GUIDE D'UTILISATION ET DE SUPERVISION

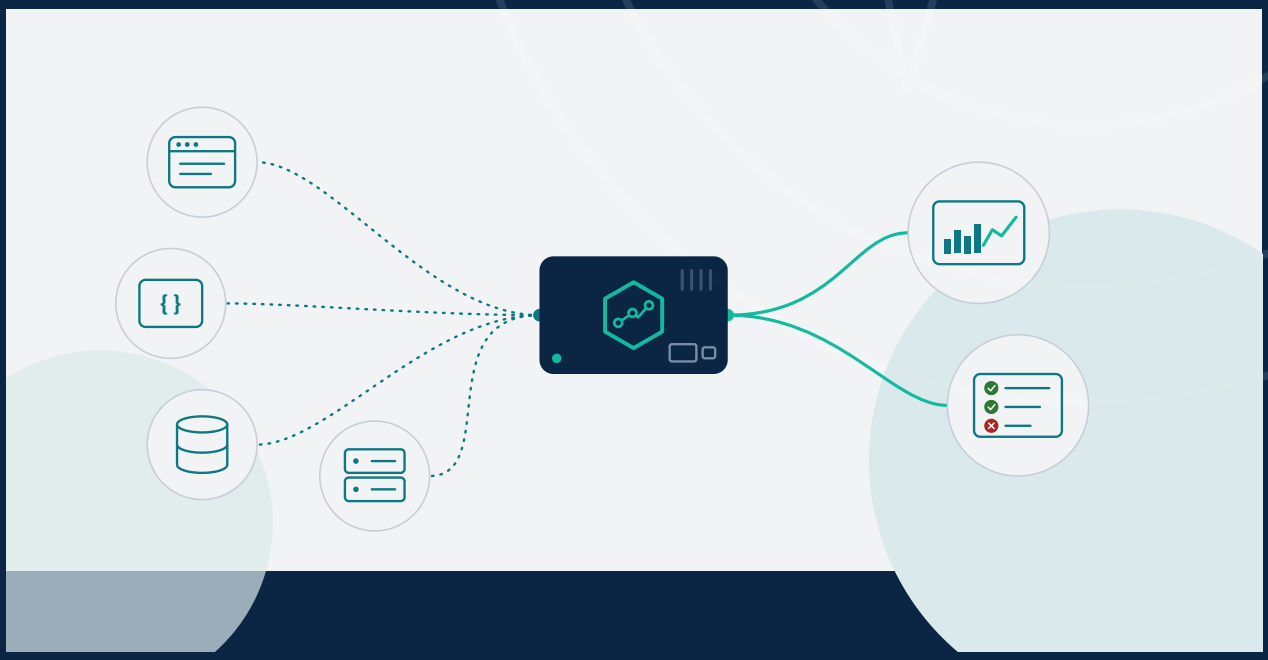
SYNOVEX

Guide d'utilisation et de supervision

Matériel : Raspberry Pi 5

Éditions : Essential, Professional, Enterprise

Public : Équipes d'exploitation, administrateurs applicatifs, équipes de production, responsables de supervision



SYNOVEX — SIREN 884 713 009

Table des matières

Informations sur le document 4

Chapitre 1 — Comprendre la supervision applicative 7
1.1 Qu'est-ce que la supervision ? 7
1.2 Disponibilité, performance et fiabilité 7
1.3 Les niveaux de supervision 9
1.4 Pourquoi un serveur disponible ne garantit pas le bon fonctionnement d'une application 10
1.5 Limites des contrôles techniques classiques 12
1.6 L'intérêt de la supervision synthétique 13
1.7 Détection proactive et réduction du temps de détection 15
1.8 Une supervision complémentaire, pas un remplacement 17
1.9 L'apport de la supervision applicative aux équipes 18

Chapitre 2 — Éditions, points d'observation et fonctionnement autonome 20
2.1 Périmètre de la solution 20
2.2 Vue d'ensemble 22
2.3 Cycle de vie d'une exécution 24

Chapitre 3 — Principes de fonctionnement 27
3.1 La notion de contrôle 27
3.2 Les cibles supervisées 30
3.3 La fréquence d'exécution 31
3.4 Le délai maximal d'exécution (timeout) 32
3.5 Nouvelles tentatives et confirmation d'un incident 33
3.6 Les seuils de performance 36
3.7 Le résultat d'une exécution 36
3.8 Les états et les erreurs 38
3.9 Mesures de performance 42
3.10 Erreurs d'exécution et échecs de la cible 42
3.11 Classes d'exécution et concurrence 43
3.12 Rétention et exploitation des données 44
3.13 Incident réseau, applicatif ou fonctionnel 46
3.14 Fonctionnement autonome 47

Chapitre 4 — Contrôles réseau et techniques 50
4.1 Vue d'ensemble 50
4.2 Choisir le bon contrôle 52
4.3 Contrôle ICMP (icmp) 54
4.4 Contrôle TCP (tcp) 57
4.5 Contrôle DNS (dns) 60
4.6 Contrôle HTTP et HTTPS (http) 63
4.7 Plugins Nagios (nagios) 72
4.8 Synthèse 78

Chapitre 5 — Contrôles applicatifs et métier 79
5.1 De la disponibilité technique à la réussite fonctionnelle 79
5.2 Superviser une API REST avec le contrôle http 80
5.3 Enchaîner des requêtes avec le contrôle api 83
5.4 Superviser un web service SOAP 87
5.5 Concevoir des contrôles fonctionnels robustes 89
5.6 Contrôles de bases de données (database) 89
5.7 Indicateurs métier 98

5.8 Détection d'anomalies 99
5.9 Précautions et sécurité 101
5.10 Synthèse 102

Chapitre 6 — Planification et plans d'exécution 103
6.1 Deux façons d'organiser les exécutions 103
6.2 Planifier un contrôle 104
6.3 Les plans d'exécution 108
6.4 Comportement d'un plan 109
6.5 Créer et piloter un plan 112
6.6 Charge et dimensionnement 114
6.7 Exemples de configurations 116
6.8 Interventions planifiées et fenêtres de maintenance 119

Chapitre 7 — Consultation et interprétation des résultats 120
7.1 Où se trouvent les résultats 120
7.2 Consultation dans l'interface d'administration 121
7.3 Consultation par l'API REST 123
7.4 Lire un résultat d'exécution 127
7.5 Méthode d'analyse d'un échec 129
7.6 Bagotage, nouvelles tentatives et confirmation des incidents 133
7.7 Indicateurs : temps de réponse, percentiles et disponibilité 134
7.8 Historique, rétention et agrégats 137
7.9 État de santé de la sonde 137
7.10 Exporter les données 140
7.11 Synthèse 141

Chapitre 8 — Diagnostic des contrôles 143
8.1 Méthode générale de diagnostic 143
8.2 Contrôles techniques et applicatifs 145
8.3 Types d'erreur et actions de premier niveau 148

Chapitre 9 — Références 150
9.1 Types d'erreur 150
9.2 États et codes retour Nagios 151
9.3 Glossaire 152

SYNOVEX

Guide d'utilisation et de supervision

Sonde de supervision applicative autonome pour Raspberry Pi 5

Information	Valeur
Document	Guide d'utilisation et de supervision — identifiant SYN-DOC-03
Révision documentaire	B
Public	Équipes d'exploitation, administrateurs applicatifs, équipes de production, responsables de supervision
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 s'adresse aux équipes qui créent, exploitent et suivent au quotidien les contrôles de supervision exécutés par SYNOVEX. Il présente les principes de la supervision applicative, les éditions et les points d'observation, le fonctionnement des contrôles (fréquence, délais, nouvelles tentatives, seuils, états), les contrôles réseau, techniques, applicatifs et métier, la planification et les plans d'exécution, puis la consultation et l'interprétation des résultats.

La sonde est supposée installée et mise en service (guide de démarrage rapide, SYN-DOC-01, ou guide d'installation et d'administration, SYN-DOC-02). Les scénarios navigateur Puppeteer font l'objet du guide de développement des scénarios (SYN-DOC-04).

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 — Comprendre la supervision applicative

Ce chapitre présente les notions sur lesquelles repose la sonde. Il ne suppose aucune connaissance préalable de la supervision et peut être lu par des interlocuteurs non techniques : responsables métier, chefs de projet, direction. Les lecteurs familiers des outils de supervision peuvent se limiter aux § 1.4 à § 1.8.

1.1 Qu'est-ce que la supervision ?

La **supervision** (en anglais *monitoring*) désigne l'ensemble des moyens qui permettent de connaître en permanence l'état d'un système informatique et d'être averti lorsqu'il s'écarte de son fonctionnement normal. Elle repose sur trois activités complémentaires :

1. **Mesurer** : collecter régulièrement des informations sur les composants et les services (un serveur répond-il ? en combien de temps ? une page contient-elle le texte attendu ?).
2. **Évaluer** : comparer ces mesures à des critères définis à l'avance (seuils, résultats attendus) pour en déduire un **état** : normal, dégradé ou en panne.
3. **Alerter et conserver** : prévenir les bonnes personnes lorsqu'un état se dégrade, et garder l'historique pour analyser les incidents, mesurer la qualité de service et anticiper les problèmes.

Le terme **observabilité**, apparu plus récemment, désigne la capacité à comprendre le comportement interne d'un système à partir des données qu'il produit : métriques, traces et journaux. La supervision répond à la question « quelque chose ne va-t-il pas ? ». L'observabilité aide à répondre à « pourquoi ? ». La sonde contribue aux deux : elle détecte les dysfonctionnements et fournit les éléments qui permettent d'en comprendre la cause (étape en échec, type d'erreur, capture d'écran, durées détaillées).

1.2 Disponibilité, performance et fiabilité

Trois notions structurent toute démarche de supervision. Elles sont souvent confondues ; les distinguer permet de choisir les bons contrôles.

Notion	Question posée	Exemple de mesure	Exemple de dysfonctionnement
Disponibilité	Le service est-il accessible et rend-il le service attendu ?	Pourcentage d'exécutions réussies sur une période	Le site renvoie une erreur ; la connexion des utilisateurs échoue
Performance	Le service répond-il dans un délai acceptable ?	Temps de réponse d'une page, durée d'un parcours, temps par étape	La recherche de produits prend 12 secondes au lieu de 2
Fiabilité	Le service se comporte-t-il de façon constante dans le temps ?	Fréquence des échecs intermittents, variabilité des temps de réponse	Une erreur sur dix tentatives, imprévisible

La disponibilité s'exprime généralement en pourcentage sur une période. Le tableau suivant montre ce que représentent concrètement quelques objectifs courants :

Disponibilité visée	Indisponibilité tolérée par mois (30 jours)	Indisponibilité tolérée par an
99 %	7 h 12 min	3 j 15 h 36 min

Disponibilité visée	Indisponibilité tolérée par mois (30 jours)	Indisponibilité tolérée par an
99,5 %	3 h 36 min	1 j 19 h 48 min
99,9 %	43 min 12 s	8 h 45 min 36 s
99,99 %	4 min 19 s	52 min 34 s

Ces objectifs n'ont de sens que si la disponibilité est mesurée **du point de vue de l'utilisateur**. Un serveur disponible à 99,99 % ne garantit rien si l'application qu'il héberge est inutilisable une demi-journée par mois. C'est précisément ce que la supervision applicative permet de mesurer.

> **Information** — Un objectif de niveau de service se formalise souvent par un **SLO** (*Service Level Objective*), par exemple « 99,5 % des parcours de commande réussissent en moins de 8 secondes sur un mois glissant ». Les résultats de la sonde, exportés vers votre plateforme d'observabilité (chapitre 2 du guide d'intégration et d'observabilité), permettent de calculer ces indicateurs.

1.3 Les niveaux de supervision

On distingue habituellement plusieurs niveaux de supervision, du plus matériel au plus fonctionnel. Chacun répond à des questions différentes et s'appuie sur des outils différents.

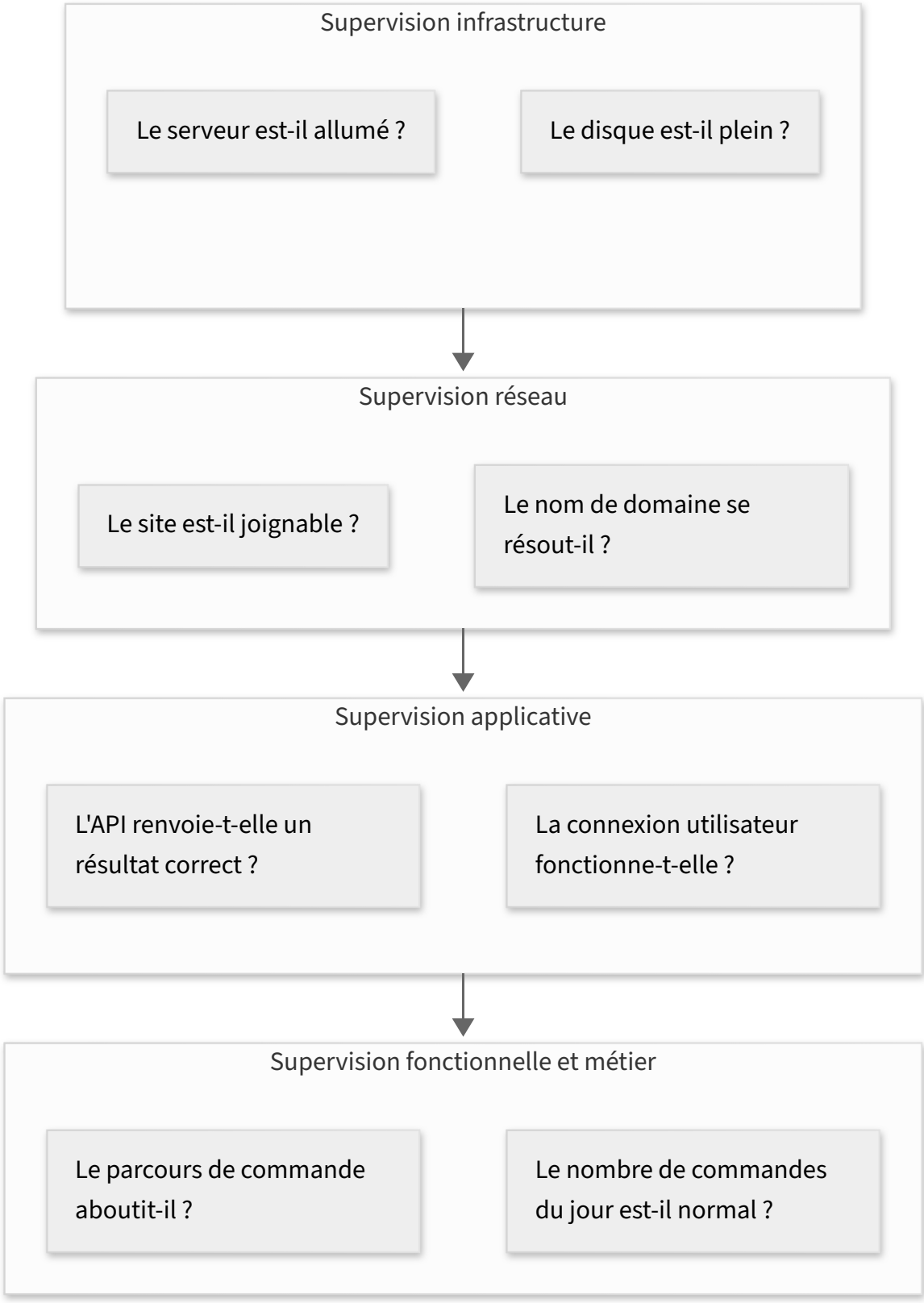


Schéma 1-1 — Les niveaux de supervision, du matériel au métier

1.3.1 Supervision de l'infrastructure

Elle surveille les ressources matérielles et système : processeur, mémoire, disques, température, état des services du système d'exploitation, sauvegardes. Elle est généralement assurée par des agents installés sur les serveurs ou par des protocoles d'administration (SNMP, WMI) et par des outils tels que Nagios, Centreon ou Zabbix. Elle est indispensable pour anticiper la saturation d'une ressource et diagnostiquer une panne matérielle.

1.3.2 Supervision du réseau

Elle vérifie que les équipements et les services sont joignables et que les communications s'établissent : réponse au ping, ouverture des ports, résolution des noms, latence, perte de paquets. Elle détecte les coupures de liens, les erreurs de routage ou de filtrage et les défaillances de DNS.

1.3.3 Supervision applicative

Elle vérifie que les applications rendent le service attendu, en les sollicitant comme le ferait un utilisateur ou une application cliente : requête vers une API et analyse de la réponse, ouverture d'une page web et vérification de son contenu, connexion avec un compte de test, requête sur une base de données. Elle ne se contente pas de constater qu'un service « répond » : elle vérifie qu'il répond **correctement**.

1.3.4 Supervision fonctionnelle et métier

Elle va plus loin en vérifiant des enchaînements complets et des indicateurs propres à l'activité : un parcours d'achat aboutit-il ? le nombre de commandes enregistrées depuis le début de la journée est-il cohérent avec l'activité habituelle ? un traitement nocturne a-t-il produit les données attendues ? Elle traduit la santé du système d'information dans les termes du métier.

La sonde couvre les niveaux réseau, applicatif et fonctionnel. Elle s'appuie sur la supervision de l'infrastructure existante sans s'y substituer.

1.4 Pourquoi un serveur disponible ne garantit pas le bon fonctionnement d'une application

Une application moderne repose sur de nombreux composants : serveurs web, serveurs d'application, bases de données, files de messages, services tiers (paiement, authentification, cartographie), réseaux de diffusion de contenu, scripts exécutés dans le navigateur. Chacun peut fonctionner individuellement alors que l'ensemble ne rend plus le service. Les situations suivantes, toutes rencontrées en exploitation, illustrent cet écart. Elles sont fictives dans leurs détails.

1.4.1 Le site fonctionne, mais le bouton de commande est défaillant

Une boutique en ligne s'affiche normalement. Les serveurs web répondent, leur processeur est peu chargé, la base de données est disponible : la supervision de l'infrastructure ne signale rien. Pourtant, après une mise à jour, un script exécuté dans le navigateur provoque une erreur au moment où l'utilisateur clique sur **Valider ma commande**. Le bouton ne réagit plus. Aucune commande n'aboutit, et l'incident n'est découvert que plusieurs heures plus tard, par les réclamations des clients ou par l'équipe commerciale qui constate l'absence de ventes.

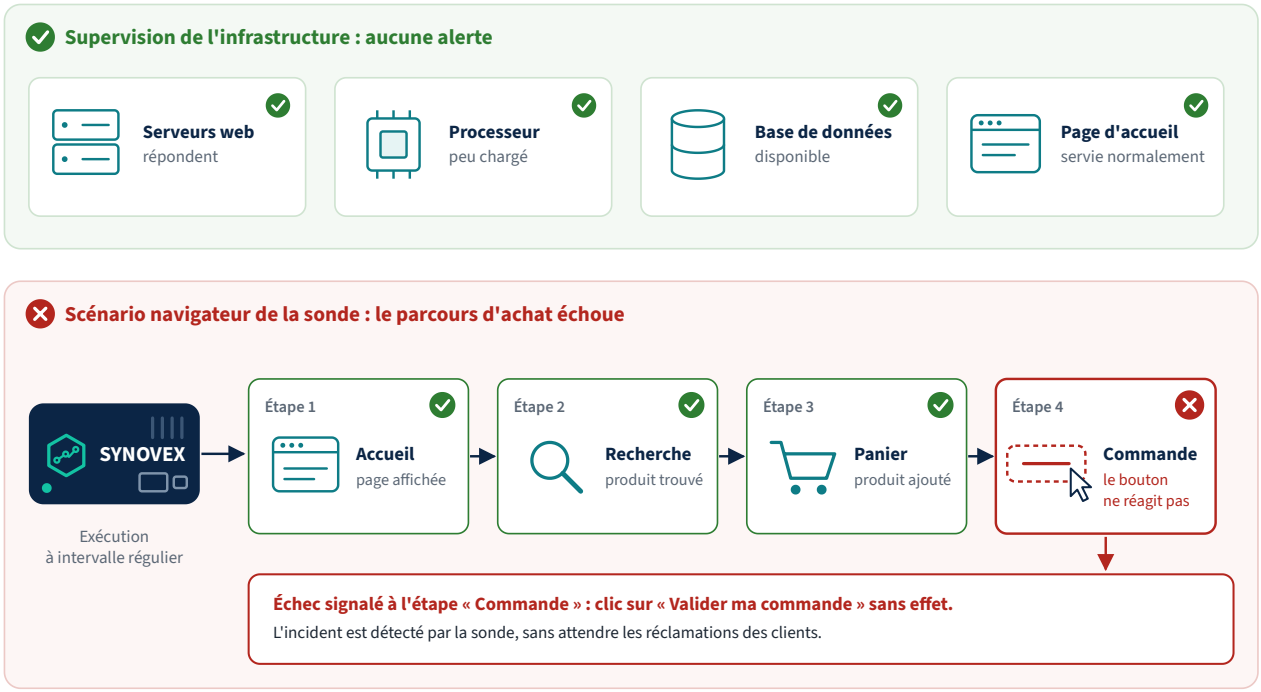


Figure 1-1 — Exemple : l'infrastructure ne signale aucune anomalie, mais le parcours d'achat échoue à l'étape de commande ; le scénario de la sonde le détecte

Seul un contrôle qui **reproduit le parcours d'achat dans un vrai navigateur**, jusqu'au clic sur ce bouton, peut détecter ce type de défaut. C'est le rôle des scénarios navigateur (chapitres 3 à 6 du guide de développement des scénarios).

1.4.2 L'API répond HTTP 200, mais le résultat est faux

Une API de calcul de prix répond avec le code HTTP `200 OK`, qui signifie « requête traitée avec succès ». Un contrôle qui se limite au code de retour la considère donc comme disponible. Or, à la suite d'une erreur de configuration, l'API renvoie un prix à zéro pour tous les articles, ou une liste vide au lieu de la liste des produits disponibles.

Un contrôle applicatif doit donc **vérifier le contenu de la réponse** : présence d'un champ, valeur attendue, nombre d'éléments, cohérence d'un montant. Les contrôles HTTP et API de la sonde permettent d'exprimer ces vérifications, appelées **assertions** (chapitres 4 et 5).

1.4.3 La base de données répond, mais les traitements métier échouent

Le serveur de base de données accepte les connexions et répond aux requêtes simples. Mais un traitement planifié chargé d'intégrer les commandes de la nuit a échoué silencieusement : la table des commandes à préparer est vide ce matin. Les entrepôts n'expédient rien, et la supervision technique reste au vert.

Une requête de contrôle qui vérifie qu'**au moins un enregistrement a été intégré depuis minuit**, ou que la date de la dernière intégration est récente, signale immédiatement l'anomalie. Les contrôles de bases de données de la sonde (chapitre 5) reposent sur ce principe.

1.4.4 Le site est accessible, mais l'authentification ne fonctionne plus

La page de connexion d'une application s'affiche. Mais l'annuaire d'entreprise ou le fournisseur d'identité utilisé pour vérifier les mots de passe est inaccessible depuis le serveur d'application, ou son certificat a expiré. Tous les utilisateurs reçoivent le message « identifiant ou mot de passe incorrect ». Le service est, de fait, totalement indisponible, alors que tous les contrôles techniques sont positifs.

Un scénario qui **se connecte avec un compte de test** et vérifie l'affichage de l'espace personnel détecte ce dysfonctionnement en quelques minutes.

1.4.5 Le nombre de commandes chute anormalement

Aucune erreur technique n'est visible, mais le nombre de commandes enregistrées entre 10 h et 11 h est cinq fois inférieur à celui des semaines précédentes à la même heure. Les causes possibles sont multiples : un moyen de paiement refusé pour une catégorie de cartes, une page produit qui ne s'affiche plus sur certains navigateurs, une campagne de communication renvoyant vers une adresse erronée.

La surveillance d'un **indicateur métier**, le volume de commandes sur une période, avec un seuil minimal adapté à l'heure de la journée, révèle des problèmes qu'aucun contrôle technique ne peut voir. La sonde permet de suivre ces indicateurs par des requêtes sur les bases de données et de les publier comme métriques (chapitre 5).

1.4.6 L'application fonctionne techniquement, mais le parcours devient inutilisable

Toutes les pages s'affichent et tous les services répondent. Mais une modification de la mise en page fait qu'une bannière d'information recouvre le bouton **Suivant** du formulaire d'inscription, ou que la page de résultats de recherche met vingt secondes à s'afficher. Techniquement, rien n'est « en panne ». Pour l'utilisateur, le service est inutilisable.

Un scénario navigateur qui **mesure la durée de chaque étape** et échoue si un élément n'est pas cliquable signale ces dégradations, en précisant l'étape concernée.

1.4.7 Synthèse

Situation	Supervision infrastructure	Supervision réseau	Contrôle HTTP simple (code retour)	Supervision applicative et fonctionnelle
Bouton de commande défaillant	Ne détecte pas	Ne détecte pas	Ne détecte pas	Détecte (scénario navigateur)
API 200 avec résultat faux	Ne détecte pas	Ne détecte pas	Ne détecte pas	Détecte (assertion sur le contenu)
Traitement métier en échec	Ne détecte pas	Ne détecte pas	Ne détecte pas	Détecte (requête de contrôle)
Authentification cassée	Ne détecte pas	Ne détecte pas	Ne détecte pas	Détecte (scénario de connexion)
Chute des commandes	Ne détecte pas	Ne détecte pas	Ne détecte pas	Détecte (indicateur métier)
Parcours inutilisable ou très lent	Ne détecte pas	Ne détecte pas	Partiellement	Détecte (mesure par étape)
Serveur arrêté	Détecte	Détecte	Détecte	Détecte
Disque saturé, avant impact	Détecte	Ne détecte pas	Ne détecte pas	Ne détecte pas

La dernière ligne rappelle que la supervision de l'infrastructure reste indispensable : elle détecte les risques **avant** qu'ils aient un impact visible. Les deux approches se complètent.

1.5 Limites des contrôles techniques classiques

Les contrôles techniques classiques présentent quatre limites que la supervision applicative compense.

Ils observent les composants, pas le service. Un contrôle porte sur un serveur, un processus ou un port. Le service rendu à l'utilisateur résulte de la coopération de dizaines de composants, dont certains échappent à votre supervision : services tiers, réseaux de diffusion, scripts exécutés dans le navigateur.

Ils se limitent souvent à la présence d'une réponse. Un port ouvert, un processus actif ou un code HTTP `200` indiquent que le composant répond, pas qu'il répond correctement.

Ils sont vus depuis l'intérieur. Un agent installé sur un serveur ne voit pas le chemin parcouru par l'utilisateur : résolution DNS, pare-feu, proxy, équilibreur de charge, certificat présenté au navigateur. Une erreur sur ce chemin reste invisible.

Ils ignorent la dimension métier. Un contrôle technique ne sait pas qu'une journée sans commande est anormale, ni qu'un délai de réponse acceptable pour un traitement de fond est inacceptable pour un client sur une page de paiement.

1.6 L'intérêt de la supervision synthétique

1.6.1 Définition

La **supervision synthétique** consiste à exécuter, à intervalles réguliers et de façon automatisée, des transactions **artificielles** et reproductibles qui imitent l'usage réel d'un service : chargement d'une page, appel d'une API, parcours utilisateur complet. Le terme « synthétique » signifie que ces transactions sont générées par l'outil de supervision et non par de vrais utilisateurs.

Ses propriétés font sa valeur :

Propriété	Bénéfice
Régularité	Le service est vérifié en permanence, y compris la nuit, le week-end et pendant les périodes creuses, quand aucun utilisateur ne pourrait signaler un problème
Reproductibilité	Le même parcours est exécuté à l'identique : toute variation de résultat ou de durée provient du service, et non d'un comportement différent de l'utilisateur
Objectivité	Les mesures sont comparables dans le temps, d'une version applicative à l'autre, d'un site à l'autre
Précision du diagnostic	En cas d'échec, l'étape fautive, le message d'erreur et une capture d'écran sont disponibles immédiatement
Indépendance	Le contrôle ne dépend ni du volume de trafic ni de la bonne volonté des utilisateurs à signaler les problèmes
Choix du point d'observation	Le contrôle est exécuté depuis l'emplacement où la sonde est installée : réseau interne, site distant, réseau d'un client ou d'un partenaire, autre région géographique

Lorsque plusieurs sondes sont installées dans des environnements différents, on parle de **supervision synthétique distribuée**. Elle permet de distinguer quatre questions complémentaires :

1. **la disponibilité technique du service** : le service répond-il correctement ?
2. **la disponibilité perçue depuis un point distant** : le service est-il accessible depuis le réseau d'un client, d'une agence ou d'un partenaire ?
3. **la performance observée** : quels sont les temps de réponse, la latence et les éventuelles dégradations depuis ce point d'observation ?
4. **le fonctionnement applicatif** : les contrôles configurés confirment-ils le comportement attendu des services et des parcours applicatifs ?

Un service peut répondre correctement depuis le centre de données tout en restant inaccessible ou lent depuis une agence, un pays ou le réseau d'un client : seule une mesure prise depuis ce point d'observation le révèle. Chaque sonde rend compte de ce qu'elle observe ; la comparaison des résultats entre sondes et l'analyse des causes restent du ressort de vos équipes et de vos outils de supervision (§ 2.2.1).

1.6.2 Navigation automatisée

Pour les applications web, la forme la plus complète de supervision synthétique pilote un **véritable navigateur**. La sonde utilise Chromium, le projet de navigateur libre sur lequel repose Google Chrome, en mode **headless**, c'est-à-dire sans affichage à l'écran. Elle le pilote avec **Puppeteer**, une bibliothèque qui permet de commander le navigateur par programme : ouvrir une page, remplir un formulaire, cliquer, attendre l'apparition d'un élément, vérifier un texte.

Contrairement à une simple requête HTTP, le navigateur exécute tout ce que la page contient : scripts, chargements différés, appels d'API effectués depuis la page, redirections d'authentification. Il voit donc ce que voit l'utilisateur. Le chapitre 3 du guide de développement des scénarios détaille ce fonctionnement et ses limites.

1.6.3 Supervision synthétique et mesure de l'expérience réelle

La supervision synthétique est souvent comparée à la **mesure de l'expérience des utilisateurs réels** (en anglais *Real User Monitoring*, RUM), qui collecte des mesures dans le navigateur de chaque visiteur.

Critère	Supervision synthétique (la sonde)	Mesure de l'expérience réelle (RUM)
Source des mesures	Transactions générées par la sonde	Navigation des utilisateurs réels
Fonctionne sans trafic	Oui	Non
Détecte un problème avant les utilisateurs	Oui, si le parcours concerné est contrôlé	Non : il faut que des utilisateurs soient touchés
Représentativité	Parcours choisis, conditions constantes	Diversité réelle des appareils, réseaux et usages
Couverture des applications internes	Oui, depuis tout emplacement où la sonde peut joindre l'application	Dépend de l'instrumentation des applications
Diagnostic d'un échec	Étape, erreur, capture d'écran	Statistiques agrégées, erreurs remontées par les navigateurs
Mise en œuvre	Sans modification des applications	Instrumentation des pages à prévoir

Les deux approches sont complémentaires. La sonde vous alerte au plus tôt sur les parcours critiques, y compris en l'absence de trafic. La mesure réelle révèle des problèmes propres à certains utilisateurs que les parcours synthétiques ne couvrent pas.

1.7 Détection proactive et réduction du temps de détection

1.7.1 Les temps d'un incident

Le déroulement d'un incident se décompose en plusieurs durées successives, qui forment ensemble la durée totale

d'impact pour les utilisateurs :

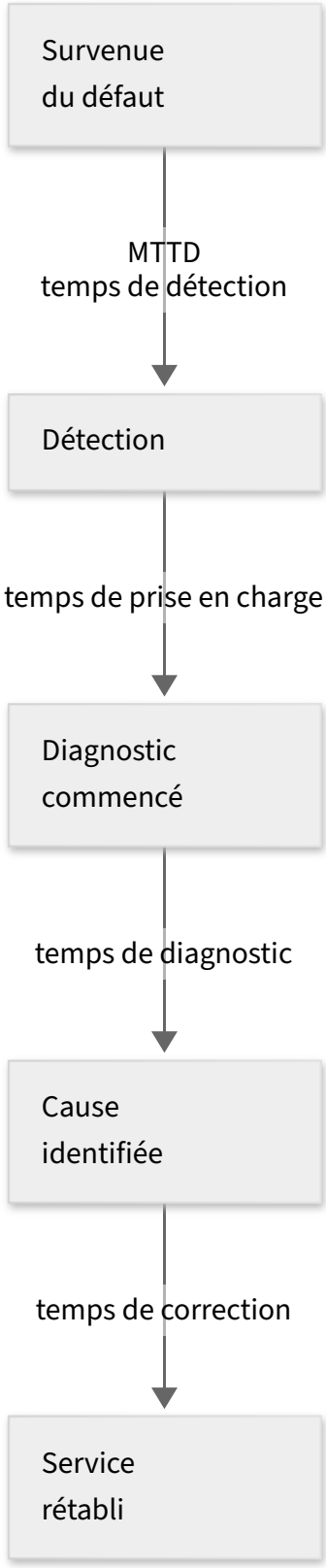


Schéma 1-2 — Les temps d'un incident : détection, diagnostic, rétablissement

Le **MTTD** (*Mean Time To Detect*, temps moyen de détection) mesure le délai entre la survenue d'un défaut et sa détection. Le **MTTR** (*Mean Time To Restore* ou *To Repair*, temps moyen de rétablissement) mesure le délai jusqu'au retour à la normale. Sans supervision applicative, la détection dépend souvent des signalements des utilisateurs : elle peut prendre plusieurs heures, voire un jour entier pour un défaut survenu la nuit ou touchant une fonction peu utilisée.

1.7.2 Ce qui détermine le temps de détection avec la sonde

Avec la sonde, le temps de détection dépend principalement de trois paramètres que vous maîtrisez :

Paramètre	Effet	Où le régler
Fréquence d'exécution du contrôle	Un contrôle exécuté toutes les 5 minutes détecte un défaut en 5 minutes au plus, en moyenne 2 min 30 s	Chapitre 6
Nouvelles tentatives avant confirmation	Elles évitent les fausses alertes dues à un incident passager, mais retardent la confirmation d'un vrai défaut	Chapitre 3
Délai de notification de votre plateforme d'alerte	Règles d'alerte, regroupement, astreinte	Chapitre 2 du guide d'intégration et d'observabilité

Par exemple, un parcours exécuté toutes les 5 minutes avec une nouvelle tentative après 1 minute confirme un défaut persistant entre 1 et 6 minutes environ après sa survenue (hors durée d'exécution du contrôle), au lieu de plusieurs heures en l'absence de contrôle.

1.7.3 Réduction du temps de diagnostic

La sonde réduit aussi le temps de diagnostic, souvent plus long que la détection. Lorsqu'un parcours échoue, elle indique l'étape concernée, le type d'erreur (élément introuvable, délai dépassé, erreur de certificat, assertion non vérifiée…), le message détaillé, la durée de chaque étape et, pour les scénarios navigateur, une capture d'écran de la page au moment de l'échec. L'équipe d'exploitation sait immédiatement s'il s'agit d'un problème réseau, applicatif ou fonctionnel (chapitre 3) et peut solliciter directement l'équipe compétente.

1.8 Une supervision complémentaire, pas un remplacement

La supervision applicative **complète** les dispositifs existants. Le tableau suivant situe la sonde par rapport aux autres familles d'outils couramment présentes dans les entreprises.

Famille d'outils	Exemples	Ce qu'elle apporte	Ce que la sonde ajoute
Supervision d'infrastructure et réseau	Nagios, Centreon, Zabbix, outils SNMP	État des serveurs, ressources, équipements, services système	Vision du service rendu, parcours, contrôle du contenu ; plugins compatibles Nagios et intégration à ces outils (chapitre 2 du guide d'intégration et d'observabilité)
Supervision des performances applicatives (APM)	Dynatrace, Datadog APM, autres outils à agents	Analyse détaillée du code serveur, des transactions et des dépendances internes, à partir du trafic réel	Mesure active et continue, même sans trafic ; vue utilisateur depuis le réseau ; corrélation possible avec les traces de l'APM (chapitre 2 du guide d'intégration et d'observabilité)
Métriques et tableaux de bord	Prometheus, Grafana	Collecte, stockage et visualisation des séries temporelles, alerting	Métriques de disponibilité et de performance des services, prêtes à être collectées
Mesure de l'expérience réelle	Outils RUM	Diversité des utilisateurs réels	Détection anticipée, parcours critiques, applications internes

Famille d'outils	Exemples	Ce qu'elle apporte	Ce que la sonde ajoute
Tests automatisés en intégration continue	Suites de tests fonctionnels	Vérification avant mise en production, sur des environnements de test	Vérification permanente **après** mise en production, dans les conditions réelles d'exploitation

La sonde s'insère dans ce paysage : ses résultats sont publiés vers vos plateformes existantes, où ils rejoignent vos tableaux de bord, vos règles d'alerte et vos processus d'astreinte. Vos équipes conservent leurs outils et leurs habitudes.

1.9 L'apport de la supervision applicative aux équipes

1.9.1 Pour les équipes informatiques

Les équipes d'exploitation sont alertées sur les incidents qui affectent réellement les utilisateurs, avec les éléments nécessaires à leur qualification. Elles peuvent prioriser leurs interventions selon l'impact réel, plutôt que selon le nombre d'alarmes techniques. Les équipes de développement et de qualité vérifient après chaque mise en production que les parcours critiques fonctionnent toujours, et repèrent les dégradations de performance introduites par une nouvelle version. Les responsables d'infrastructure disposent d'indicateurs objectifs pour piloter les engagements de service, internes comme externes, et pour dialoguer avec leurs prestataires.

1.9.2 Pour les métiers

Les responsables métier obtiennent une mesure de la disponibilité exprimée dans leurs termes : « ce mois-ci, le parcours de commande a réussi dans 99,7 % des exécutions évaluables », plutôt que « le serveur web a été disponible ». Ce taux est une proportion d'exécutions, non une proportion du temps : il s'accompagne de la couverture de la mesure, c'est-à-dire de la part des exécutions attendues qui ont effectivement produit un résultat évaluable, car les périodes sans mesure et les changements de fréquence peuvent dissocier les deux grandeurs. Une disponibilité exprimée en temps, par exemple dans un engagement de niveau de service, suppose une méthode de calcul explicitement convenue avec les parties prenantes (§ 7.7.3). Les indicateurs d'activité (commandes, transactions, inscriptions) surveillés par la sonde les alertent sur des anomalies qu'eux seuls savent interpréter, et que la seule supervision technique ne révèle pas. La sonde fournit enfin un langage commun aux équipes informatiques et métier pour décrire la qualité de service.

1.9.3 Bonnes conditions de réussite

L'expérience montre que la valeur d'un projet de supervision applicative dépend moins de l'outil que de quelques choix d'organisation :

Condition	Recommandation
Choisir les bons parcours	Commencer par trois à cinq parcours critiques pour l'activité (connexion, recherche, commande, paiement en environnement de test), plutôt que de chercher à tout couvrir
Définir les attendus avec les métiers	Formaliser pour chaque parcours le résultat attendu et le délai acceptable
Disposer de comptes et de données de test	Prévoir des comptes dédiés, des articles de test et des environnements où les actions sont sans conséquence (§ 2.6 du guide d'installation et d'administration)
Organiser la réaction aux alertes	Désigner qui reçoit les alertes, à quelle heure, et quelle est la conduite à tenir
Maintenir les scénarios	Associer la mise à jour des scénarios au cycle de mise en production des applications (chapitre 6 du guide de développement des scénarios)

La suite de la documentation décrit comment la sonde met en œuvre ces principes : son architecture (chapitre 2), puis son installation et son utilisation.

Chapitre 2 — Éditions, points d'observation et fonctionnement autonome

2.1 Périmètre de la solution

2.1.1 Éditions

Les fonctions disponibles dépendent de l'édition acquise : SYNOVEX Essential, SYNOVEX Professional ou SYNOVEX Enterprise. Dans les tableaux de cette documentation, les éditions sont désignées par leur seul nom (Essential, Professional, Enterprise). Toutes les éditions partagent le même système. L'édition est inscrite sur la carte microSD lors de sa préparation, sous la forme d'un **certificat d'édition** qui active les modules correspondants : aucune clé n'est à saisir et aucune activation en ligne n'est nécessaire. Le droit d'utilisation du logiciel intégré est perpétuel : il ne nécessite ni abonnement ni renouvellement. Chaque version majeure bénéficie d'une période de support d'au moins cinq ans, dont la date de fin vous est indiquée à l'achat : pendant cette période, les correctifs de sécurité sont publiés pour toutes les sondes. Un contrat de maintenance facultatif donne en outre accès aux nouvelles versions fonctionnelles ; son échéance n'affecte aucune fonction acquise (§ 8.4.7 du guide d'installation et d'administration). Pour passer à une édition supérieure, vous remplacez la carte par une carte de cette édition et vous y transférez votre configuration (§ 8.6 du guide d'installation et d'administration), sous réserve que le Raspberry Pi 5 corresponde aux exigences de l'édition visée (chapitre 2 du guide d'installation et d'administration).

Fonction	Essential	Professional	Enterprise
Contrôles ICMP, TCP, DNS	Oui	Oui	Oui
Contrôles HTTP/HTTPS, y compris vérification du contenu et du certificat	Oui	Oui	Oui
Plugins compatibles Nagios fournis avec la sonde	Oui	Oui	Oui
Plugins compatibles Nagios personnalisés	—	Oui	Oui
Contrôles d'API (suites de requêtes, extraction de valeurs)	—	Oui	Oui
Contrôles de bases de données (PostgreSQL, MySQL/MariaDB, SQL Server, Oracle) et indicateurs métier	—	Oui	Oui
Scénarios navigateur déclaratifs et scriptés (Puppeteer, Chromium headless)	—	—	Oui
Interface d'administration, API REST, planification, plans d'exécution, historique local	Oui	Oui	Oui
Export OpenTelemetry (OTLP), endpoint Prometheus, plugin Nagios/Centreon	Oui	Oui	Oui

Aucune édition n'impose de limite technique au nombre de contrôles ou de scénarios navigateur. La capacité réelle dépend du matériel, de la fréquence et de la durée des contrôles : la méthode de dimensionnement (calcul de la charge, seuils à respecter, surveillance dans la durée) figure au chapitre 6 (§ 6.6), et le § 10.1 du guide d'installation et d'administration récapitule, sous forme d'aide-mémoire, les prérequis et la configuration matérielle minimale.

