du site peut les activer à des instants proches, sauf si vous leur attribuez des décalages `security.criticalStagger` différents (§ 8.4.11 du guide d'installation et d'administration), qui réduisent ce risque sans l'exclure.

Chapitre 5 — Dépannage des intégrations

5.1 Destinations d'export et intégrations

5.1.1 Tester une destination

Avant toute analyse, lancez le test intégré de la destination concernée. Il envoie un point de test et renvoie le code HTTP, le corps de la réponse d'erreur et la latence.

```bash
curl --cacert sonde-ca.pem -X POST \
     -H @"$HOME/.sva/entete-auth" \
     https://sonde-paris-01.example.internal/api/v1/exporters/dynatrace/test
```

5.1.2 Tableau des symptômes

Ce tableau reprend, dans la démarche de diagnostic de ce chapitre, les cas du tableau de dépannage des exports du chapitre 2 (§ 2.16).

Symptôme	Causes possibles	Vérifications	Actions correctives
HTTP 401	Clé ou jeton d'ingestion absent, invalide ou révoqué ; mauvais format d'en-tête (`Api-Token` pour Dynatrace, en-tête de clé d'API pour Datadog)	Test de la destination ; secret référencé dans les en-têtes	Mettre à jour le secret ; vérifier le format exact de l'en-tête (chapitre 2)
HTTP 403	Jeton valide mais sans les droits d'ingestion requis ; pour Datadog, endpoint ne correspondant pas au « site » du compte	Portées du jeton Dynatrace (ingestion de métriques, logs, traces) ; site Datadog	Ajouter les portées nécessaires ; utiliser l'endpoint du bon site Datadog
HTTP 400 sur les métriques vers Dynatrace ou l'intake direct Datadog	**Temporalité cumulative** envoyée à une plateforme qui n'accepte que le **delta**	Paramètre `metrics.temporality` de la destination	Régler `metrics.temporality` sur `delta` (préréglages `dynatrace` et `datadog-agentless`)
Succès partiel signalé, certaines métriques absentes dans Dynatrace	Requête mêlant métriques acceptées et refusées (type non pris en charge, temporalité)	Journal de la sonde (`sva.export.failures`)	Utiliser le préréglage `dynatrace` , qui fixe la temporalité delta et des histogrammes à limites explicites
HTTP 404	Chemin d'endpoint incorrect, notamment pour le récepteur OTLP de Prometheus (`/api/v1/otlp/v1/metrics`)	URL complète testée	Pour Prometheus, renseigner l'URL complète dans `signalEndpoints.metrics`

Symptôme	Causes possibles	Vérifications	Actions correctives
HTTP 413	Lot trop volumineux	Taille des lots (1 000 points par lot par défaut)	Réduire le volume (désactiver `metrics.stepHistograms`, allonger `metrics.exportInterval`) ; si l'erreur persiste, contacter le support
HTTP 429	Limitation de débit de la plateforme ; quota d'ingestion atteint	En-tête `Retry-After` ; consommation dans la plateforme	La sonde réessaie automatiquement ; réduire le volume (désactiver les histogrammes par étape, espacer l'export) ; ajuster le quota
HTTP 5xx, erreurs réseau	Plateforme indisponible ; proxy ; pare-feu	Test de la destination ; connectivité vers l'endpoint	Vérifier le flux sortant (§ 6.1) ; la sonde réessaie pendant `retry.maxElapsed` puis abandonne le lot (`sva.export.dropped`)
Prometheus ne collecte pas la sonde (cible `DOWN`)	Pare-feu ; jeton absent ou de mauvais rôle ; certificat non reconnu par Prometheus	Page des cibles de Prometheus (message d'erreur) ; `curl` vers `/metrics` avec le jeton	Ouvrir le port 443 depuis Prometheus ; utiliser un jeton de rôle `metrics` ; renseigner `tls_config.ca_file`
Données reçues par le récepteur OTLP de Prometheus mais invisibles	Récepteur non activé (`--web.enable-otlp-receiver`) ; ingestion hors ordre refusée ; noms traduits avec suffixes	Métriques internes de Prometheus sur la traduction OTLP	Activer le récepteur ; configurer `out_of_order_time_window` ; rechercher les noms traduits (`sva_check_duration_seconds…`)
Métriques présentes mais aucun tableau de bord ni alerte	Comportement normal : l'ingestion OTLP ne crée ni tableau de bord ni alerte	—	Importer les modèles fournis par l'éditeur (`exemples/grafana/sva-tableau-de-bord.json` pour Grafana, `exemples/prometheus/sva-regles-alerte.yml` pour les règles d'alerte Prometheus) ou créer les tableaux de bord et alertes (chapitre 2)
Nagios ou Centreon affiche `UNKNOWN` pour `check_sva`	Sonde injoignable depuis le serveur Nagios ; jeton invalide ; identifiant de contrôle erroné ; texte `MAINTENANCE` (code 3) : la sonde est en état de maintenance pour l'activation d'une mise à jour, situation normale et temporaire (§ 9.7 du guide d'installation et d'administration)	Exécuter le plugin en ligne de commande sur le serveur Nagios	Corriger le jeton (rôle `metrics`), l'identifiant du contrôle (option `--check`) et le flux HTTPS
Trous dans les courbes après une coupure réseau	Comportement documenté : les métriques ne sont pas rejouées après une indisponibilité de la destination	`sva.export.dropped`	L'historique des résultats reste disponible sur la sonde, dans la limite de sa rétention locale (chapitre 7 du guide d'utilisation et de supervision). Pour réduire les pertes, utiliser un collecteur OpenTelemetry avec file persistante (chapitre 2) : voir l'encadré ci-dessous sur sa portée
Absence de données de la sonde dans la plateforme alors qu'elle fonctionne	Export interrompu (voir lignes précédentes) ; heure de la sonde décalée ; activation d'une mise à jour en cours (absence normale pendant le redémarrage)	Test de la destination ; `timedatectl`	Corriger l'export ; corriger la synchronisation NTP