2.1.2 Offres matérielles

La sonde est proposée sous deux formes :

Offre	Contenu	Ce que vous fournissez
microSD préconfigurée	Carte microSD contenant le système et le logiciel de la sonde	Raspberry Pi 5 de capacité mémoire adaptée à l'édition, alimentation, refroidissement, boîtier, câble Ethernet
Boîtier complet	Raspberry Pi 5 assemblé dans son boîtier, refroidissement, carte microSD installée	Alimentation et câble Ethernet, sauf mention contraire de la liste de colisage jointe au boîtier et de votre bon de commande ; emplacement d'installation

Un SSD NVMe facultatif, monté sur la carte d'extension M.2 HAT+ officielle de Raspberry Pi, peut compléter la carte microSD comme volume de données. Il est surtout utile pour l'édition Enterprise ; la carte microSD reste le support principal et suffit à un usage standard (§ 8.3.4 du guide d'installation et d'administration).

Le contenu exact de chaque offre et les références de matériel sont détaillés au chapitre 2 du guide d'installation et d'administration.

> **Information** — SYNOVEX est conçu et documenté pour Raspberry Pi 5. Son architecture logicielle peut permettre une utilisation sur d'autres ordinateurs monocartes, notamment certains modèles Orange Pi. Toutefois, ces plateformes n'ont pas fait l'objet d'une validation officielle et leur fonctionnement n'est pas garanti. SYNOVEX est pris en charge exclusivement sur **Raspberry Pi 5** raccordé en **Ethernet** (§ 2.1.5 du guide d'installation et d'administration).

2.1.3 Ce que la sonde fait et ne fait pas

La sonde **observe depuis l'extérieur** le comportement des services, à la manière d'un utilisateur ou d'une application cliente. Elle n'installe aucun agent sur les serveurs supervisés et n'a pas besoin d'accès privilégié à ces serveurs. En contrepartie, elle ne mesure pas l'utilisation du processeur, de la mémoire ou des disques des serveurs : cette supervision des ressources reste du ressort de vos outils existants, que la sonde complète sans les remplacer (chapitre 1).

La sonde n'est pas non plus un outil de test de charge. Ses contrôles sont volontairement peu fréquents et légers, pour ne pas perturber les services observés. Elle n'est pas davantage un outil de mesure de l'expérience des utilisateurs réels : elle mesure des parcours représentatifs, reproduits à l'identique, ce qui la rend complémentaire des solutions de mesure en conditions réelles (§ 1.6.3).

2.1.4 Évolutions prévues

Les évolutions suivantes sont envisagées pour des versions ultérieures. Elles ne font pas partie des fonctions actuelles de la sonde : ne les prenez pas en compte pour préparer un déploiement.

- **Console centrale.** Une console centrale facultative permettra de gérer plusieurs sondes depuis un point unique. Elle sera hébergée par l'éditeur ou installée dans votre infrastructure, au choix. La sonde restera pleinement autonome : la console ne sera jamais nécessaire à son fonctionnement. L'adresse de la console hébergée par l'éditeur est réservée (`https://fleet-synovex.incitypark.fr`) ; la sonde ne l'utilise pas.
- **Connexion Wi-Fi.** La sonde se raccorde aujourd'hui exclusivement en Ethernet ; une connexion Wi-Fi est à l'étude.

2.2 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 (§ 2.2.1). 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.

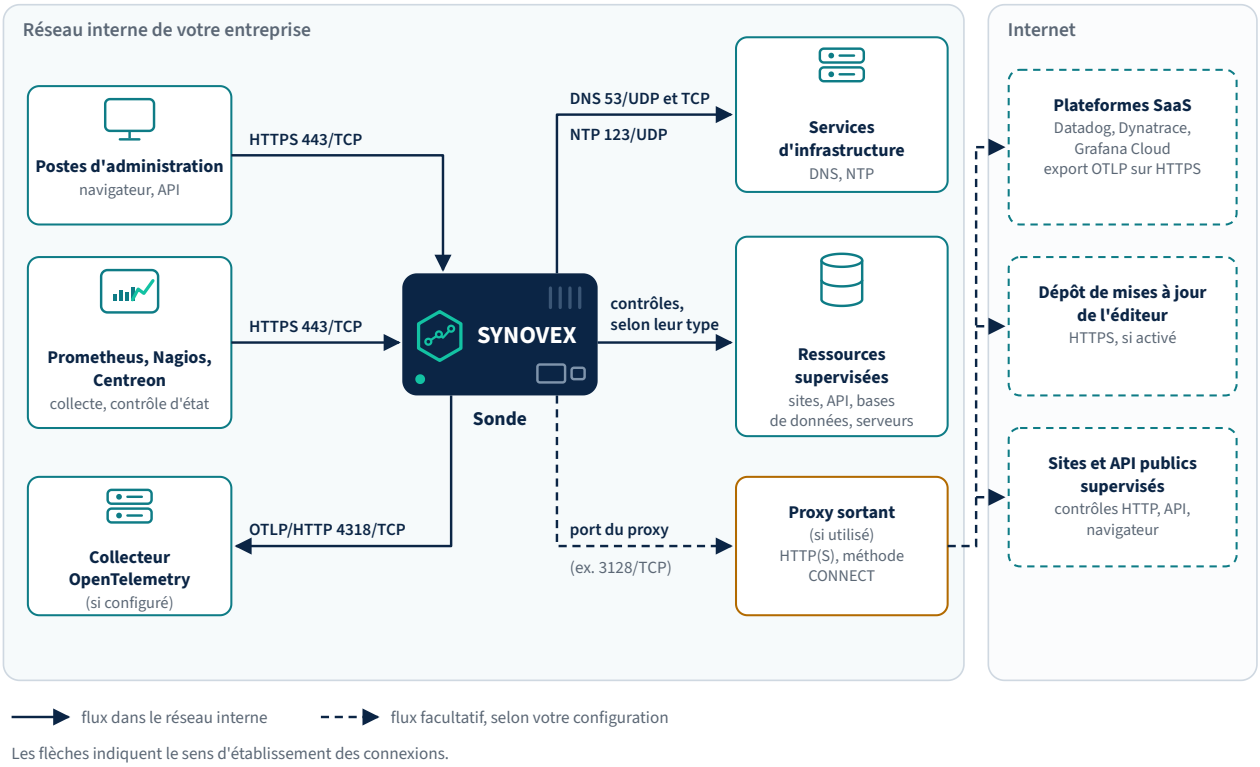


Figure 2-1 — Schéma d'implantation type : sonde dans le réseau interne, plateformes de supervision, accès Internet sortant via proxy

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.

2.2.1 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 du guide d'intégration et d'observabilité), 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 du guide d'intégration et d'observabilité) ;
- **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.

2.3 Cycle de vie d'une exécution

Toutes les exécutions suivent le même cycle, quel que soit le type de contrôle. L'exemple ci-dessous décrit celui d'un scénario navigateur, le plus complet.

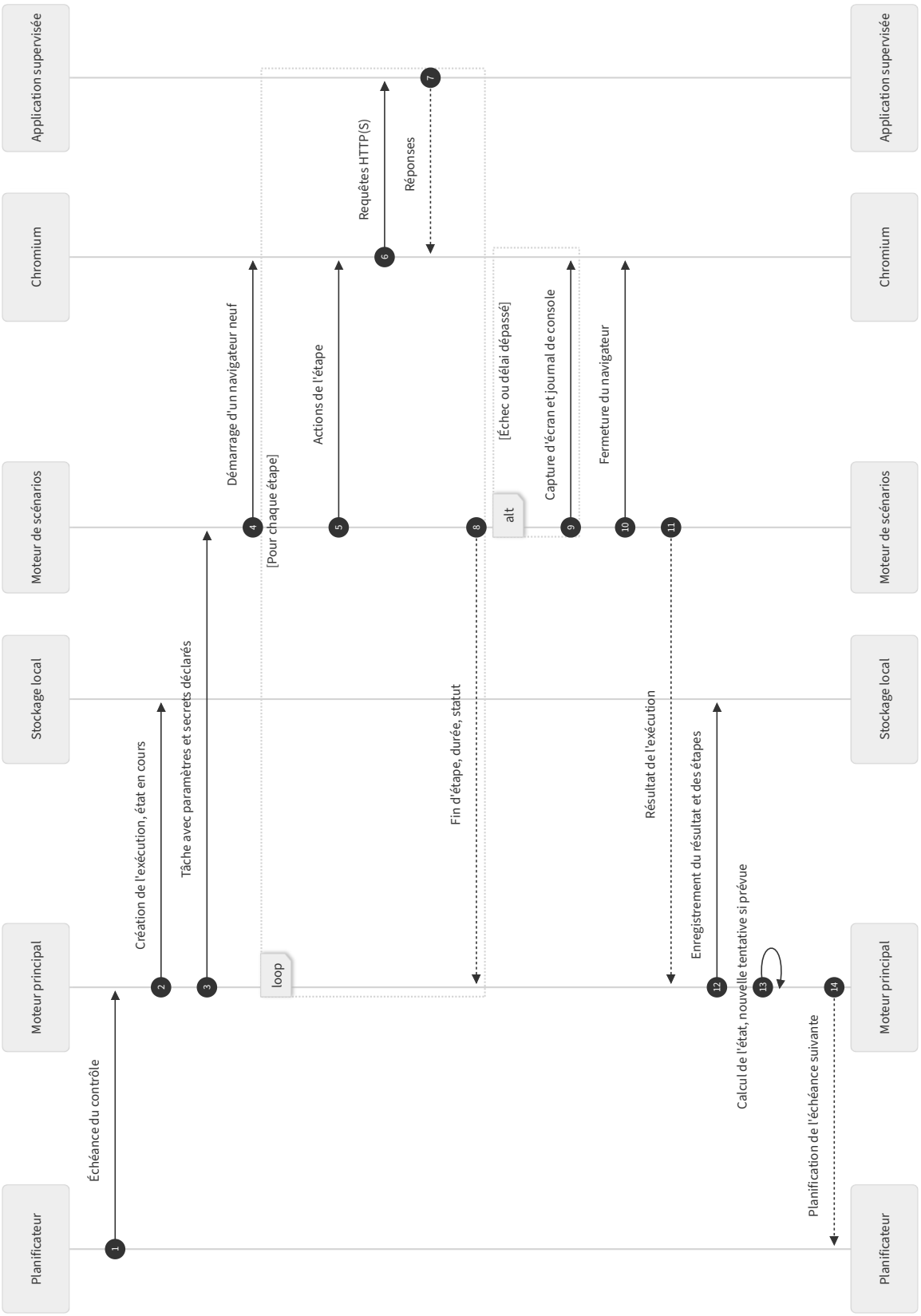


Schéma 2-1 — Cycle de vie d'une exécution (scénario navigateur)

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

Les étapes communes à tous les contrôles sont les suivantes :

1. **Déclenchement.** Le planificateur déclenche le contrôle à son échéance, ou un utilisateur le déclenche manuellement depuis l'interface ou l'API.
2. **Préparation.** Le moteur principal crée l'exécution, résout les paramètres et récupère les secrets déclarés par le contrôle.
3. **Exécution.** Le contrôle est exécuté par le composant approprié, dans la limite du délai maximal (*timeout*) configuré.
4. **Évaluation.** Le résultat est comparé aux critères du contrôle : assertions, seuils de durée, codes retour.
5. **Confirmation.** En cas d'échec, le contrôle peut être relancé selon les nouvelles tentatives configurées avant que l'état `CRITICAL` soit retenu.
6. **Enregistrement.** Le résultat, les étapes, les mesures et les éventuels artefacts sont enregistrés localement.
7. **Publication.** Les métriques, la trace de l'exécution et les journaux associés sont mis à disposition des destinations configurées.
8. **Replanification.** L'échéance suivante est calculée.

Le chapitre 3 détaille les notions d'état, de délai, de nouvelle tentative et de seuil utilisées à chaque étape.

Chapitre 3 — Principes de fonctionnement

Ce chapitre présente les notions communes à tous les contrôles exécutés par la sonde : comment un contrôle est défini, quand il s'exécute, comment son résultat est calculé et comment l'interpréter. Ces principes s'appliquent aussi bien à une simple vérification ICMP qu'à un parcours navigateur en plusieurs étapes. Les chapitres suivants s'appuient sur eux :

- le chapitre 4 détaille les contrôles réseau et techniques ;
- le chapitre 5, les contrôles applicatifs et métier ;
- le chapitre 6, la planification ;
- les chapitres 3 à 6 du guide de développement des scénarios, les scénarios Puppeteer.

Les exemples de ce chapitre utilisent des noms de domaine et des adresses fictifs, réservés à la documentation (`example.com`, `192.0.2.0/24`).

3.1 La notion de contrôle

3.1.1 Définition

Un **contrôle** est l'unité de base de la supervision. Il décrit une vérification que la sonde doit réaliser de façon répétée :

- **quoi** vérifier : le type de contrôle et sa cible ;
- **comment** juger le résultat : les assertions, les règles et les seuils ;
- **quand** l'exécuter : la planification ;
- **dans quelles limites** : le délai maximal (timeout) et les nouvelles tentatives.

Chaque lancement d'un contrôle produit une **exécution**. Une exécution a un début, une durée, un **état** (`OK`, `WARNING`, `CRITICAL`, `UNKNOWN`) et, le cas échéant, une description de l'erreur rencontrée.

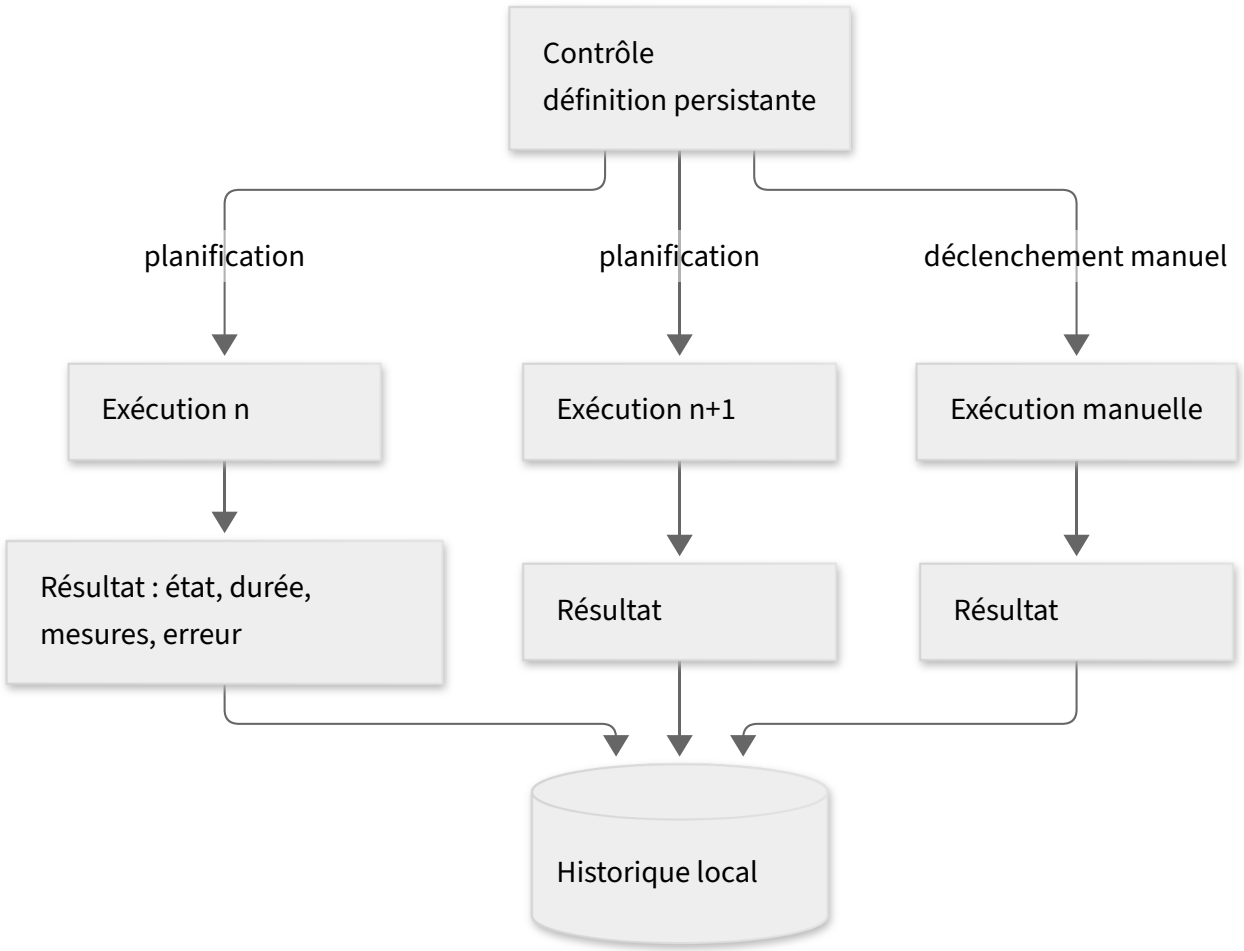


Schéma 3-1 — Un contrôle et ses exécutions

Les contrôles sont stockés **sur la sonde elle-même**. Ils continuent d'être exécutés quand aucune plateforme externe n'est joignable (§ 3.14).

3.1.2 Types de contrôles

La sonde prend en charge huit types de contrôles. Leur disponibilité dépend de l'édition acquise.

Type	Vérifie	Édition minimale	Chapitre
`icmp`	La joignabilité d'un équipement (ping), la latence et la perte de paquets	Essential	chapitre 2 du guide d'installation et d'administration
`tcp`	L'ouverture d'un port TCP et, en option, un échange simple	Essential	chapitre 2 du guide d'installation et d'administration
`dns`	La résolution d'un nom et la conformité de la réponse	Essential	chapitre 2 du guide d'installation et d'administration
`http`	Une requête HTTP ou HTTPS : code de retour, en-têtes, contenu, durée, certificat	Essential	chapitre 2 du guide d'installation et d'administration

Type	Vérifie	Édition minimale	Chapitre
`nagios`	L'exécution d'un plugin compatible Nagios	Essential (plugins fournis), Professional (plugins personnalisés)	chapitre 2 du guide d'installation et d'administration
`api`	Une suite de requêtes HTTP enchaînées, avec réutilisation de valeurs d'une réponse à l'autre	Professional	chapitre 3 du guide d'installation et d'administration
`database`	Le résultat d'une requête ou d'une procédure SQL au regard de règles	Professional	chapitre 3 du guide d'installation et d'administration
`browser`	Un parcours utilisateur dans un navigateur Chromium sans interface (headless)	Enterprise	chapitres 1 à 4 du guide d'installation et d'administration du guide de développement des scénarios

3.1.3 Structure commune

Quel que soit son type, un contrôle partage un ensemble de propriétés communes. Il les complète par une section propre à son type (`http`, `database`, `browser`…).

Propriété	Rôle	Exemple
`id`	Identifiant stable, choisi par vous. Il apparaît dans les résultats, l'API et la télémétrie	`portail-accueil`
`name`	Libellé lisible	`Portail client : page d'accueil`
`type`	Type de contrôle (§ 3.1.2)	`http`
`enabled`	Contrôle actif ou suspendu	`true`
`tags`	Étiquettes libres (8 au maximum) servant au filtrage et au regroupement	`{ "application": "portail" }`
`schedule`	Planification (§ 3.3 et chapitre 6)	`{ "interval": "1m" }`
`timeout`	Durée maximale d'une exécution (§ 3.4)	`15s`
`retry`	Nouvelles tentatives avant de confirmer un échec (§ 3.5)	`{ "attempts": 1, "interval": "20s" }`
`thresholds`	Seuils de performance (§ 3.6)	`{ "warningDuration": "3s" }`
`secrets`	Liste des secrets que le contrôle est autorisé à utiliser (chapitre 6 du guide d'installation et d'administration)	`["API_TOKEN"]`
`ipFamily`	Famille d'adresses utilisée pour joindre la cible : `auto` (valeur par défaut), `ipv4` ou `ipv6` (voir ci-dessous)	`auto`

Le paramètre `ipFamily` détermine la version du protocole IP employée pour joindre la cible :

Valeur	Comportement
`auto` (défaut)	IPv6 si l'IPv6 est activé sur la sonde (chapitre 5 du guide d'installation et d'administration) **et** si la cible possède une adresse AAAA ; IPv4 dans tous les autres cas
`ipv4`	IPv4 uniquement
`ipv6`	IPv6 uniquement. L'IPv6 doit être activé sur la sonde

> **Information** — La sonde fonctionne toujours en IPv4. L'IPv6 est facultatif et **désactivé par défaut** : il s'active dans la section `network` des paramètres (objet `ipv6`, chapitre 5 du guide d'installation et d'administration). Le comportement de chaque type de contrôle vis-à-vis d'IPv6 est précisé aux chapitres 4 et 5 (§ 4.1.3).

L'identifiant doit commencer par une lettre minuscule ou un chiffre et ne contenir que des lettres minuscules, des chiffres et des tirets (2 à 63 caractères). Choisissez-le avec soin : il sert de clé dans l'historique, les tableaux de bord et les alertes. Le renommer revient à créer un nouveau contrôle du point de vue des outils d'observabilité.

Les durées s'écrivent sous la forme d'un nombre suivi d'une unité : `500ms`, `30s`, `5m`, `1h`.

> **Conseil** — Adoptez une convention de nommage dès le premier contrôle, par exemple `<application>-<fonction>` (`boutique-connexion`, `boutique-panier`, `erp-api-stock`). Les recherches, les filtres et les alertes en seront grandement simplifiés.

3.1.4 Exemple complet

L'exemple suivant, fictif, vérifie toutes les minutes que la page d'accueil d'un portail répond avec le code HTTP 200 et contient le texte attendu :

```json
{
  "id": "portail-accueil",
  "name": "Portail client : page d'accueil",
  "type": "http",
  "enabled": true,
  "tags": { "application": "portail", "environment": "production" },
  "schedule": { "interval": "1m", "jitter": "10s" },
  "timeout": "15s",
  "retry": { "attempts": 1, "interval": "20s" },
  "thresholds": { "warningDuration": "3s", "criticalDuration": "10s" },
  "http": {
    "method": "GET",
    "url": "https://portail.example.com/",
    "assertions": [
      { "kind": "status", "operator": "eq", "value": 200 },
      { "kind": "body", "operator": "contains", "value": "Espace client" }
    ]
  }
}
```

Le chapitre 6 du guide d'installation et d'administration explique comment créer ce contrôle par l'interface d'administration ou par l'API, et le chapitre 4 détaille chacune des options du type `http`.

3.2 Les cibles supervisées

3.2.1 Nature des cibles

La **cible** d'un contrôle est la ressource vérifiée : une adresse IP, un nom d'hôte, un port, une URL, un nom DNS, une base de données ou une application web. La sonde exécute ses contrôles **depuis le réseau où elle est installée**. Elle observe donc les ressources telles qu'un utilisateur ou un système situé au même endroit les perçoit, avec le même routage, les mêmes règles de pare-feu, le même serveur proxy et les mêmes serveurs DNS.

Cette position est déterminante pour l'interprétation des résultats :

- une ressource interne (`erp.example.internal`) n'est vérifiable que si la sonde est raccordée à un réseau qui y donne accès ;
- une ressource publique (`www.example.com`) est vérifiée **telle qu'elle est vue depuis votre site**, y compris à travers votre accès Internet ;
- deux sondes placées sur deux sites différents peuvent observer des résultats différents pour une même cible. C'est précisément ce qui permet de distinguer une panne générale d'un problème local.

3.2.2 Accès réseau aux cibles

Pour chaque cible, vérifiez avant de créer le contrôle que :

1. le nom de la cible est résolu par les serveurs DNS configurés sur la sonde (chapitre 5 du guide d'installation et d'administration) ;
2. les flux nécessaires sont autorisés depuis l'adresse de la sonde vers la cible (port, protocole) ;
3. un passage par le proxy est configuré si la cible n'est joignable que par lui (chapitre 5 du guide d'installation et d'administration) ;
4. les certificats présentés par la cible sont reconnus. Si elle utilise un certificat émis par une autorité de certification interne, importez cette autorité (chapitre 5 du guide d'installation et d'administration).

> ! **Attention** — Une cible inaccessible pour des raisons de configuration réseau (flux non ouvert, proxy absent) produit des échecs qui ne reflètent pas l'état réel de l'application. Validez chaque nouveau contrôle par une exécution d'essai (§ 6.6.5 du guide d'installation et d'administration) avant de l'activer.

3.2.3 Respect des cibles

La sonde génère du trafic réel vers les cibles. Ce trafic doit rester raisonnable et prévu :

- informez les équipes responsables des applications vérifiées, en particulier lorsqu'un contrôle utilise un compte de test ou s'authentifie ;
- évitez des fréquences inutilement élevées (§ 3.3.3) ;
- certaines cibles disposent de protections anti-robot ou de limitations de débit. Un contrôle trop fréquent peut les déclencher et faire bloquer l'adresse de la sonde ;
- les contrôles qui modifient des données (création de panier, commande de test) doivent viser un environnement de test ou des données de test identifiées (chapitre 5 du guide de développement des scénarios).

3.3 La fréquence d'exécution

3.3.1 Intervalle ou expression cron

La planification d'un contrôle s'exprime de l'une des deux façons suivantes, **jamais les deux à la fois** :

Mode	Syntaxe	Comportement	Usage typique
Intervalle	`{ "interval": "5m" }`	Une exécution toutes les 5 minutes, de façon continue	Disponibilité, temps de réponse
Expression cron	`{ "cron": "*/15 8-20 * * 1-5", "timezone": "Europe/Paris" }`	Exécutions aux instants décrits par l'expression, dans le fuseau horaire indiqué	Contrôles liés à des horaires (heures ouvrées, traitements nocturnes)

Un contrôle peut aussi n'avoir **aucune planification propre** (`"schedule": null`). Il n'est alors exécuté que manuellement ou au sein d'un plan d'exécution (chapitre 6).

Le chapitre 6 détaille la syntaxe des expressions cron, la gestion des fuseaux horaires et le dimensionnement de la charge.

3.3.2 La gigue

La **gigue** (`jitter`) est un décalage aléatoire, compris entre zéro et la valeur indiquée, appliqué au démarrage de chaque exécution. Elle évite que tous les contrôles ayant la même fréquence démarrent exactement au même instant, ce qui provoquerait des pics de charge sur la sonde comme sur les cibles. Exemple : `"jitter": "10s"` sur un intervalle d'une minute.

La gigue est particulièrement utile après un redémarrage de la sonde : sans elle, tous les contrôles seraient lancés simultanément (§ 3.14.2).

3.3.3 Choisir une fréquence

La fréquence fixe le **délai maximal de détection** d'un incident. Avec un intervalle de 5 minutes et une nouvelle tentative, un incident survenu juste après une exécution réussie est confirmé au plus tard après environ 5 minutes, plus l'intervalle de nouvelle tentative et la durée des exécutions.

Type de contrôle	Fréquence courante	Justification
`icmp`, `tcp`, `dns`	1 minute	Contrôles très légers ; détection rapide
`http`	1 à 5 minutes	Bon compromis entre réactivité et charge
`nagios`	Selon le plugin, souvent 1 à 5 minutes	Dépend du coût du plugin
`api`	2 à 10 minutes	Plusieurs requêtes par exécution
`database`	5 à 15 minutes, ou cron	Requêtes parfois coûteuses ; indicateurs métier à rythme horaire
`browser`	5 à 15 minutes	Exécution coûteuse en mémoire et en temps ; une seule à la fois par défaut

> **Information** — Une fréquence plus élevée améliore la réactivité, mais augmente le volume de données conservées (§ 3.12), la charge de la sonde et le trafic vers les cibles. Pour un parcours navigateur, une exécution toutes les 5 minutes représente déjà 288 parcours par jour.

3.4 Le délai maximal d'exécution (timeout)

Le **timeout** est la durée au-delà de laquelle une exécution est interrompue et considérée en échec. Il protège la sonde contre les exécutions bloquées et garantit que chaque contrôle produit un résultat dans un délai connu.

Quand le timeout est atteint :

1. la sonde interrompt l'exécution. Pour un plugin ou un scénario navigateur, elle met fin à l'ensemble des processus créés, y compris les processus secondaires de Chromium ;
2. l'exécution est enregistrée avec l'état `CRITICAL` et le type d'erreur `timeout` ;
3. pour un scénario navigateur, une capture d'écran de l'état de la page est produite lorsque c'est encore possible (chapitre 7).

Règles de réglage :

Règle	Explication
Le timeout doit dépasser nettement la durée normale	Prévoyez au moins deux à trois fois la durée habituelle, pour absorber les variations normales sans fausse alerte
Il doit rester inférieur à l'intervalle	Une exécution qui dure plus longtemps que l'intervalle retarde les suivantes (§ 6.6)
Il doit tenir compte de la plateforme	Le Raspberry Pi 5 est moins rapide qu'un poste de travail récent, notamment pour le lancement de Chromium et le rendu des pages
Les seuils de performance sont distincts	Le timeout signale une exécution qui n'aboutit pas ; la lenteur se signale avec `thresholds` (§ 3.6)

Type	Timeout indicatif
`icmp`, `tcp`, `dns`	5 à 10 s
`http`	10 à 30 s
`api`	30 à 60 s
`database`	30 à 60 s
`nagios`	10 à 60 s, selon le plugin
`browser`	60 à 180 s, selon le nombre d'étapes

Ces valeurs sont des points de départ. Ajustez-les après quelques jours d'observation des durées réelles (chapitre 7).

3.5 Nouvelles tentatives et confirmation d'un incident

3.5.1 Principe

Un échec isolé n'est pas toujours le signe d'un incident : une perte de paquet, un délai réseau ponctuel ou un redémarrage applicatif de quelques secondes peuvent faire échouer une exécution unique. Pour éviter les fausses alertes, la sonde peut **confirmer un échec** par une ou plusieurs nouvelles tentatives avant de le publier.

```json
"retry": { "attempts": 2, "interval": "30s" }
```

Avec ce réglage, un échec déclenche jusqu'à deux nouvelles exécutions, espacées de 30 secondes :

- si l'une d'elles réussit, l'échec n'est pas publié ;
- si toutes échouent, l'échec est confirmé et l'état `CRITICAL` est publié.

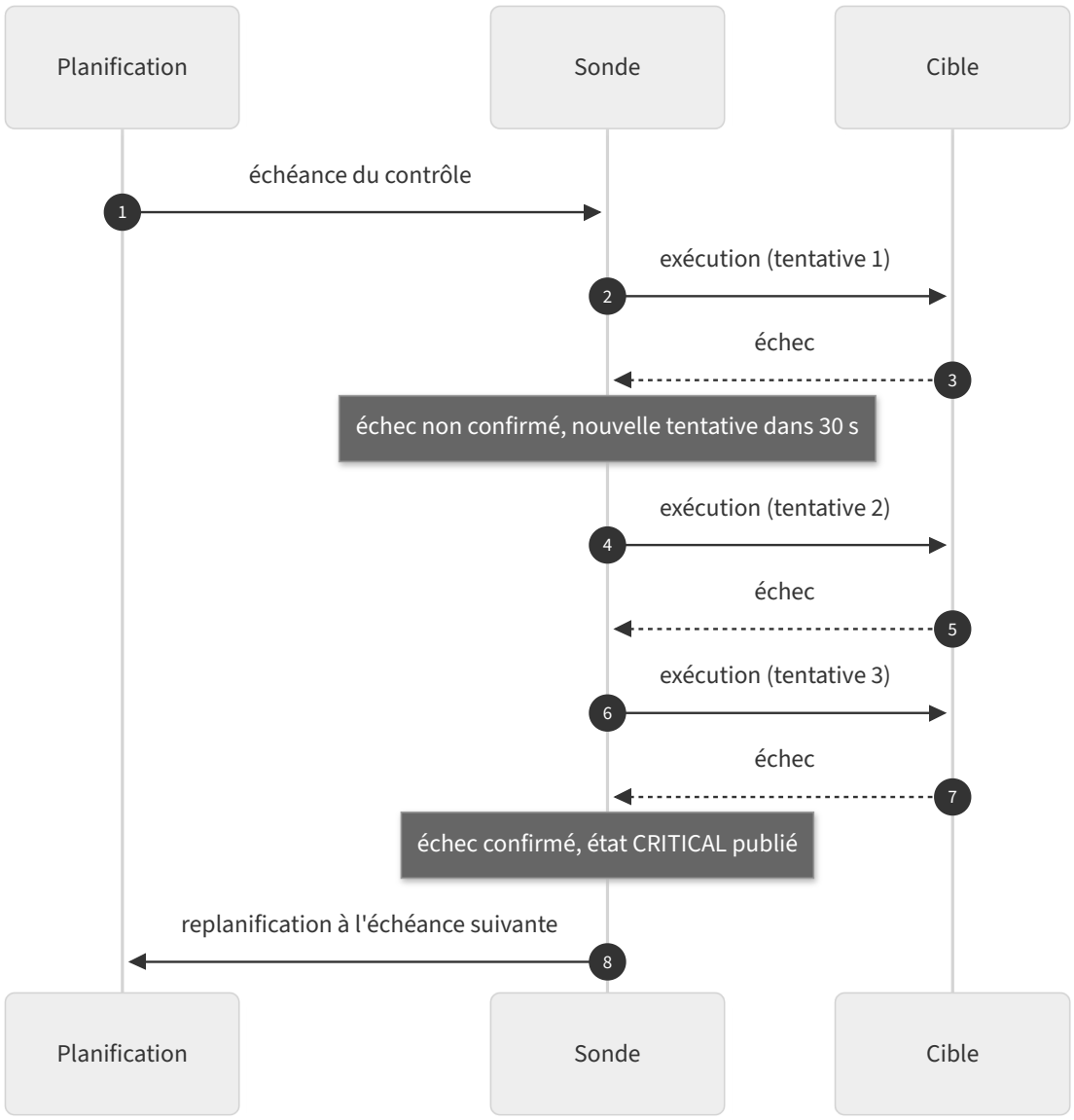


Schéma 3-2 — Nouvelles tentatives avant confirmation d'un échec

3.5.2 Valeurs et effets

Propriété	Valeurs	Défaut
`attempts`	0 à 5 nouvelles tentatives	0 (aucune)
`interval`	Durée entre deux tentatives	`15s`

Chaque tentative est enregistrée comme une exécution à part entière, avec son numéro de tentative (`attempt`). Elles restent toutes consultables dans l'historique, ce qui permet de repérer les instabilités intermittentes même lorsqu'elles n'ont pas abouti à une alerte.

> ✓ **Conseil** — Une nouvelle tentative suffit dans la plupart des cas. Au-delà de deux, le délai de détection s'allonge sensiblement pour un bénéfice faible. Pour les scénarios navigateur, tenez compte de la durée du scénario : trois tentatives d'un parcours de 60 secondes retardent la confirmation de plusieurs minutes et occupent le moteur de scénarios pendant ce temps.

Budget d'exécution. Dans le pire des cas, toutes les tentatives atteignent le timeout. Une exécution complète, nouvelles tentatives comprises, peut alors durer :

`(attempts + 1) × timeout + attempts × interval`

Par exemple, `"timeout": "90s"` et `"retry": { "attempts": 2, "interval": "30s" }` donnent 3 × 90 + 2 × 30 = 330 secondes, soit 5 min 30 s. Ce budget doit rester inférieur à l'intervalle de planification du contrôle, ou à la durée maximale du plan qui le contient ; sinon, des échéances sont sautées (§ 6.2.1 et § 6.6).

Parcours à effet de bord et idempotence. Une opération est **idempotente** lorsque la répéter produit le même effet que l'exécuter une seule fois : une consultation l'est, une création de commande ne l'est pas. Une nouvelle tentative **rejoue l'intégralité** du contrôle. Or une exécution interrompue de façon ambiguë (timeout, coupure réseau, interruption par l'activation d'une mise à jour avec le type d'erreur `maintenance`, § 3.14.2) a pu produire son effet sur l'application avant d'être interrompue : la sonde ne peut pas le savoir. La nouvelle tentative, ou l'exécution suivante, peut alors **rejouer cet effet** : deuxième commande de test, deuxième réservation, deuxième message.

Pour un scénario transactionnel, même en environnement de recette :

Recommandation	Raison
Fixer `"retry": { "attempts": 0 }`	Aucune répétition automatique d'une opération qui n'est pas idempotente
À défaut, rendre l'opération idempotente côté application, avec une **clé d'opération** préfixée pour signaler un objet de test et **réutilisée telle quelle à chaque répétition de la même opération logique** (même échéance de planification, nouvelles tentatives comprises), l'application refusant une seconde création avec la même clé	L'application ne reconnaît un doublon que si la clé est identique d'une tentative à l'autre : une référence tirée au hasard ou horodatée **à chaque exécution** change à chaque tentative et ne déduplique rien ; elle rend seulement chaque objet créé identifiable et rattachable à l'exécution qui l'a produit
Confier le **nettoyage** des objets de test à un mécanisme propre à l'environnement de recette (tâche planifiée de purge des objets marqués comme objets de test, réinitialisation périodique), indépendant de l'exécution suivante du scénario	Une étape de nettoyage placée à la fin du scénario ne s'exécute pas si celui-ci est interrompu avant son terme
Ne jamais exécuter un tel parcours en production sans l'accord du responsable de l'application	Voir § 5.9.1

Le § 5.8 du guide de développement des scénarios présente un parcours de commande en environnement de test, auquel ces règles s'appliquent.

3.5.3 Bagotage

On parle de **bagotage** (*flapping*) lorsqu'un contrôle alterne rapidement entre réussite et échec. C'est souvent le symptôme d'un problème réel mais intermittent : saturation, équilibrage de charge vers un serveur défaillant, instabilité réseau. Les nouvelles tentatives masquent une partie de ces alternances. L'historique des exécutions et des tentatives permet de les mettre en évidence (chapitre 7).

3.6 Les seuils de performance

Une exécution peut réussir fonctionnellement tout en étant anormalement lente. Les **seuils de performance** (`thresholds`) permettent de qualifier cette lenteur :

Seuil	Effet si la durée totale de l'exécution le dépasse
`warningDuration`	L'exécution réussie passe à l'état `WARNING`
`criticalDuration`	L'exécution réussie passe à l'état `CRITICAL`

Exemple : avec `"warningDuration": "3s"` et `"criticalDuration": "10s"`, une page qui répond correctement en 4 secondes produit l'état `WARNING`, et en 12 secondes l'état `CRITICAL`. Au-delà du timeout, l'exécution est interrompue (§ 3.4).

Pour un contrôle `http` ou `api`, une assertion de type `duration` permet également de fixer une limite sur la durée d'une requête particulière (chapitres 4 et 5). Pour un scénario navigateur, la durée de chaque étape est mesurée et consultable, ce qui permet d'identifier l'étape responsable d'une lenteur (chapitre 7).

> ✓ **Conseil** — Fixez les seuils à partir des durées réellement observées sur une période représentative (au moins une semaine, incluant les heures de pointe), et non à partir d'objectifs théoriques. Un seuil trop bas génère des alertes `WARNING` permanentes qui finissent par être ignorées.

3.7 Le résultat d'une exécution

3.7.1 Contenu d'un résultat

Chaque exécution produit un résultat normalisé, identique dans sa structure pour tous les types de contrôles. Il est stocké sur la sonde, consultable dans l'interface d'administration et par l'API (`GET /api/v1/runs/{runId}`), et transmis aux destinations d'export configurées (chapitre 2 du guide d'intégration et d'observabilité).

Élément	Contenu	Exemple
Identifiant d'exécution	Identifiant unique, triable chronologiquement (ULID)	`01M7SHD1K2X7Q9N3V4T6W8Y0ZA`
Contrôle et révision	Identifiant du contrôle et numéro de version de sa configuration au moment de l'exécution	`boutique-connexion`, révision 7
Déclenchement	`schedule` (planification), `plan` (plan d'exécution), `manual`, `retry` (nouvelle tentative), `dryRun` (essai)	`retry`
Tentative	Numéro de tentative (§ 3.5)	2
Début et durée	Horodatage UTC et durée en millisecondes	`2026-11-19T09:15:02Z`, 14 650 ms
État	`OK`, `WARNING`, `CRITICAL` ou `UNKNOWN`	`CRITICAL`
Erreur	Type d'erreur normalisé (§ 3.8.5), étape concernée, message explicatif	`assertion_failed`, étape « Vérifier l'accès à l'espace client »
Étapes	Pour les scénarios et suites d'API : nom, état et durée de chaque étape	5 étapes
Mesures	Valeurs propres au type : code HTTP, phases de la requête, latence ICMP, valeur d'un indicateur métier, données de performance d'un plugin	`http.status_code = 200`

Élément	Contenu	Exemple
Sortie	Texte produit par un plugin Nagios	`HTTP OK: HTTP/1.1 200 OK - 512 bytes`
Artefacts	Fichiers de diagnostic : capture d'écran, journal de la console du navigateur, extrait réseau	`failure.png`
Trace	Identifiant de trace OpenTelemetry, pour retrouver l'exécution dans une plateforme d'observabilité	`4bf92f35…`

3.7.2 Lecture d'un résultat en échec

Le résultat d'un échec répond à trois questions, dans cet ordre :

1. **Quoi ?** L'état et le type d'erreur indiquent la nature du problème (délai dépassé, élément introuvable, code HTTP inattendu…).
2. **Où ?** L'étape en échec situe le problème dans le parcours.
3. **Pourquoi ?** Le message d'erreur, la capture d'écran et le journal de la console fournissent les éléments de diagnostic.

Exemple d'un résultat en échec, au format de l'API (fictif). Il s'agit de l'exécution du scénario déclaratif `boutique-connexion` (§ 4.3.1 du guide de développement des scénarios), repris à l'identique au chapitre 7 (§ 7.4) :

JSON

```
{
  "id": "01M7SHD1K2X7Q9N3V4T6W8Y0ZA",
  "checkId": "boutique-connexion",
  "checkRevision": 7,
  "planRunId": null,
  "trigger": "retry",
  "status": "CRITICAL",
  "attempt": 2,
  "startedAt": "2026-11-19T09:15:02.114Z",
  "durationMs": 14650,
  "error": {
    "type": "assertion_failed",
    "step": "Vérifier l'accès à l'espace client",
    "message": "Élément [data-testid=account-menu] non visible après 10000 ms"
  },
  "steps": [
    { "index": 0, "name": "Ouvrir la page de connexion", "status": "OK", "startedOffsetMs": 50,
    "durationMs": 1830 },
    { "index": 1, "name": "Saisir l'identifiant", "status": "OK", "startedOffsetMs": 1880,
    "durationMs": 160 },
    { "index": 2, "name": "Saisir le mot de passe", "status": "OK", "startedOffsetMs": 2040,
    "durationMs": 140 },
    { "index": 3, "name": "Valider le formulaire", "status": "OK", "startedOffsetMs": 2180,
    "durationMs": 2410 },
    { "index": 4, "name": "Vérifier l'accès à l'espace client", "status": "CRITICAL",
    "startedOffsetMs": 4590, "durationMs": 10004 }
  ],
  "measurements": { "browser.launch_ms": 1180 },
  "artifacts": [
    { "name": "failure.png", "kind": "screenshot", "bytes": 182344 },
    { "name": "console.json", "kind": "console", "bytes": 2210 }
  ],
  "trace": { "traceId": "4bf92f3577b34da6a3ce929d0e0e4736" }
}
```

Ce résultat se lit ainsi : il s'agit de la deuxième tentative (`attempt` vaut 2, `trigger` vaut `retry`), l'échec est donc confirmé. La page de connexion s'est ouverte, l'authentification a été soumise, mais l'espace client n'est pas apparu dans le délai de 10 secondes de l'étape, d'où la durée de cette étape et la durée totale de l'exécution. L'incident est donc probablement fonctionnel (authentification refusée ou page d'accueil du compte en erreur) plutôt que réseau. La capture d'écran le confirmera. Le chapitre 7 détaille la consultation et l'interprétation des résultats.

> **Information** — Les messages d'erreur sont produits avec **masquage des secrets** : si la valeur d'un secret apparaît dans un message, un journal ou une sortie de plugin, elle est remplacée avant tout enregistrement.

3.8 Les états et les erreurs

3.8.1 Les états

La sonde utilise cinq états. Ils reprennent la sémantique des plugins Nagios, familière aux équipes d'exploitation, et l'état `STALE`, qui signale l'absence de résultat récent.

État	Signification	Origine
`OK`	Le contrôle a réussi et respecte les seuils	Toutes les assertions et règles sont satisfaites, durée inférieure à `warningDuration` ; plugin retournant le code 0
`WARNING`	Le contrôle a réussi, mais avec une dégradation	Durée supérieure à `warningDuration` ; règle de sévérité `warning` non satisfaite (chapitre 5) ; certificat proche de l'expiration (chapitre 4) ; plugin retournant le code 1
`CRITICAL`	Le contrôle a échoué : la cible ne fonctionne pas comme attendu	Assertion ou règle non satisfaite, erreur réseau, timeout, durée supérieure à `criticalDuration` ; plugin retournant le code 2
`UNKNOWN`	Le contrôle n'a pas pu être évalué	Problème de la sonde ou de la configuration, et non de la cible : défaillance de Chromium, plugin introuvable, ressources insuffisantes, redémarrage pendant l'exécution, exécution interrompue par le drainage d'une activation de mise à jour (type d'erreur `maintenance`) ; plugin retournant le code 3
`STALE`	Aucun résultat récent n'est disponible	Contrôle actif sans exécution terminée depuis plus de **deux intervalles** de planification. Ce calcul est suspendu pendant l'état de maintenance de la sonde et pendant un intervalle après la reprise (§ 3.14.2)

Les quatre premiers états qualifient **une exécution**. `STALE` qualifie **un contrôle** : il est calculé à partir de l'absence d'exécution récente, et non produit par une exécution.

> **Attention** — `UNKNOWN` ne doit pas être traité comme un état anodin. Il signifie que la supervision ne fonctionne pas pour ce contrôle : un incident réel sur la cible pourrait passer inaperçu. Un `UNKNOWN` persistant appelle une intervention sur la sonde ou sur la configuration du contrôle (chapitre 8).

3.8.2 Règles de calcul de l'état

L'état d'une exécution est déterminé dans l'ordre suivant :

1. si l'exécution n'a pas pu être menée à son terme pour une raison propre à la sonde (moteur de scénarios indisponible, navigateur défaillant, mémoire insuffisante, redémarrage, interruption par le drainage d'une activation de mise à jour), l'état est `UNKNOWN` ;
2. sinon, si le timeout est atteint, si une erreur réseau ou protocolaire survient, si une assertion ou une règle de sévérité critique n'est pas satisfaite, ou si le code d'un scénario scripté lève une erreur (`script_error`), l'état est

`CRITICAL` ;

3. sinon, si la durée dépasse `criticalDuration`, l'état est `CRITICAL` ;
4. sinon, si la durée dépasse `warningDuration`, ou si une règle de sévérité `warning` n'est pas satisfaite, l'état est `WARNING` ;
5. sinon, l'état est `OK`.

Pour un contrôle `nagios`, le code de retour du plugin fixe directement l'état (0 → `OK`, 1 → `WARNING`, 2 → `CRITICAL`, 3 → `UNKNOWN`). Les seuils de durée de la sonde s'y ajoutent. Un code de retour en dehors de 0 à 3 est traité comme `UNKNOWN`, avec le type d'erreur `plugin_error`.

Une fois les nouvelles tentatives épuisées (§ 3.5), l'état de la dernière tentative devient **l'état courant du contrôle**. C'est cet état qui est publié vers les plateformes d'observabilité et renvoyé par `GET /api/v1/checks/{id}/status`.

3.8.3 Cycle de vie de l'état d'un contrôle

Outre les cinq états définis au § 3.8.1, le diagramme suivant fait apparaître deux **phases du cycle de vie** d'un contrôle, qui ne sont pas des états de résultat :

- **Inactif** : le contrôle est désactivé (`"enabled": false`). Il n'est pas planifié et n'est pas soumis au calcul de `STALE`, réservé aux contrôles actifs ; une exécution d'essai reste possible (§ 6.6.5 du guide d'installation et d'administration) ;
- **En attente** : le contrôle est actif, mais aucune de ses exécutions n'est encore terminée, par exemple juste après sa création ou son activation. Dans la réponse de `GET /api/v1/checks/{id}/status`, le champ `lastRun` vaut alors `null`.

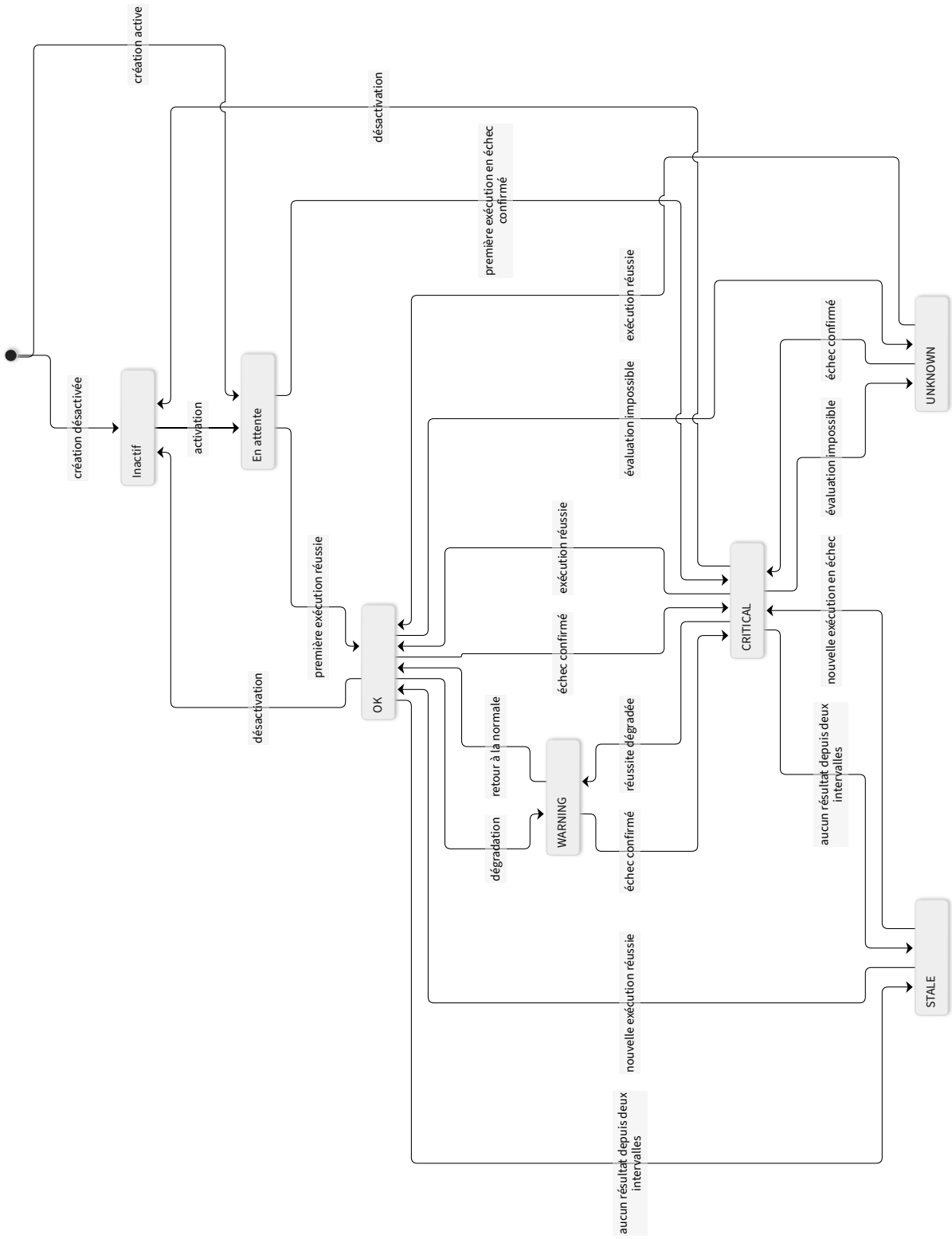


Schéma 3-3 — Cycle de vie de l'état d'un contrôle

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

Pour la lisibilité, le diagramme ne représente pas toutes les transitions : tout état peut évoluer vers n'importe quel autre selon le résultat de l'exécution suivante, et tout contrôle peut être désactivé à tout moment. Chaque changement d'état d'un contrôle produit un événement `check.status.changed`, consultable par l'API (§ 6.11 du guide d'installation et d'administration) et exporté sous forme de log (chapitre 2 du guide d'intégration et d'observabilité).

3.8.4 Le cas de l'état STALE

`STALE` peut apparaître à deux niveaux.

Niveau	Calculé par	Causes typiques
Contrôle	La sonde	File d'attente saturée (trop de contrôles pour les ressources disponibles), plan d'exécution bloquant les contrôles qu'il contient, moteur de scénarios arrêté
Sonde entière	La plateforme d'observabilité, par une alerte sur absence de signal	Sonde éteinte, débranchée, en panne, ou privée de réseau ; redémarrage lors de l'activation d'une mise à jour

La sonde ne peut évidemment pas signaler elle-même sa propre disparition. La détection d'une sonde muette repose donc sur une alerte configurée dans la plateforme d'observabilité : le chapitre 2 du guide d'intégration et d'observabilité fournit les règles correspondantes pour chaque plateforme.

État de maintenance de la sonde. Outre son fonctionnement normal, la sonde peut se trouver en **état de maintenance** : elle y entre au début de l'activation d'une mise à jour (§ 3.14.2) et en sort après son redémarrage sur la nouvelle version. Cet état concerne la sonde entière, et non un contrôle. Il est exposé par `/health`, qui répond `{"status": "MAINTENANCE"}` avec le code HTTP 503, par la métrique `sva.agent.maintenance` (Prometheus : `sva_agent_maintenance`), par les événements OTLP `maintenance.started` et `maintenance.ended`, et par `check_sva`, qui renvoie le code 3 (`UNKNOWN`) avec le texte `MAINTENANCE` (chapitre 7 et chapitre 2 du guide d'intégration et d'observabilité). Pendant la maintenance et pendant un intervalle après la reprise, la sonde **suspend le calcul de `STALE`** pour ses contrôles : l'absence d'exécution due à l'activation ne produit pas d'état `STALE`. La durée de cet intervalle est un intervalle de planification de chaque contrôle après la reprise.

3.8.5 Les types d'erreur

Lorsqu'une exécution échoue (hors simple dépassement d'un seuil de durée ou de temps de réponse), le résultat précise un **type d'erreur** choisi dans une liste fermée (champ `error.type`). Cette liste permet des statistiques et des alertes fiables, ce que ne permettent pas des messages libres.

Le type d'erreur détermine l'état produit, selon la distinction présentée au § 3.10 :

- **`CRITICAL`** pour les erreurs qui mettent en cause la cible ou le parcours vérifié : `timeout`, `dns_error`, `connection_refused`, `connection_reset`, `tls_error`, `certificate_expired`, `http_status`, `assertion_failed`, `element_not_found`, `navigation_error`, `database_error`, `auth_failed`, ainsi que `script_error`, qui signale un parcours non vérifié ;
- **`UNKNOWN`** pour les erreurs qui empêchent l'évaluation : `browser_crash`, `plugin_error` (plugin qui n'a pas pu s'exécuter jusqu'à son terme ou dont le code retour est en dehors de 0 à 3, § 3.8.2), `secret_missing` (secret référencé par le contrôle mais absent de la sonde), `resource_exhausted`, `runner_crash`, `agent_restart`, `maintenance` (§ 3.14.2), `unknown`, et `no_data` par défaut (paramètre `onEmpty`, chapitre 5).

Le tableau de référence de tous les types d'erreur, avec leur signification, leur origine probable et la première vérification à mener, figure au chapitre 7 (§ 7.5.2). Le chapitre 8 les reprend sous forme de fiches de diagnostic.

3.9 Mesures de performance

3.9.1 Ce qui est mesuré

Chaque exécution mesure au minimum sa **durée totale**. Selon le type, la sonde collecte des mesures complémentaires qui permettent de localiser l'origine d'une lenteur.

Type	Mesures principales
`icmp`	Temps aller-retour (RTT), taux de perte de paquets
`tcp`	Durée d'établissement de la connexion
`dns`	Durée de résolution
`http`	Durée par phase : résolution DNS, connexion TCP, négociation TLS, délai avant le premier octet, transfert ; code HTTP ; durée de validité restante du certificat
`api`	Durée et code de chaque requête de la suite
`database`	Durée de la requête ; valeurs des indicateurs exportés
`nagios`	Données de performance (*perfdata*) publiées par le plugin
`browser`	Durée de chaque étape du parcours

3.9.2 Interpréter une durée

La durée mesurée par la sonde est celle observée **depuis son emplacement réseau**, avec ses propres ressources. Elle diffère de celle mesurée par un utilisateur sur un poste de travail, en particulier pour un parcours navigateur : le Raspberry Pi 5 exécute JavaScript et effectue le rendu des pages moins vite qu'un ordinateur récent.

Ce qui compte n'est donc pas la valeur absolue, mais **l'évolution** de la mesure dans le temps, pour une même sonde et un même contrôle. Une augmentation durable de 30 % du temps d'une étape est un signal fiable, même si la valeur absolue n'est pas directement comparable à l'expérience d'un utilisateur.

> ✓ **Conseil** — Pour comparer des sites entre eux, utilisez des sondes de même édition et de même configuration matérielle, avec des contrôles identiques.

3.10 Erreurs d'exécution et échecs de la cible

La distinction entre un problème **de la cible** et un problème **de la supervision** est fondamentale. Elle conditionne l'équipe à alerter et l'action à mener.

	Échec de la cible	Erreur d'exécution
Ce qui est en cause	L'application, le service ou le réseau supervisé	La sonde, sa configuration ou le contrôle lui-même
État produit	`CRITICAL` (ou `WARNING`)	`UNKNOWN`
Exemples	Code HTTP 500, élément absent de la page, base de données injoignable, valeur métier hors seuil	Chromium défaillant, mémoire insuffisante, plugin introuvable
Équipe concernée	Exploitation ou responsable de l'application	Administrateur de la sonde

	Échec de la cible	Erreur d'exécution
Action	Diagnostiquer l'application	Corriger la configuration ou intervenir sur la sonde (chapitre 8)

Certaines situations se situent à la frontière. Une erreur de programmation dans un scénario scripté (`script_error`) relève de la supervision, mais produit volontairement l'état `CRITICAL` : le parcours n'a pas été vérifié et ne doit pas passer inaperçu. De même, un contrôle dont le sélecteur ne correspond plus à la page, parce que l'application a évolué, produit un `CRITICAL` (`element_not_found`), alors que l'application fonctionne peut-être correctement. Ces **faux positifs** sont inhérents à la supervision synthétique. Le chapitre 6 du guide de développement des scénarios décrit les pratiques qui les limitent (sélecteurs robustes, assertions explicites), et le chapitre 8 la manière de les diagnostiquer.

3.11 Classes d'exécution et concurrence

3.11.1 Principe

La sonde exécute plusieurs contrôles en parallèle, dans les limites de ses ressources. Pour qu'un contrôle coûteux ne bloque pas les contrôles légers, chaque type est rattaché à une **classe d'exécution** dotée de sa propre limite de **concurrence**, c'est-à-dire du nombre maximal d'exécutions simultanées.

Classe	Types de contrôles	Concurrence par défaut	Composant
Légère	`icmp`, `tcp`, `dns`, `http`, `api`	16	Moteur principal (ICMP : environnement isolé)
Plugins et bases	`nagios`, `database`	4	Environnement isolé (plugins) et moteur principal (bases)
Navigateur	`browser`	**1** (2 configurable sur une sonde de 8 Go)	Moteur de scénarios

Les valeurs de concurrence sont données à titre indicatif ; les valeurs en vigueur pour chaque version sont indiquées dans les notes de version.

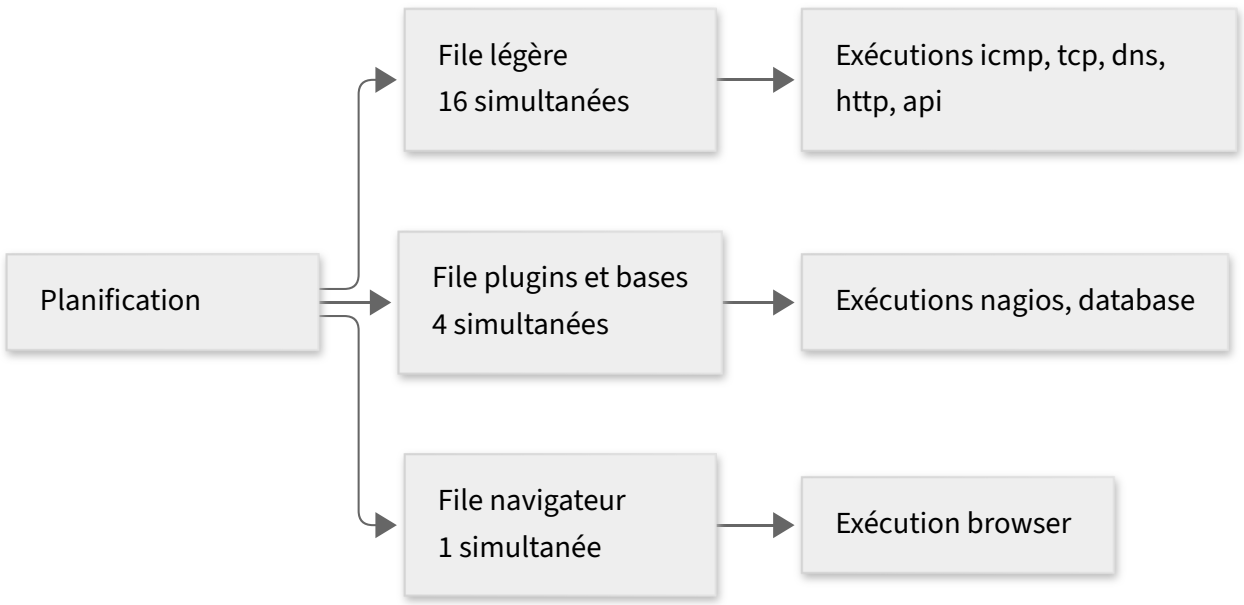


Schéma 3-4 — Files d'exécution et concurrence

3.11.2 Effets de la concurrence

Lorsqu'une file est pleine, les exécutions suivantes **attendent** qu'une place se libère. Ce retard est mesuré : c'est le **retard de planification**, publié dans les métriques de santé de la sonde (chapitre 2 du guide d'intégration et d'observabilité).

Un retard de planification durable indique que la sonde est sous-dimensionnée pour la charge configurée. C'est fréquent dans la classe navigateur, limitée à une exécution simultanée. Le chapitre 6 (§ 6.6) explique comment calculer la charge d'une sonde et la ramener dans des limites saines.

> ! **Attention** — Porter la concurrence de la classe navigateur au-delà de 2, ou à 2 sans mesure préalable de la consommation de vos scénarios, expose à des défaillances de Chromium par manque de mémoire, qui produisent des états `UNKNOWN` (`browser_crash`, `resource_exhausted`). Préférez réduire la fréquence des scénarios ou les regrouper dans un plan d'exécution.

3.12 Rétention et exploitation des données

3.12.1 Données conservées sur la sonde

La sonde conserve localement l'historique de ses exécutions. Elle reste ainsi exploitable sans aucune plateforme externe.

Donnée	Durée de conservation par défaut	Usage
Exécutions détaillées (résultat, étapes, mesures, erreurs)	14 jours	Diagnostic d'incidents récents

Donnée	Durée de conservation par défaut	Usage
Agrégats horaires (nombre d'exécutions, d'échecs, durées médiane et 95e centile)	400 jours	Disponibilité et tendances sur l'année
Artefacts (captures d'écran, journaux de console)	7 jours, dans la limite de 500 Mo	Diagnostic visuel des échecs
Journal d'audit des opérations d'administration	365 jours	Traçabilité (chapitre 6 du guide d'installation et d'administration)

Ces durées sont configurables par un administrateur (§ 6.9 du guide d'installation et d'administration), dans la limite de l'espace de stockage disponible. Lorsque l'espace se raréfie, la sonde supprime d'abord les données les plus anciennes, en commençant par les artefacts, et poursuit l'exécution des contrôles.

> **Attention** — La rétention ne concerne que les données conservées **sur la sonde**. Les objets créés par un scénario dans l'application supervisée (commandes, comptes, paniers de test) ne sont jamais supprimés par la sonde. Leur nettoyage relève de l'environnement de recette, indépendamment des exécutions des scénarios (§ 3.5.2).

> **Information** — Le **95e centile** (p95) d'une série de durées est la valeur en dessous de laquelle se situent 95 % des mesures. Il reflète l'expérience des exécutions les plus lentes sans être faussé par quelques valeurs extrêmes, contrairement au maximum.

3.12.2 Volume de données

À titre indicatif, 50 contrôles exécutés toutes les minutes produisent environ 72 000 exécutions par jour, soit de l'ordre de 300 Mo sur 14 jours. Cette volumétrie est compatible avec une carte microSD de 32 Go. Elle justifie toutefois une fréquence raisonnable et le recours aux agrégats pour l'analyse de long terme.

SSD NVMe facultatif. Un SSD NVMe, raccordé par la carte M.2 HAT+ officielle de Raspberry Pi, peut accueillir les données volumineuses : historique des exécutions, agrégats, artefacts, file d'export et journaux persistants. Il est surtout utile pour l'édition Enterprise et pour les rétentions longues ; la microSD suffit à un usage standard. La partition de démarrage, les deux emplacements système, le certificat d'édition, la configuration et les secrets restent sur la microSD. Si le SSD est absent ou défaillant, les contrôles continuent, l'historique local est conservé sur la microSD en mode réduit, et l'indicateur de santé `storage` passe en `CRITICAL`. Le matériel compatible et l'activation sont décrits aux chapitres 2 et 8 du guide d'installation et d'administration (§ 8.3.4 du guide d'installation et d'administration).

3.12.3 Exploitation des données

Les données de la sonde s'exploitent de trois manières complémentaires :

Usage	Moyen	Chapitre
Diagnostic immédiat, consultation sans outil tiers	Interface d'administration de la sonde, API REST	chapitre 4 du guide d'installation et d'administration du guide d'installation et d'administration et chapitre 5 du guide d'installation et d'administration
Tableaux de bord, alertes, historique de long terme, vue multi-sondes	Plateformes d'observabilité via OpenTelemetry ou Prometheus	chapitre 1 du guide d'intégration et d'observabilité

Usage	Moyen	Chapitre
Intégration aux outils de supervision existants	Plugin `check_sva` pour Nagios et Centreon	chapitre 1 du guide d'intégration et d'observabilité

La sonde reste dans tous les cas la **source de référence** des résultats : les plateformes externes en reçoivent une copie.

3.13 Incident réseau, applicatif ou fonctionnel

Un même symptôme vu par un utilisateur (« le site ne marche pas ») peut provenir de causes de natures très différentes. En combinant plusieurs types de contrôles sur une même application, on oriente rapidement le diagnostic vers la bonne équipe.

Nature de l'incident	Définition	Contrôles qui le révèlent	Signature typique
Réseau	La cible n'est pas joignable ou l'est mal : routage, pare-feu, DNS, saturation de lien	`icmp`, `tcp`, `dns`	`dns_error`, `connection_refused`, `timeout` sur des contrôles légers ; perte de paquets, latence en hausse
Technique / applicatif	La cible est joignable, mais le service applicatif répond mal ou pas du tout	`http`, `api`, `database`, `nagios`	`http_status` (5xx), `tls_error`, `database_error`, durées en forte hausse ; contrôles réseau `OK`
Fonctionnel	Le service répond techniquement, mais ne rend pas le service attendu	Assertions de contenu, `api` avec validation, `database` avec règles métier, `browser`	`assertion_failed`, `element_not_found`, `auth_failed` ; codes HTTP 200 ; contrôles réseau et techniques `OK`

La lecture combinée des états oriente le diagnostic :

Ping (`icmp`)	Port (`tcp`)	Page (`http`)	Parcours (`browser`)	Orientation probable
`CRITICAL`	`CRITICAL`	`CRITICAL`	`CRITICAL`	Incident réseau, ou serveur arrêté
`OK`	`CRITICAL`	`CRITICAL`	`CRITICAL`	Service arrêté ou flux filtré
`OK`	`OK`	`CRITICAL` (`http_status` 5xx)	`CRITICAL`	Incident applicatif côté serveur
`OK`	`OK`	`OK`	`CRITICAL` (`assertion_failed`)	Incident fonctionnel : parcours cassé, alors que le site répond
`OK`	`OK`	`WARNING`	`WARNING`	Dégradation de performance
`OK`	`OK`	`OK`	`UNKNOWN`	Problème de la sonde ou du scénario, pas de l'application

> **Information** — Certains équipements filtrent ICMP. Un contrôle `icmp` en échec sur une cible dont le port applicatif répond n'indique pas nécessairement un incident : vérifiez d'abord qu'ICMP est autorisé vers cette cible avant d'en faire un critère d'alerte.

> **Conseil** — Pour chaque application importante, combinez au moins un contrôle technique (`tcp` ou `http`) et un contrôle fonctionnel (assertion de contenu, `api` ou `browser`). Cette combinaison distingue d'emblée « l'application est arrêtée » de « l'application fonctionne mal ».

3.14 Fonctionnement autonome

3.14.1 Indépendance vis-à-vis des plateformes externes

La sonde est conçue pour fonctionner **sans dépendance externe**. Sa configuration, sa planification, l'exécution des contrôles et l'historique des résultats résident entièrement sur l'appareil. Elle n'envoie **aucune donnée à l'éditeur** : ses seuls échanges avec lui sont la vérification et le téléchargement des mises à jour, lorsque vous les activez (chapitre 8 du guide d'installation et d'administration).

Situation	Comportement de la sonde
Plateforme d'observabilité injoignable	Les contrôles continuent normalement. L'envoi de télémétrie est retenté pendant une durée limitée, puis les données non transmises sont abandonnées et comptabilisées. L'historique complet reste consultable localement
Accès Internet coupé	Les contrôles des ressources internes continuent. Ceux qui visent des ressources externes passent logiquement en `CRITICAL` : c'est le reflet fidèle de la situation vue depuis le site
Serveur NTP injoignable	La sonde conserve l'heure courante tant qu'elle n'est pas redémarrée. Après un redémarrage sans source de temps, les horodatages peuvent être faux tant que l'heure n'est pas resynchronisée : le Raspberry Pi 5 ne conserve l'heure hors tension qu'avec une pile optionnelle
Console centrale (évolution prévue) injoignable	Sans objet actuellement. Avec la console centrale prévue, la sonde continuera avec la dernière configuration reçue

> ! **Attention** — Une sonde privée d'accès vers ses destinations d'export continue de superviser, mais les alertes configurées dans ces plateformes ne se déclenchent plus. Configurez dans chaque plateforme une alerte d'absence de signal (chapitre 2 du guide d'intégration et d'observabilité), afin d'être prévenu de toute interruption de la remontée des données.

> → **Évolution prévue** — Une version ultérieure permettra de retransmettre, après le rétablissement d'une destination d'export, les traces et événements produits pendant l'interruption. Seules les données encore présentes dans la file d'envoi au moment du rétablissement sont transmises.

3.14.2 Comportement au redémarrage

Après une coupure électrique ou un redémarrage, la sonde reprend son activité **sans intervention**.

1. Le système démarre, puis les services de la sonde sont lancés automatiquement.
2. La sonde vérifie l'intégrité de sa base de données. En cas d'anomalie, elle restaure la dernière sauvegarde de configuration (chapitre 8 du guide d'installation et d'administration).
3. Les exécutions interrompues par l'arrêt sont clôturées avec l'état `UNKNOWN` et le type d'erreur `agent_restart`.
4. La planification est recalculée. Les exécutions manquées pendant l'arrêt ne sont **pas rattrapées** : chaque contrôle reprend à sa prochaine échéance, avec une gigue qui évite un démarrage simultané de tous les contrôles.
5. Les envois vers les destinations d'export reprennent dès que le réseau est disponible.

Une coupure électrique peut entraîner la perte des tout derniers résultats enregistrés. La configuration, conservée avec un niveau de durabilité supérieur, n'est pas affectée tant que le support de stockage respecte les écritures synchrones ; une panne de la carte elle-même relève de la sauvegarde de la configuration (chapitre 8 du guide d'installation et d'administration).

Mises à jour. Une mise à jour du système est téléchargée et installée sur l'emplacement système inactif **sans interrompre les contrôles**. Seule son **activation**, c'est-à-dire le redémarrage sur la nouvelle version, interrompt la supervision. Elle a lieu :

- dans l'une des fenêtres de maintenance définies par le client (section `updates`, paramètre `windows`, chapitre 8 du guide d'installation et d'administration). Une activation automatique ordinaire est annoncée au moins `updates.notice` à l'avance (24 heures par défaut) ; à défaut, elle est reportée à la fenêtre suivante. Un correctif critique est annoncé au moins 15 minutes à l'avance ;
- au moment choisi par un administrateur qui l'applique (`POST /api/v1/updates/apply`), immédiatement après confirmation explicite et sans préavis, ou à une heure donnée par `scheduledAt`, avec le préavis qui en résulte ;
- pour un correctif critique avec `criticalPolicy: asap`, dès que possible (§ 8.4.11 du guide d'installation et d'administration).

Aucune fenêtre n'est définie par défaut : sans fenêtre, aucune activation automatique ordinaire n'a lieu, la mise à jour reste en attente et l'état de santé `updates` passe en `WARNING` (`CRITICAL` pour un correctif critique) ; seul un correctif critique en mode `asap`, choisi explicitement, est activé sans fenêtre. La matrice complète figure au § 8.4.2 du guide d'installation et d'administration. Si la nouvelle version n'est pas saine, la sonde revient automatiquement à la version précédente, au prix d'un redémarrage supplémentaire.

Au début d'une activation, la sonde **draine** ses exécutions avant de redémarrer :

1. Elle passe en **état de maintenance** (§ 3.8.4) : `/health` répond `MAINTENANCE` (HTTP 503) et l'événement `maintenance.started` est émis.
2. Aucune exécution nouvelle ne démarre.
3. Les exécutions en cours disposent du délai `updates.drainTimeout` (5 minutes par défaut, 30 minutes au maximum) pour se terminer normalement.
4. Au-delà de ce délai, les exécutions encore en cours sont interrompues avec l'état `UNKNOWN` et le type d'erreur `maintenance` (§ 3.8.5).
5. La sonde redémarre sur la nouvelle version, puis reprend son activité selon les étapes ci-dessus ; l'événement `maintenance.ended` signale la fin de l'état de maintenance.

Le calcul de `STALE` est suspendu pendant la maintenance et pendant un intervalle après la reprise (§ 3.8.4). L'état de maintenance **ne suffit pas à supprimer les alertes** dans vos plateformes, car la sonde est injoignable pendant son redémarrage : alignez leurs périodes de maintenance sur vos fenêtres (chapitre 2 du guide d'intégration et d'observabilité). 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).

> **Information** — Une sonde unique ne supervise pas pendant son propre redémarrage. Lorsque la continuité de la supervision est essentielle, déployez plusieurs sondes par site, avec des fenêtres de maintenance qui ne se chevauchent pas (chapitre 8 du guide d'installation et d'administration).

> **Avertissement** — Des coupures électriques répétées sollicitent fortement la carte microSD et peuvent, à terme, l'endommager. Préférez toujours un arrêt propre (chapitre 8 du guide d'installation et d'administration) et alimentez la sonde par une alimentation conforme (chapitre 2 du guide d'installation et d'administration), idéalement secourue par onduleur.

3.14.3 Ce que la sonde ne fait pas seule

Pour fixer des attentes justes :

- la sonde **ne crée pas d'alertes** par courriel ou messagerie. Les alertes sont produites par les plateformes vers lesquelles elle exporte ses données, ou par Nagios/Centreon via `check_sva` (chapitre 2 du guide d'intégration et d'observabilité) ;
- la sonde **ne corrige pas** les incidents qu'elle détecte. Elle les constate et fournit les éléments de diagnostic ;
- la sonde **n'observe que ce qu'on lui demande** : un parcours non couvert par un contrôle n'est pas supervisé.

Chapitre 4 — Contrôles réseau et techniques

Ce chapitre décrit les cinq types de contrôles qui vérifient la disponibilité et la performance des **couches techniques** : joignabilité réseau, ouverture de ports, résolution de noms, réponses des serveurs web et plugins compatibles avec l'écosystème Nagios. Ce sont les contrôles les plus légers de la sonde. Ils constituent le socle de toute supervision, y compris lorsque des contrôles applicatifs (chapitre 5) ou des scénarios navigateur (chapitres 3 à 6 du guide de développement des scénarios) sont déployés.

Les principes communs à tous les contrôles (fréquence, timeout, nouvelles tentatives, seuils, états) sont expliqués au chapitre 3. La planification et les plans d'exécution sont traités au chapitre 6, la lecture des résultats au chapitre 7. Le présent chapitre se concentre sur ce qui est propre à chaque type de contrôle.

4.1 Vue d'ensemble

4.1.1 Types de contrôles techniques

Type	Ce qui est vérifié	Édition minimale	Composant exécutant
`icmp`	Un équipement répond au ping ; latence et taux de perte	Essential	Environnement d'exécution isolé (utilitaire `fping`)
`tcp`	Un port TCP accepte les connexions ; dialogue simple ; poignée de main TLS	Essential	Moteur principal
`dns`	Un nom est résolu, par le serveur attendu, avec la valeur attendue	Essential	Moteur principal
`http`	Un serveur web ou une API répond avec le code, les en-têtes, le contenu et le délai attendus ; validité du certificat	Essential	Moteur principal
`nagios`	Toute vérification réalisable par un plugin au format Nagios	Essential (plugins fournis) · Professional (plugins personnalisés)	Environnement d'exécution isolé

Le contrôle `http` couvre aussi la supervision d'API et de web services. Ces usages, qui relèvent de la supervision applicative, sont détaillés au chapitre 5.

4.1.2 Conventions des exemples

Tous les exemples de ce chapitre sont **fictifs** :

- noms de domaine en `example.com`, `example.org` et `example.internal` ;
- adresses IP dans les plages réservées à la documentation (`192.0.2.0/24`, `198.51.100.0/24`, `203.0.113.0/24` en IPv4, `2001:db8::/32` en IPv6).

Remplacez-les par les valeurs de votre environnement.

Les contrôles sont présentés au format JSON de l'API d'administration, c'est-à-dire la forme qu'ils prennent dans `POST /api/v1/checks` (chapitre 6 du guide d'installation et d'administration). L'interface d'administration propose les mêmes paramètres sous forme de formulaire. Chaque exemple est complet et peut être soumis tel quel, après adaptation des cibles.

> ✓ **Conseil** — Avant d'activer un nouveau contrôle, exécutez-le une première fois en mode essai : `POST /api/v1/checks/{id}/runs?dryRun=true`, ou le bouton d'exécution d'essai de l'interface. Le mode essai réalise le contrôle sans enregistrer d'état ni transmettre de télémétrie. Vous vérifiez ainsi les paramètres sans déclencher d'alerte (§ 6.6.5 du guide d'installation et d'administration).

Les extraits de résultats présentés dans ce chapitre illustrent la structure d'un résultat d'exécution (chapitre 7). Les noms des mesures (`measurements`) propres à chaque type de contrôle, tels qu'ils figurent dans ces extraits, sont ceux du produit. Les blocs « Résultat attendu » comportent tous les champs obligatoires du résultat d'exécution (`id`, `checkId`, `checkRevision`, `status`, `startedAt`, `durationMs`) ; les champs facultatifs sans intérêt pour le type de contrôle présenté (étapes, artefacts, trace…) y sont omis. Les blocs signalés comme **extraits (projection)** ne reprennent que quelques champs : ils ne constituent pas une réponse complète de l'API. L'exemple de réponse intégrale figure au § 7.4.

4.1.3 Cibles IPv6

La sonde fonctionne toujours en IPv4 et peut, en complément, utiliser IPv6 lorsque celui-ci est activé dans ses paramètres réseau (section `network`, objet `ipv6`, chapitre 5 du guide d'installation et d'administration). Il est désactivé par défaut. Les contrôles peuvent alors viser des cibles IPv6.

La famille d'adresses employée par un contrôle est fixée par le paramètre commun `ipFamily` (§ 3.1.3) :

- `auto` (défaut) : IPv6 si l'IPv6 est activé sur la sonde et si la cible possède une adresse AAAA, IPv4 sinon ;
- `ipv4` : IPv4 uniquement ;
- `ipv6` : IPv6 uniquement.

Lorsque la cible est désignée par une adresse IP littérale, la famille est celle de cette adresse. Lorsqu'elle est désignée par un nom et que ce nom n'a pas d'adresse dans la famille demandée, l'exécution échoue avec le type d'erreur `dns_error`. Un contrôle `ipFamily: "ipv6"` est refusé à l'enregistrement (`422`) tant que l'IPv6 est désactivé sur la sonde. Si l'IPv6 est désactivé après coup, ces contrôles passent en `UNKNOWN` jusqu'à sa réactivation ou leur modification.

Type	Comportement en IPv6
`icmp`	`fping` envoie des requêtes *ICMPv6 Echo*. Les équipements et pare-feu intermédiaires doivent autoriser l'ICMPv6 Echo depuis l'adresse IPv6 de la sonde
`tcp`	Connexion, poignée de main TLS et dialogue `send` / `expect` identiques à IPv4, vers l'adresse IPv6 retenue
`dns`	Le type d'enregistrement interrogé reste fixé par `dns.recordType` : `AAAA` vérifie les adresses IPv6 d'un nom, quelle que soit la famille utilisée pour joindre le serveur. Les serveurs de `dns.servers` peuvent être des adresses IPv4 ou IPv6 ; chacun est interrogé dans la famille de son adresse
`http`	Requête vers l'adresse IPv6 retenue. Une adresse littérale s'écrit entre crochets dans l'URL : `https://[2001:db8:20::10]/`. La vérification du certificat porte alors sur cette adresse, qui doit figurer dans le certificat du serveur
`nagios`	Selon les options propres à chaque plugin (par exemple `-6` pour la plupart des plugins de **Monitoring Plugins**)

L'exemple fictif suivant vérifie la joignabilité d'un routeur par son adresse IPv6 :

```json
{
  "id": "routeur-site-lyon-ipv6",
  "name": "Routeur du site de Lyon : joignabilité IPv6",
  "description": "Ping ICMPv6 vers l'adresse IPv6 du routeur de Lyon (exemple fictif).",
  "type": "icmp",
  "tags": { "site": "lyon", "domaine": "reseau" },
  "schedule": { "interval": "1m" },
  "timeout": "10s",
  "ipFamily": "ipv6",
  "icmp": {
    "host": "2001:db8:10::1",
    "count": 5,
    "maxPacketLossPercent": 20
  }
}
```

Avec une adresse littérale, `ipFamily` est facultatif : il est indiqué ici pour rendre l'intention explicite. Pour une cible désignée par un nom qui possède des adresses IPv4 et IPv6, `"ipFamily": "ipv6"` force la vérification du chemin IPv6, et un second contrôle en `"ipFamily": "ipv4"` permet de suivre les deux chemins séparément.

Les contrôles vers des cibles IPv6 fonctionnent selon les règles ci-dessus, pour chacun des types du tableau.

4.2 Choisir le bon contrôle

4.2.1 Arbre de décision

Le choix d'un type de contrôle dépend de la question à laquelle vous voulez répondre. Le schéma suivant oriente ce choix. Il inclut les types applicatifs, pour montrer la frontière avec le chapitre 5 et les chapitres 3 à 6 du guide de développement des scénarios.