Information — Un collecteur OpenTelemetry à file persistante **réduit les pertes** de données ; il ne garantit pas leur livraison. Distinguez les deux segments du trajet :

- **de la sonde au collecteur** : la sonde réessaie l'envoi pendant `retry.maxElapsed`, puis abandonne le lot. Un lot abandonné n'atteint jamais le collecteur, qui ne peut préserver que les données qu'il a reçues. Ces abandons sont observables par la métrique `sva.export.dropped` ;
- **du collecteur à la destination** : la file persistante conserve les données reçues pendant une indisponibilité de la destination, dans la limite de sa taille et de sa durée de conservation. Au-delà, le collecteur abandonne à son tour des données ; surveillez ses propres métriques de file et d'abandon.

Placer le collecteur au plus près de la sonde, sur un réseau fiable, réduit le risque sur le premier segment. En cas de perte, l'historique local de la sonde reste la référence, dans la limite de sa rétention (chapitre 7 du guide d'utilisation et de supervision).

Chapitre 6 — Références

6.1 Tableau des flux réseau

Ce tableau reprend, à l'identique, la matrice des flux de la configuration réseau, pour la transmettre à l'équipe chargée des pare-feu.

Sens	Source	Destination	Port / protocole	Usage	Obligatoire
Entrant	Postes d'administration	Sonde	443/TCP (HTTPS)	Interface d'administration, API	Oui
Entrant	Serveur Prometheus	Sonde	443/TCP (HTTPS)	Collecte de `/metrics`	Si Prometheus en mode collecte
Entrant	Serveur Nagios ou Centreon	Sonde	443/TCP (HTTPS)	Plugin `check_sva`	Si Nagios ou Centreon
Entrant	Postes d'administration	Sonde	80/TCP (HTTP)	Redirection permanente vers HTTPS, aucun contenu servi	Non (désactivable)
Entrant	Postes du réseau local	Sonde	5353/UDP (mDNS)	Résolution du nom `.local` de la sonde ; annonce active par défaut	Non (désactivable dans la section `network`)
Entrant	Postes d'administration autorisés	Sonde	22/TCP (SSH)	Maintenance avancée	Non : désactivé par défaut ; activation temporaire, 4 heures par défaut, 72 heures au maximum (§ 5.9.4 du guide d'installation et d'administration)
Couche 2 (sans flux IP)	Sonde	Commutateur (authentificateur 802.1X)	EAPOL (*EAP over LAN*) : trames Ethernet de type 0x888E, aucun port ni adresse IP	Authentification 802.1X EAP-TLS du port ; le commutateur relaie vers le serveur RADIUS	Si le port est contrôlé par 802.1X (§ 5.2.6 du guide d'installation et d'administration)
Entrant et sortant	Sonde	Routeurs et voisins du lien local	ICMPv6 (découverte des voisins, NDP, dont les annonces de routeur utilisées en mode `slaac`)	Adressage et résolution d'adresses IPv6	Si l'IPv6 est activé (§ 5.2.5 du guide d'installation et d'administration)
Sortant	Sonde	Serveurs DHCP	67-68/UDP	Adressage	Si DHCP
Sortant	Sonde	Serveurs DNS	53/UDP et 53/TCP	Résolution de noms	Oui
Sortant	Sonde	Serveurs NTP	123/UDP	Synchronisation de l'heure	Oui
Sortant	Sonde	Dépôt de mises à jour de l'éditeur, ou miroir interne	443/TCP (HTTPS)	Mises à jour en ligne : vérification et téléchargement des images signées ; adresse par défaut `https://updates-synovex.incitypark.fr`, inscrite dans le profil éditeur	Si mises à jour en ligne
Sortant	Sonde	Proxy de l'entreprise	Port du proxy (par exemple 3128/TCP)	Relais des flux HTTP(S) sortants (§ 5.5 du guide d'installation et d'administration)	Si proxy
Sortant	Sonde	Plateformes SaaS (Datadog, Dynatrace, Grafana Cloud)	443/TCP (HTTPS, OTLP/HTTP)	Export OpenTelemetry	Si configurées