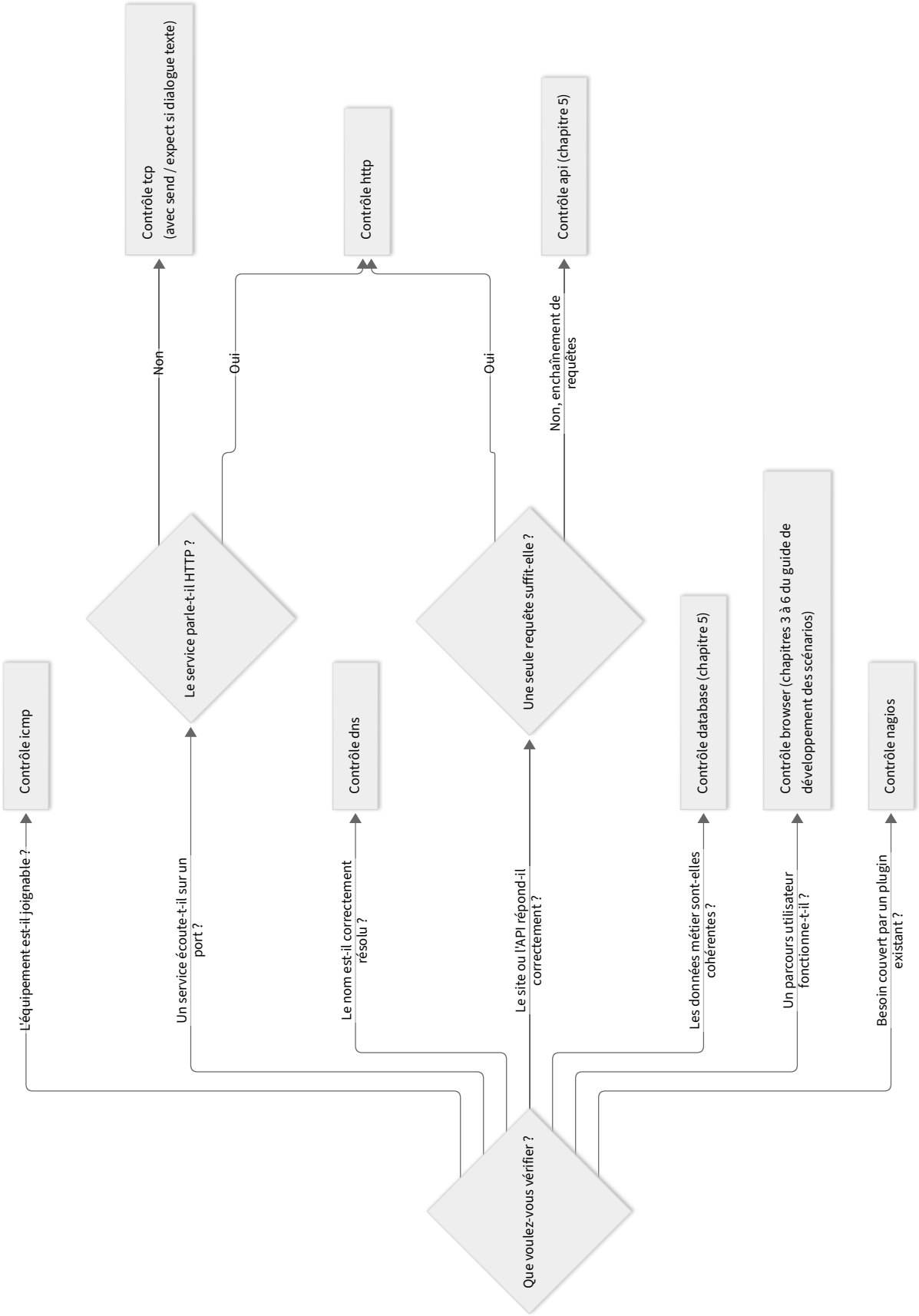


Schéma 4-1 — Choisir un type de contrôle

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

4.2.2 Superviser par couches

Un service informatique repose sur plusieurs couches. Un contrôle unique ne permet pas toujours de localiser une panne : un contrôle HTTP en échec peut révéler un problème de résolution de noms, de réseau, de serveur ou d'application. Superposer des contrôles de niveaux différents sur un même service accélère considérablement le diagnostic.

Exemple pour le site web fictif `www.example.com` :

Couche	Contrôle	Ce que son échec indique
Résolution de noms	`dns` sur `www.example.com`	Le nom n'est plus résolu, ou pointe vers une mauvaise adresse
Réseau	`icmp` sur la passerelle ou le répartiteur de charge	Le chemin réseau est coupé
Transport	`tcp` sur le port 443	Le service n'écoute plus, ou un pare-feu bloque le flux
Protocole	`http` sur la page d'accueil	Le serveur web répond mal (erreur 5xx, contenu inattendu, lenteur)
Certificat	`http` avec surveillance de l'expiration	Le certificat expire bientôt ou est invalide
Application	`browser` sur le parcours de connexion (Enterprise)	L'application est inutilisable pour l'utilisateur

Lors d'un incident, l'état combiné de ces contrôles désigne directement la couche défaillante. Si `dns` et `tcp` sont `OK` mais `http` est `CRITICAL`, la cause se situe au niveau du serveur web ou de l'application, pas du réseau.

> ✓ **Conseil** — N'empilez pas systématiquement toutes les couches pour chaque service. Réservez cette approche aux services critiques. Pour les autres, un contrôle `http` bien configuré suffit souvent.

4.3 Contrôle ICMP (`icmp`)

4.3.1 Objectif et intérêt

Le contrôle `icmp` envoie une série de requêtes *ICMP Echo* (le « ping ») vers un équipement et mesure :

- le **temps d'aller-retour** (RTT, *Round-Trip Time*) : délai entre l'envoi d'une requête et la réception de la réponse ;
- le **taux de perte** : proportion de requêtes restées sans réponse.

Il répond à la question la plus élémentaire de la supervision : l'équipement est-il joignable depuis la sonde ? Il convient particulièrement aux équipements qui n'exposent aucun service applicatif interrogeable : routeurs, commutateurs, pare-feu, liens inter-sites, imprimantes, équipements industriels.

Le contrôle `icmp` ne vérifie **pas** qu'un service fonctionne. Un serveur peut répondre au ping alors que son application est arrêtée. À l'inverse, de nombreux serveurs et pare-feu bloquent volontairement l'ICMP : un échec ne signifie pas toujours une panne.

4.3.2 Fonctionnement

Le contrôle est exécuté par l'environnement d'exécution isolé de la sonde, au moyen de l'utilitaire système `fping`. Pour chaque exécution, la sonde :

1. résout le nom de l'hôte s'il ne s'agit pas d'une adresse IP, en IPv4 ou en IPv6 selon le paramètre `ipFamily` (§ 4.1.3) ;
2. envoie le nombre de requêtes défini par `count` ;
3. calcule le temps d'aller-retour moyen et le taux de perte ;
4. compare ces valeurs aux seuils.

4.3.3 Paramètres

Paramètre	Obligatoire	Description	Défaut
`icmp.host`	Oui	Nom d'hôte ou adresse IP (IPv4 ou IPv6) de l'équipement	—
`icmp.count`	Non	Nombre de requêtes par exécution (1 à 20)	3
`icmp.maxPacketLossPercent`	Non	Taux de perte maximal toléré, en pourcentage. Au-delà, l'état est `CRITICAL`	0
`icmp.warningRttMs`	Non	Temps d'aller-retour moyen, en millisecondes, au-delà duquel l'état est `WARNING`	Aucun
`icmp.criticalRttMs`	Non	Temps d'aller-retour moyen, en millisecondes, au-delà duquel l'état est `CRITICAL`	Aucun

Les paramètres communs (`schedule`, `timeout`, `retry`, `tags`…) sont décrits au chapitre 3.

4.3.4 Détermination de l'état

Situation	État	Type d'erreur
Taux de perte inférieur ou égal au seuil et RTT moyen sous `warningRttMs`	`OK`	—
Taux de perte inférieur ou égal au seuil, mais RTT moyen supérieur à `warningRttMs`	`WARNING`	—
Taux de perte inférieur ou égal au seuil, mais RTT moyen supérieur à `criticalRttMs`	`CRITICAL`	—
Taux de perte supérieur à `maxPacketLossPercent`, y compris aucune réponse	`CRITICAL`	`timeout`
Nom d'hôte non résolu	`CRITICAL`	`dns_error`
Utilitaire indisponible ou erreur interne de la sonde	`UNKNOWN`	`plugin_error`

4.3.5 Exemple : joignabilité d'un routeur de site distant

Le routeur fictif du site de Lyon, d'adresse `192.0.2.1`, est contrôlé toutes les minutes. Cinq requêtes sont envoyées par exécution :

- jusqu'à une requête perdue sur cinq (20 %) est tolérée ;
- un RTT moyen supérieur à 50 ms déclenche un `WARNING`.

```json
{
  "id": "routeur-site-lyon",
  "name": "Routeur du site de Lyon : joignabilité",
  "description": "Vérifie la liaison inter-sites vers le routeur de Lyon (exemple fictif).",
  "type": "icmp",
  "tags": { "site": "lyon", "domaine": "reseau" },
  "schedule": { "interval": "1m" },
  "timeout": "10s",
  "retry": { "attempts": 2, "interval": "15s" },
  "icmp": {
    "host": "192.0.2.1",
    "count": 5,
    "maxPacketLossPercent": 20,
    "warningRttMs": 50
  }
}
```

Les deux nouvelles tentatives à 15 secondes d'intervalle évitent qu'une perte ponctuelle sur la liaison ne déclenche une alerte. L'état `CRITICAL` n'est retenu que si les trois exécutions successives échouent (chapitre 3).

4.3.6 Résultat attendu

Exemple de résultat d'exécution réussie :

```json
{
  "id": "01M7SD3NCWRFNSFJGQGVKWDBH3",
  "checkId": "routeur-site-lyon",
  "checkRevision": 1,
  "status": "OK",
  "startedAt": "2025-11-19T08:00:00.412Z",
  "durationMs": 4120,
  "measurements": {
    "icmp.rtt_avg_ms": 12.4,
    "icmp.packet_loss_percent": 0
  }
}
```

Ces deux mesures alimentent les métriques `sva.icmp.rtt` et `sva.icmp.packet_loss` (chapitre 2 du guide d'intégration et d'observabilité).

4.3.7 Erreurs courantes

Symptôme	Cause probable	Action
`CRITICAL` permanent alors que l'équipement fonctionne	L'équipement ou un pare-feu intermédiaire bloque les requêtes ICMP	Faire autoriser l'ICMP Echo depuis l'adresse de la sonde, ou remplacer le contrôle par un contrôle `tcp` sur un port ouvert
États alternant entre `OK` et `CRITICAL`	Liaison instable, ou limitation du débit ICMP par l'équipement (les routeurs donnent souvent une faible priorité à l'ICMP)	Augmenter `count` et `maxPacketLossPercent` ; configurer des nouvelles tentatives
`WARNING` fréquents sans impact pour les utilisateurs	Seuil `warningRttMs` trop strict pour une liaison longue distance	Mesurer la latence normale pendant quelques jours, puis fixer le seuil avec une marge
`CRITICAL` avec `dns_error`	Le nom de l'équipement n'est pas résolu par les serveurs DNS de la sonde	Vérifier la configuration DNS de la sonde (chapitre 5 du guide d'installation et d'administration) ou utiliser l'adresse IP

> **Information** — Le temps d'aller-retour ICMP mesure la latence réseau, pas le temps de réponse d'une application. Pour suivre la performance perçue par les utilisateurs, utilisez les mesures du contrôle `http` (§ 4.6) ou d'un scénario navigateur.

4.4 Contrôle TCP (`tcp`)

4.4.1 Objectif et intérêt

Le contrôle `tcp` vérifie qu'un **port TCP** accepte les connexions sur un hôte donné, et mesure le temps d'établissement de la connexion. Il permet en outre :

- d'effectuer une **poignée de main TLS** (*TLS handshake*), la négociation initiale d'une connexion chiffrée, pour vérifier qu'un service chiffré répond correctement ;
- de mener un **dialogue texte simple** : envoyer une chaîne de caractères (`send`), puis vérifier que la réponse contient une chaîne attendue (`expect`).

C'est le contrôle de référence pour les services qui ne parlent pas HTTP : serveurs de messagerie (SMTP, IMAP), annuaires (LDAP), bases de données (au niveau réseau), serveurs de fichiers, files de messages, caches, protocoles propriétaires.

Il est plus fiable que le ping pour vérifier qu'un serveur est opérationnel, puisqu'il atteste qu'un processus écoute effectivement sur le port. Il ne valide en revanche ni l'authentification, ni le bon fonctionnement applicatif du service.

4.4.2 Paramètres

Paramètre	Obligatoire	Description	Défaut
`tcp.host`	Oui	Nom d'hôte ou adresse IP (IPv4 ou IPv6, § 4.1.3)	—
`tcp.port`	Oui	Port TCP (1 à 65535)	—
`tcp.tls`	Non	Si `true`, effectue une poignée de main TLS après la connexion	`false`
`tcp.send`	Non	Chaîne envoyée après l'établissement de la connexion (1 024 caractères maximum). Les séquences JSON `\r` et `\n` permettent d'envoyer des retours chariot et des sauts de ligne	Aucune
`tcp.expect`	Non	Chaîne qui doit figurer dans les données reçues (1 024 caractères maximum)	Aucune
`thresholds.warningDuration`	Non	Durée totale au-delà de laquelle l'état est `WARNING`	Aucun

> **Information** — La validation du certificat lors de la poignée de main TLS du contrôle `tcp` (chaîne de confiance, nom d'hôte) suit le magasin de certificats de la sonde : le magasin système, complété des autorités de certification importées dans la section `trust` (§ 5.7.3 du guide d'installation et d'administration). Pour une surveillance détaillée d'un certificat, y compris de sa date d'expiration, utilisez un contrôle `http` (§ 4.6.7) lorsque le service est un service web.

4.4.3 Détermination de l'état

Situation	État	Type d'erreur
Connexion établie, poignée de main TLS réussie si demandée, chaîne attendue reçue si définie, durée sous le seuil	`OK`	—

Situation	État	Type d'erreur
Idem, mais durée supérieure à `thresholds.warningDuration`	`WARNING`	—
Connexion refusée (aucun processus n'écoute, ou rejet actif par un pare-feu)	`CRITICAL`	`connection_refused`
Pas de réponse dans le délai imparti (filtrage silencieux, hôte éteint)	`CRITICAL`	`timeout`
Connexion interrompue par le serveur	`CRITICAL`	`connection_reset`
Échec de la poignée de main TLS	`CRITICAL`	`tls_error`
Chaîne `expect` absente de la réponse	`CRITICAL`	`assertion_failed`
Nom d'hôte non résolu	`CRITICAL`	`dns_error`

4.4.4 Exemple : port d'un serveur de base de données

Ce contrôle vérifie que le serveur PostgreSQL fictif `db-ventes.example.internal` accepte les connexions sur son port standard. Il ne remplace pas un contrôle `database` (chapitre 5), qui exécute une requête réelle. Il suffit en revanche à distinguer une panne réseau d'une panne applicative.

```json
{
  "id": "db-ventes-port",
  "name": "Serveur de base Ventes : port PostgreSQL",
  "type": "tcp",
  "tags": { "application": "ventes", "domaine": "base_de_donnees" },
  "schedule": { "interval": "1m" },
  "timeout": "5s",
  "retry": { "attempts": 1, "interval": "20s" },
  "thresholds": { "warningDuration": "500ms" },
  "tcp": {
    "host": "db-ventes.example.internal",
    "port": 5432
  }
}
```

4.4.5 Exemple : bannière d'un serveur de messagerie

Un serveur SMTP envoie une bannière commençant par le code `220` dès l'ouverture de la connexion. Vérifier cette bannière garantit que le service de messagerie est prêt à recevoir des messages, et pas seulement que le port est ouvert.

```json
{
  "id": "smtp-relais",
  "name": "Relais SMTP interne : bannière de service",
  "type": "tcp",
  "tags": { "application": "messagerie" },
  "schedule": { "interval": "2m" },
  "timeout": "10s",
  "tcp": {
    "host": "smtp.example.internal",
    "port": 25,
    "expect": "220"
  }
}
```

Aucune chaîne n'est envoyée : la sonde se contente de lire la bannière. Le contrôle ne transmet aucun message et ne s'authentifie pas.

4.4.6 Exemple : dialogue texte avec un cache

Certains services acceptent des commandes texte. L'exemple suivant envoie la commande `PING` à un serveur de cache compatible avec le protocole Redis, qui doit répondre `+PONG`. La séquence `\r\n` termine la commande, conformément au protocole.

JSON

```
{
  "id": "cache-sessions",
  "name": "Cache des sessions : réponse au PING",
  "type": "tcp",
  "tags": { "application": "portail" },
  "schedule": { "interval": "1m" },
  "timeout": "5s",
  "tcp": {
    "host": "cache.example.internal",
    "port": 6379,
    "send": "PING\r\n",
    "expect": "+PONG"
  }
}
```

> **Attention** — N'envoyez jamais par `send` une commande qui modifie l'état du service : écriture, suppression, arrêt. Le contrôle s'exécute à intervalle régulier, potentiellement des milliers de fois par jour. Limitez-vous aux commandes de lecture ou de diagnostic prévues par le protocole.

4.4.7 Exemple : service chiffré (TLS)

Pour un annuaire LDAP accessible en LDAPS (port 636), l'option `tls` vérifie que la négociation chiffrée aboutit :

JSON

```
{
  "id": "annuaire-ldaps",
  "name": "Annuaire d'entreprise : LDAPS",
  "type": "tcp",
  "tags": { "application": "annuaire" },
  "schedule": { "interval": "2m" },
  "timeout": "10s",
  "tcp": {
    "host": "ldap.example.internal",
    "port": 636,
    "tls": true
  }
}
```

4.4.8 Résultat attendu

JSON

```
{
  "id": "01M7SDCT5TPA021PRK6SRM48FC",
  "checkId": "smtp-relais",
  "checkRevision": 1,
  "status": "OK",
  "startedAt": "2025-11-19T08:05:00.218Z",
  "durationMs": 38,
  "measurements": {
    "tcp.connect_ms": 4,
    "tcp.response_received": true
  }
}
```

La durée totale de l'exécution alimente la métrique `sva.check.duration` (chapitre 2 du guide d'intégration et d'observabilité) ; la durée d'établissement de la connexion est conservée dans les mesures du résultat.

4.4.9 Erreurs courantes

Symptôme	Cause probable	Action
`CRITICAL` / `connection_refused`	Le service est arrêté, écoute sur une autre interface ou un autre port, ou un pare-feu rejette la connexion	Vérifier l'état du service sur le serveur ; vérifier le port ; vérifier les règles de filtrage entre la sonde et la cible
`CRITICAL` / `timeout`	Un pare-feu ignore silencieusement les paquets, ou l'hôte est éteint	Faire ouvrir le flux depuis l'adresse de la sonde (chapitre 5 du guide d'installation et d'administration) ; tester la joignabilité avec un contrôle `icmp`
`CRITICAL` / `assertion_failed` alors que le service fonctionne	La chaîne attendue diffère (bannière personnalisée, autre version) ou la réponse arrive en plusieurs fragments après la lecture	Exécuter le contrôle en mode essai et lire la réponse reçue dans le détail de l'exécution ; choisir une chaîne `expect` plus courte et stable
`CRITICAL` / `tls_error`	Certificat expiré ou non reconnu, protocole TLS non pris en charge, service non chiffré sur ce port	Vérifier le certificat du service ; vérifier que le port est bien un port TLS (et non STARTTLS, qui commence en clair)
Le contrôle provoque des entrées d'erreur dans les journaux du service	Connexions ouvertes puis fermées sans dialogue complet, considérées comme anormales par certains services	En informer l'équipe responsable ; espacer la fréquence ; préférer un plugin Nagios dédié au protocole (§ 4.7)

> **Information** — Les services qui utilisent STARTTLS (SMTP sur le port 587, IMAP sur 143, LDAP sur 389) débutent la connexion en clair et ne basculent en chiffré qu'après une commande explicite. Le contrôle `tcp` avec `tls: true` ne convient pas à ce cas. Utilisez le plugin Nagios adapté (`check_smtp` avec l'option `-S` , par exemple), décrit au § 4.7.

4.5 Contrôle DNS (`dns`)

4.5.1 Objectif et intérêt

Le DNS (*Domain Name System*) traduit les noms (`www.example.com`) en adresses IP et fournit d'autres informations : serveurs de messagerie d'un domaine, alias, enregistrements texte. Quand le DNS est défaillant, tous les services qui en dépendent deviennent inaccessibles, même s'ils fonctionnent parfaitement.

Le contrôle `dns` interroge un ou plusieurs serveurs DNS et vérifie :

- que le nom est **résolu** (la requête aboutit et renvoie au moins une réponse) ;
- que la réponse **contient les valeurs attendues** (adresse IP, nom de serveur de messagerie…) ;
- que le **délai de résolution** reste acceptable.

Cas d'usage typiques :

- détecter une erreur de modification d'une zone DNS : enregistrement supprimé ou adresse modifiée par erreur ;
- vérifier que les serveurs DNS internes répondent, en interrogeant chaque serveur individuellement ;
- vérifier la cohérence entre la vue interne et la vue externe d'un domaine (*split-horizon*) ;
- contrôler les enregistrements de messagerie (MX) ou les enregistrements texte (TXT) utilisés pour la sécurité des messages ;

- détecter une redirection anormale du trafic par une modification non autorisée d'un enregistrement.

4.5.2 Paramètres

Paramètre	Obligatoire	Description	Défaut
`dns.name`	Oui	Nom à résoudre	—
`dns.recordType`	Non	Type d'enregistrement : `A`, `AAAA`, `CNAME`, `MX`, `TXT`, `NS`, `SRV`, `PTR`, `SOA`	`A`
`dns.servers`	Non	Serveurs DNS à interroger (4 au maximum), désignés par une adresse IPv4 ou IPv6 (§ 4.1.3). S'il est absent, la sonde utilise ses propres serveurs DNS (chapitre 5 du guide d'installation et d'administration)	Serveurs de la sonde
`dns.expect`	Non	Valeurs qui doivent figurer dans la réponse (16 au maximum)	Aucune
`thresholds.warningDuration`	Non	Délai de résolution au-delà duquel l'état est `WARNING`	Aucun

Types d'enregistrements :

Type	Contenu	Exemple de valeur attendue
`A`	Adresse IPv4	`203.0.113.10`
`AAAA`	Adresse IPv6	`2001:db8::10`
`CNAME`	Alias vers un autre nom	`www-prod.example.net`
`MX`	Serveurs de messagerie du domaine	`mx1.example.com`
`TXT`	Texte libre (politiques de messagerie, vérifications de domaine)	`v=spf1 include:_spf.example.com -all`
`NS`	Serveurs faisant autorité pour la zone	`ns1.example.com`
`SRV`	Localisation d'un service (hôte et port)	`ldap.example.internal`
`PTR`	Résolution inverse (adresse vers nom)	`www.example.com`
`SOA`	Informations d'autorité de la zone	Serveur primaire de la zone

> **Information** — Lorsque plusieurs valeurs sont listées dans `expect`, toutes doivent être présentes dans la réponse. La réponse peut contenir d'autres valeurs en plus. Lorsque plusieurs serveurs sont listés dans `servers`, chacun est interrogé et chacun doit répondre correctement.

4.5.3 Détermination de l'état

Situation	État	Type d'erreur
Réponse obtenue, valeurs attendues présentes, délai sous le seuil	`OK`	—
Idem, mais délai supérieur à `thresholds.warningDuration`	`WARNING`	—
Nom inexistant (NXDOMAIN), absence d'enregistrement du type demandé, échec du serveur (SERVFAIL)	`CRITICAL`	`dns_error`
Serveur DNS ne répondant pas dans le délai	`CRITICAL`	`timeout`
Réponse obtenue, mais une valeur attendue est absente	`CRITICAL`	`assertion_failed`

4.5.4 Exemple : adresse publique d'un site

Ce contrôle vérifie que le nom fictif `www.example.com` pointe toujours vers l'adresse publique attendue. Il interroge deux résolveurs : l'un interne, l'autre représentant un résolveur externe de l'entreprise.

```json
{
  "id": "dns-site-public",
  "name": "DNS : adresse du site public",
  "type": "dns",
  "tags": { "application": "site_public", "domaine": "dns" },
  "schedule": { "interval": "5m" },
  "timeout": "10s",
  "retry": { "attempts": 1, "interval": "30s" },
  "thresholds": { "warningDuration": "300ms" },
  "dns": {
    "name": "www.example.com",
    "recordType": "A",
    "servers": ["192.0.2.53", "198.51.100.53"],
    "expect": ["203.0.113.10"]
  }
}
```

Si quelqu'un modifie l'enregistrement par erreur, le contrôle passe en `CRITICAL` avec le type d'erreur `assertion_failed`, alors même que le nom continue d'être résolu.

4.5.5 Exemple : enregistrements de messagerie

```json
{
  "id": "dns-mx-domaine",
  "name": "DNS : serveurs de messagerie du domaine",
  "type": "dns",
  "tags": { "application": "messagerie", "domaine": "dns" },
  "schedule": { "interval": "15m" },
  "timeout": "10s",
  "dns": {
    "name": "example.com",
    "recordType": "MX",
    "expect": ["mx1.example.com", "mx2.example.com"]
  }
}
```

4.5.6 Exemple : résolution interne d'un service applicatif

```json
{
  "id": "dns-api-interne",
  "name": "DNS interne : nom de l'API de commandes",
  "type": "dns",
  "tags": { "application": "commandes", "domaine": "dns" },
  "schedule": { "interval": "5m" },
  "timeout": "5s",
  "dns": {
    "name": "api-commandes.example.internal",
    "recordType": "A"
  }
}
```

Sans `expect`, le contrôle vérifie uniquement que le nom est résolu, quelle que soit l'adresse. Cette forme convient aux services dont l'adresse change légitimement (répartition de charge dynamique, plateformes en nuage).

4.5.7 Résultat attendu

```json
{
  "id": "01M7SDNZ8B64S4DFX3Q75QPQA1",
  "checkId": "dns-site-public",
  "checkRevision": 1,
  "status": "OK",
  "startedAt": "2025-11-19T08:10:00.331Z",
  "durationMs": 42,
  "measurements": {
    "dns.answers": "203.0.113.10",
    "dns.lookup_ms": 18
  }
}
```

Le délai de résolution alimente la métrique `sva.dns.lookup.duration` (chapitre 2 du guide d'intégration et d'observabilité).

4.5.8 Erreurs courantes

Symptôme	Cause probable	Action
`CRITICAL` / `dns_error` sur un nom interne uniquement	Les serveurs DNS de la sonde ne connaissent pas la zone interne (sonde configurée avec des résolveurs publics)	Configurer les serveurs DNS internes sur la sonde (chapitre 5 du guide d'installation et d'administration) ou dans `dns.servers`
`CRITICAL` / `assertion_failed` alors que le service fonctionne	L'adresse a été modifiée volontairement (migration, changement d'hébergeur), ou plusieurs adresses sont renvoyées en rotation	Mettre à jour `expect` ; ne lister que les valeurs stables
`CRITICAL` / `timeout` sur un serveur précis	Le serveur DNS est arrêté, ou le flux UDP/TCP 53 est filtré entre la sonde et ce serveur	Vérifier le service DNS ; faire ouvrir le port 53 en UDP et TCP
Résultats différents selon le serveur interrogé	Vue interne et vue externe différentes (comportement voulu), ou réplication incomplète entre serveurs	Créer un contrôle par vue avec les valeurs attendues propres à chacune ; vérifier la réplication des zones
Le contrôle reste `OK` après une modification de zone	Le résolveur interrogé renvoie la valeur conservée en cache jusqu'à expiration de sa durée de vie (TTL)	Interroger directement les serveurs faisant autorité dans `dns.servers`

4.6 Contrôle HTTP et HTTPS (`http`)

4.6.1 Objectif et intérêt

Le contrôle `http` envoie une requête HTTP ou HTTPS et vérifie la réponse : code de statut, en-têtes, contenu, délai. Il mesure aussi les **phases** de la requête, ce qui permet de localiser une lenteur, et surveille le **certificat** des sites HTTPS.

C'est le contrôle le plus utilisé de la sonde. Il convient pour :

- les sites web et les pages d'accueil ;
- les points de contrôle de santé des applications (*health checks*), souvent exposés sous des chemins comme `/health` ou `/status` ;
- les API REST et les web services : voir le chapitre 5 pour les usages applicatifs avancés ;
- la surveillance des certificats TLS ;
- la vérification des redirections : passage forcé en HTTPS, nouvelles adresses après une migration.

Contrairement à un navigateur, le contrôle `http` n'exécute pas le JavaScript des pages et ne charge pas les ressources associées (images, feuilles de style, scripts). Il vérifie une seule requête et sa réponse. Pour vérifier le comportement d'une application web telle qu'un utilisateur la voit, utilisez un scénario navigateur (chapitres 3 à 6 du guide de développement des scénarios).

4.6.2 Paramètres de la requête

Paramètre	Obligatoire	Description	Défaut
`http.url`	Oui	URL complète, `http://` ou `https://` (2 048 caractères maximum). Une adresse IPv6 littérale s'écrit entre crochets (§ 4.1.3)	—
`http.method`	Non	`GET`, `HEAD`, `POST`, `PUT`, `PATCH`, `DELETE`, `OPTIONS`	`GET`
`http.headers`	Non	En-têtes de la requête (32 au maximum). Les valeurs sensibles référencent un secret : `${secret:NOM}`	Aucun
`http.body`	Non	Corps de la requête (64 Kio maximum), pour `POST`, `PUT` et `PATCH`	Aucun
`http.followRedirects`	Non	Suivre les redirections (codes 301, 302, 303, 307, 308)	`true`
`http.tls.verify`	Non	Vérifier le certificat du serveur (chaîne de confiance et nom)	`true`
`http.tls.caRef`	Non	Restreint la confiance de ce contrôle à une seule autorité de certification importée dans la section `trust` (§ 5.7.3 du guide d'installation et d'administration). Sans cette propriété, le magasin complet de la sonde s'applique	Aucune
`http.tls.certificateExpiryWarningDays`	Non	Nombre de jours avant l'expiration du certificat en dessous duquel l'état est `WARNING` (0 à 365)	Aucun
`http.tls.certificateExpiryCriticalDays`	Non	Nombre de jours avant l'expiration du certificat en dessous duquel l'état est `CRITICAL` (0 à 365)	Aucun
`http.proxy`	Non	`"none"` : le contrôle contourne le proxy global de la sonde (chapitre 5 du guide d'installation et d'administration). C'est la seule valeur admise : un contrôle ne peut pas définir un autre proxy	Proxy global, avec ses exclusions
`http.assertions`	Non	Liste des vérifications à appliquer à la réponse (20 au maximum)	Voir § 4.6.4

Le délai maximal de la requête est fixé par le paramètre commun `timeout` du contrôle (chapitre 3).

4.6.3 Méthodes HTTP

Méthode	Usage en supervision	Précaution
`GET`	Cas général : récupérer une page ou une ressource	Aucune
`HEAD`	Vérifier l'existence d'une ressource et ses en-têtes sans télécharger le contenu : économie de bande passante pour les gros fichiers	Certains serveurs traitent mal `HEAD` ; les assertions sur le corps sont alors impossibles

Méthode	Usage en supervision	Précaution
`POST`	Interroger un service de recherche, un web service SOAP, un point d'authentification	Vérifier que la requête **ne crée ni ne modifie** de données
`PUT`, `PATCH`, `DELETE`	Rarement utiles ; à réserver aux environnements de test	Ces méthodes modifient des données : **ne jamais les utiliser en production**
`OPTIONS`	Vérifier les méthodes autorisées ou la configuration CORS	Aucune

> **Avertissement** — Une requête de supervision est répétée en permanence. Une requête `POST` qui crée une commande, un ticket ou un compte, exécutée toutes les minutes, produit plus de 1 400 enregistrements par jour. N'utilisez que des requêtes sans effet de bord, ou des environnements de test dédiés (§ 5.9).

4.6.4 Assertions

Une **assertion** est une condition que la réponse doit satisfaire. Si au moins une assertion échoue, l'état est `CRITICAL` avec le type d'erreur `assertion_failed` (ou `http_status` pour une assertion sur le code de statut).

Chaque assertion comporte :

- un **type** (`kind`) ;
- un **opérateur** (`operator`) ;
- une **valeur attendue** (`value`) ;
- selon le type, un **chemin** (`path`).

Type (`kind`)	`path`	Ce qui est comparé	Exemple
`status`	—	Code de statut HTTP de la réponse finale	Égal à `200`
`header`	Nom de l'en-tête	Valeur de l'en-tête de réponse	`Content-Type` contient `application/json`
`body`	—	Corps de la réponse, en texte	Contient `Bienvenue`
`jsonPath`	Expression JSONPath	Valeur extraite d'une réponse JSON	`$.status` égal à `UP`
`xpath`	Expression XPath	Valeur extraite d'une réponse XML	`//*[local-name()='etat']` égal à `actif`
`duration`	—	Durée totale de la requête, en millisecondes	Inférieure à `2000`

Opérateurs disponibles :

Opérateur	Signification	Types de valeurs
`eq` / `ne`	Égal / différent	Nombre, texte, booléen
`lt` / `lte`	Strictement inférieur / inférieur ou égal	Nombre
`gt` / `gte`	Strictement supérieur / supérieur ou égal	Nombre
`contains` / `notContains`	Contient / ne contient pas	Texte
`matches`	Correspond à une expression régulière	Texte (expression régulière)
`exists`	L'élément est présent (en-tête, chemin JSON ou XML)	Aucune valeur requise

Une **expression JSONPath** désigne une valeur dans un document JSON : `$` représente la racine, `$.status` le champ `status` de premier niveau, `$.items[0].id` le champ `id` du premier élément du tableau `items`. Une **expression XPath** joue le même rôle pour les documents XML.

> **Information** — Si aucune assertion de type `status` n'est définie, une réponse dont le code est supérieur ou égal à 400 est considérée comme un échec (`CRITICAL` , `http_status`). Définissez toujours une assertion `status` explicite : le comportement du contrôle est alors sans ambiguïté.

4.6.5 Phases mesurées

Pour chaque exécution, la sonde décompose la durée de la requête en phases. Cette décomposition est l'outil principal de diagnostic d'une lenteur.

Phase	Mesure	Une valeur élevée indique…
`dns`	Résolution du nom d'hôte	Un serveur DNS lent ou surchargé
`connect`	Établissement de la connexion TCP	Une latence réseau, une saturation, un pare-feu lent
`tls`	Négociation TLS (HTTPS uniquement)	Un serveur surchargé en calcul cryptographique, une chaîne de certificats longue
`ttfb`	Délai jusqu'au premier octet de réponse (*Time To First Byte*), une fois la requête envoyée	Un traitement lent côté serveur : application, base de données
`transfer`	Réception du corps de la réponse	Une réponse volumineuse ou une bande passante insuffisante

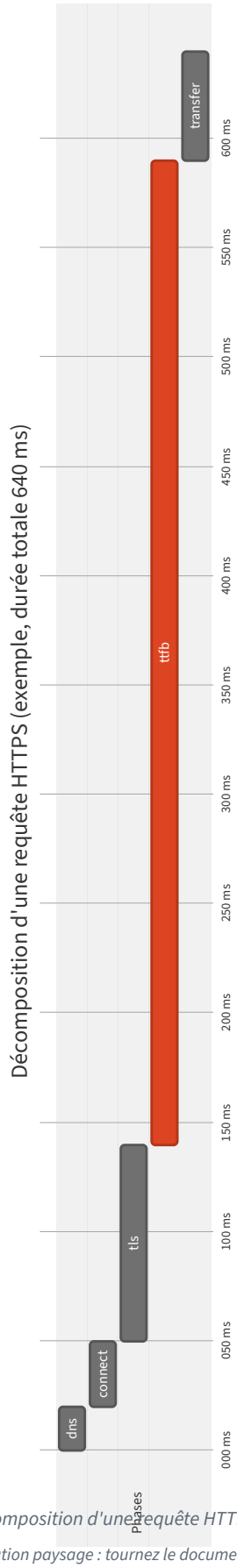


Schéma 4-2 — Décomposition d'une requête HTTPS en phases (exemple)

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

Dans cet exemple, l'essentiel du temps est consommé par la phase `ttfb` : le serveur met 450 ms à produire sa réponse. Le réseau n'est pas en cause. Ces phases sont publiées sous la métrique `sva.http.phase.duration` (chapitre 2 du guide d'intégration et d'observabilité).

4.6.6 Redirections

Par défaut (`followRedirects: true`), la sonde suit les redirections, et les assertions portent sur la **réponse finale**. Pour vérifier la redirection elle-même, par exemple pour garantir que le site non chiffré renvoie bien vers HTTPS, désactivez le suivi et vérifiez le code et l'en-tête `Location` (§ 4.6.10).

4.6.7 Certificats et expiration

Pour une URL en `https://`, la sonde vérifie le certificat présenté par le serveur :

Vérification	Échec	État
Le certificat est signé par une autorité de confiance (magasin de la sonde ou `caRef`)	Certificat autosigné ou autorité inconnue	`CRITICAL` / `tls_error`
Le nom du serveur correspond au certificat	Certificat émis pour un autre nom	`CRITICAL` / `tls_error`
Le certificat est en cours de validité	Certificat expiré	`CRITICAL` / `certificate_expired`
L'expiration est au-delà du délai de prévenance	Expiration dans moins de `certificateExpiryWarningDays` jours	`WARNING`
L'expiration est au-delà du délai critique	Expiration dans moins de `certificateExpiryCriticalDays` jours	`CRITICAL`

La durée restante avant expiration est publiée en permanence dans la métrique `sva.tls.certificate.expiry` (en secondes), ce qui permet de construire un tableau de bord de tous les certificats supervisés.

> ✓ **Conseil** — Fixez `certificateExpiryWarningDays` à une valeur compatible avec votre processus de renouvellement. Pour des certificats renouvelés manuellement avec une validation administrative, 30 jours est un minimum. Pour des certificats renouvelés automatiquement, une valeur plus courte (7 à 14 jours) signale le dysfonctionnement du renouvellement automatique sans fausse alerte.

4.6.8 Sites internes et autorité de certification d'entreprise

Les applications internes sont souvent protégées par des certificats émis par l'autorité de certification (AC) de l'entreprise, inconnue du magasin standard de la sonde. Dans ce cas :

1. Importez le certificat de l'AC interne, au format PEM, dans la section `trust` des paramètres de la sonde (§ 5.7.3 du guide d'installation et d'administration). Ce magasin s'ajoute au magasin système et s'applique aux contrôles `http`, `api`, `tcp` (avec TLS) et `browser`, ainsi qu'aux exports.
2. Le cas échéant, référencez-la explicitement dans le contrôle avec `http.tls.caRef`.

> ! **Avertissement** — Le paramètre `tls.verify: false` désactive toute vérification du certificat. Le contrôle ne détecte plus alors une usurpation de serveur, ni l'expiration d'un certificat. Réservez cette option à un diagnostic temporaire, jamais à un contrôle permanent.

4.6.9 Exemple : page d'accueil d'un site

Ce contrôle vérifie que la page d'accueil du site fictif répond avec le code 200, contient le texte attendu, ne présente pas de message d'erreur applicatif, répond en moins de 3 secondes et dispose d'un certificat valide au moins 21 jours.

```json
{
  "id": "site-public-accueil",
  "name": "Site public : page d'accueil",
  "type": "http",
  "tags": { "application": "site_public", "environment": "production" },
  "schedule": { "interval": "1m", "jitter": "10s" },
  "timeout": "15s",
  "retry": { "attempts": 1, "interval": "30s" },
  "thresholds": { "warningDuration": "2s" },
  "http": {
    "method": "GET",
    "url": "https://www.example.com/",
    "headers": { "User-Agent": "Supervision-Sonde (exemple)" },
    "assertions": [
      { "kind": "status", "operator": "eq", "value": 200 },
      { "kind": "body", "operator": "contains", "value": "Bienvenue sur Example" },
      { "kind": "body", "operator": "notContains", "value": "Erreur interne" },
      { "kind": "duration", "operator": "lt", "value": 3000 }
    ],
    "tls": { "verify": true, "certificateExpiryWarningDays": 21 }
  }
}
```

Lecture de cet exemple :

- Au-delà de 2 secondes (`warningDuration`), l'état passe à `WARNING`. Au-delà de 3 secondes (assertion `duration`), il passe à `CRITICAL`.
- L'assertion `notContains` détecte les pages d'erreur renvoyées avec un code 200. Ce cas est fréquent lorsqu'un cadre applicatif intercepte les exceptions.
- L'en-tête `User-Agent` identifie la sonde dans les journaux du serveur web. Les équipes d'exploitation peuvent ainsi distinguer le trafic de supervision de celui des utilisateurs, et l'exclure de leurs statistiques d'audience.

4.6.10 Exemple : redirection vers HTTPS

```json
{
  "id": "site-public-redirection-https",
  "name": "Site public : redirection HTTP vers HTTPS",
  "type": "http",
  "tags": { "application": "site_public" },
  "schedule": { "interval": "15m" },
  "timeout": "10s",
  "http": {
    "method": "GET",
    "url": "http://www.example.com/",
    "followRedirects": false,
    "assertions": [
      { "kind": "status", "operator": "eq", "value": 301 },
      { "kind": "header", "path": "Location", "operator": "matches", "value":
    "^https://www\\.example\\.com/" }
    ]
  }
}
```

Dans la valeur JSON de l'expression régulière, le point est échappé deux fois (`\\.`) : une fois pour l'expression régulière, une fois pour la syntaxe JSON.

4.6.11 Exemple : point de contrôle de santé d'une application

De nombreuses applications exposent un point de contrôle de santé renvoyant un document JSON. L'exemple suivant interroge le point de santé fictif d'une application interne, protégée par une AC d'entreprise et par un jeton d'accès stocké comme secret.

```json
{
  "id": "crm-sante",
  "name": "CRM : point de contrôle de santé",
  "type": "http",
  "tags": { "application": "crm", "environment": "production" },
  "schedule": { "interval": "1m" },
  "timeout": "10s",
  "secrets": ["CRM_HEALTH_TOKEN"],
  "http": {
    "method": "GET",
    "url": "https://crm.example.internal/actuator/health",
    "headers": {
      "Accept": "application/json",
      "Authorization": "Bearer ${secret:CRM_HEALTH_TOKEN}"
    },
    "assertions": [
      { "kind": "status", "operator": "eq", "value": 200 },
      { "kind": "header", "path": "Content-Type", "operator": "contains", "value":
   "application/json" },
      { "kind": "jsonPath", "path": "$.status", "operator": "eq", "value": "UP" },
      { "kind": "jsonPath", "path": "$.components.db.status", "operator": "eq", "value": "UP" }
    ],
    "tls": { "verify": true, "caRef": "ac-interne-example", "certificateExpiryWarningDays": 30 }
  }
}
```

Le secret `CRM_HEALTH_TOKEN` est déclaré dans la liste `secrets` du contrôle. Il n'est transmis au moteur d'exécution que pour ce contrôle, et sa valeur n'apparaît ni dans les résultats ni dans les journaux (chapitre 6 du guide d'installation et d'administration).

> **Information** — Pour une authentification HTTP Basic, construisez l'en-tête `Authorization: Basic ${secret:NOM}`. Le secret contient alors la chaîne `utilisateur:motdepasse` encodée en Base64, et non le mot de passe en clair.

4.6.12 Exemple : téléchargement d'un fichier publié

La méthode `HEAD` vérifie la présence d'un fichier volumineux sans le télécharger. Ici, il s'agit d'un catalogue fictif publié chaque nuit.

```json
{
  "id": "catalogue-publie",
  "name": "Catalogue tarifaire : fichier publié",
  "type": "http",
  "tags": { "application": "catalogue" },
  "schedule": { "cron": "0 7-19 * * 1-5", "timezone": "Europe/Paris" },
  "timeout": "20s",
  "http": {
    "method": "HEAD",
    "url": "https://fichiers.example.com/catalogue/catalogue-courant.csv",
    "assertions": [
      { "kind": "status", "operator": "eq", "value": 200 },
      { "kind": "header", "path": "Content-Length", "operator": "exists" }
    ]
  }
}
```

4.6.13 Résultat attendu

```json
{
  "id": "01M7SH7FTQAA30DSWTVCMWPNPD",
  "checkId": "site-public-accueil",
  "checkRevision": 1,
  "status": "OK",
  "startedAt": "2025-11-19T09:12:00.087Z",
  "durationMs": 640,
  "measurements": {
    "http.status_code": 200,
    "http.dns_ms": 20,
    "http.connect_ms": 30,
    "http.tls_ms": 90,
    "http.ttfb_ms": 450,
    "http.transfer_ms": 50,
    "tls.certificate_days_remaining": 74
  }
}
```

En cas d'échec, le résultat précise l'assertion en cause. Exemple de message, consultable dans le détail de l'exécution (chapitre 7). Ce bloc est un **extrait (projection)** limité aux champs `status` et `error` ; la réponse complète est illustrée au § 7.4 :

```json
{
  "status": "CRITICAL",
  "error": {
    "type": "assertion_failed",
    "message": "Assertion body contains « Bienvenue sur Example » : texte absent de la réponse (28
    413 octets reçus)"
  }
}
```

4.6.14 Erreurs courantes

Symptôme	Cause probable	Action
`CRITICAL` / `tls_error` sur un site interne	Certificat émis par une AC interne non importée sur la sonde	Importer l'AC dans la section `trust` (§ 4.6.8)
`CRITICAL` / `tls_error` après une intervention	Chaîne de certificats incomplète sur le serveur (certificat intermédiaire manquant). Les navigateurs la compensent parfois, la sonde non	Corriger la configuration du serveur web pour qu'il présente la chaîne complète
`CRITICAL` / `http_status` avec le code 403	Le pare-feu applicatif (WAF) ou un filtrage géographique bloque la sonde	Faire autoriser l'adresse de la sonde ; ajouter un `User-Agent` identifiable
`CRITICAL` / `http_status` avec le code 407	Un proxy exige une authentification	Configurer le proxy de la sonde (chapitre 5 du guide d'installation et d'administration), ou l'exclure pour les destinations internes (liste d'exclusions, ou `http.proxy: "none"` dans le contrôle)
`CRITICAL` / `assertion_failed` sur `body` alors que la page s'affiche dans un navigateur	Contenu généré par JavaScript côté navigateur : absent de la réponse HTTP brute	Choisir un texte présent dans le HTML initial, ou utiliser un scénario navigateur (chapitres 3 à 6 du guide de développement des scénarios)
`CRITICAL` / `http_status` avec le code 429	Le serveur limite le nombre de requêtes	Réduire la fréquence du contrôle ; faire exempter la sonde des limites
Assertion `jsonPath` en échec alors que la valeur semble correcte	Type différent (nombre `200` contre texte `"200"`, booléen `true` contre texte `"true"`) ou chemin incorrect	Vérifier le type de la valeur dans la réponse brute affichée par le mode essai
États alternant entre `OK` et `WARNING`	`warningDuration` proche du temps de réponse habituel	Fixer le seuil à partir des percentiles observés (chapitre 7)
`CRITICAL` / `timeout` uniquement aux heures de pointe	Saturation de l'application ou du réseau	Analyser les phases ; c'est souvent un incident réel à signaler

4.7 Plugins Nagios (`nagios`)

4.7.1 Objectif et intérêt

Nagios est un logiciel de supervision historique. Son format de **plugins**, de petits programmes autonomes qui réalisent une vérification et renvoient un résultat normalisé, est devenu un standard de fait, repris par Centreon, Icinga, Naemon, Shinken et de nombreux autres outils. Des milliers de plugins existent.

Le contrôle `nagios` permet à la sonde d'exécuter ces plugins. Ce type répond à deux besoins :

- **couvrir des protocoles** que les contrôles natifs ne traitent pas en profondeur : dialogue SMTP complet avec STARTTLS, LDAP avec recherche, temps NTP, SSH, FTP, IMAP… ;
- **réutiliser l'existant** : une entreprise qui dispose de plugins personnalisés pour ses applications peut les exécuter depuis la sonde, au plus près des utilisateurs (édition Professional).

4.7.2 Contrat d'un plugin Nagios

Un plugin respecte deux conventions.

1. Le code de retour indique l'état :

Code de retour	État Nagios	État de la sonde
0	OK	`OK`
1	WARNING	`WARNING`
2	CRITICAL	`CRITICAL`
3	UNKNOWN	`UNKNOWN`
Autre valeur, ou arrêt anormal	—	`UNKNOWN` (`plugin_error`)

2. La sortie standard contient une ligne de texte lisible, éventuellement suivie, après le caractère `|`, de **données de performance** (*perfdata*) :

```
SMTP OK - 0.112 sec. response time|time=0.112094s;;;0.000000
```
TEXTE

Chaque donnée de performance suit le format :

```
'libellé'=valeur[unité];[seuil_warning];[seuil_critical];[min];[max]
```
TEXTE

Par exemple, `rta=12.400000ms;100.000000;500.000000;0.000000` indique un temps d'aller-retour moyen (`rta`) de 12,4 ms, un seuil d'avertissement à 100 ms, un seuil critique à 500 ms et une valeur minimale de 0.

La sonde conserve la sortie texte dans le résultat de l'exécution (4 096 caractères au maximum, secrets masqués). Elle n'exporte sous forme de métriques que les données de performance explicitement listées dans `perfdataExport` (§ 4.7.5).

4.7.3 Plugins fournis

L'image de la sonde intègre les plugins du projet **Monitoring Plugins** (paquet Debian `monitoring-plugins`), le jeu de plugins standard de l'écosystème Nagios. La liste exacte des plugins présents dans l'image est indiquée dans les notes de version. Les plus utiles depuis une sonde sont les suivants.

Plugin	Usage	Options principales
`check_http` / `check_curl`	Vérification HTTP avancée (contenu, redirections, certificat)	`-H` hôte, `-u` chemin, `-S` HTTPS, `--sni`, `-s` texte attendu, `-C` jours avant expiration du certificat
`check_smtp`	Dialogue SMTP, y compris STARTTLS	`-H` hôte, `-p` port, `-S` STARTTLS, `-e` réponse attendue
`check_imap`, `check_pop`	Disponibilité des services de messagerie en réception	`-H` hôte, `-p` port, `-S` SSL
`check_ldap`	Connexion et recherche dans un annuaire LDAP	`-H` hôte, `-b` base de recherche, `-T` STARTTLS, `-S` LDAPS
`check_ssh`	Disponibilité et version d'un service SSH	`-H` hôte, `-p` port, `-r` version attendue
`check_ntp_time`	Écart de l'heure d'un serveur NTP par rapport à l'horloge locale	`-H` serveur, `-w` / `-c` seuils d'écart en secondes
`check_dig`	Requête DNS détaillée avec vérification de la réponse	`-H` serveur DNS, `-l` nom, `-T` type, `-a` réponse attendue

Plugin	Usage	Options principales
`check_ftp`	Disponibilité d'un serveur FTP	`-H` hôte, `-p` port
`check_ping`	Ping avec seuils de latence et de perte. Il dépend de la commande `ping`, qui exige des privilèges réseau absents de l'environnement isolé : préférez le contrôle natif `icmp` (§ 4.3)	`-H` hôte, `-w` / `-c` sous la forme `latence,perte%`

Chaque plugin documente ses options avec l'argument `--help`. La documentation de référence est publiée par le projet Monitoring Plugins (monitoring-plugins.org).

> **Attention** — Certains plugins requièrent des privilèges élevés, car ils utilisent des sockets bruts ou des ports réservés : `check_icmp` et `check_dhcp`, par exemple. L'environnement d'exécution isolé de la sonde ne dispose pas de ces privilèges, qui ne lui sont pas accordés afin de maintenir son isolation. `check_icmp` et `check_dhcp` ne sont donc pas pris en charge. Pour le ping, utilisez le contrôle natif `icmp` (§ 4.3). Les plugins qui vérifient les ressources **locales** d'un serveur (`check_disk`, `check_load`, `check_procs`…) n'ont pas de sens sur la sonde, puisqu'ils mesureraient la sonde elle-même et non vos serveurs.

4.7.4 Paramètres

Paramètre	Obligatoire	Description
`nagios.plugin`	Oui	Nom de l'exécutable, **sans chemin** (lettres, chiffres, `_`, `.`, `-` ; 64 caractères au maximum). La sonde le recherche uniquement dans les répertoires de plugins autorisés
`nagios.args`	Non	Liste des arguments, **un élément par argument** (32 au maximum, 512 caractères chacun)
`nagios.perfdataExport`	Non	Libellés des données de performance à convertir en métriques (10 au maximum)

4.7.5 Exécution sécurisée

La sonde exécute les plugins **sans interpréteur de commandes** (*shell*) : le programme est lancé directement, avec la liste d'arguments fournie. Ce choix de conception protège la sonde contre les injections de commandes, mais il a des conséquences pratiques :

- chaque argument est un élément distinct de la liste `args` : `["-H", "smtp.example.internal"]` et non `["-H smtp.example.internal"]` ;
- les constructions propres au shell sont **sans effet** : redirections (`>`, `2>&1`), enchaînements (`;`, `&&`), tubes (`|`), variables (`$HOME`), substitutions (`$(…)`) ;
- un plugin ne peut pas être désigné par un chemin arbitraire.

Les plugins s'exécutent dans l'environnement isolé de la sonde, sous un compte système dédié, sans accès à la base de données de la sonde ni aux secrets non déclarés. Leur consommation de mémoire et de processeur est limitée.

Délais. Le timeout du contrôle (paramètre commun `timeout`) interrompt le plugin s'il ne s'est pas terminé. La plupart des plugins disposent aussi de leur propre délai, l'option `-t`. Fixez-le **inférieur** au timeout du contrôle : le plugin s'arrête alors proprement et renvoie un message explicite, au lieu d'être interrompu par la sonde.

Secrets. La référence `${secret:NOM}` est substituée dans les arguments `nagios.args` : elle permet de transmettre un mot de passe à un plugin. Le secret doit être déclaré dans la liste `secrets` du contrôle.

> **Avertissement** — Un argument de ligne de commande peut être visible par d'autres processus du système pendant l'exécution du plugin. Lorsqu'un plugin propose de lire ses identifiants depuis un fichier ou une variable d'environnement, préférez cette méthode. Dans tous les cas, utilisez des comptes de supervision aux droits minimaux.

Conversion des données de performance. Pour chaque libellé listé dans `perfdataExport`, la sonde publie la valeur sous la métrique `sva.plugin.perfdata`, avec l'attribut `sva.perfdata.label` (chapitre 2 du guide d'intégration et d'observabilité). Les libellés non listés restent visibles dans la sortie texte, mais ne produisent pas de métrique. Cette sélection explicite évite qu'un plugin produisant des libellés variables (un par fichier, par utilisateur…) ne multiplie le nombre de séries de métriques.

4.7.6 Exemple : messagerie sortante avec STARTTLS

JSON

```
{
  "id": "smtp-soumission-starttls",
  "name": "Messagerie : soumission SMTP avec STARTTLS",
  "type": "nagios",
  "tags": { "application": "messagerie" },
  "schedule": { "interval": "5m" },
  "timeout": "30s",
  "nagios": {
    "plugin": "check_smtp",
    "args": ["-H", "smtp.example.internal", "-p", "587", "-S", "-w", "2", "-c", "5", "-t", "20"],
    "perfdataExport": ["time"]
  }
}
```

Le plugin se connecte au port 587, négocie le chiffrement par STARTTLS (`-S`), et renvoie `WARNING` au-delà de 2 secondes et `CRITICAL` au-delà de 5 secondes. Son propre délai (`-t 20`) est inférieur au timeout du contrôle (30 secondes).

4.7.7 Exemple : heure d'un serveur NTP interne

Une dérive de l'heure perturbe l'authentification Kerberos, la validité des jetons et la corrélation des journaux. Ce contrôle vérifie que le serveur NTP de l'entreprise délivre une heure cohérente avec celle de la sonde.

JSON

```
{
  "id": "ntp-interne-ecart",
  "name": "Serveur NTP interne : écart de temps",
  "type": "nagios",
  "tags": { "domaine": "infrastructure" },
  "schedule": { "interval": "15m" },
  "timeout": "20s",
  "nagios": {
    "plugin": "check_ntp_time",
    "args": ["-H", "ntp1.example.internal", "-w", "0.5", "-c", "1", "-t", "10"],
    "perfdataExport": ["offset"]
  }
}
```

> **Information** — Ce contrôle compare l'heure du serveur NTP à celle de la sonde, elle-même synchronisée par NTP (chapitre 5 du guide d'installation et d'administration). Si la sonde utilise ce même serveur comme unique source de temps, l'écart sera toujours faible. Pour un contrôle significatif, synchronisez la sonde sur une source indépendante, ou comparez plusieurs serveurs entre eux.

4.7.8 Exemple : recherche dans un annuaire LDAP

```json
{
  "id": "annuaire-recherche",
  "name": "Annuaire : connexion et recherche LDAP",
  "type": "nagios",
  "tags": { "application": "annuaire" },
  "schedule": { "interval": "5m" },
  "timeout": "30s",
  "nagios": {
    "plugin": "check_ldap",
    "args": ["-H", "ldap.example.internal", "-b", "dc=example,dc=internal", "-T", "-w", "1", "-c",
    "3", "-t", "20"],
    "perfdataExport": ["time"]
  }
}
```

4.7.9 Résultat attendu

```json
{
  "id": "01M7SHP4MTG5ZRSMPT8FDSNWZ5",
  "checkId": "smtp-soumission-starttls",
  "checkRevision": 1,
  "status": "OK",
  "startedAt": "2025-11-19T09:20:00.154Z",
  "durationMs": 214,
  "output": "SMTP OK - 0.112 sec. response time",
  "measurements": {
    "perfdata.time": 0.112094
  }
}
```

4.7.10 Plugins personnalisés (édition Professional)

À partir de l'édition Professional, un administrateur peut ajouter ses propres plugins. Le plugin se dépose par `POST /api/v1/plugins?name=<nom>` : le corps de la requête contient le fichier exécutable (`application/octet-stream`, 1 Mo au maximum). Le nom doit commencer par `check_` et ne contenir que des minuscules, des chiffres et des tirets bas (`check_[a-z0-9_]`, 40 caractères au plus après le préfixe). Un script doit commencer par sa ligne d'interpréteur (`#!/bin/sh` ou `#!/usr/bin/env python3`). La liste des plugins déposés s'obtient par `GET /api/v1/plugins`, et `DELETE /api/v1/plugins/{name}` retire un plugin ; ce retrait est refusé (`409`) tant qu'un contrôle l'utilise. Chaque dépôt et chaque retrait sont journalisés. Le dépôt et le retrait d'un plugin personnalisé sont réservés au rôle `admin`, car ils reviennent à installer ou à modifier du code exécutable sur le réseau de l'entreprise (chapitre 6 du guide d'installation et d'administration) : un utilisateur `operator` reçoit une réponse `403`.

Un plugin personnalisé doit respecter les règles suivantes :

Règle	Justification
Respecter les codes de retour 0 à 3 et le format de sortie du § 4.7.2	Interprétation correcte de l'état
Être non interactif : aucune saisie au clavier, aucune confirmation	La sonde n'attend aucune interaction
Se terminer de lui-même avant le timeout du contrôle	Éviter les interruptions forcées (`CRITICAL`, `timeout`)
Ne dépendre que des interpréteurs et bibliothèques présents dans l'image (interpréteurs `sh`, fourni par `dash`, et `python3` ; utilitaire `curl`)	L'environnement isolé n'installe aucun paquet supplémentaire

Règle	Justification
N'écrire aucun fichier en dehors de son répertoire temporaire	Isolation et protection de la carte microSD
Ne jamais afficher de secret dans sa sortie	La sortie est stockée et peut être exportée
Produire des libellés de perfdata **fixes**	Maîtrise de la cardinalité des métriques

Squelette pédagogique d'un plugin en shell POSIX, qui vérifie l'âge d'un horodatage publié par une application sur une URL texte. Exemple fictif :

SHELL

```
#!/bin/sh
# check_fraicheur_export — exemple pédagogique de plugin personnalisé.
# Usage : check_fraicheur_export <URL> <seuil_warning_s> <seuil_critical_s>
# L'URL renvoie l'horodatage Unix (en secondes) du dernier export réussi.

URL="$1"; WARN="$2"; CRIT="$3"

if [ -z "$URL" ] || [ -z "$WARN" ] || [ -z "$CRIT" ]; then
  echo "UNKNOWN - arguments manquants"
  exit 3
fi

# Récupération de l'horodatage (délai maximal 10 s)
DERNIER=$(curl -fsS --max-time 10 "$URL") || {
  echo "CRITICAL - impossible de lire $URL"
  exit 2
}

case "$DERNIER" in
  ''|*[!0-9]*) echo "UNKNOWN - réponse inattendue"; exit 3 ;;
esac

AGE=$(( $(date +%s) - DERNIER ))
PERF="age=${AGE}s;${WARN};${CRIT};0"

if [ "$AGE" -ge "$CRIT" ]; then
  echo "CRITICAL - dernier export il y a ${AGE} s|${PERF}"; exit 2
elif [ "$AGE" -ge "$WARN" ]; then
  echo "WARNING - dernier export il y a ${AGE} s|${PERF}"; exit 1
fi
echo "OK - dernier export il y a ${AGE} s|${PERF}"
exit 0
```

Ce squelette utilise l'utilitaire `curl`, présent dans l'environnement d'exécution isolé. Le contrôle correspondant :

JSON

```
{
  "id": "export-comptable-fraicheur",
  "name": "Export comptable : fraîcheur du dernier export",
  "type": "nagios",
  "tags": { "application": "comptabilite" },
  "schedule": { "interval": "10m" },
  "timeout": "30s",
  "nagios": {
    "plugin": "check_fraicheur_export",
    "args": ["https://compta.example.internal/export/dernier", "7200", "14400"],
    "perfdataExport": ["age"]
  }
}
```

4.7.11 Erreurs courantes

Symptôme	Cause probable	Action
`UNKNOWN` / `plugin_error` dès la première exécution	Plugin introuvable dans les répertoires autorisés, nom mal orthographié, fichier non exécutable	Vérifier le nom exact ; pour un plugin personnalisé, vérifier son dépôt et ses droits d'exécution
`UNKNOWN` avec un message d'usage du plugin	Arguments regroupés dans un seul élément de `args`	Séparer chaque option et chaque valeur en éléments distincts
Le plugin fonctionne en ligne de commande sur un serveur, mais pas sur la sonde	Dépendance absente de l'image, privilèges requis, ou construction de shell utilisée (tube, redirection)	Vérifier les dépendances ; réécrire l'appel sans shell ; préférer un contrôle natif
`CRITICAL` / `timeout` après exactement la durée du timeout	Plugin bloqué, ou délai du plugin supérieur au timeout du contrôle	Fixer l'option `-t` du plugin en dessous du timeout du contrôle
Aucune métrique de perfdata dans la plateforme d'observabilité	Libellé absent de `perfdataExport`, ou orthographe différente (sensible à la casse)	Copier le libellé exact depuis la sortie du plugin, visible dans le résultat de l'exécution
Sortie tronquée	Sortie supérieure à 4 096 caractères	Réduire la verbosité du plugin ; seule la première ligne est utile à l'état

4.8 Synthèse

Besoin	Contrôle recommandé	Section
Joignabilité d'un équipement réseau	`icmp`	§ 4.3
Service non HTTP en écoute	`tcp`	§ 4.4
Bannière ou dialogue texte simple	`tcp` avec `send` / `expect`	§ 4.4.5
Service chiffré dès la connexion	`tcp` avec `tls`	§ 4.4.7
Service chiffré par STARTTLS	`nagios` (`check_smtp -S`, `check_ldap -T`…)	§ 4.7
Résolution d'un nom, valeur d'un enregistrement	`dns`	§ 4.5
Page web, point de santé, redirection	`http`	§ 4.6
Expiration des certificats	`http` avec `certificateExpiryWarningDays`	§ 4.6.7
Protocole spécifique couvert par un plugin standard	`nagios`	§ 4.7.3
Vérification propre à l'entreprise	`nagios`, plugin personnalisé (Professional)	§ 4.7.10
Réponse fonctionnelle d'une API, enchaînement de requêtes	`http`, `api`	Chapitre 5
Cohérence des données métier	`database`	Chapitre 5
Parcours utilisateur complet	`browser`	Chapitres 3 à 6 du guide de développement des scénarios

Chapitre 5 — Contrôles applicatifs et métier

Les contrôles techniques du chapitre 4 vérifient que les services répondent. Ce chapitre traite d'une question différente : les applications **rendent-elles le service attendu** ? Une API qui répond `200` avec un résultat erroné, une base de données accessible dont les traitements nocturnes n'ont pas tourné, une activité commerciale qui s'effondre sans qu'aucun serveur ne soit en panne : ces situations échappent à la supervision technique et sont précisément celles que les contrôles applicatifs et métier permettent de détecter.

Ce chapitre présente :

- la supervision d'API avec le contrôle `http` et avec le contrôle `api`, qui enchaîne plusieurs requêtes ;
- la supervision de web services SOAP ;
- la conception de contrôles fonctionnels robustes ;
- les contrôles de bases de données (`database`) ;
- la publication d'indicateurs métier et la détection d'anomalies.

Les parcours dans un navigateur, troisième niveau de la supervision applicative, font l'objet des chapitres 3 à 6 du guide de développement des scénarios.

> **Information** — Les contrôles `api` et `database` requièrent l'édition **Professional** ou supérieure. Le contrôle `http` avec assertions de contenu est disponible dès l'édition Essential. Tous les exemples de ce chapitre sont fictifs : domaines en `example.com` et `example.internal`, comptes et données de test.

5.1 De la disponibilité technique à la réussite fonctionnelle

Une application peut être **techniquement disponible** sans être **fonctionnellement utilisable**. Le tableau suivant illustre l'écart entre ces deux niveaux et le type de contrôle qui permet de le détecter.

Situation	Contrôle technique	Contrôle applicatif ou métier	Contrôle adapté
L'API de stock répond `200`, mais tous les articles apparaissent indisponibles à la suite d'une erreur de synchronisation	`OK` : le serveur répond	`CRITICAL` : l'article témoin, toujours en stock, apparaît indisponible	`http` + `jsonPath`
L'authentification par jeton est cassée après une mise à jour du serveur d'identité	`OK` : les deux serveurs répondent	`CRITICAL` : l'obtention du jeton ou l'appel authentifié échoue	`api`
Le web service d'annuaire répond, mais ne trouve plus aucune adresse	`OK` : réponse SOAP valide	`CRITICAL` : l'adresse témoin n'est plus trouvée	`http` + `xpath`
Le traitement nocturne de facturation n'a pas tourné	`OK` : la base répond	`CRITICAL` : la dernière exécution réussie date de plus de 26 heures	`database`
Les commandes chutent de 70 % à cause d'un moyen de paiement défaillant chez un prestataire	`OK` : tous les serveurs fonctionnent	`CRITICAL` : le volume de commandes de l'heure est anormalement bas	`database` + indicateur métier

Situation	Contrôle technique	Contrôle applicatif ou métier	Contrôle adapté
Des messages s'accumulent dans une file de traitement	`OK` : le serveur de files répond	`WARNING` à partir de 100 messages en attente depuis plus de 15 minutes, `CRITICAL` à partir de 500 (§ 5.6.8)	`database` ou `http` sur l'API d'administration de la file

La supervision applicative **complète** la supervision technique, elle ne la remplace pas. Lorsqu'un contrôle applicatif échoue, les contrôles techniques du même service indiquent si la cause est une panne d'infrastructure (§ 4.2.2) ou un dysfonctionnement purement fonctionnel.

5.2 Superviser une API REST avec le contrôle `http`

5.2.1 Principe

Une **API REST** (*Representational State Transfer*) est une interface de programmation accessible par HTTP, qui échange généralement des documents JSON. La plupart des applications modernes en exposent une, à destination de leur interface web, de leurs applications mobiles ou de partenaires.

Le contrôle `http`, décrit en détail au chapitre 4 (§ 4.6), suffit lorsqu'**une seule requête** permet de vérifier le bon fonctionnement : consultation d'une ressource témoin, recherche, calcul. Sa valeur applicative tient aux **assertions sur le contenu** : il ne vérifie pas seulement que l'API répond, mais qu'elle répond **juste**.

5.2.2 Choisir une donnée témoin

Un contrôle fonctionnel efficace repose sur une **donnée témoin** : un objet dont vous connaissez la valeur attendue, stable dans le temps et réservé à la supervision. Par exemple :

- un article témoin, toujours en stock, au prix fixe ;
- un client de test, avec une adresse connue ;
- un compte de démonstration, avec un solde figé ;
- une recherche dont le résultat est connu à l'avance.

> ✓ **Conseil** — Créez les données témoins en accord avec les équipes applicatives et documentez-les. Une donnée témoin supprimée ou modifiée lors d'un nettoyage provoque une fausse alerte. Une donnée témoin trop « réelle » peut fausser les statistiques métier : une commande de test comptabilisée dans le chiffre d'affaires, par exemple. Les applications prévoient souvent un marqueur pour exclure ces données des rapports.

5.2.3 Exemple : consultation d'une ressource témoin

L'API de stock fictive doit indiquer que l'article témoin `ARTICLE-TEMOIN` est disponible, avec une quantité positive et un prix inchangé.

```json
{
  "id": "api-stock-article-temoin",
  "name": "API de stock : article témoin",
  "description": "Vérifie que l'API de stock renvoie l'état connu de l'article témoin (exemple
    fictif).",
  "type": "http",
  "tags": { "application": "stock", "environment": "production" },
  "schedule": { "interval": "1m" },
```

```
  "timeout": "10s",
  "retry": { "attempts": 1, "interval": "20s" },
  "thresholds": { "warningDuration": "800ms" },
  "secrets": ["API_STOCK_TOKEN"],
  "http": {
    "method": "GET",
    "url": "https://api.example.com/v1/stock/ARTICLE-TEMOIN",
    "headers": {
      "Accept": "application/json",
      "Authorization": "Bearer ${secret:API_STOCK_TOKEN}"
    },
    "assertions": [
      { "kind": "status", "operator": "eq", "value": 200 },
      { "kind": "header", "path": "Content-Type", "operator": "contains", "value":
   "application/json" },
      { "kind": "jsonPath", "path": "$.reference", "operator": "eq", "value": "ARTICLE-TEMOIN" },
      { "kind": "jsonPath", "path": "$.available", "operator": "eq", "value": true },
      { "kind": "jsonPath", "path": "$.quantity", "operator": "gt", "value": 0 },
      { "kind": "jsonPath", "path": "$.price.amount", "operator": "eq", "value": 19.9 },
      { "kind": "duration", "operator": "lt", "value": 2000 }
    ]
  }
}
```

Chaque assertion couvre un risque différent :

Assertion	Défaillance détectée
`status` égal à `200`	Erreur serveur, ressource introuvable, refus d'accès
`Content-Type` contient `application/json`	Page d'erreur HTML renvoyée par un proxy ou un répartiteur de charge à la place de l'API
`$.reference`	Réponse portant sur un autre objet : erreur de routage, cache corrompu
`$.available` et `$.quantity`	Données de stock incohérentes, synchronisation défaillante
`$.price.amount`	Erreur de tarification, chargement d'un mauvais référentiel de prix
`duration` inférieure à 2 000 ms	Dégradation sévère des performances

5.2.4 Exemple : recherche par requête POST

Certaines API de recherche attendent un corps de requête JSON. La requête suivante recherche un terme dont le résultat est connu. Elle ne crée et ne modifie aucune donnée.

```json
{
  "id": "api-catalogue-recherche",
  "name": "API catalogue : recherche d'un terme témoin",
  "type": "http",
  "tags": { "application": "catalogue", "environment": "production" },
  "schedule": { "interval": "5m" },
  "timeout": "15s",
  "secrets": ["API_CATALOGUE_KEY"],
  "http": {
    "method": "POST",
    "url": "https://api.example.com/v2/catalogue/recherche",
    "headers": {
      "Content-Type": "application/json",
      "Accept": "application/json",
      "X-Api-Key": "${secret:API_CATALOGUE_KEY}"
    },
    "body": "{\"terme\": \"lampe de bureau témoin\", \"page\": 1, \"taille\": 10}",
    "assertions": [
      { "kind": "status", "operator": "eq", "value": 200 },
      { "kind": "jsonPath", "path": "$.total", "operator": "gte", "value": 1 },
      { "kind": "jsonPath", "path": "$.resultats[0].reference", "operator": "eq", "value": "LAMPE-
    TEMOIN-01" },
      { "kind": "body", "operator": "notContains", "value": "\"erreurs\"" }
    ]
  }
}
```

Le corps de la requête est une chaîne de caractères : les guillemets du document JSON envoyé sont échappés (`\"`).

5.2.5 Validation du contenu : choisir le bon opérateur

Besoin	Opérateur	Exemple
Valeur exacte connue et stable	`eq`	Référence, code de statut métier, booléen
Valeur numérique dans une plage	`gt`, `gte`, `lt`, `lte` (une assertion par borne)	Quantité positive, délai de livraison inférieur à 5 jours
Présence d'un champ, sans contrainte sur sa valeur	`exists`	Un identifiant de transaction est fourni
Format attendu, valeur variable	`matches`	Date au format `^\d{4}-\d{2}-\d{2}$`, identifiant de type `^CMD-[0-9]+$`
Absence d'un message d'erreur	`notContains` (sur `body`)	Pas de champ `"erreurs"`, pas de `"exception"`
Valeur différente d'un état anormal	`ne`	`$.mode` différent de `"maintenance"`

> **Attention** — N'écrivez pas d'assertion `eq` sur une valeur qui change légitimement : horodatage, identifiant généré, compteur, prix soumis à promotion. Le contrôle échouerait à chaque évolution normale. Utilisez `exists`, `matches` ou une plage numérique.

5.3 Enchaîner des requêtes avec le contrôle `api`

5.3.1 Principe

Certaines vérifications nécessitent **plusieurs requêtes successives**. La réponse de l'une fournit une donnée indispensable à la suivante. Cas typiques :

- obtenir un **jeton d'accès** auprès d'un serveur d'autorisation, puis appeler l'API protégée ;
- rechercher un objet, puis consulter son détail à partir de l'identifiant renvoyé ;
- vérifier la cohérence entre deux services : le stock affiché par le catalogue doit correspondre à celui de l'entrepôt.

Le contrôle `api` exécute une liste ordonnée de requêtes, de 1 à 20. Chaque requête :

- porte un **nom** (`name`), qui devient une **étape** du résultat : vous savez ainsi précisément quelle requête a échoué ;
- accepte tous les paramètres d'une requête `http` (§ 4.6.2), y compris ses propres assertions ;
- peut **extraire des variables** de sa réponse (`extract`), réutilisables dans les requêtes suivantes avec la syntaxe `${var:nom}`.

Le contournement du proxy global (`proxy: "none"`, § 4.6.2) se déclare pour l'ensemble du contrôle, dans `api.proxy`, et non requête par requête. De même, le paramètre commun `ipFamily` (§ 3.1.3) s'applique à toutes les requêtes du contrôle ; les cibles IPv6 suivent les mêmes règles que pour le contrôle `http` (§ 4.1.3).

Dès qu'une requête échoue (erreur réseau ou assertion non satisfaite), la suite est interrompue et le contrôle passe en `CRITICAL`. L'étape en échec et le type d'erreur sont indiqués dans le résultat.

5.3.2 Extraction de variables

Le paramètre `extract` associe un nom de variable à une expression, dont la nature est déclarée explicitement dans le champ `type` :

- `jsonpath` : expression **JSONPath**, pour une réponse JSON ;
- `xpath` : expression **XPath**, pour une réponse XML ;
- `regex` : **expression régulière** appliquée au corps de la réponse en texte. Si l'expression comporte un groupe de capture, la valeur extraite est celle du premier groupe ; sinon, c'est la totalité de la correspondance. Seule la première correspondance est retenue.

Champ	Obligatoire	Description	Défaut
`type`	Oui	`jsonpath`, `xpath` ou `regex`	—
`expression`	Oui	Expression d'extraction	—
`reveal`	Non	Si `true`, la valeur extraite apparaît en clair dans les résultats et les journaux	`false`

> **Attention** — Les deux graphies diffèrent par la casse : `"type": "jsonpath"` (tout en minuscules) dans `extract`, mais `"kind": "jsonPath"` (avec un P majuscule) dans les assertions. Une graphie inversée est refusée à l'enregistrement du contrôle.

```json
"extract": {
  "jeton": { "type": "jsonpath", "expression": "$.access_token" },
  "idCommande": { "type": "jsonpath", "expression": "$.commandes[0].id", "reveal": true }
}
```

Les variables extraites sont disponibles dans l'URL, les en-têtes et le corps des requêtes **suivantes** de la même exécution. Elles ne sont pas conservées d'une exécution à l'autre.

> **Information** — Une variable extraite peut contenir une donnée sensible, comme un jeton d'accès. La sonde masque dans les résultats et les journaux les valeurs des secrets déclarés, ainsi que, par défaut, celles des variables extraites. Réservez `"reveal": true` aux valeurs non sensibles utiles au diagnostic, comme un identifiant de commande témoin. Ne conservez dans les assertions et les messages que ce qui est nécessaire au diagnostic.

5.3.3 Exemple : jeton OAuth 2.0 puis appel d'une API protégée

OAuth 2.0 est le protocole d'autorisation le plus répandu pour les API. Dans le flux dit *client credentials*, une application s'authentifie auprès d'un serveur d'autorisation avec un identifiant et un secret client, puis obtient un **jeton d'accès** de courte durée. Ce jeton accompagne ensuite chaque appel à l'API.

Ce flux implique deux composants distincts : le serveur d'autorisation et l'API. Il peut échouer à chaque étape. Le contrôle `api` les vérifie ensemble, en séparant clairement les responsabilités.

JSON

```
{
  "id": "api-commandes-oauth",
  "name": "API commandes : authentification OAuth 2.0 et consultation",
  "description": "Obtient un jeton (client credentials) puis consulte la commande témoin (exemple
    fictif).",
  "type": "api",
  "tags": { "application": "commandes", "environment": "production" },
  "schedule": { "interval": "5m" },
  "timeout": "30s",
  "retry": { "attempts": 1, "interval": "1m" },
  "thresholds": { "warningDuration": "3s" },
  "secrets": ["OAUTH_CLIENT_ID", "OAUTH_CLIENT_SECRET"],
  "api": {
    "requests": [
      {
        "name": "Obtenir un jeton d'accès",
        "method": "POST",
        "url": "https://auth.example.com/oauth2/token",
        "headers": { "Content-Type": "application/x-www-form-urlencoded" },
        "body":
    "grant_type=client_credentials&client_id=${secret:OAUTH_CLIENT_ID}&client_secret=${secret:OAUTH_C
    LIENT_SECRET}&scope=commandes.lecture",
        "assertions": [
          { "kind": "status", "operator": "eq", "value": 200 },
          { "kind": "jsonPath", "path": "$.access_token", "operator": "exists" },
          { "kind": "jsonPath", "path": "$.token_type", "operator": "eq", "value": "Bearer" }
        ],
        "extract": { "jeton": { "type": "jsonpath", "expression": "$.access_token" } }
      },
      {
        "name": "Rechercher la commande témoin",
        "method": "GET",
        "url": "https://api.example.com/v1/commandes?reference=CMD-TEMOIN-0001",
        "headers": {
          "Accept": "application/json",
          "Authorization": "Bearer ${var:jeton}"
        },
        "assertions": [
          { "kind": "status", "operator": "eq", "value": 200 },
          { "kind": "jsonPath", "path": "$.total", "operator": "eq", "value": 1 }
        ],
        "extract": { "idCommande": { "type": "jsonpath", "expression": "$.commandes[0].id",
    "reveal": true } }
```

```
      },
      {
        "name": "Consulter le détail de la commande",
        "method": "GET",
        "url": "https://api.example.com/v1/commandes/${var:idCommande}",
        "headers": {
          "Accept": "application/json",
          "Authorization": "Bearer ${var:jeton}"
        },
        "assertions": [
          { "kind": "status", "operator": "eq", "value": 200 },
          { "kind": "jsonPath", "path": "$.reference", "operator": "eq", "value": "CMD-TEMOIN-0001"
    },
          { "kind": "jsonPath", "path": "$.lignes[0].quantite", "operator": "eq", "value": 2 },
          { "kind": "duration", "operator": "lt", "value": 1500 }
        ]
      }
    ]
  }
}
```

> **Attention** — Dans un corps `application/x-www-form-urlencoded`, les secrets sont substitués tels quels. Si une valeur contient des caractères réservés (`&`, `=`, `+`, `%`, espace), enregistrez le secret sous sa forme déjà encodée (par exemple `%2B` pour `+`), faute de quoi le serveur d'autorisation recevra une valeur tronquée ou altérée.

Lecture de l'exemple :

1. **Première requête.** Le jeton est demandé avec le périmètre (*scope*) minimal `commandes.lecture`. Le compte client utilisé pour la supervision ne doit disposer d'aucun droit d'écriture.
2. **Deuxième requête.** Le jeton extrait (`${var:jeton}`) authentifie la recherche. L'identifiant interne de la commande témoin est extrait de la réponse.
3. **Troisième requête.** Cet identifiant construit l'URL du détail, dont le contenu est vérifié.

Secrets et hôtes autorisés. Chaque secret ne peut être transmis qu'aux hôtes de sa liste `allowedHosts`, fixée par un administrateur (§ 6.8.1 du guide d'installation et d'administration). Dans cet exemple, les secrets `OAUTH_CLIENT_ID` et `OAUTH_CLIENT_SECRET` doivent être créés avec `"allowedHosts": ["auth.example.com"]` : ils ne sont injectés que dans la requête adressée au serveur d'autorisation, jamais dans celles adressées à `api.example.com`. Ici, les secrets figurent dans le **corps** de la première requête : la liste s'applique à tout élément porteur d'un secret, et pas seulement aux en-têtes. Si une réponse redirige vers un autre hôte, les en-têtes porteurs d'un secret ne sont pas retransmis, et un corps porteur d'un secret renvoyé lors d'une redirection n'est transmis qu'à un hôte de la liste (§ 6.8.1 du guide d'installation et d'administration). Un utilisateur `operator` peut référencer ces secrets dans un contrôle `api`, mais il ne peut ni les lire ni étendre la liste des hôtes autorisés.

> **Attention** — La variable `${var:jeton}`, extraite d'une réponse, n'est pas un secret de la sonde : la liste `allowedHosts` ne la concerne pas. Vérifiez que les requêtes qui l'emploient visent uniquement l'API prévue.

5.3.4 Résultat attendu et localisation de l'échec

Lorsque le serveur d'autorisation est défaillant, le résultat indique sans ambiguïté que le problème se situe à la première étape, et non dans l'API de commandes :

```json
{
  "checkId": "api-commandes-oauth",
  "status": "CRITICAL",
  "durationMs": 10042,
  "error": {
    "type": "timeout",
    "step": "Obtenir un jeton d'accès",
    "message": "Délai dépassé lors de la requête POST https://auth.example.com/oauth2/token"
  },
  "steps": [
    { "index": 0, "name": "Obtenir un jeton d'accès", "status": "CRITICAL", "durationMs": 10042 }
  ]
}
```

Les durées et les échecs par étape sont publiés dans les métriques `sva.step.duration` et `sva.step.failures` (chapitre 2 du guide d'intégration et d'observabilité). Vous pouvez ainsi suivre séparément le temps de réponse du serveur d'autorisation et celui de l'API.

> ✓ **Conseil** — Donnez aux requêtes des **noms fixes et explicites**, rédigés du point de vue fonctionnel (« Obtenir un jeton d'accès »). Ne faites jamais figurer dans un nom une valeur variable, comme un identifiant ou une date : chaque nom distinct crée de nouvelles séries de métriques.

5.3.5 Erreurs courantes des contrôles `api`

Symptôme	Cause probable	Action
Échec à l'étape d'authentification avec le code 401	Secret client expiré, révoqué ou modifié	Renouveler le secret avec l'équipe responsable du serveur d'autorisation, puis le remplacer sur la sonde (chapitre 6 du guide d'installation et d'administration)
Échec à l'étape d'authentification avec le code 400	Paramètres du flux incorrects (périmètre inconnu, type de flux non autorisé pour ce client)	Comparer le corps de la requête avec la documentation du serveur d'autorisation
Échec à la deuxième étape avec le code 401, alors que la première réussit	Variable mal référencée (`${var:…}` mal orthographié), ou jeton émis pour une autre audience	Vérifier le nom de la variable dans `extract` et dans l'en-tête ; vérifier l'audience du jeton
Échec avec le code 403	Le compte de supervision n'a pas les droits sur la ressource témoin	Ajuster les droits du compte, en restant en lecture seule
Échec d'authentification alors que le secret est correct	L'hôte de la requête ne figure pas dans la liste `allowedHosts` du secret : le secret n'a pas été injecté	Faire vérifier ou compléter la liste par un administrateur (§ 6.8 du guide d'installation et d'administration)
Variable vide, URL invalide	Expression d'extraction ne correspondant pas à la réponse (structure modifiée, tableau vide)	Exécuter le contrôle en mode essai et examiner la réponse de l'étape précédente

5.4 Superviser un web service SOAP

5.4.1 Principe

SOAP (*Simple Object Access Protocol*) est un protocole d'échange de messages XML, encore très présent dans les systèmes d'information : progiciels, échanges interentreprises, services administratifs. Un appel SOAP est une requête HTTP `POST` dont le corps est une **enveloppe XML**, et dont la réponse est elle aussi une enveloppe XML.

La sonde supervise les web services SOAP avec le contrôle `http` :

- le corps de la requête contient l'enveloppe ;
- les en-têtes précisent le type de contenu et, selon la version de SOAP, l'action appelée ;
- les assertions `body` et `xpath` vérifient la réponse.

Le contrôle `api` permet d'enchaîner plusieurs appels SOAP.

5.4.2 Exemple : recherche d'une adresse dans un annuaire

Cet exemple reprend l'intention d'un scénario historique de supervision : vérifier qu'un web service d'annuaire de messagerie retrouve une adresse connue, ce qui prouve que le service fonctionne de bout en bout, jusqu'à sa source de données. Toutes les valeurs sont fictives :

- service `https://annuaire.example.internal/ws/annuaire` ;
- opération `rechercherAdresse` ;
- adresse témoin `test-supervision@example.com`.