Sens	Source	Destination	Port / protocole	Usage	Obligatoire
Sortant	Sonde	Collecteur OpenTelemetry ou Agent Datadog du client	4318/TCP (OTLP/HTTP) ou 4317/TCP (OTLP/gRPC), selon le protocole de la destination (§ 2.1.4)	Export OpenTelemetry	Si configurés
Sortant	Sonde	Proxy inverse placé devant le récepteur OTLP de Prometheus	Port du proxy inverse (par exemple 443/TCP, HTTPS)	Export des métriques en mode envoi (§ 2.5.3)	Si configuré
Sortant	Sonde	Ressources supervisées	Selon les contrôles : ICMP ou ICMPv6 ; ports TCP des services ; 53 ; 80 et 443 ; SGBD : 5432 (PostgreSQL), 3306 (MySQL/MariaDB), 1433 (SQL Server), 1521 (Oracle)	Contrôles	Oui, selon les contrôles
Sortant	Sonde	Console centrale	443/TCP (HTTPS, authentification mutuelle mTLS)	Gestion multi-sondes	Non : évolution prévue, facultative

Information — Les ports des SGBD indiqués sont les ports par défaut. Ils peuvent différer dans votre environnement. Les ports des contrôles sont ceux que vous configurez. La sonde n'utilise aucun serveur de licences en ligne et n'envoie aucune télémétrie à l'éditeur.

Information — Si l'IPv6 est activé, les flux de ce tableau s'appliquent à l'identique en IPv6 : le pare-feu local de la sonde applique les mêmes règles dans les deux familles d'adresses. Les échanges RADIUS ont lieu entre le commutateur et le serveur RADIUS : ils ne partent pas de la sonde. Le mécanisme de mise à jour A/B ne crée aucun flux supplémentaire, et une mise à jour hors ligne n'en exige aucun vers l'éditeur.

6.2 États et codes retour Nagios

6.2.1 États de la sonde

État	Signification	Valeur de `sva.check.status`	Code retour de `check_sva`
`OK`	Contrôle réussi, dans les seuils	0	0
`WARNING`	Contrôle réussi, mais seuil de performance dépassé, ou plugin ayant renvoyé 1	1	1
`CRITICAL`	Échec technique ou fonctionnel confirmé (après les éventuelles nouvelles tentatives)	2	2
`UNKNOWN`	Le contrôle n'a pas pu être évalué (problème de la sonde, défaillance du navigateur, plugin ayant renvoyé 3)	3	3
`STALE`	Aucun résultat depuis deux intervalles pour un contrôle actif. Calcul suspendu pendant l'état de maintenance et pendant un intervalle après la reprise	—	3
Sonde en état de maintenance	Activation d'une mise à jour en cours, du drainage au retour en service (`/health` : `MAINTENANCE`) ; ce n'est pas un état de contrôle	—	3, avec le texte `MAINTENANCE`

6.2.2 Convention des plugins Nagios

Code retour	État Nagios	État de la sonde
0	OK	`OK`
1	WARNING	`WARNING`
2	CRITICAL	`CRITICAL`
3	UNKNOWN	`UNKNOWN`
Autre code, ou plugin interrompu	—	`UNKNOWN` (`plugin_error`)

Sortie d'un plugin : une ligne de texte lisible, suivie facultativement du caractère `|` et des données de performance au format `label=valeur[unité];avertissement;critique;min;max`.

Exemple de sortie de `check_sva` :

```
TEXTE
CRITICAL - boutique-connexion : étape 'Vérifier l'accès à l'espace client' (assertion_failed) |
    duration=14.65s;8;;
```