JSON

```
{
  "id": "ws-annuaire-recherche",
  "name": "Web service annuaire : recherche de l'adresse témoin",
  "description": "Appel SOAP 1.1 de l'opération rechercherAdresse sur l'adresse de test (exemple
    fictif).",
  "type": "http",
  "tags": { "application": "annuaire_messagerie", "environment": "production" },
  "schedule": { "interval": "5m" },
  "timeout": "15s",
  "retry": { "attempts": 1, "interval": "30s" },
  "http": {
    "method": "POST",
    "url": "https://annuaire.example.internal/ws/annuaire",
    "headers": {
      "Content-Type": "text/xml; charset=utf-8",
      "SOAPAction": "\"urn:annuaire#rechercherAdresse\""
    },
    "body": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><soapenv:Envelope
    xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\" xmlns:ann=\"urn:annuaire\">
    <soapenv:Body><ann:rechercherAdresse><ann:email>test-supervision@example.com</ann:email>
    </ann:rechercherAdresse></soapenv:Body></soapenv:Envelope>",
    "assertions": [
      { "kind": "status", "operator": "eq", "value": 200 },
      { "kind": "body", "operator": "notContains", "value": "Fault>" },
      { "kind": "xpath", "path": "//*[local-name()='trouve']", "operator": "eq", "value": "0" },
      { "kind": "xpath", "path": "//*[local-name()='email']", "operator": "eq", "value": "test-
    supervision@example.com" }
    ],
    "tls": { "verify": true, "caRef": "ac-interne-example" }
  }
}
```

Points d'attention :

Élément	Explication
`Content-Type: text/xml` et `SOAPAction`	En-têtes de SOAP 1.1. En SOAP 1.2, le type est `application/soap+xml` et l'action figure dans un paramètre `action` de ce même en-tête
`notContains` `Fault>`	Une erreur SOAP (*SOAP Fault*) peut être renvoyée avec le code HTTP 500, mais certains serveurs la renvoient avec le code 200. Cette assertion couvre les deux cas
`//*[local-name()='trouve']`	Les réponses SOAP utilisent des **espaces de noms** XML dont le préfixe varie d'un serveur à l'autre. La fonction `local-name()` désigne un élément par son nom local, quel que soit son préfixe. C'est la forme la plus robuste
Corps sur une ligne	Le corps est une chaîne JSON : les guillemets de l'enveloppe sont échappés (`\"`). Préparez et testez l'enveloppe dans un outil dédié, puis convertissez-la en chaîne JSON

5.4.3 Migration des projets SoapUI

Les clients de la génération précédente de la solution utilisaient des projets SoapUI pour superviser leurs web services. Ces projets ne sont pas exécutables par la sonde. Leurs éléments se transposent comme suit :

Élément SoapUI	Équivalent sur la sonde
Projet et suite de tests	Un ou plusieurs contrôles `http` ou `api`
Étape de requête SOAP ou REST (*Test Step*)	Requête d'un contrôle `http`, ou requête nommée d'un contrôle `api`
Enveloppe SOAP de la requête	`http.body`
Assertion *Contains* / *Not Contains*	Assertion `body` avec `contains` / `notContains`
Assertion *XPath Match*	Assertion `xpath`
Assertion *Valid HTTP Status Codes*	Assertion `status`
Assertion *Response SLA*	Assertion `duration`, ou `thresholds`
Transfert de propriété entre étapes (*Property Transfer*)	`extract` et `${var:…}` du contrôle `api`
Script Groovy	Pas d'équivalent direct : plugin personnalisé (§ 4.7.10) ou découpage en contrôles plus simples

> **Information** — La transposition est manuelle. Elle est l'occasion de revoir les assertions : un simple *Contains* sur un fragment XML est souvent remplaçable par une assertion `xpath`, plus précise.

> **Évolution prévue** — Un outil d'import des projets SoapUI est prévu dans une version ultérieure pour accompagner la migration.

5.5 Concevoir des contrôles fonctionnels robustes

Un contrôle fonctionnel n'a de valeur que s'il est **fiable** : il doit échouer quand le service est réellement dégradé, et seulement dans ce cas. Les principes suivants s'appliquent aux contrôles `http`, `api` et `database`, ainsi qu'aux scénarios navigateur.

Principe	Mise en œuvre
Vérifier le résultat métier, pas seulement la réponse	Une assertion sur une valeur fonctionnelle (quantité, référence, état) en plus du code HTTP
Utiliser des données témoins dédiées	Objets réservés à la supervision, documentés, exclus des statistiques
Préférer des invariants à des valeurs exactes	« La quantité est positive » plutôt que « la quantité vaut 148 »
Rester en lecture seule	Aucune création, modification ou suppression de données en production (§ 5.9)
Couvrir les erreurs « silencieuses »	Assertions `notContains` sur les marqueurs d'erreur renvoyés avec un code 200
Séparer les responsabilités	Une étape nommée par composant (authentification, recherche, détail)
Calibrer les seuils de performance sur l'historique	Valeurs fondées sur les percentiles observés (chapitre 7)
Maintenir les contrôles avec les applications	Revue des contrôles à chaque évolution de l'API ou du schéma de données

✓ **Conseil** — Associez chaque contrôle fonctionnel à une **fiche descriptive** : ce qu'il vérifie, la donnée témoin utilisée, le responsable applicatif à prévenir, la conduite à tenir en cas d'échec. Le champ `description` du contrôle peut en contenir un résumé.

5.6 Contrôles de bases de données (`database`)

5.6.1 Principe et intérêt

Le contrôle `database` se connecte à une base de données, exécute une **requête SQL** ou une **procédure stockée**, puis vérifie le résultat au moyen de **règles**. Il répond à des questions que ni le réseau ni les API ne permettent de traiter :

- les traitements planifiés (*batchs*) ont-ils tourné, et ont-ils réussi ?
- des données s'accumulent-elles anormalement : messages en attente, commandes bloquées, rejets ?
- un volume d'activité est-il cohérent avec l'heure et le jour ?
- un référentiel est-il complet, et les données sont-elles cohérentes entre deux tables ?

Le contrôle `database` permet aussi de **publier des valeurs** issues de la requête sous forme d'indicateurs métier (§ 5.7).

5.6.2 Bases de données prises en charge

Le moteur principal de la sonde se connecte aux bases de données au moyen de pilotes **JDBC** (*Java Database Connectivity*), l'interface standard d'accès aux bases de données de la plateforme Java.

Moteur (`engine`)	SGBD	Port par défaut	Remarques
`postgresql`	PostgreSQL	5432	—
`mysql`	MySQL	3306	—

Moteur (`engine`)	SGBD	Port par défaut	Remarques
`mariadb`	MariaDB	3306	—
`sqlserver`	Microsoft SQL Server	1433	Chiffrement de la connexion : `tls`. Autres options de connexion, dont les instances nommées : propriétés JDBC `properties`, liste fermée
`oracle`	Oracle Database	1521	Le champ `database` désigne le nom de service Oracle

Les versions de chaque SGBD prises en charge dépendent des pilotes intégrés ; elles sont indiquées dans les notes de version.

5.6.3 Paramètres

Paramètre	Obligatoire	Description
`database.engine`	Oui	`postgresql`, `mysql`, `mariadb`, `sqlserver` ou `oracle`
`database.host`	Oui	Nom d'hôte ou adresse IP du serveur. Une adresse IPv6 est admise ; la famille d'adresses suit le paramètre commun `ipFamily` (§ 3.1.3, et § 4.1.3)
`database.port`	Non	Port, si différent du port par défaut
`database.database`	Non	Nom de la base (ou nom de service Oracle)
`database.username`	Oui	Compte de connexion, en lecture seule (§ 5.6.4)
`database.passwordSecret`	Oui	Nom du secret contenant le mot de passe. Ce secret doit aussi figurer dans la liste `secrets` du contrôle, et l'hôte `database.host` (avec son port, le cas échéant) dans la liste `allowedHosts` du secret (§ 6.8.1 du guide d'installation et d'administration)
`database.tls`	Non	Si `true`, chiffre la connexion au serveur (TLS). Défaut : `false`. Le chiffrement ne protège contre l'interception que si le certificat du serveur est vérifié (tableau ci-dessous)
`database.properties`	Non	Propriétés de connexion JDBC complémentaires (nom et valeur texte). Seules les clés d'une liste fermée, propre à chaque moteur, sont acceptées ; cette liste est indiquée dans les notes de version
`database.query`	L'un des deux	Requête SQL à exécuter (10 000 caractères maximum)
`database.procedure`	L'un des deux	Nom de la procédure stockée à appeler
`database.params`	Non	Paramètres de la procédure (nom et valeur)
`database.rules`	Oui	Règles appliquées au résultat (1 à 20)
`database.onEmpty`	Non	État produit lorsque le résultat ne contient aucune ligne : `OK`, `WARNING`, `CRITICAL` ou `UNKNOWN` (§ 5.6.6). Défaut : `UNKNOWN`
`database.exportValues`	Non	Colonnes publiées comme indicateurs métier (10 au maximum, § 5.7)

Le délai maximal (connexion et exécution) est fixé par le paramètre commun `timeout` du contrôle.

Sécurité de la connexion, délais et annulation selon le moteur. La sonde applique un délai de requête **côté pilote**, par la méthode JDBC standard `Statement.setQueryTimeout`. Ce délai seul ne suffit pas : si la connexion est perdue, le serveur peut poursuivre l'exécution. Fixez donc aussi une limite **côté serveur** sur le compte de supervision. Le tableau suivant rappelle, pour information, les mécanismes standards de chaque moteur et de son pilote JDBC de

référence. Les options effectivement appliquées par la sonde lorsque `tls` vaut `true`, la correspondance entre `timeout` et le délai du pilote, et la liste des propriétés `properties` autorisées sont indiquées dans les notes de version.

Moteur	Vérification du certificat (option du pilote de référence)	Limite côté serveur	Annulation sur délai dépassé
PostgreSQL	`sslmode=verify-full` (pilote pgJDBC) : chaîne de confiance et nom d'hôte	`statement_timeout`, fixable pour le rôle (`ALTER ROLE … SET statement_timeout`, § 5.6.4)	Le pilote envoie une demande d'annulation au serveur par une connexion distincte
MySQL	`sslMode=VERIFY_IDENTITY` (MySQL Connector/J 8 et ultérieur)	`max_execution_time` (en millisecondes), variable de session ou globale, ou indication `/*+ MAX_EXECUTION_TIME(90000) */` dans la requête ; ne s'applique qu'aux instructions `SELECT`	Le pilote demande l'arrêt de la requête (`KILL QUERY`) par une connexion distincte
MariaDB	`sslMode=verify-full` (MariaDB Connector/J 3)	`max_statement_time` (en secondes), par session, globalement ou par compte (`ALTER USER … WITH MAX_STATEMENT_TIME 90`)	Mécanisme propre au pilote
SQL Server	`encrypt=true` et `trustServerCertificate=false` (pilote Microsoft JDBC) ; `hostNameInCertificate` si le nom du certificat diffère de l'hôte	Pas de délai d'exécution natif par compte équivalent à `statement_timeout` : la limite repose sur le délai du pilote	Le pilote envoie un signal d'interruption (*attention*) au serveur
Oracle Database	Connexion `TCPS` avec vérification du nom du serveur (`oracle.net.ssl_server_dn_match=true`)	Profil limitant le temps processeur par appel (`CPU_PER_CALL`, pris en compte si `RESOURCE_LIMIT` est actif), ou Resource Manager (directive `CANCEL_SQL`)	Le pilote demande l'annulation ; la requête se termine par l'erreur `ORA-01013`

Une annulation interrompt une requête ou une procédure **en cours de route**. Pour une requête de lecture qualifiée sans effet, y compris les fonctions qu'elle appelle (§ 5.6.5), l'interruption ne laisse aucune écriture métier. Elle n'est pas pour autant neutre : si le pilote n'obtient pas l'annulation (connexion perdue, demande non traitée par le serveur), la requête peut se poursuivre côté serveur et conserver des ressources (processeur, mémoire, verrous, transaction ouverte) jusqu'à la limite côté serveur ou la fin de la session. Après une annulation ou un délai dépassé, la connexion doit être ramenée à un état sain (transaction annulée) ou fermée, et ses ressources libérées, avant toute réutilisation. Pour une procédure qui écrirait, des effets partiels pourraient subsister : c'est une raison supplémentaire d'exiger des procédures non modifiantes (§ 5.6.5).

> **Attention** — Faites valider la limite côté serveur par l'administrateur de la base : elle doit rester **supérieure** au `timeout` du contrôle, pour que l'erreur constatée soit celle de la sonde (`timeout`) et non une erreur SQL.

5.6.4 Préparer un compte de supervision en lecture seule

La sonde doit se connecter avec un **compte dédié**, disposant uniquement du droit de lecture sur les objets nécessaires. Ce compte limite les conséquences d'une erreur dans une requête ou d'une compromission des identifiants. Les commandes suivantes sont des exemples, à adapter et à faire exécuter par l'administrateur de la base. Les noms d'objets sont fictifs.

> **Avertissement** — Le compte de supervision est **dédié** à la sonde et ses droits sont **vérifiés côté serveur**, et non seulement déclarés : la sonde ne peut pas contrôler les droits du compte qu'on lui confie. Vérifiez aussi qu'une tentative d'écriture avec ce compte est **refusée**.

Faites vérifier par l'administrateur de la base les **droits effectifs** du compte, y compris ceux hérités de rôles, ceux accordés à tous les comptes (`PUBLIC`) et les droits par défaut sur les objets futurs. Outils usuels : `has_table_privilege` et la commande `\dp` de `psql` (PostgreSQL), `SHOW GRANTS FOR` (MySQL, MariaDB), `sys.fn_my_permissions` exécutée sous l'identité du compte (SQL Server), vues `SESSION_PRIVS` et `SESSION_ROLES` (Oracle). Renouvelez cette vérification à chaque évolution des rôles de la base.

PostgreSQL :

```sql
CREATE ROLE supervision_ro LOGIN PASSWORD '********';
GRANT CONNECT ON DATABASE ventes TO supervision_ro;
GRANT USAGE ON SCHEMA public TO supervision_ro;
GRANT SELECT ON public.commandes TO supervision_ro;
-- Limite la durée de toute requête de ce compte à 90 secondes : valeur supérieure
-- au timeout des contrôles qui utilisent ce compte (30 secondes au plus ici)
ALTER ROLE supervision_ro SET statement_timeout = '90s';
```

MySQL / MariaDB :

```sql
CREATE USER 'supervision_ro'@'192.0.2.50' IDENTIFIED BY '********';
GRANT SELECT ON ventes.commandes TO 'supervision_ro'@'192.0.2.50';
```

Le compte MySQL est restreint à l'adresse de la sonde (`192.0.2.50` dans cet exemple).

Microsoft SQL Server :

```sql
CREATE LOGIN supervision_ro WITH PASSWORD = '********';
USE Exploitation;
CREATE USER supervision_ro FOR LOGIN supervision_ro;
GRANT SELECT ON dbo.HistoriqueTraitements TO supervision_ro;
```

Oracle Database :

```sql
CREATE USER supervision_ro IDENTIFIED BY "********";
GRANT CREATE SESSION TO supervision_ro;
GRANT SELECT ON FACTURATION.SUIVI_LOTS TO supervision_ro;
GRANT EXECUTE ON FACTURATION.PKG_SUPERVISION TO supervision_ro;
```

> **Avertissement** — N'utilisez jamais un compte applicatif ou un compte d'administration de la base pour la supervision. Si la requête doit appeler une procédure stockée, vérifiez avec l'équipe responsable que cette procédure ne réalise aucune écriture.

5.6.5 Requête ou procédure stockée

Forme	Usage	Paramètre
Requête	Cas général : comptage, recherche de la dernière exécution, calcul d'écart	`query`
Procédure stockée	Logique de contrôle déjà écrite et maintenue par l'équipe applicative, ou accès aux tables non autorisé directement	`procedure` + `params`

Les deux formes sont exclusives : un contrôle utilise soit `query`, soit `procedure`.

Pour une procédure, les paramètres sont transmis par leur nom, avec des valeurs de type texte, nombre, booléen ou nulle :

```json
"procedure": "PKG_SUPERVISION.CONTROLE_LOTS",
"params": { "p_application": "FACTURATION", "p_fenetre_heures": 26 }
```

> **Avertissement** — Le droit d'exécuter une procédure peut donner accès à **davantage** que les droits du compte de supervision : un compte « en lecture seule » doté du droit `EXECUTE` peut écrire au travers d'une procédure.

Selon le moteur, une procédure s'exécute avec les droits de son propriétaire : c'est le comportement par défaut d'Oracle (`AUTHID DEFINER`) et de MySQL et MariaDB (`SQL SECURITY DEFINER`) ; c'est aussi le cas des fonctions PostgreSQL déclarées `SECURITY DEFINER` ; sous SQL Server, le chaînage des propriétaires (*ownership chaining*) permet à la procédure d'accéder aux objets du même propriétaire sans droit direct du compte.

Chaque procédure utilisée par la sonde doit être **qualifiée comme non modifiante** par l'équipe qui la maintient, avant sa mise en supervision et à chacune de ses évolutions :

- aucune écriture (`INSERT`, `UPDATE`, `DELETE`, `MERGE`), aucune modification de structure, aucune validation de transaction (`COMMIT`), y compris dans une table de journalisation ou d'audit ;
- aucune transaction autonome (par exemple `PRAGMA AUTONOMOUS_TRANSACTION` sous Oracle) ni effet de bord externe : envoi de courrier électronique, dépôt de message dans une file, appel d'un service, lancement d'un traitement ;
- même vérification pour les procédures et fonctions **appelées par** la procédure (droits transitifs), ainsi que pour toute fonction définie dans la base et appelée par une **requête de lecture** (`SELECT`) du contrôle : une telle fonction peut écrire, s'exécuter avec les droits de son propriétaire ou produire un effet externe, et se qualifie donc comme une procédure, moteur par moteur ;
- exécution compatible avec une interruption à tout instant (annulation sur délai dépassé, § 5.6.3).

La procédure doit renvoyer un **jeu de résultats** dont les colonnes sont nommées, puisque les règles s'appliquent à ces colonnes. Le mode de restitution du résultat selon le SGBD (curseur de sortie Oracle, jeu de résultats SQL Server…) est propre à chaque SGBD ; les conventions applicables sont indiquées dans les notes de version.

5.6.6 Règles

Une **règle** compare la valeur d'une colonne du résultat à une valeur de référence.

Élément	Valeurs	Description
`column`	Nom de colonne	Colonne du résultat à vérifier. Donnez un alias explicite dans la requête (`AS nb_commandes`)

Élément	Valeurs	Description
`operator`	`eq`, `ne`, `lt`, `lte`, `gt`, `gte`	Égal, différent, strictement inférieur, inférieur ou égal, strictement supérieur, supérieur ou égal. Les comparaisons d'ordre ne s'appliquent qu'aux valeurs numériques
`value`	Nombre, texte ou booléen	Valeur de référence
`severity`	`critical` (défaut), `warning`	État produit si la règle n'est pas respectée
`scope`	`allRows` (défaut), `anyRow`, `firstRow`	Lignes concernées : toutes les lignes doivent respecter la règle, au moins une ligne, ou seulement la première

Calcul de l'état :

1. si au moins une règle de sévérité `critical` n'est pas respectée, l'état est `CRITICAL` (`assertion_failed`) ;
2. sinon, si au moins une règle de sévérité `warning` n'est pas respectée, l'état est `WARNING` ;
3. sinon, l'état est `OK`, sous réserve des seuils de durée (`thresholds`).

Une règle s'exprime comme la **condition normale** : elle décrit ce qui doit être vrai pour que tout aille bien. Exemple : « le nombre de commandes doit être supérieur à 0 » s'écrit `{ "column": "nb", "operator": "gt", "value": 0 }`.

Pour définir deux niveaux d'alerte sur une même colonne, déclarez deux règles : une règle `warning` au seuil le plus exigeant, une règle `critical` au seuil le plus large.

> **Attention** — Une requête qui ne renvoie **aucune ligne** ne permet pas d'évaluer les règles. L'état produit est alors fixé par le paramètre `onEmpty` : `UNKNOWN` par défaut, avec le type d'erreur `no_data`. Il est fortement recommandé d'écrire des requêtes qui renvoient **toujours exactement une ligne**, au moyen de fonctions d'agrégation (`COUNT`, `MAX`, `MIN`, `SUM`, `COALESCE`). Par exemple, `SELECT COUNT(*) AS nb …` renvoie `0` et non un résultat vide.

Erreurs d'exécution :

Situation	État	Type d'erreur
Serveur injoignable, base inexistante, erreur SQL	`CRITICAL`	`database_error`
Identifiants refusés	`CRITICAL`	`auth_failed`
Délai dépassé (connexion ou requête)	`CRITICAL`	`timeout`
Aucune ligne renvoyée	Selon `onEmpty` (`UNKNOWN` par défaut)	`no_data`
Règle non respectée	Selon `severity`	`assertion_failed`

5.6.7 Exemple : traitement nocturne (SQL Server)

L'application fictive d'exploitation enregistre chaque exécution de ses traitements dans une table d'historique. Le contrôle vérifie que le traitement de facturation a réussi au cours des dernières 26 heures. La marge de 2 heures au-delà d'un cycle quotidien absorbe les variations normales de l'heure de fin. Le contrôle vérifie aussi qu'aucun traitement n'est en erreur.

```json
{
  "id": "batch-facturation-nocturne",
  "name": "Traitement de facturation : dernière exécution réussie",
  "type": "database",
  "tags": { "application": "facturation", "environment": "production" },
  "schedule": { "cron": "*/30 6-20 * * *", "timezone": "Europe/Paris" },
  "timeout": "30s",
  "secrets": ["DB_EXPLOITATION_PASSWORD"],
  "database": {
    "engine": "sqlserver",
    "host": "sql-exploitation.example.internal",
    "port": 1433,
    "database": "Exploitation",
    "username": "supervision_ro",
    "passwordSecret": "DB_EXPLOITATION_PASSWORD",
    "query": "SELECT COALESCE(DATEDIFF(MINUTE, MAX(CASE WHEN Statut = 'SUCCES' THEN DateFin END),
    SYSUTCDATETIME()), 99999) AS minutes_depuis_succes, COALESCE(SUM(CASE WHEN Statut = 'ERREUR' AND
    DateFin > DATEADD(HOUR, -26, SYSUTCDATETIME()) THEN 1 ELSE 0 END), 0) AS erreurs_26h FROM
    dbo.HistoriqueTraitements WHERE Traitement = 'FACTURATION'",
    "rules": [
      { "column": "minutes_depuis_succes", "operator": "lte", "value": 1560, "severity": "critical",
    "scope": "firstRow" },
      { "column": "erreurs_26h", "operator": "eq", "value": 0, "severity": "warning", "scope":
    "firstRow" }
    ],
    "exportValues": [{ "column": "minutes_depuis_succes", "indicator":
    "facturation_minutes_depuis_succes" }]
  }
}
```

Décryptage :

- `COALESCE(…, 99999)` garantit une valeur même si aucun succès n'a jamais été enregistré. La règle échoue alors clairement.
- La règle `critical` porte sur l'absence de succès depuis plus de 1 560 minutes (26 heures).
- La règle `warning` signale un échec intermédiaire suivi d'une relance réussie : la situation est rétablie, mais mérite d'être examinée.
- La planification par expression `cron` limite le contrôle aux heures où une intervention est possible (de 6 h à 20 h). La planification est détaillée au chapitre 6.

5.6.8 Exemple : messages en attente (MariaDB)

```json
{
  "id": "file-integration-en-attente",
  "name": "File d'intégration : messages en attente",
  "type": "database",
  "tags": { "application": "integration" },
  "schedule": { "interval": "5m" },
  "timeout": "20s",
  "secrets": ["DB_INTEGRATION_PASSWORD"],
  "database": {
    "engine": "mariadb",
    "host": "db-integration.example.internal",
    "database": "integration",
    "username": "supervision_ro",
    "passwordSecret": "DB_INTEGRATION_PASSWORD",
    "query": "SELECT COUNT(*) AS en_attente_15min FROM messages WHERE etat = 'EN_ATTENTE' AND
   date_reception < NOW() - INTERVAL 15 MINUTE",
    "rules": [
      { "column": "en_attente_15min", "operator": "lt", "value": 100, "severity": "warning" },
      { "column": "en_attente_15min", "operator": "lt", "value": 500, "severity": "critical" }
    ],
    "exportValues": [{ "column": "en_attente_15min", "indicator": "messages_en_attente" }]
  }
}
```

À 120 messages bloqués depuis plus de 15 minutes, seule la règle `warning` est enfreinte : l'état est `WARNING`. À 600 messages, la règle `critical` l'est aussi : l'état est `CRITICAL`.

5.6.9 Exemple : vérification par une procédure stockée (Oracle)

```json
{
  "id": "facturation-controle-lots",
  "name": "Facturation : contrôle de cohérence des lots",
  "type": "database",
  "tags": { "application": "facturation" },
  "schedule": { "cron": "0 7,13 * * 1-5", "timezone": "Europe/Paris" },
  "timeout": "1m",
  "secrets": ["DB_FACTURATION_PASSWORD"],
  "database": {
    "engine": "oracle",
    "host": "ora-facturation.example.internal",
    "port": 1521,
    "database": "FACTPRD",
    "username": "supervision_ro",
    "passwordSecret": "DB_FACTURATION_PASSWORD",
    "procedure": "PKG_SUPERVISION.CONTROLE_LOTS",
    "params": { "p_application": "FACTURATION", "p_fenetre_heures": 26 },
    "rules": [
      { "column": "STATUT", "operator": "eq", "value": "OK", "scope": "allRows" }
    ]
  }
}
```

La procédure renvoie une ligne par lot contrôlé, avec une colonne `STATUT`. La règle `scope: allRows` exige que **tous** les lots soient conformes. Avec `scope: anyRow`, il suffirait qu'un seul le soit.

5.6.10 Bonnes pratiques des requêtes de supervision

Une requête de supervision s'exécute à intervalle régulier, en permanence. Elle doit être **légère** et **sans effet sur l'activité de production**.

Pratique	Raison
Requêtes agrégées renvoyant une seule ligne	Résultat toujours évaluable, transfert minimal
Filtrer sur une fenêtre de temps récente (`date > maintenant - 1 heure`) et sur des colonnes indexées	Éviter le parcours complet de grandes tables
Vérifier le plan d'exécution avec l'administrateur de la base avant la mise en production	Une requête anodine peut se révéler coûteuse sur une grande volumétrie
Interroger une réplique en lecture lorsqu'elle existe	Aucun impact sur la base principale
Fixer une limite de durée côté serveur (par exemple `statement_timeout` pour PostgreSQL), en plus du timeout du contrôle	Une requête interrompue par la sonde peut continuer à s'exécuter sur le serveur
Adapter la fréquence au rythme du phénomène observé	Un traitement quotidien n'a pas besoin d'être vérifié toutes les minutes
Ne jamais écrire : ni `INSERT` , `UPDATE` , `DELETE` , ni procédure modifiant des données	Sécurité et intégrité des données
Éviter `SELECT *` et ne lire que les colonnes utiles aux règles	Performance ; aucune donnée personnelle inutile dans les résultats

5.6.11 Correspondance avec l'ancien format des scénarios BDD

Les clients de la génération précédente de la solution décrivaient leurs contrôles de bases de données dans des fichiers texte structurés en sections. Ce format **n'est pas utilisé** par la sonde actuelle. Le tableau suivant permet de transposer manuellement un contrôle existant.

> → **Évolution prévue** — Un outil d'import des anciens fichiers de scénarios BDD est prévu dans une version ultérieure pour faciliter la migration. La transposition est manuelle.

Ancien format	Contrôle `database`
Section `[DATABASE]` , clé `HOST`	`database.host`
Clé `PORT`	`database.port`
Clé `BDD_TYPE` : `MYSQL` , `POSTGRESQL` , `SQLSERVER2008` , `SQLSERVER2016` , `ORACLE`	`database.engine` : `mysql` , `postgresql` , `sqlserver` , `sqlserver` , `oracle`
Clé `USERNAME`	`database.username`
Clé `PASSWORD` (en clair dans le fichier)	Secret créé sur la sonde (chapitre 6 du guide d'installation et d'administration), référencé par `database.passwordSecret` et déclaré dans `secrets`
Section `[SQL]`	`database.query`
Section `[PROC]`	`database.procedure`
Section `[PARAM]` , lignes `nom;valeur`	`database.params` : `{ "nom": valeur }`
Section `[RULES]` , lignes `COLONNE;OPÉRATEUR;VALEUR`	`database.rules` : `{ "column": …, "operator": …, "value": … }`
Opérateurs `LT` , `LTE` , `GT` , `GTE` , `EQ`	`lt` , `lte` , `gt` , `gte` , `eq`

Ancien format	Contrôle `database`
Règle appliquée à chaque ligne du résultat	`scope: "allRows"` (valeur par défaut)

> **Information** — La version 2008 de SQL Server n'est plus prise en charge par son éditeur. La compatibilité des pilotes actuels avec cette version n'est pas garantie ; les versions prises en charge sont indiquées dans les notes de version.

> **Avertissement** — Les anciens fichiers contenaient les mots de passe en clair. Lors de la migration, supprimez ces fichiers des postes et des partages après transposition, et profitez-en pour renouveler les mots de passe concernés.

5.7 Indicateurs métier

5.7.1 Principe

Un **indicateur métier** est une valeur qui mesure l'activité ou la santé fonctionnelle d'une application : nombre de commandes de l'heure, montant encaissé, nombre d'inscriptions, messages en attente, délai depuis la dernière synchronisation. Contrairement à un état (OK ou en échec), un indicateur se **suit dans le temps** : sa courbe révèle les tendances, les ruptures et les cycles d'activité.

Le paramètre `exportValues` d'un contrôle `database` publie la valeur d'une ou plusieurs colonnes du résultat, à chaque exécution. Chaque entrée associe :

- `column` : la colonne du résultat ;
- `indicator` : le nom de l'indicateur, en minuscules, chiffres et tirets bas (`commandes_heure`, `messages_en_attente`).

La valeur est publiée sous la métrique `sva.indicator.value`, avec les attributs `sva.check.id` et `sva.indicator.name` (chapitre 2 du guide d'intégration et d'observabilité). Elle est également conservée dans l'historique de la sonde et affichée dans le détail des exécutions (chapitre 7). La valeur publiée est celle de la **première ligne** du résultat.

5.7.2 Règles de nommage

Règle	Exemple correct	À éviter
Nom fonctionnel, stable dans le temps	`commandes_heure`	`count_1`
Unité ou période explicite dans le nom	`montant_encaisse_jour_eur`	`montant`
Aucune valeur variable dans le nom	`commandes_heure`	`commandes_2025_11_19`
Dix indicateurs au plus par contrôle	—	Un indicateur par magasin, par produit… (cardinalité incontrôlée)

> **Conseil** — Pour suivre un indicateur par entité (par magasin, par canal de vente), créez un contrôle par entité **significative**, en nombre limité. La sonde n'est pas un entrepôt de données : les analyses fines par dimension relèvent des outils décisionnels de l'entreprise.

5.8 Détection d'anomalies

5.8.1 Seuils fixes

Dans la version actuelle, la détection d'anomalies repose sur des **seuils fixes**, exprimés par les règles des contrôles `database` et par les assertions des contrôles `http` et `api`. C'est une méthode simple, explicable et prévisible. Elle convient lorsque la valeur normale de l'indicateur est stable.

5.8.2 Tenir compte des cycles d'activité

L'activité d'une entreprise varie selon l'heure et le jour : un volume de commandes normal à 3 heures du matin serait alarmant à 11 heures. Trois techniques permettent d'adapter les seuils aux cycles, sans fonction d'apprentissage.

1. Plusieurs contrôles, plusieurs plages horaires. Créez un contrôle par période d'activité, chacun avec ses propres seuils et une planification `cron` limitée à sa période :

Contrôle	Planification (`cron`)	Seuil critique
Commandes, heures ouvrées	`*/15 9-18 * * 1-5`	Moins de 40 commandes par heure
Commandes, samedi en journée	`*/15 9-18 * * 6`	Moins de 25 commandes par heure
Commandes, soirée du lundi au samedi	`*/15 19-22 * * 1-6`	Moins de 15 commandes par heure

Les plans d'exécution offrent aussi des **fenêtres d'activité** (`activeWindows`), qui suspendent l'exécution en dehors des plages définies (chapitre 6).

2. Comparaison calculée dans la requête. La requête SQL compare elle-même la période en cours à une référence historique : même heure les semaines précédentes, moyenne glissante. La règle porte alors sur un **ratio**, indépendant du niveau absolu d'activité (§ 5.8.3).

3. Détection par la plateforme d'observabilité. Les indicateurs publiés (`sva.indicator.value`) peuvent alimenter les fonctions de détection d'anomalies de Datadog, de Dynatrace ou de Grafana (chapitre 2 du guide d'intégration et d'observabilité). La sonde fournit la mesure, la plateforme l'analyse.

> → **Évolution prévue** — Une détection d'anomalies intégrée à la sonde, fondée sur la comparaison automatique avec l'historique local (même créneau des semaines précédentes, écart en pourcentage), est prévue dans une version ultérieure. Elle n'est pas disponible actuellement.

5.8.3 Exemple : chute du nombre de commandes

Une boutique en ligne fictive constate qu'un incident chez un prestataire de paiement peut faire chuter les commandes sans qu'aucun serveur ne soit en panne. Le contrôle suivant compare, toutes les 15 minutes :

- le nombre de commandes validées au cours des **60 dernières minutes** ;
- la moyenne des commandes validées sur la **même fenêtre horaire** lors des **quatre semaines précédentes**.

JSON

```
{
  "id": "commandes-chute-activite",
  "name": "Boutique : chute anormale des commandes",
  "description": "Compare les commandes de la dernière heure à la moyenne des 4 semaines précédentes
    au même créneau (exemple fictif).",
  "type": "database",
  "tags": { "application": "boutique", "environment": "production" },
  "schedule": { "cron": "*/15 8-22 * * *", "timezone": "Europe/Paris" },
  "timeout": "30s",
```

```
  "secrets": ["DB_REPORTING_PASSWORD"],
  "database": {
    "engine": "postgresql",
    "host": "db-reporting.example.internal",
    "port": 5432,
    "database": "ventes",
    "username": "supervision_ro",
    "passwordSecret": "DB_REPORTING_PASSWORD",
    "query": "WITH courant AS (SELECT COUNT(*) AS nb FROM commandes WHERE statut = 'VALIDEE' AND
    est_test = false AND date_validation >= now() - interval '60 minutes'), reference AS (SELECT
    COUNT(*) / 4.0 AS nb FROM commandes c, generate_series(1, 4) AS s(semaine) WHERE c.statut =
    'VALIDEE' AND c.est_test = false AND c.date_validation >= now() - (s.semaine * interval '7 days')
    - interval '60 minutes' AND c.date_validation < now() - (s.semaine * interval '7 days')) SELECT
    courant.nb AS commandes_heure, reference.nb AS reference_heure, CASE WHEN reference.nb < 10 THEN
    1.0 ELSE round(courant.nb / reference.nb, 2) END AS ratio FROM courant, reference",
    "rules": [
      { "column": "ratio", "operator": "gte", "value": 0.7, "severity": "warning", "scope":
    "firstRow" },
      { "column": "ratio", "operator": "gte", "value": 0.4, "severity": "critical", "scope":
    "firstRow" }
    ],
    "exportValues": [
      { "column": "commandes_heure", "indicator": "commandes_heure" },
      { "column": "reference_heure", "indicator": "commandes_heure_reference" },
      { "column": "ratio", "indicator": "commandes_ratio_reference" }
    ]
  }
}
```

Fonctionnement :

Élément	Rôle
`courant`	Commandes validées des 60 dernières minutes, commandes de test exclues (`est_test = false`)
`reference`	Moyenne, sur les 4 semaines précédentes, des commandes validées dans la même fenêtre de 60 minutes
`ratio`	Activité actuelle rapportée à la référence : 1,0 = activité habituelle ; 0,5 = activité réduite de moitié
`CASE WHEN reference.nb < 10 THEN 1.0`	Neutralise le contrôle lorsque l'activité de référence est trop faible pour être significative (nuit, jour férié) : un ratio calculé sur quelques commandes produirait des fausses alertes
Règle `warning`	Activité inférieure à 70 % de la normale
Règle `critical`	Activité inférieure à 40 % de la normale
`exportValues`	Publie les trois valeurs : les courbes `commandes_heure` et `commandes_heure_reference` se superposent dans un tableau de bord et rendent l'écart immédiatement lisible

Résultat d'une exécution lors d'un incident :

```json
{
  "checkId": "commandes-chute-activite",
  "status": "CRITICAL",
  "durationMs": 412,
  "error": {
    "type": "assertion_failed",
    "message": "Règle ratio gte 0.4 (critical) non respectée : valeur 0.31"
  },
  "measurements": {
    "indicator.commandes_heure": 22,
    "indicator.commandes_heure_reference": 71.5,
    "indicator.commandes_ratio_reference": 0.31
  }
}
```

Attention — Les jours atypiques (soldes, jours fériés, campagnes promotionnelles, panne lors d'une des semaines de référence) faussent la référence. Prévoyez de suspendre le contrôle ou d'ajuster ses seuils lors de ces périodes, et informez les équipes métier de son mode de calcul. Faites valider la requête par l'administrateur de la base : son coût dépend de l'indexation de la colonne de date.

5.9 Précautions et sécurité

Les contrôles applicatifs interagissent avec des applications de production et des données de l'entreprise. Ils exigent davantage de précautions que les contrôles techniques.

5.9.1 Ne jamais modifier de données de production

Règle	Application
Lecture seule par défaut	Requêtes `GET`, recherches, consultations ; comptes en lecture seule
Aucune transaction réelle	Pas de commande, de paiement, de réservation, de virement, d'envoi de message à un client réel
Opérations d'écriture uniquement en environnement de test	Création-lecture-suppression d'objets de test sur une plateforme de recette, avec des données fictives
Données témoins identifiées	Marquage des objets de test pour les exclure des statistiques, de la facturation et des traitements métier

Avertissement — Un contrôle qui écrit des données s'exécute des centaines de fois par jour. Une erreur de conception peut créer des milliers d'objets, déclencher des envois de courriers électroniques ou de SMS à de vrais clients, ou fausser la comptabilité. Toute exception à la lecture seule doit être validée par le responsable de l'application et limitée à un environnement de test.

5.9.2 Protéger les identifiants

- Stockez tous les mots de passe, clés d'API et secrets clients comme **secrets** de la sonde (chapitre 6 du guide d'installation et d'administration). Ne les écrivez jamais en clair dans une URL, un en-tête ou une requête.
- Déclarez dans chaque contrôle **uniquement** les secrets dont il a besoin.

- Limitez la liste `allowedHosts` de chaque secret aux seuls hôtes qui doivent le recevoir (§ 6.8.1 du guide d'installation et d'administration) : serveur d'autorisation, API, serveur de base de données. Un secret n'est jamais injecté vers un hôte absent de cette liste, ni retransmis après une redirection vers un autre hôte.
- Utilisez des comptes de supervision **dédiés**, aux droits minimaux **vérifiés côté serveur** (§ 5.6.4), et renouvelez leurs secrets selon la politique de l'entreprise.

5.9.3 Maîtriser l'impact sur les applications

- Adaptez la fréquence au besoin. Un contrôle toutes les minutes sur une API coûteuse peut représenter une charge significative.
- Prévenez les équipes applicatives et sécurité de la mise en place des contrôles. Les requêtes de la sonde ne doivent pas être prises pour une attaque par les pare-feu applicatifs ou les outils de détection d'intrusion.
- Identifiez le trafic de supervision (en-tête `User-Agent` explicite, compte dédié) pour qu'il puisse être exclu des statistiques d'audience et d'activité.
- Respectez les quotas des API de partenaires : certaines facturent chaque appel.

5.9.4 Limiter les données collectées

Les résultats des contrôles sont stockés sur la sonde et peuvent être exportés vers des plateformes tierces. N'interrogez et ne vérifiez que les données strictement nécessaires. Évitez toute donnée personnelle réelle dans les requêtes, les données témoins et les assertions.

5.10 Synthèse

Besoin	Contrôle	Édition	Section
Réponse correcte d'une API (une requête)	`http` + `jsonPath`	Essential	§ 5.2
Authentification puis appel d'une API	`api`	Professional	§ 5.3
Web service SOAP	`http` + `xpath` (ou `api`)	Essential (Professional pour `api`)	§ 5.4
Traitements planifiés, files, cohérence de données	`database`	Professional	§ 5.6
Suivi d'un indicateur d'activité	`database` + `exportValues`	Professional	§ 5.7
Chute ou pic d'activité	`database` avec ratio calculé	Professional	§ 5.8
Parcours utilisateur dans un navigateur	`browser`	Enterprise	Chapitres 3 à 6 du guide de développement des scénarios

Chapitre 6 — Planification et plans d'exécution

Ce chapitre explique comment organiser dans le temps l'exécution des contrôles, par deux mécanismes :

- la **planification individuelle** de chaque contrôle ;
- les **plans d'exécution** (*playlists*), qui enchaînent plusieurs contrôles dans un ordre déterminé.

Il montre aussi comment évaluer la charge d'une sonde, pour garantir que les contrôles s'exécutent à l'heure prévue.

Les notions de fréquence, de gigue, de timeout et de concurrence sont introduites au chapitre 3. Ce chapitre en détaille la mise en œuvre. Les exemples utilisent des applications et des noms de domaine fictifs.

6.1 Deux façons d'organiser les exécutions

	Planification individuelle	Plan d'exécution
Porte sur	Un contrôle	Une liste ordonnée de contrôles
Définie dans	La propriété `schedule` du contrôle	Un objet `plan` distinct
Ordre d'exécution	Indépendant des autres contrôles	Séquentiel, dans l'ordre de la liste
Dépendances	Aucune	Arrêt possible au premier échec (`stopOnFailure`)
Plages horaires	Par expression cron	Par expression cron et plages actives (`activeWindows`)
Usage typique	Disponibilité d'un service, contrôles réseau, indicateurs métier	Parcours enchaînés, scénarios dépendants, maîtrise de la charge du moteur de scénarios

Les deux mécanismes se combinent : un contrôle peut avoir sa propre planification **et** figurer dans un ou plusieurs plans. Il peut aussi n'être exécuté que par un plan.

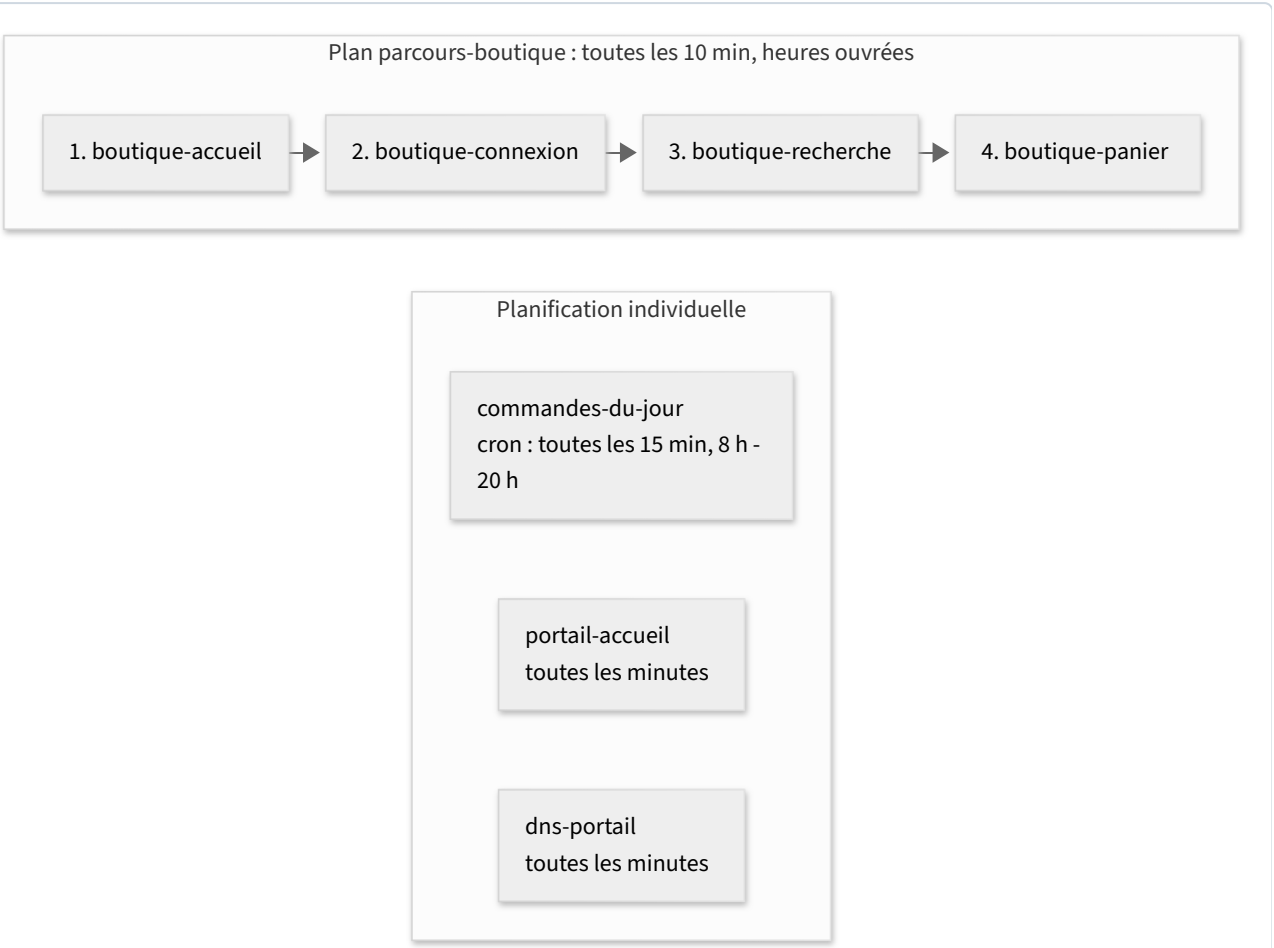


Schéma 6-1 — Planification individuelle et plan d'exécution

6.2 Planifier un contrôle

6.2.1 Par intervalle

L'intervalle est le mode le plus simple et le plus courant : le contrôle s'exécute à intervalles réguliers, en continu.

```json
"schedule": { "interval": "5m", "jitter": "20s" }
```

Propriété	Rôle	Format
`interval`	Écart entre deux exécutions successives	Durée : `30s`, `1m`, `5m`, `1h`
`jitter`	Décalage aléatoire maximal ajouté à chaque échéance (§ 3.3.2)	Durée

Les intervalles sont comptés d'un **début d'exécution** au suivant. Si une exécution dure encore au moment de l'échéance suivante, cette échéance est **sautée** : un même contrôle ne s'exécute jamais deux fois en parallèle. Une échéance sautée ne produit aucune exécution : elle se repère dans l'historique du contrôle, et une saturation durable se mesure par le retard de planification (métrique `sva.agent.scheduler.lag`, chapitre 2 du guide d'intégration et d'observabilité).

6.2.2 Par expression cron

Une **expression cron** décrit des instants d'exécution selon le calendrier. C'est le mode adapté aux contrôles qui n'ont de sens qu'à certains moments : heures ouvrées, fin d'un traitement de nuit, ouverture d'un service.

```json
"schedule": { "cron": "*/15 8-20 * * 1-5", "timezone": "Europe/Paris" }
```

La sonde utilise le format cron standard à **cinq champs**, sans champ des secondes :

Position	Champ	Valeurs	Exemple
1	Minute	0-59	`*/15` : toutes les 15 minutes
2	Heure	0-23	`8-20` : de 8 h à 20 h inclus
3	Jour du mois	1-31	`*` : tous les jours
4	Mois	1-12	`*` : tous les mois
5	Jour de la semaine	0-7 (0 et 7 = dimanche, 1 = lundi)	`1-5` : du lundi au vendredi

Opérateurs : `*` (toutes les valeurs), `,` (liste), `-` (plage), `/` (pas).

Expression	Signification
`*/5 * * * *`	Toutes les 5 minutes
`0 * * * *`	Toutes les heures, à l'heure pile
`*/15 8-20 * * 1-5`	Toutes les 15 minutes de 8 h 00 à 20 h 45, du lundi au vendredi
`30 6 * * *`	Chaque jour à 6 h 30, par exemple après un traitement de nuit
`0 9,14 * * 1-5`	À 9 h et 14 h, les jours ouvrés
`0 7 1 * *`	Le premier jour de chaque mois à 7 h

> **Information** — Dans la plage `8-20` du champ des heures, la dernière exécution d'une expression `*/15 8-20 * * *` a lieu à **20 h 45**, et non à 20 h 00 : le champ des heures inclut toute l'heure 20. Pour arrêter à 20 h 00 exactement, combinez deux planifications, ou utilisez les plages actives d'un plan (§ 6.4.4).

Une gigue (`jitter`) peut aussi s'appliquer à une planification cron. Elle évite que tous les contrôles planifiés « à l'heure pile » démarrent simultanément.

6.2.3 Fuseau horaire et changement d'heure

La propriété `timezone` indique le fuseau horaire dans lequel l'expression cron est interprétée, sous la forme d'un identifiant de la base IANA (`Europe/Paris`, `America/Montreal`, `UTC`). Sans indication, l'expression est interprétée en **UTC**.

> **Attention** — Une expression cron interprétée en UTC décale vos plages horaires d'une ou deux heures par rapport à l'heure de Paris, selon la saison. Indiquez toujours le fuseau horaire des utilisateurs de l'application supervisée.

Lors des changements d'heure, certaines heures locales n'existent pas (au printemps) ou existent deux fois (à l'automne). Une heure planifiée qui n'existe pas déclenche l'exécution à la première minute valide suivante ; une heure qui existe deux fois ne produit qu'une seule exécution. Évitez de planifier des contrôles critiques entre 2 h et 3 h

du matin dans les fuseaux soumis à l'heure d'été.

Les règles détaillées appliquées lors des deux bascules sont les suivantes, dans cet ordre :

1. **Calcul des échéances** dans le fuseau `timezone` de la planification (contrôle ou plan).
2. **Heure inexistante** (bascule de printemps) : l'échéance est reportée à la première minute valide suivante.
3. **Heure répétée** (bascule d'automne) : l'échéance ne donne lieu qu'à une exécution, lors de la **première occurrence** de l'heure répétée (heure d'été).
4. **Déduplication** : plusieurs échéances ramenées à la même minute ne donnent qu'**une seule exécution**.
5. **Gigue** : le décalage aléatoire (`jitter`) s'applique ensuite, à cette exécution unique.

Exemple, dans le fuseau `Europe/Paris`, la nuit de la bascule de printemps (de 2 h 00 à 3 h 00) : avec l'expression `*/30 2-3 * * *`, les échéances de 2 h 00 et de 2 h 30 n'existent pas et sont reportées à 3 h 00, où se trouve déjà une échéance. Le contrôle s'exécute **une seule fois** à 3 h 00 (plus la gigue), puis à 3 h 30.

Plages actives et fenêtres lors des bascules. Les bornes des plages actives d'un plan (§ 6.4.4) et l'heure de début des fenêtres de maintenance des mises à jour (chapitre 8 du guide d'installation et d'administration) sont des heures **locales**, exprimées dans leur fuseau :

Situation	Effet
Fenêtre de maintenance qui commence dans l'heure inexistante	Elle commence à la première minute valide suivante
Fenêtre de maintenance qui commence dans l'heure répétée	Elle n'a lieu qu'une fois, à la **première occurrence** de l'heure répétée (heure d'été)
Plage active qui franchit minuit et contient la bascule (par exemple de 22:00 à 04:00, la nuit du samedi au dimanche du changement d'heure)	Ses bornes restent fixes en heure locale : elle dure une heure de moins en temps réel au printemps, une heure de plus à l'automne
Fenêtre de maintenance qui contient la bascule	Une fenêtre est définie par un début local et une durée (`windows[].start` , `windows[].duration`), et non par deux bornes. La durée est un temps écoulé depuis le début effectif, et la fin locale se décale d'une heure cette nuit-là (§ 8.4.8 du guide d'installation et d'administration)
Plage active dont une borne tombe dans l'heure inexistante ou répétée	Même principe que pour les échéances : au printemps, la borne est reportée à la première minute valide suivante ; à l'automne, elle est interprétée à sa **première occurrence** (heure d'été). Une plage qui se termine dans l'heure répétée prend donc fin lors de son premier passage

Correspondance avec les outils de mise en sourdine. Lorsque vous reproduisez une plage ou une fenêtre de la sonde dans un outil externe (intervalles de temps d'Alertmanager, temps d'arrêt programmés, périodes de maintenance), configurez-le dans **le même fuseau horaire IANA** que la sonde, et non en UTC ou dans un décalage fixe comme `+01:00` : sinon, les deux périodes se décalent d'une heure à chaque bascule. Le même fuseau est nécessaire mais ne suffit pas à garantir la même résolution des heures ambiguës : chaque outil applique ses propres règles à une heure inexistante ou répétée, et un outil défini par deux bornes civiles ne calcule pas nécessairement la fin d'une fenêtre comme la sonde. Vérifiez donc, sur la première bascule suivant la mise en service, que la sourdine couvre effectivement la période de la sonde, ou évitez de placer les bornes entre 2 h et 3 h du matin.

Les horodatages des résultats restent enregistrés en UTC, quel que soit le fuseau de planification. L'interface d'administration les affiche dans le fuseau configuré dans les paramètres `time` (§ 6.9 du guide d'installation et d'administration).

6.2.4 Choisir entre intervalle et cron

Besoin	Mode recommandé
Surveiller en continu la disponibilité d'un service	Intervalle
Contrôler uniquement pendant les heures d'ouverture d'un service	Cron (ou plan avec plages actives)
Vérifier le résultat d'un traitement planifié (import nocturne, clôture)	Cron, après l'heure de fin prévue du traitement
Suivre un indicateur métier à rythme régulier	Cron, aligné sur l'heure pile ou le quart d'heure pour faciliter la lecture des courbes
Contrôler un environnement de recette disponible en journée	Cron

6.2.5 Valeurs recommandées

Type	Intervalle conseillé	Gigue conseillée	Timeout conseillé
`icmp`, `tcp`, `dns`	1 min	10 s	5 à 10 s
`http`	1 à 5 min	10 s pour 1 min, 30 s pour 5 min	10 à 30 s
`api`	2 à 10 min	30 s	30 à 60 s
`nagios`	1 à 5 min	10 s pour 1 min, 30 s pour 5 min	Selon le plugin
`database`	5 à 15 min ou cron	30 s	30 à 60 s
`browser`	5 à 15 min, de préférence au sein d'un plan	30 s	60 à 180 s

Une règle simple s'applique à tous les types : la gigue doit rester nettement inférieure à l'intervalle (de l'ordre de 10 à 25 % au plus), et le timeout inférieur à l'intervalle.

6.2.6 Contrôle sans planification propre

Un contrôle dont la planification vaut `null` n'est jamais lancé de lui-même. Il n'est exécuté que :

- au sein d'un plan d'exécution ;
- manuellement, depuis l'interface ou par `POST /api/v1/checks/{id}/runs`.

C'est la configuration habituelle des contrôles qui n'ont de sens qu'à l'intérieur d'un parcours enchaîné, comme l'ajout au panier, qui suppose une connexion préalable.

```json
{
  "id": "boutique-panier",
  "name": "Boutique : ajout d'un article témoin au panier (recette)",
  "type": "browser",
  "schedule": null,
  "timeout": "90s",
  "secrets": ["BOUTIQUE_USER", "BOUTIQUE_PASSWORD"],
  "browser": {
    "mode": "script",
    "scriptRef": "boutique-panier@sha256:9a7e31c4d2b8",
    "params": { "baseUrl": "https://recette.boutique.example.com" }
  }
}
```

6.3 Les plans d'exécution

6.3.1 Intérêt

Un **plan d'exécution**, ou *playlist*, est une liste ordonnée de contrôles exécutés **l'un après l'autre**, selon une planification propre au plan. Il répond à trois besoins.

Besoin	Apport du plan
Enchaîner des contrôles dépendants	Inutile de vérifier le panier si la connexion a échoué : avec `stopOnFailure`, le plan s'arrête au premier échec, et l'alerte porte sur la vraie cause
Maîtriser la charge	Les scénarios navigateur sont coûteux et la sonde n'en exécute qu'un à la fois par défaut. Les regrouper dans un plan garantit une exécution ordonnée, sans accumulation dans la file d'attente
Reproduire un parcours de bout en bout	La séquence des contrôles reflète l'ordre réel des actions des utilisateurs : accueil, connexion, recherche, panier

6.3.2 Structure d'un plan

Un plan est décrit par un objet JSON dont la structure est validée par le schéma `plan.schema.json` de la spécification.

Propriété	Obligatoire	Rôle	Défaut
`id`	Oui	Identifiant stable (mêmes règles que pour un contrôle)	—
`name`	Oui	Libellé lisible	—
`enabled`	Non	Plan actif ou suspendu	`true`
`schedule`	Oui	Planification du plan : `interval` ou `cron`, `timezone`, `jitter` (§ 6.2)	—
`activeWindows`	Non	Plages horaires pendant lesquelles le plan s'exécute	Toujours
`stopOnFailure`	Non	Interrompre le plan au premier contrôle en échec	`false`
`maxDuration`	Non	Durée maximale de l'ensemble du plan	`30m`
`items`	Oui	Liste ordonnée des contrôles (1 à 50)	—

Chaque élément de `items` désigne un contrôle par son identifiant (`checkId`). Il peut aussi préciser `continueOnFailure`, qui déroge localement à `stopOnFailure` (§ 6.4.2).

Exemple :

```json
{
  "id": "parcours-boutique",
  "name": "Parcours complet boutique (recette)",
  "enabled": true,
  "schedule": { "cron": "*/10 * * * *", "timezone": "Europe/Paris", "jitter": "30s" },
  "activeWindows": [
    { "days": ["mon", "tue", "wed", "thu", "fri"], "from": "07:00", "to": "21:00" }
  ],
  "stopOnFailure": true,
  "maxDuration": "8m",
  "items": [
    { "checkId": "boutique-accueil" },
    { "checkId": "boutique-connexion" },
    { "checkId": "boutique-recherche", "continueOnFailure": true },
    { "checkId": "boutique-panier" }
  ]
}
```

Ce plan s'exécute toutes les 10 minutes, du lundi au vendredi de 7 h à 21 h, heure de Paris. Il enchaîne quatre scénarios et s'arrête au premier échec, sauf pour la recherche : un échec de la recherche n'empêche pas de vérifier le panier.

6.4 Comportement d'un plan

6.4.1 Déroulement

À chaque échéance de sa planification, et si l'on se trouve dans une plage active, le plan :

1. vérifie qu'aucune exécution précédente du même plan n'est encore en cours. Si c'est le cas, l'échéance est **sautée** ;
2. exécute le premier contrôle de la liste et attend sa fin, **nouvelles tentatives comprises** ;
3. passe au contrôle suivant, ou s'arrête selon les règles de `stopOnFailure` (§ 6.4.2) ;
4. se termine lorsque tous les contrôles ont été traités, que le plan a été interrompu par un échec, ou que la durée maximale `maxDuration` est atteinte (§ 6.4.3).

Chaque contrôle exécuté par un plan produit un résultat ordinaire, avec ses propres état, durée, étapes et erreur. Ce résultat porte le type de déclenchement `plan` et un identifiant d'exécution de plan (`planRunId`). Celui-ci permet de retrouver tous les résultats d'un même passage. Chaque contrôle met à jour son propre état courant, exactement comme s'il avait été lancé seul.

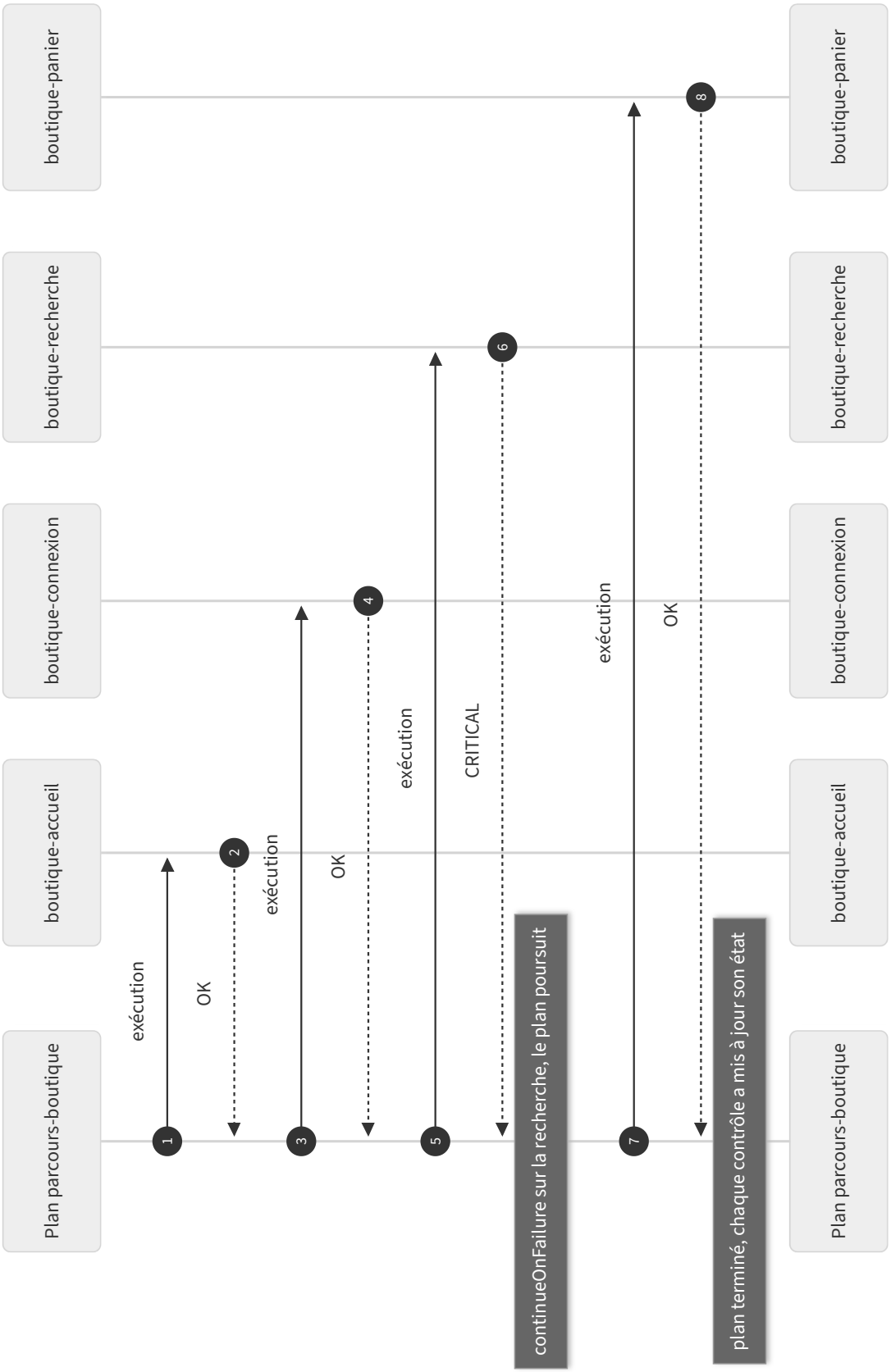


Schéma 6-2 — Déroulement d'un plan d'exécution

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

6.4.2 Arrêt sur échec : stopOnFailure et continueOnFailure

`stopOnFailure` du plan	`continueOnFailure` de l'élément	Comportement si l'élément est en échec
`false` (défaut)	absent	Le plan continue
`false`	`false`	Le plan s'arrête
`true`	absent	Le plan s'arrête
`true`	`true`	Le plan continue

Est considéré comme un **échec** un état final `CRITICAL` ou `UNKNOWN`, après nouvelles tentatives. Un état `WARNING` (réussite dégradée) n'interrompt pas le plan.

Les contrôles non exécutés à cause d'un arrêt ne produisent aucun résultat : leur état courant reste celui de leur dernière exécution.

✓ **Conseil** — Activez `stopOnFailure` lorsque les contrôles suivants dépendent réellement du précédent (connexion, puis actions authentifiées). Utilisez `continueOnFailure` pour les vérifications indépendantes insérées dans le parcours, comme la recherche ou une page d'information.

! **Attention** — Lorsqu'un plan s'arrête sur échec, les contrôles situés après le point d'arrêt ne sont plus vérifiés tant que l'échec persiste. Si l'un d'eux n'est exécuté que par le plan (planification `null`), son état peut devenir `STALE` (§ 6.4.5). C'est le comportement attendu : il indique que cette partie du parcours n'a pas pu être évaluée.

6.4.3 Durée maximale du plan

`maxDuration` borne la durée totale d'un passage. Lorsqu'elle est atteinte, le contrôle en cours est mené à son terme dans la limite de son propre timeout, puis le plan s'arrête. Les contrôles restants ne sont pas exécutés.

Fixez `maxDuration` à une valeur comprise entre :

- la somme des timeouts des contrôles du plan, si vous voulez qu'aucun contrôle ne soit privé d'exécution ;
- l'intervalle du plan, pour qu'un passage se termine avant l'échéance suivante.

Si la première valeur dépasse la seconde, le plan est trop chargé pour sa fréquence (§ 6.6).

6.4.4 Plages actives

Les **plages actives** (`activeWindows`) limitent l'exécution d'un plan à certains jours et horaires, exprimés dans le fuseau horaire de sa planification.

```json
"activeWindows": [
  { "days": ["mon", "tue", "wed", "thu", "fri"], "from": "07:00", "to": "21:00" },
  { "days": ["sat"], "from": "09:00", "to": "13:00" }
]
```

Propriété	Valeurs
`days`	`mon`, `tue`, `wed`, `thu`, `fri`, `sat`, `sun`
`from`, `to`	Heure au format `HH:MM` (00:00 à 23:59)

Un plan peut compter jusqu'à 14 plages. En dehors de toute plage, les échéances du plan sont ignorées. Une plage peut franchir minuit : une heure de début postérieure à l'heure de fin, par exemple de 22:00 à 02:00, indique un franchissement de minuit. Le comportement des plages lors des changements d'heure est décrit au § 6.2.3.

Les plages actives servent par exemple à :

- superviser un environnement de recette uniquement pendant les heures où il est disponible ;
- ne pas générer de trafic synthétique pendant les traitements de nuit d'une application ;
- adapter la supervision aux horaires d'ouverture d'un service.

6.4.5 Plans, états et STALE

Un contrôle est déclaré `STALE` lorsqu'il ne produit aucun résultat pendant plus de deux intervalles de sa planification (§ 3.8.4). Pour un contrôle exécuté uniquement par un plan, la référence est la planification **du plan**. En dehors des plages actives, aucun contrôle n'est déclaré `STALE` pour cause d'absence d'exécution.

6.4.6 Un contrôle dans plusieurs plans

Un même contrôle peut figurer dans plusieurs plans, par exemple un scénario de connexion partagé entre deux parcours. Chaque exécution met à jour l'état courant du contrôle, quel que soit le plan à l'origine. Si deux plans lancent le même contrôle simultanément, la seconde demande est ignorée tant que la première n'est pas terminée.

> ✓ **Conseil** — Évitez de multiplier les plans contenant les mêmes scénarios navigateur : chaque passage consomme du temps de moteur de scénarios. Un parcours unique, bien construit, est généralement préférable à plusieurs plans redondants.

6.5 Créer et piloter un plan

6.5.1 Opérations disponibles

Opération	Méthode et chemin	Rôle minimal
Lister les plans	`GET /api/v1/plans`	`viewer`
Lire un plan	`GET /api/v1/plans/{id}`	`viewer`
Créer	`POST /api/v1/plans`	`operator`
Remplacer	`PUT /api/v1/plans/{id}`	`operator`
Modifier partiellement	`PATCH /api/v1/plans/{id}`	`operator`
Activer, désactiver	`POST /api/v1/plans/{id}/enable`, `POST /api/v1/plans/{id}/disable`	`operator`
Lister les résultats d'un passage de plan	`GET /api/v1/runs?planRunId=…`	`viewer`
Supprimer	`DELETE /api/v1/plans/{id}`	`operator`
Exécuter immédiatement	`POST /api/v1/plans/{id}/runs`	`operator`

Les principes généraux de l'API (authentification, ETag et `If-Match`, format des erreurs) sont décrits au chapitre 6 du guide d'installation et d'administration. Les exemples réutilisent la fonction `sapi` définie au § 6.3.3 du guide d'installation et d'administration.

6.5.2 Procédure de création

Objectif : mettre en place un plan qui enchaîne quatre scénarios de la boutique de recette toutes les 10 minutes, en heures ouvrées.

Prérequis :

- rôle `operator` ;
- édition Enterprise, puisque le plan contient des scénarios navigateur ;
- les quatre contrôles existent et ont été validés individuellement par une exécution d'essai (§ 6.6.5 du guide d'installation et d'administration) ;
- les contrôles destinés à n'être exécutés que par le plan ont une planification `null` (§ 6.2.6).

Étapes :

1. Rédigez la définition du plan dans un fichier `parcours-boutique.json` (exemple du § 6.3.2). Pour la première mise en service, indiquez `"enabled": false`.
2. Créez le plan :

   ```bash
   sapi -X POST -H 'Content-Type: application/json' \
     --data @parcours-boutique.json "$SONDE/api/v1/plans"
   ```

3. Lancez une exécution immédiate et observez le déroulement (§ 6.5.3).
4. Si le résultat est conforme, activez le plan : `POST /api/v1/plans/{id}/enable`.

Résultat attendu : le plan apparaît dans la liste ; ses exécutions sont visibles dans l'historique de chacun des contrôles, avec le déclenchement `plan`.

Vérification : après au moins deux échéances, vérifiez dans l'historique des exécutions (chapitre 7) que les quatre contrôles sont exécutés dans l'ordre, et que la durée totale d'un passage reste nettement inférieure à 10 minutes.

En cas de problème :

- réponse `422` : un identifiant de `items` ne désigne aucun contrôle existant, ou la planification est invalide ;
- plan qui ne s'exécute jamais : vérifiez les plages actives et le fuseau horaire.

6.5.3 Exécution immédiate d'un plan

L'exécution immédiate lance un passage complet du plan, sans attendre son échéance. Elle sert à valider un plan après sa création ou sa modification, et à confirmer qu'un parcours est rétabli après un incident.

```bash
sapi -X POST -D - "$SONDE/api/v1/plans/parcours-boutique/runs"
```

La sonde répond `202 Accepted`. Le déroulement se suit dans l'interface d'administration, par le flux d'événements en temps réel (§ 6.11 du guide d'installation et d'administration) ou en consultant l'historique des contrôles concernés. L'exécution immédiate d'un plan désactivé, ou en dehors de ses plages actives, est autorisée.

6.5.4 Modifier l'ordre ou le contenu d'un plan

La modification d'un plan (`PUT` ou `PATCH`) prend effet au **prochain passage**. Un passage en cours se poursuit avec l'ancienne définition.

Retirer un contrôle d'un plan ne supprime pas le contrôle. S'il n'a pas de planification propre, il n'est plus exécuté du tout : pensez à lui en attribuer une, ou à le désactiver, pour que la situation reste explicite.

6.6 Charge et dimensionnement

6.6.1 Pourquoi calculer la charge

Chaque classe d'exécution dispose d'un nombre limité d'exécutions simultanées (§ 3.11) : 16 pour les contrôles légers, 4 pour les plugins et les bases de données, **une seule** par défaut pour les scénarios navigateur. Si la charge demandée dépasse la capacité d'une classe, les exécutions s'accumulent dans la file d'attente. Leur retard grandit, et les contrôles finissent par devenir `STALE`.

La classe navigateur est de loin la plus exposée. Avant de multiplier les scénarios, il convient donc de calculer la charge.

6.6.2 Méthode de calcul

Pour une classe d'exécution, le **taux d'occupation** se calcule ainsi :

```
TEXTE
Taux d'occupation = Somme, pour chaque contrôle, de (durée moyenne × exécutions par heure)
                    ─────────────────────────────────────────────────────────────────────
                    3 600 secondes × concurrence de la classe
```

Le nombre d'exécutions par heure vaut 3 600 divisé par l'intervalle en secondes. Pour un contrôle cron, comptez les exécutions de l'heure la plus chargée.

Taux d'occupation	Appréciation
Moins de 50 %	Confortable : marge suffisante pour les nouvelles tentatives, les exécutions manuelles et les ralentissements
50 à 70 %	Acceptable ; à surveiller
Plus de 70 %	Risqué : la moindre série de nouvelles tentatives ou de timeouts crée des retards
Plus de 100 %	Saturé : des exécutions sont retardées en permanence, puis des contrôles deviennent `STALE`

> **Information** — Utilisez les durées réellement observées sur la sonde (chapitre 7), et non celles mesurées sur un poste de travail. Comptez aussi les **nouvelles tentatives** : un scénario en échec avec une nouvelle tentative consomme deux fois son temps d'exécution, et un timeout consomme la totalité de la durée maximale.

6.6.3 Exemple chiffré

Une sonde Enterprise (concurrence navigateur : 1) doit exécuter six scénarios navigateur, chacun d'une durée moyenne de 45 secondes.

Configuration initiale : chaque scénario est planifié individuellement toutes les 5 minutes, soit 12 exécutions par heure.

```
TEXTE
Charge = 6 scénarios × 45 s × 12 exécutions/h = 3 240 s par heure
Taux   = 3 240 / (3 600 × 1) = 90 %
```

Le taux de 90 % est trop élevé. Il suffit que deux scénarios échouent par timeout (par exemple 120 secondes chacun, nouvelle tentative comprise) pour saturer la file pendant plusieurs minutes. Les exécutions se décalent, et l'état de certains scénarios devient `STALE`.

Configuration corrigée : les six scénarios sont regroupés dans un plan exécuté toutes les 10 minutes.

```
TEXTE
Durée d'un passage = 6 × 45 s = 270 s (4 min 30 s)
Charge             = 270 s × 6 passages/h = 1 620 s par heure
Taux               = 1 620 / 3 600 = 45 %
```

Le taux de 45 % laisse une marge confortable. La durée d'un passage, 4 min 30 s, reste inférieure à l'intervalle de 10 minutes : une `maxDuration` de 8 minutes est cohérente.

Alternatives si une fréquence de 5 minutes est indispensable pour certains parcours :

Option	Effet
Ne conserver à 5 minutes que les deux parcours les plus critiques ; passer les autres à 15 minutes	Taux de 50 % : 2 × 45 × 12 + 4 × 45 × 4 = 1 800 s par heure
Optimiser les scénarios (chapitre 6 du guide de développement des scénarios) : supprimer les attentes inutiles, bloquer les ressources tierces	Durée moyenne réduite, donc charge réduite
Porter la concurrence navigateur à 2 sur une sonde de 8 Go	Capacité doublée, au prix d'une consommation mémoire accrue (§ 3.11.2). Valeur indicative
Répartir les scénarios sur une seconde sonde	Capacité doublée et vision depuis un second site

6.6.4 Contrôles légers

Pour les contrôles légers, la charge est rarement un problème. Exemple : 60 contrôles `http` toutes les minutes, d'une durée moyenne de 0,8 seconde.

```
TEXTE
Charge = 60 × 0,8 s × 60 exécutions/h = 2 880 s par heure
Taux   = 2 880 / (3 600 × 16) = 5 %
```

La vigilance porte alors davantage sur le **volume de données** produit (§ 3.12.2) et sur la charge imposée aux cibles que sur la capacité de la sonde.

6.6.5 Surveiller la charge dans la durée

Le calcul initial doit être confirmé par l'observation. La sonde publie des métriques de santé (chapitre 2 du guide d'intégration et d'observabilité) qui permettent de suivre :

- le **retard de planification** par classe d'exécution : l'écart entre l'heure prévue et l'heure effective de démarrage ;
- la **profondeur des files d'attente** par classe ;
- les **échéances sautées**, visibles dans l'historique des contrôles (intervalles sans exécution).

Un retard de planification qui augmente régulièrement, ou des échéances sautées fréquentes, imposent de revoir la planification selon les options du § 6.6.3.

6.7 Exemples de configurations

6.7.1 Supervision continue d'un portail

Contrôles légers planifiés individuellement, toutes les minutes, avec une gigue qui étale les exécutions :

```json
{
  "id": "portail-tcp-443",
  "name": "Portail client : port HTTPS",
  "type": "tcp",
  "schedule": { "interval": "1m", "jitter": "10s" },
  "timeout": "5s",
  "retry": { "attempts": 1, "interval": "15s" },
  "tcp": { "host": "portail.example.com", "port": 443, "tls": true }
}
```

6.7.2 Indicateur métier aux heures d'activité

Contrôle de base de données exécuté tous les quarts d'heure pendant les heures d'activité commerciale, du lundi au samedi (édition Professional, chapitre 5) :

```json
{
  "id": "commandes-du-jour",
  "name": "Volume de commandes de la journée",
  "type": "database",
  "schedule": { "cron": "*/15 9-19 * * 1-6", "timezone": "Europe/Paris", "jitter": "30s" },
  "timeout": "30s",
  "secrets": ["DB_REPORTING_PASSWORD"],
  "database": {
    "engine": "postgresql",
    "host": "db-reporting.example.internal",
    "port": 5432,
    "database": "ventes",
    "username": "supervision_ro",
    "passwordSecret": "DB_REPORTING_PASSWORD",
    "query": "SELECT count(*) AS nb FROM commandes WHERE created_at >= current_date",
    "rules": [
      { "column": "nb", "operator": "gt", "value": 0, "severity": "critical" }
    ],
    "exportValues": [ { "column": "nb", "indicator": "commandes_jour" } ]
  }
}
```

À 9 h 00, le nombre de commandes du jour peut être légitimement faible : la règle se limite donc à vérifier qu'il n'est pas nul. Le chapitre 5 explique comment construire des règles adaptées à l'heure de la journée.

6.7.3 Vérification d'un traitement de nuit

Contrôle exécuté une fois par jour, à 6 h 30, après la fin prévue d'un import nocturne :

```json
{
  "id": "import-nuit-statut",
  "name": "Import nocturne : statut du dernier traitement",
  "type": "database",
  "schedule": { "cron": "30 6 * * *", "timezone": "Europe/Paris" },
  "timeout": "60s",
  "secrets": ["DB_ERP_PASSWORD"],
  "database": {
    "engine": "sqlserver",
    "host": "erp-db.example.internal",
    "port": 1433,
    "database": "erp",
    "username": "supervision_ro",
    "passwordSecret": "DB_ERP_PASSWORD",
    "query": "SELECT TOP 1 statut FROM journal_imports ORDER BY date_fin DESC",
    "rules": [
      { "column": "statut", "operator": "eq", "value": "TERMINE", "severity": "critical", "scope":
    "firstRow" }
    ]
  }
}
```

Attention — Un contrôle exécuté une seule fois par jour devient `STALE` s'il ne produit aucun résultat pendant deux jours. Une défaillance de ce contrôle n'est donc signalée par l'état `STALE` qu'avec un retard important. Pour les traitements critiques, ajoutez un second contrôle plus tardif (par exemple à 8 h), ou surveillez les exécutions en échec dans votre plateforme d'observabilité (chapitre 2 du guide d'intégration et d'observabilité).

6.7.4 Parcours de recette en heures ouvrées

Le plan `parcours-boutique` présenté au § 6.3.2 enchaîne quatre scénarios navigateur exécutés uniquement par le plan. Le contrôle de connexion correspondant, sans planification propre, est le suivant :

```json
{
  "id": "boutique-connexion",
  "name": "Boutique : connexion du compte de test (recette)",
  "type": "browser",
  "schedule": null,
  "timeout": "60s",
  "retry": { "attempts": 1, "interval": "30s" },
  "thresholds": { "warningDuration": "20s" },
  "secrets": ["BOUTIQUE_USER", "BOUTIQUE_PASSWORD"],
  "browser": {
    "mode": "steps",
    "steps": [
      { "name": "Ouvrir la page de connexion", "action": "goto", "url": "${param:baseUrl}/connexion"
    },
      { "name": "Saisir l'identifiant", "action": "fill", "selector": "[data-testid=login-email]",
    "value": "${secret:BOUTIQUE_USER}" },
      { "name": "Saisir le mot de passe", "action": "fill", "selector": "[data-testid=login-
    password]", "value": "${secret:BOUTIQUE_PASSWORD}" },
      { "name": "Valider", "action": "click", "selector": "[data-testid=login-submit]",
    "waitForNavigation": true },
      { "name": "Vérifier l'espace client", "action": "assertVisible", "selector": "[data-
    testid=account-menu]" }
    ],
    "params": { "baseUrl": "https://recette.boutique.example.com" },
    "captureOnFailure": true
  }
}
```

La construction des scénarios déclaratifs et scriptés est détaillée au chapitre 4 du guide de développement des scénarios.

6.7.5 Plan de vérification matinale

Plan exécuté une fois par jour ouvré à 7 h 45, avant l'arrivée des utilisateurs, qui vérifie dans l'ordre les briques d'une application et s'arrête au premier échec :

```json
{
  "id": "verification-matinale-erp",
  "name": "ERP : vérification avant ouverture",
  "schedule": { "cron": "45 7 * * 1-5", "timezone": "Europe/Paris" },
  "stopOnFailure": true,
  "maxDuration": "10m",
  "items": [
    { "checkId": "erp-dns" },
    { "checkId": "erp-tcp-443" },
    { "checkId": "erp-portail-http" },
    { "checkId": "import-nuit-statut" },
    { "checkId": "erp-connexion" }
  ]
}
```

Ce plan s'exécute en complément de la supervision continue. En cas d'échec, l'équipe d'exploitation dispose d'un quart d'heure pour intervenir avant l'ouverture. L'ordre des contrôles (DNS, port, page, données, connexion) situe immédiatement le niveau de la panne (§ 3.13).

6.8 Interventions planifiées et fenêtres de maintenance

Pendant une intervention planifiée sur une application supervisée (mise à jour, maintenance de base de données), les contrôles concernés échouent légitimement. Pour éviter des alertes inutiles :

1. avant l'intervention, désactivez les contrôles concernés (§ 6.6.4 du guide d'installation et d'administration) et les plans concernés (`POST /api/v1/plans/{id}/disable`, § 6.5.1) ;
2. après l'intervention, réactivez-les, puis lancez une exécution immédiate pour confirmer le retour à la normale ;
3. si vous utilisez une plateforme d'observabilité, déclarez-y aussi une période de maintenance pour suspendre les alertes. Les données remontées pendant l'intervention restent ainsi disponibles pour analyse.

> ✓ **Conseil** — Intégrez ces étapes à votre procédure de gestion des changements. Avec l'API, elles s'automatisent facilement en début et en fin d'intervention (§ 6.13 du guide d'installation et d'administration).

> → **Évolution prévue** — Des **fenêtres de maintenance** natives, qui suspendront automatiquement un ensemble de contrôles et de plans sur une période définie à l'avance, sont prévues dans une version ultérieure. Elles éviteront le risque d'oublier de réactiver un contrôle après une intervention.

Ces fenêtres ne doivent pas être confondues avec les **fenêtres de maintenance des mises à jour** de la sonde (section `updates`, paramètre `windows`, chapitre 8 du guide d'installation et d'administration). Une fenêtre de mise à jour ne suspend aucun contrôle : elle fixe seulement les créneaux où la sonde peut **activer** une mise à jour, c'est-à-dire redémarrer sur la nouvelle version. L'activation, elle, draine les exécutions en cours puis interrompt la supervision le temps du redémarrage (§ 3.14.2).

> ✓ **Conseil** — Placez les fenêtres de mise à jour en dehors des créneaux où vos plans d'exécution sont les plus importants, et tenez compte de la durée maximale de vos plans au regard du délai de drainage `updates.drainTimeout`.

Chapitre 7 — Consultation et interprétation des résultats

Ce chapitre explique où trouver les résultats produits par la sonde et comment les lire. Il propose ensuite une méthode pour transformer un résultat en diagnostic. Il s'adresse aux exploitants qui consultent la sonde au quotidien, et aux administrateurs qui automatisent la collecte des résultats.

Les notions de contrôle, d'exécution, d'état et de seuil sont définies au chapitre 3. Ce chapitre les suppose connues. La résolution des incidents propres à la sonde (démarrage, réseau, Chromium) est traitée au chapitre 8. La transmission des résultats vers une plateforme d'observabilité externe fait l'objet du chapitre 2 du guide d'intégration et d'observabilité.

> **Information** — Tous les noms d'hôtes, adresses et comptes de ce chapitre sont fictifs. La sonde d'exemple s'appelle `sonde-paris-01.example.internal`.

7.1 Où se trouvent les résultats

La sonde conserve elle-même l'intégralité de ses résultats. Elle ne dépend d'aucun serveur externe pour les produire ni pour les restituer : c'est la conséquence directe de son fonctionnement autonome (chapitre 2). Trois moyens de consultation coexistent.

Moyen	Public	Usage typique	Section
Interface d'administration	Exploitants, administrateurs	Consultation visuelle, analyse d'un échec, déclenchement manuel	§ 7.2
API REST	Administrateurs, outils d'automatisation	Intégration à des scripts, extraction de données, supervision par un outil tiers	§ 7.3
Plateformes d'observabilité	Équipes d'exploitation, centres de supervision	Vue multi-sondes, alertes, historique long terme	Chapitre 2 du guide d'intégration et d'observabilité

Ces trois moyens reposent sur les mêmes données, enregistrées dans la base locale de la sonde. Une exécution consultée dans l'interface, lue par l'API ou retrouvée sous forme de trace dans une plateforme externe a le même identifiant. On peut donc passer de l'un à l'autre sans ambiguïté.

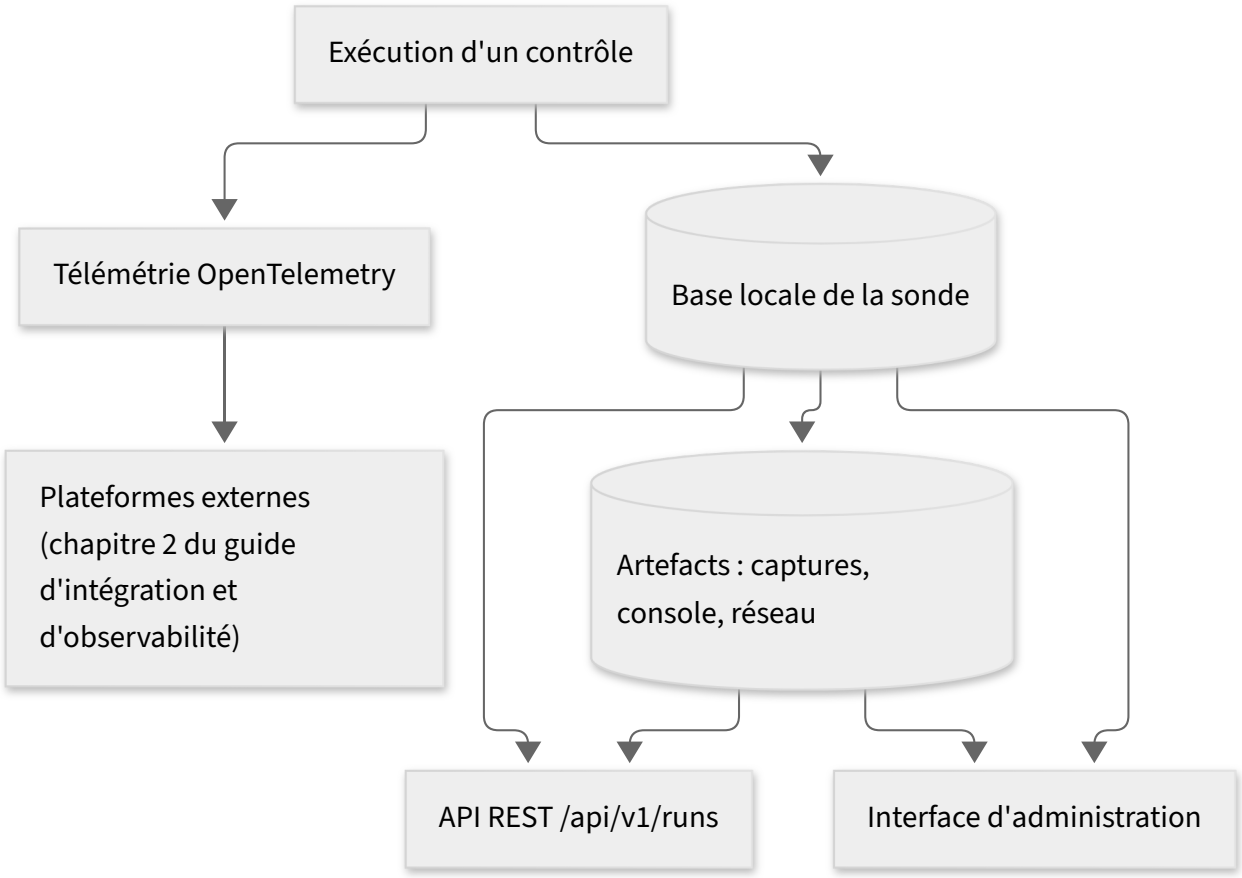


Schéma 7-1 — Où se trouvent les résultats

7.2 Consultation dans l'interface d'administration

L'interface d'administration est accessible depuis un navigateur à l'adresse `https://<adresse-ou-nom-de-la-sonde>/`, avec un compte disposant au minimum du rôle `viewer` (chapitre 6 du guide d'installation et d'administration). Elle n'utilise que l'API publique de la sonde : tout ce qui y est affiché est également disponible par l'API (§ 7.3).

> **Information** — Les descriptions ci-dessous portent sur les informations présentées par l'interface d'administration, pas sur la disposition de ses écrans. Les mêmes informations sont accessibles par l'API REST.

7.2.1 Tableau de bord

Le tableau de bord est la page d'accueil de l'interface. Il répond en un coup d'œil à la question : « Y a-t-il un problème en ce moment ? »

Il présente :

- **une synthèse par état** : le nombre de contrôles actifs en `OK`, `WARNING`, `CRITICAL`, `UNKNOWN` et `STALE` ;

- **la liste des contrôles**, avec pour chacun son nom, son type, son état courant, la date du dernier changement d'état, la date de la dernière exécution et sa durée ;
- **des filtres** par état, par type de contrôle et par étiquette (`tags`, par exemple `application=boutique`) ;
- **l'état de santé de la sonde** elle-même (§ 7.9), signalé de manière distincte : un problème de la sonde n'est pas un problème des applications supervisées.

L'affichage se met à jour automatiquement à chaque fin d'exécution, grâce au flux d'événements décrit au § 7.3.6. Il n'est pas nécessaire de recharger la page.

> ✓ **Conseil** — Triez la liste par date de dernier changement d'état. Les contrôles qui viennent de passer en `CRITICAL` apparaissent ainsi en tête, avant ceux qui sont en échec depuis longtemps et déjà pris en charge.

7.2.2 Fiche d'un contrôle

En sélectionnant un contrôle, on accède à sa fiche. Elle regroupe :

- la définition du contrôle : cible, fréquence, timeout, nouvelles tentatives, seuils (lecture seule pour le rôle `viewer`) ;
- l'état courant et sa date de début ;
- l'historique des exécutions récentes, avec leur état et leur durée ;
- l'évolution de la durée d'exécution sur la période de rétention détaillée ;
- les actions disponibles selon le rôle : exécution immédiate, activation, désactivation (rôle `operator` au minimum).

7.2.3 Détail d'une exécution

Le détail d'une exécution est l'écran d'analyse par excellence. Il présente toutes les informations enregistrées pour cette exécution :

- l'état final, la durée totale, la date de début, le numéro de tentative et l'origine du lancement (planification, plan d'exécution, lancement manuel, nouvelle tentative, exécution d'essai) ;
- pour un échec : le **type d'erreur** (`error.type`), l'**étape** concernée et le **message** détaillé ;
- pour un scénario ou une suite de requêtes API : la **liste des étapes**, avec pour chacune son état et sa durée ;
- les **mesures** propres au type de contrôle : phases d'une requête HTTP, temps aller-retour ICMP, jours restants avant l'expiration d'un certificat, valeurs d'indicateurs métier, données de performance d'un plugin ;
- pour un plugin Nagios : la **sortie texte** du plugin ;
- les **artefacts** associés (§ 7.2.4) ;
- l'**identifiant de trace**, qui permet de retrouver la même exécution dans une plateforme d'observabilité (chapitre 2 du guide d'intégration et d'observabilité).

La liste des étapes se lit comme une frise chronologique. L'étape en échec est mise en évidence, et les étapes suivantes n'ont pas été exécutées : un scénario s'arrête à la première étape en échec. La durée de chaque étape permet de repérer une lenteur localisée, par exemple une authentification qui prend habituellement 2 secondes et qui en prend soudain 9.

7.2.4 Artefacts

Un artefact est un fichier produit par une exécution pour faciliter le diagnostic. Par défaut, les artefacts sont produits **uniquement en cas d'échec**, afin de limiter l'occupation du stockage et l'usure de la carte microSD.

Artefact	Type (`kind`)	Contenu	Contrôles concernés
Capture d'écran	`screenshot`	Image de la page au moment de l'échec	`browser`
Journal de console	`console`	Erreurs et avertissements JavaScript émis par la page	`browser`
Extrait réseau	`network`	Résumé des requêtes réseau de la page : adresses, statuts, durées, en nombre borné	`browser`
Code HTML	`html`	Contenu de la page au moment de l'échec, si activé	`browser`
Journal	`log`	Messages produits par le contrôle (`log.info`, `log.warn` dans un scénario scripté)	Tous

> **Avertissement** — Une capture d'écran ou un extrait de page peut contenir des données affichées par l'application supervisée. Le navigateur masque les champs de mot de passe, mais une page peut réafficher une information sensible (nom, adresse, numéro de commande). Utilisez exclusivement des comptes de test sans données réelles. Limitez l'accès aux artefacts aux personnes habilitées : rôle `viewer` au minimum, à attribuer avec discernement.

Les artefacts sont soumis à une double limite de rétention : 7 jours **et** 500 Mo par défaut (§ 7.8). La première limite atteinte déclenche la suppression des artefacts les plus anciens. Un artefact supprimé reste référencé dans le détail de l'exécution, mais n'est plus téléchargeable.

7.3 Consultation par l'API REST

L'API REST permet d'intégrer les résultats de la sonde à des outils tiers : scripts d'exploitation, outils de reporting, plateformes d'automatisation. Ses principes généraux (authentification, format des erreurs, pagination) sont présentés au chapitre 6 du guide d'installation et d'administration. Sa référence complète figure au § 3.3 de la référence technique et API.

Les exemples ci-dessous utilisent `curl`, disponible sur la plupart des systèmes, et un jeton d'API de rôle `viewer` 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 jeton n'apparaît jamais dans les arguments d'une commande, ni dans l'historique du shell. Le fichier `sonde-ca.pem` contient le certificat de la sonde, récupéré et vérifié selon le § 4.4.4 du guide d'installation et d'administration, ou celui de l'autorité de certification qui l'a émis (chapitre 5 du guide d'installation et d'administration).

```bash
# Variables utilisées dans les exemples de ce chapitre (valeurs fictives),
# définies comme au § 6.3.3 du guide d'installation et d'administration : adresse de la sonde et
    certificat vérifié
export SONDE="https://sonde-paris-01.example.internal"
export CA="$HOME/.sva/sonde-paris-01.pem"
# Le jeton n'est pas placé dans une variable : il est lu dans le fichier
# "$HOME/.sva/entete-auth" (droits 600), créé une fois selon le § 6.3.3 du guide d'installation et
    d'administration.
```

> **Attention** — N'utilisez pas l'option `-k` (`--insecure`) de `curl` pour interroger l'API. Elle désactive la vérification du certificat et expose le jeton à une interception. Utilisez `--cacert` avec le certificat de la sonde ; la seule exception est le téléchargement initial de ce certificat, immédiatement suivi de la vérification de son empreinte (§ 4.4.4 du guide d'installation et d'administration).

7.3.1 Dernier état d'un contrôle

La ressource `GET /api/v1/checks/{id}/status` renvoie l'état consolidé d'un contrôle. Elle est accessible avec un jeton de rôle `metrics`, le plus restreint, qui convient aux outils de supervision tiers comme le plugin Nagios `check_sva` (chapitre 2 du guide d'intégration et d'observabilité).

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

Exemple de réponse :

```json
{
  "checkId": "boutique-connexion",
  "status": "CRITICAL",
  "since": "2026-11-19T09:15:16.764Z",
  "lastRun": {
    "id": "01M7SHD1K2X7Q9N3V4T6W8Y0ZA",
    "startedAt": "2026-11-19T09:15:02.114Z",
    "durationMs": 14650,
    "errorType": "assertion_failed",
    "step": "Vérifier l'accès à l'espace client"
  }
}
```

> **Information** — Les champs de cette réponse (`checkId`, `status`, `since`, `lastRun`) sont décrits dans la description OpenAPI de la sonde (`GET /api/v1/openapi.json`).

7.3.2 Historique des exécutions

La ressource `GET /api/v1/runs` renvoie l'historique des exécutions, du plus récent au plus ancien. L'ordre est **total** : les exécutions sont triées par date de début (`startedAt`), puis par identifiant (`id`) en cas d'égalité. Elle accepte les filtres suivants.

Paramètre	Description	Exemple
`checkId`	Limite aux exécutions d'un contrôle	`checkId=boutique-connexion`
`status`	Limite à un état	`status=CRITICAL`
`from`, `to`	Bornes de la période (RFC 3339, UTC)	`from=2026-11-19T00:00:00Z`
`limit`	Nombre d'éléments par page	`limit=100`
`cursor`	Curseur de la page suivante, fourni par la réponse précédente	`cursor=eyJ…`

Exemple : les exécutions en échec d'un contrôle sur une journée.

```bash
curl --cacert "$CA" \
     -H @"$HOME/.sva/entete-auth" \
     "$SONDE/api/v1/runs?checkId=boutique-connexion&status=CRITICAL&from=2026-11-
    19T00:00:00Z&to=2026-11-20T00:00:00Z&limit=100"
```

La réponse a la forme suivante :

```json
{
  "items": [
    { "id": "01M7SHD1K2X7Q9N3V4T6W8Y0ZA", "checkId": "boutique-connexion", "status": "CRITICAL",
    "startedAt": "2026-11-19T09:15:02.114Z", "durationMs": 14650 }
  ],
  "nextCursor": "eyJiZWZvcmUiOiIwMUpEQjVLOFIy…"
}
```

Pagination. Lorsque la réponse contient `nextCursor`, d'autres résultats existent : répétez la requête avec `cursor=<valeur de nextCursor>`, sans modifier les autres paramètres, jusqu'à ce que `nextCursor` soit absent.

Garanties de lecture (à l'intention des intégrateurs). Les règles suivantes concernent les scripts d'export et les outils qui doivent obtenir un ensemble exhaustif ou reproductible. La pagination par curseur garantit qu'aucune exécution n'est répétée et qu'aucune exécution déjà enregistrée au début de la lecture n'est omise, sous réserve de la rétention. Une exécution enregistrée **pendant** la lecture peut en revanche manquer (ligne « Exécutions nouvelles »).

Règle	Comportement
Ordre	Ordre total (`startedAt` , `id`) : deux exécutions de même date de début restent départagées, et chaque page reprend exactement après la dernière exécution lue
Curseur	Il encode la dernière position lue. Il reste valide **24 heures** ; au-delà, la sonde répond `410 Gone` et la lecture doit reprendre depuis la première page
Exécutions nouvelles	L'historique est lu du plus récent au plus ancien, et la position d'une exécution dépend de sa date de **début** (`startedAt`), et non de sa date d'enregistrement. Une exécution enregistrée pendant la lecture n'est donc renvoyée que si sa date de début est antérieure à la dernière position lue. Sinon, qu'il s'agisse d'une exécution nouvelle ou d'une exécution longue qui s'achève pendant la lecture, elle n'apparaît que dans une lecture ultérieure
Borne de fin `to`	Pour un **export reproductible**, fixez toujours `to` : les exécutions commencées après cette borne sont exclues. Une exécution commencée **avant** `to` mais enregistrée plus tard reste en revanche dans la période et relève de la ligne précédente. Deux lectures successives de la même période ne donnent donc le même ensemble que si `to` précède l'instant de lecture d'au moins la durée maximale d'une exécution, de sorte qu'aucune exécution de la période ne reste en cours, et sous réserve de la rétention
Rétention	Les exécutions purgées par la rétention (§ 7.8) **pendant** la lecture disparaissent des pages suivantes
Instantané	La sonde ne fournit pas d'instantané : la lecture reflète l'état des données au moment de chaque page

> ! **Attention** — Pour une période proche de la limite de rétention (14 jours par défaut, § 7.8), les exécutions les plus anciennes peuvent être purgées entre deux pages. Pour un export exhaustif, choisissez une période nettement plus récente que cette limite, ou lisez-la en une seule session, sans interruption.

7.3.3 Détail d'une exécution

La ressource `GET /api/v1/runs/{runId}` renvoie le résultat complet d'une exécution, au format décrit au § 7.4.

```bash
curl --cacert "$CA" \
     -H @"$HOME/.sva/entete-auth" \
     "$SONDE/api/v1/runs/01M7SHD1K2X7Q9N3V4T6W8Y0ZA"
```

7.3.4 Téléchargement des artefacts

Chaque artefact listé dans le détail d'une exécution est téléchargeable par `GET /api/v1/runs/{runId}/artifacts/{name}`.

```bash
# Télécharger la capture d'écran de l'échec
curl --cacert "$CA" \
     -H @"$HOME/.sva/entete-auth" \
     -o failure.png \
     "$SONDE/api/v1/runs/01M7SHD1K2X7Q9N3V4T6W8Y0ZA/artifacts/failure.png"
```

Un artefact purgé par la rétention renvoie une erreur `404`, au format `application/problem+json`.

7.3.5 Exécution manuelle pour confirmer un incident

Avant d'escalader un incident, il est souvent utile de relancer immédiatement le contrôle pour vérifier que l'échec persiste. Cette action requiert le rôle `operator` : le fichier d'en-tête utilisé doit contenir un jeton de ce rôle. L'exemple suppose un second fichier, `$HOME/.sva/entete-auth-operator`, créé selon la même procédure (§ 6.3.3 du guide d'installation et d'administration) avec un jeton de rôle `operator`.

```bash
curl --cacert "$CA" -X POST \
     -H @"$HOME/.sva/entete-auth-operator" \
     "$SONDE/api/v1/checks/boutique-connexion/runs"
```

La sonde répond `202 Accepted`, avec un en-tête `Location` qui désigne l'exécution créée. Interrogez ensuite cette adresse pour obtenir le résultat de l'exécution. Une exécution manuelle est enregistrée et exportée comme toute autre exécution. Son origine (`trigger`) vaut `manual`.

Pour tester un contrôle sans modifier son état ni alimenter les plateformes externes, utilisez plutôt l'exécution d'essai `?dryRun=true`, décrite au § 6.6.5 du guide d'installation et d'administration.

7.3.6 Suivi en temps réel

La ressource `GET /api/v1/events` diffuse les événements de la sonde au format **Server-Sent Events** (SSE). Avec ce mécanisme standard, le client garde une connexion HTTPS ouverte et reçoit les événements au fil de l'eau, sans interroger la sonde à intervalles réguliers.

Événement	Émis lorsque
`run.started`	Une exécution démarre
`run.finished`	Une exécution se termine (avec son état)
`check.status.changed`	L'état consolidé d'un contrôle change, par exemple de `OK` à `CRITICAL`
`system.alert`	La sonde signale un problème la concernant : stockage, mémoire, température, export

```bash
# Suivre les événements en direct (Ctrl+C pour arrêter)
curl --cacert "$CA" -N \
     -H @"$HOME/.sva/entete-auth" \
     "$SONDE/api/v1/events"
```

Le flux d'événements sert au suivi interactif et à l'automatisation légère. Pour l'alerte de production, privilégiez les mécanismes du chapitre 2 du guide d'intégration et d'observabilité : ils sont conçus pour la haute disponibilité et la gestion des notifications.

7.4 Lire un résultat d'exécution

Le résultat d'une exécution suit un format unique, quel que soit le type de contrôle. L'exemple ci-dessous est celui du scénario déclaratif `boutique-connexion` (§ 4.3.1 du guide de développement des scénarios) en échec, sur une boutique en ligne fictive ; c'est le même que celui du chapitre 3 (§ 3.7.2).

```json
{
  "id": "01M7SHD1K2X7Q9N3V4T6W8Y0ZA",
  "checkId": "boutique-connexion",
  "checkRevision": 7,
  "planRunId": null,
  "trigger": "retry",
  "status": "CRITICAL",
  "attempt": 2,
  "startedAt": "2026-11-19T09:15:02.114Z",
  "durationMs": 14650,
  "error": {
    "type": "assertion_failed",
    "step": "Vérifier l'accès à l'espace client",
    "message": "Élément [data-testid=account-menu] non visible après 10000 ms"
  },
  "steps": [
    { "index": 0, "name": "Ouvrir la page de connexion", "status": "OK", "startedOffsetMs": 50,
    "durationMs": 1830 },
    { "index": 1, "name": "Saisir l'identifiant", "status": "OK", "startedOffsetMs": 1880,
    "durationMs": 160 },
    { "index": 2, "name": "Saisir le mot de passe", "status": "OK", "startedOffsetMs": 2040,
    "durationMs": 140 },
    { "index": 3, "name": "Valider le formulaire", "status": "OK", "startedOffsetMs": 2180,
    "durationMs": 2410 },
    { "index": 4, "name": "Vérifier l'accès à l'espace client", "status": "CRITICAL",
    "startedOffsetMs": 4590, "durationMs": 10004 }
  ],
  "measurements": { "browser.launch_ms": 1180 },
  "artifacts": [
    { "name": "failure.png", "kind": "screenshot", "bytes": 182344 },
    { "name": "console.json", "kind": "console", "bytes": 2210 }
  ],
  "trace": { "traceId": "4bf92f3577b34da6a3ce929d0e0e4736" }
}
```

Champ	Signification	Utilité pour l'analyse
`id`	Identifiant unique de l'exécution (format ULID, triable chronologiquement)	Référence à citer dans un ticket d'incident
`checkId`	Identifiant du contrôle	—

Champ	Signification	Utilité pour l'analyse
`checkRevision`	Version de la configuration du contrôle utilisée	Vérifier si une modification récente du contrôle coïncide avec l'apparition de l'échec
`planRunId`	Exécution du plan d'exécution dont fait partie celle-ci, ou `null`	Relier les contrôles enchaînés d'un même parcours (chapitre 6)
`trigger`	Origine : `schedule`, `plan`, `manual`, `retry`, `dryRun`	Distinguer une exécution planifiée d'un lancement manuel ; ici `retry`, nouvelle tentative après un premier échec
`status`	État final de l'exécution	—
`attempt`	Numéro de tentative (1 = première)	Ici 2 : la première tentative avait déjà échoué ; l'échec est confirmé (§ 7.6)
`startedAt`	Date et heure de début, en UTC	Corrélation avec les journaux de l'application
`durationMs`	Durée totale en millisecondes	—
`error.type`	Catégorie d'erreur, parmi une liste fermée (§ 7.5.2)	Classer et agréger les échecs
`error.step`	Étape en échec	Localiser le problème dans le parcours
`error.message`	Message détaillé, secrets masqués	Comprendre la cause précise
`steps[]`	Étapes avec état, décalage de début et durée	Repérer une lenteur localisée ; ici, l'étape en échec dure un peu plus de 10 secondes, le délai d'attente de l'élément
`measurements`	Mesures propres au type de contrôle	Diagnostic technique (§ 7.5.4)
`output`	Sortie texte d'un plugin Nagios (absent ici)	—
`artifacts[]`	Artefacts disponibles	Capture d'écran, console
`trace.traceId`	Identifiant de trace OpenTelemetry	Retrouver l'exécution dans une plateforme externe

> **Information** — Toutes les dates sont exprimées en UTC dans l'API. L'interface d'administration les affiche dans le fuseau horaire configuré sur la sonde. Pensez-y lorsque vous comparez avec les journaux d'une application, qui sont souvent en heure locale.

Pour les autres types de contrôles, les mesures de `measurements` sont décrites dans les chapitres correspondants : chapitre 4 pour les contrôles réseau et techniques, chapitre 5 pour les contrôles applicatifs et métier. Quelques exemples courants :

Type	Exemples de mesures
`http`	Code de statut, durée des phases DNS, connexion, TLS, premier octet et transfert, jours restants avant l'expiration du certificat
`icmp`	Temps aller-retour moyen, taux de perte
`dns`	Durée de résolution, réponses obtenues
`database`	Valeurs des indicateurs exportés, par exemple le nombre de commandes du jour
`nagios`	Données de performance déclarées exportables
`browser`	Durée de lancement du navigateur (`browser.launch_ms`), exclue de la durée de l'exécution (§ 7.7.1)

7.5 Méthode d'analyse d'un échec

Face à un contrôle en échec, la tentation est d'ouvrir immédiatement la capture d'écran. Une démarche structurée en cinq questions permet d'aller plus vite et d'éviter les conclusions hâtives.

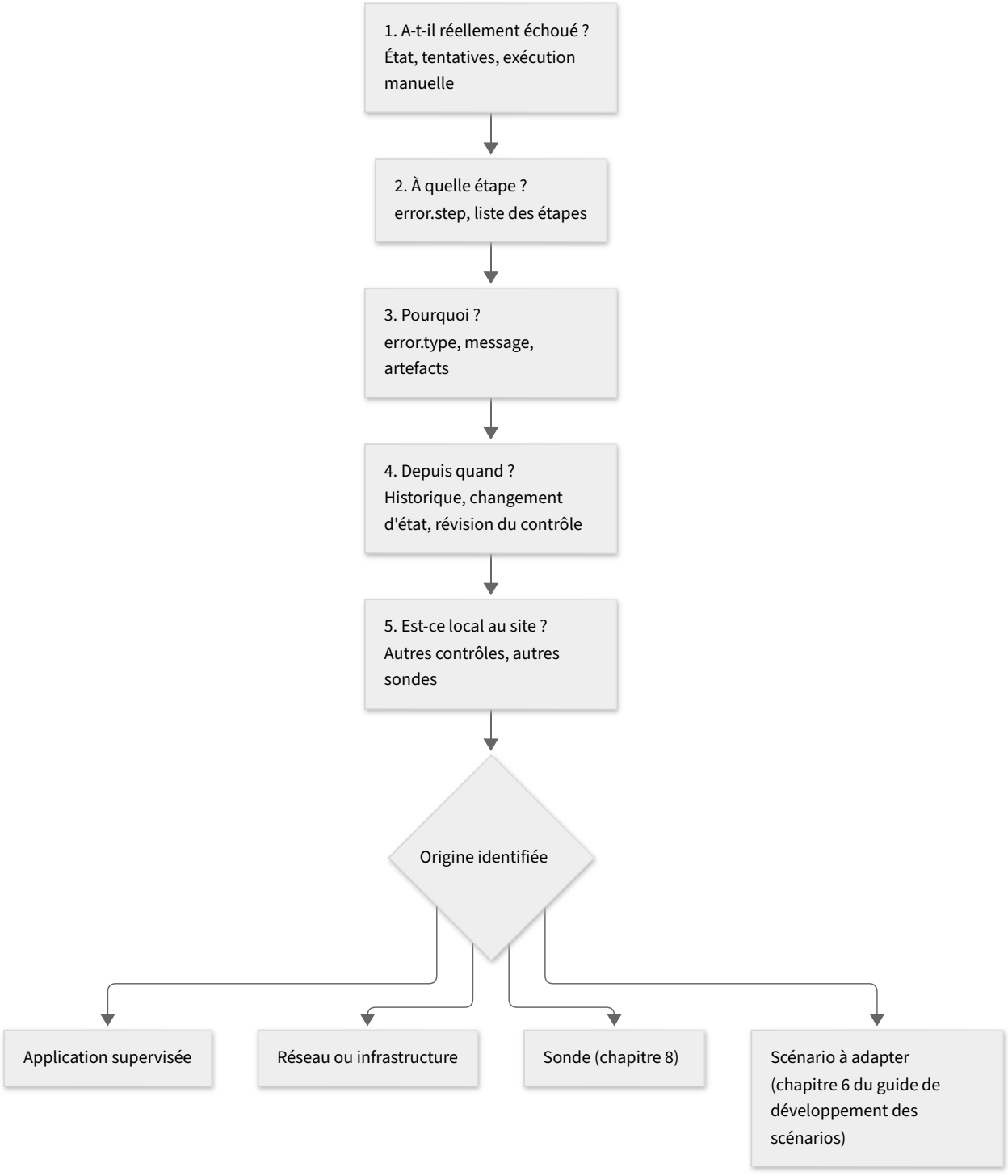


Schéma 7-2 — Méthode d'analyse d'un échec en cinq questions

7.5.1 A-t-il réellement échoué ?

Commencez par qualifier l'état.

- `CRITICAL` : la cible n'a pas satisfait les critères du contrôle. C'est un signal sur l'application ou l'infrastructure supervisée.
- `UNKNOWN` : le contrôle **n'a pas pu être évalué**. La cause se trouve en général du côté de la sonde ou de la configuration du contrôle (navigateur qui ne démarre pas, plugin introuvable, ressources insuffisantes), et non du côté de l'application. Un `UNKNOWN` de type `maintenance` survenu dans une fenêtre de maintenance des mises à jour est normal : l'exécution a été interrompue par le drainage qui précède l'activation (§ 3.14.2). Reportez-vous au chapitre 8.
- `WARNING` : le contrôle a réussi, mais un seuil de performance est dépassé, ou un plugin Nagios a retourné le code 1. L'application fonctionne, mais lentement.
- `STALE` : aucun résultat n'a été produit depuis deux intervalles. Le contrôle ne s'exécute plus, ce qui oriente vers la sonde (§ 7.9 et chapitre 8).

Vérifiez ensuite le champ `attempt`. Si le contrôle prévoit des nouvelles tentatives et que l'état `CRITICAL` survient à la dernière tentative, l'échec a été confirmé plusieurs fois (§ 7.6). En cas de doute, lancez une exécution manuelle (§ 7.3.5).

7.5.2 À quelle étape et pourquoi ?

Le champ `error.step` désigne l'étape en échec, et `error.type` la catégorie de l'erreur, choisie dans une liste fermée. Le tableau suivant est la **référence** de ces catégories : il donne, pour chacune, l'état produit, l'origine la plus probable et la piste à suivre en priorité. Les règles de calcul de l'état sont décrites au chapitre 3 (§ 3.8.2).

`error.type`	Signification	État	Origine la plus probable	Première vérification
`timeout`	Le délai maximal de l'exécution ou d'une attente a été dépassé	`CRITICAL`	Application lente ou bloquée, flux filtré sans rejet ; parfois timeout trop court	Durée habituelle de l'étape dans l'historique
`dns_error`	Le nom de la cible ne se résout pas	`CRITICAL`	Serveurs DNS de la sonde, nom erroné, enregistrement supprimé	Résolution depuis un autre poste du même réseau
`connection_refused`	La cible refuse la connexion	`CRITICAL`	Service arrêté, mauvais port, pare-feu rejetant activement	État du service sur le serveur
`connection_reset`	Connexion interrompue par la cible ou un équipement intermédiaire	`CRITICAL`	Pare-feu, répartiteur de charge, saturation	Journaux des équipements réseau
`tls_error`	Échec de la négociation TLS ou certificat non reconnu	`CRITICAL`	Autorité interne non importée, nom ne correspondant pas au certificat, protocole obsolète	Chaîne de certification (chapitre 5 du guide d'installation et d'administration)
`certificate_expired`	Certificat de la cible expiré	`CRITICAL`	Renouvellement oublié	Date d'expiration du certificat
`http_status`	Code HTTP différent de celui attendu	`CRITICAL`	Erreur applicative (5xx), ressource déplacée (404), accès refusé (401, 403)	Code obtenu dans le message

`error.type`	Signification	État	Origine la plus probable	Première vérification
`assertion_failed`	Une assertion ou une règle n'est pas satisfaite	`CRITICAL` (`WARNING` pour une règle de sévérité `warning`)	Dysfonctionnement fonctionnel, valeur métier hors seuil, ou évolution de l'application	Capture d'écran, contenu attendu, message
`element_not_found`	Un élément attendu de la page est introuvable ou invisible	`CRITICAL`	Page modifiée (sélecteur à mettre à jour), page d'erreur affichée, lenteur	Capture d'écran et code HTML
`navigation_error`	La navigation vers une page a échoué	`CRITICAL`	Site injoignable depuis le navigateur, redirection en boucle	Extrait réseau
`script_error`	Erreur de programmation dans un scénario scripté : le parcours n'a pas pu être vérifié	`CRITICAL`	Script à corriger	Message et numéro de ligne ; exécution d'essai après correction (§ 6.6.5 du guide d'installation et d'administration)
`database_error`	Erreur lors de la connexion ou de la requête SQL	`CRITICAL`	Base indisponible, identifiants, requête invalide, droits du compte	Message du pilote
`auth_failed`	Authentification refusée par la cible	`CRITICAL`	Compte de test verrouillé, mot de passe expiré ou modifié	Validité du secret (chapitre 6 du guide d'installation et d'administration)
`browser_crash`	Le navigateur Chromium s'est arrêté anormalement	`UNKNOWN`	Mémoire insuffisante sur la sonde, page très lourde	Santé de la sonde (§ 7.9), chapitre 8
`plugin_error`	Le plugin Nagios n'a pas pu s'exécuter jusqu'à son terme, ou a renvoyé un code retour en dehors de 0 à 3	`UNKNOWN`	Plugin absent ou non exécutable, arguments invalides, plugin interrompu	Sortie du plugin, nom et arguments (chapitre 4)
`no_data`	Un contrôle `database` n'a renvoyé aucune ligne : les règles ne peuvent pas être évaluées	`UNKNOWN` par défaut, selon `onEmpty`	Requête sans agrégation, filtre trop restrictif, données absentes sur la période	Requête du contrôle (chapitre 5)
`secret_missing`	Un secret référencé par le contrôle est absent de la sonde	`UNKNOWN`	Secret non créé, supprimé ou mal orthographié dans le contrôle ; configuration importée sans ses secrets	Liste des secrets de la sonde et noms déclarés dans `secrets` (§ 6.8 du guide d'installation et d'administration)
`resource_exhausted`	Ressources de la sonde insuffisantes (mémoire, processus)	`UNKNOWN`	Trop d'exécutions simultanées, scénario trop gourmand	Santé de la sonde
`runner_crash`	Le moteur de scénarios s'est arrêté pendant l'exécution	`UNKNOWN`	Incident de la sonde	Chapitre 8

`error.type`	Signification	État	Origine la plus probable	Première vérification
`agent_restart`	Le moteur principal ou la sonde a redémarré pendant l'exécution	`UNKNOWN`	Redémarrage, coupure électrique, mise à jour ; normal s'il est ponctuel	Journal d'audit, historique de mise à jour
`maintenance`	L'exécution a été interrompue au début de l'activation d'une mise à jour, faute de s'être terminée dans le délai de drainage (`updates.drainTimeout`)	`UNKNOWN`	Activation d'une mise à jour dans une fenêtre de maintenance, ou à la demande d'un administrateur	Date de l'exécution comparée aux fenêtres (`updates.windows`), journal d'audit, événements `maintenance.started` et `maintenance.ended` ; s'il se répète, allonger `drainTimeout` (30 minutes au maximum) ou déplacer la fenêtre hors des plans longs (chapitre 8 du guide d'installation et d'administration)
`unknown`	Erreur non classée	`UNKNOWN`	—	Message détaillé et journaux ; support si l'erreur se répète

Le message `error.message` complète la catégorie avec les détails : élément attendu, code HTTP reçu, délai écoulé. Pour les scénarios navigateur, la **capture d'écran** confirme souvent le diagnostic en quelques secondes. Elle révèle par exemple une page de maintenance, une bannière de consentement qui masque le bouton attendu, ou un message d'erreur de l'application.

> ✓ **Conseil** — Un échec `element_not_found` ou `assertion_failed` qui apparaît juste après une mise en production de l'application supervisée signale souvent une **évolution de la page**, et non une panne. Comparez la capture d'écran avec le comportement attendu avant de déclarer un incident. Le chapitre 6 du guide de développement des scénarios explique comment écrire des scénarios robustes à ces évolutions.

7.5.3 Depuis quand ?

La date de début de l'état courant (`since` dans `/status` , date du dernier changement d'état dans l'interface) indique depuis quand le problème est observé. Trois vérifications complètent cette information :

1. **L'historique des exécutions** : l'échec est-il continu, ou intermittent (§ 7.6) ?
2. **La révision du contrôle** (`checkRevision`) : si elle a changé au moment où l'échec est apparu, la modification du contrôle est la première suspecte. Le journal d'audit (chapitre 6 du guide d'installation et d'administration) indique qui l'a modifié et quand.
3. **Les événements extérieurs** : une mise en production, une maintenance réseau ou un changement de mot de passe du compte de test à la même heure.

7.5.4 Est-ce local au site ?

Une sonde observe les applications **depuis l'emplacement où elle est installée**. Un échec peut donc refléter un problème de l'application, ou un problème du chemin réseau entre la sonde et l'application.

Observation	Interprétation probable
Tous les contrôles d'une même application échouent, les autres sont `OK`	Problème de l'application

Observation	Interprétation probable
Tous les contrôles de la sonde échouent ou passent en `UNKNOWN`	Problème de la sonde ou de son raccordement réseau (chapitre 8)
Seuls les contrôles vers des destinations externes (Internet) échouent	Problème de sortie Internet du site : proxy, pare-feu, lien opérateur
Une autre sonde, installée sur un autre site, voit la même application `OK`	Problème du chemin réseau propre au site
Les mesures HTTP montrent une phase `dns` ou `connect` anormalement longue	Réseau ou résolution de noms, plutôt que l'application
La phase `ttfb` (attente du premier octet) est longue	Temps de traitement côté serveur

La comparaison entre plusieurs sondes installées sur des sites différents est le moyen le plus sûr de distinguer une panne applicative d'un problème local. Elle s'effectue dans une plateforme d'observabilité grâce à l'attribut `sva.site` (chapitre 2 du guide d'intégration et d'observabilité).

7.6 Bagotage, nouvelles tentatives et confirmation des incidents

On parle de **bagotage** lorsqu'un contrôle alterne rapidement entre les états `OK` et `CRITICAL`. Le bagotage a plusieurs origines : instabilité réelle de l'application, saturation intermittente du réseau, ou scénario fragile qui échoue de manière aléatoire. Il génère des alertes à répétition et finit par décrédibiliser la supervision.

La sonde dispose d'un mécanisme de **nouvelles tentatives** (`retry`, chapitre 3). Lorsqu'une exécution échoue, la sonde relance le contrôle jusqu'à `retry.attempts` fois, à `retry.interval` d'intervalle, avant de retenir l'état `CRITICAL`. Le champ `attempt` du résultat indique la tentative concernée.

Configuration	Effet	Usage conseillé
`attempts: 0`	Le premier échec déclenche l'état `CRITICAL`	Contrôles très fiables, ou lorsque la moindre indisponibilité doit être signalée
`attempts: 1`, `interval: 30s`	L'échec doit se répéter 30 secondes plus tard	Valeur raisonnable pour la plupart des scénarios navigateur
`attempts: 2` ou plus	Réduit fortement les alertes sur échecs isolés	Applications connues pour des micro-coupures ; au prix d'un délai de détection plus long

> **Attention** — Les nouvelles tentatives allongent le délai de détection d'un incident réel : avec un intervalle de 5 minutes, 2 tentatives espacées de 30 secondes, il faut jusqu'à 6 minutes pour que l'état passe à `CRITICAL`. Recherchez l'équilibre entre réactivité et fiabilité des alertes, et ne masquez pas un scénario fragile par des tentatives multiples : corrigez-le (chapitre 6 du guide de développement des scénarios).

Pour identifier le bagotage, consultez l'historique d'un contrôle sur 24 heures. Un nombre élevé de changements d'état, ou d'exécutions réussies à la deuxième tentative, est un signal à examiner. Dans une plateforme d'observabilité, le compteur `sva.check.runs` (chapitre 2 du guide d'intégration et d'observabilité) n'en donne qu'un indice indirect, dans les limites décrites ci-dessous ; la mesure elle-même repose sur l'historique des exécutions et sur les événements de changement d'état.

Ce compteur mesure exactement le **nombre d'exécutions exportées**, ventilé par contrôle (`sva.check.id`), type de contrôle (`sva.check.type`), état de l'exécution (`sva.check.status`) et type d'erreur (`error.type`). Une hausse de la part des exécutions `CRITICAL` alors que l'état consolidé reste `OK` ne révèle des échecs non confirmés que si les

tentatives non confirmées sont comptées dans ce compteur ; les notes de version précisent ce point. En outre, il ne porte ni numéro de tentative, ni état précédent, ni identifiant d'incident : il ne permet pas, à lui seul, de compter les réussites obtenues à la deuxième tentative ni le nombre de changements d'état. Pour ces deux mesures, utilisez :

- l'**historique des exécutions** (`GET /api/v1/runs`, § 7.3.2), dont chaque élément détaillé porte `attempt` et `trigger` (§ 7.4) : une exécution `OK` de tentative 2 ou plus, précédée d'un échec, est une réussite en nouvelle tentative ;
- les **événements** `check.status.changed` (§ 7.3.6), émis à chaque changement de l'état consolidé : leur nombre sur une période est le nombre de transitions d'état.

7.7 Indicateurs : temps de réponse, percentiles et disponibilité

7.7.1 Temps de réponse

La durée d'exécution d'un contrôle mesure le temps nécessaire pour accomplir le parcours ou la requête **depuis la sonde**. Elle inclut le temps réseau entre la sonde et l'application. Elle peut donc différer des temps mesurés côté serveur ou par un utilisateur situé ailleurs.

Pour un scénario navigateur, la durée totale inclut le chargement des pages, l'exécution du JavaScript de l'application et les attentes explicites du scénario. Elle exclut le lancement du navigateur, mesuré séparément (`browser.launch_ms` dans `measurements`) : une lenteur de Chromium sur la sonde n'est ainsi pas imputée à l'application. La durée de chaque étape est plus parlante que la durée totale : elle isole la partie du parcours qui se dégrade.

7.7.2 Moyenne et percentiles

La **moyenne** des temps de réponse est sensible aux valeurs extrêmes et masque les dégradations ponctuelles. Les **percentiles** décrivent mieux l'expérience réelle.

- Le **50e percentile** (p50, ou médiane) est la durée en dessous de laquelle se trouvent la moitié des exécutions. Il représente le cas typique.
- Le **95e percentile** (p95) est la durée en dessous de laquelle se trouvent 95 % des exécutions. Il représente les cas les plus lents, hors valeurs exceptionnelles.

Exemple : sur 100 exécutions d'un parcours, 94 durent environ 4 secondes et 6 durent environ 20 secondes. La moyenne vaut environ 5 secondes, ce qui semble acceptable. Le p95, lui, est proche de 20 secondes, ce qui révèle que 6 % des parcours sont très lents.

La sonde calcule les percentiles p50 et p95 dans ses **agrégats horaires** (§ 7.8). Les plateformes d'observabilité les calculent à partir des histogrammes de durée exportés (chapitre 2 du guide d'intégration et d'observabilité).

7.7.3 Disponibilité calculée

La disponibilité d'un contrôle sur une période est la proportion d'exécutions réussies parmi les exécutions **évaluables**. Il s'agit d'une proportion d'**exécutions**, et non d'une proportion du **temps** : les périodes sans mesure (sonde arrêtée, maintenance, échéances sautées) et les changements de fréquence peuvent dissocier les deux. La formule recommandée est la suivante :

```
TEXTE
                          nombre d'exécutions OK + WARNING
Disponibilité = ──────────────────────────────────────────────────
                  nombre d'exécutions OK + WARNING + CRITICAL
```

Deux choix sont implicites dans cette formule :

- **`WARNING` compte comme disponible** : l'application a fonctionné, même lentement. Pour un engagement de performance, calculez un indicateur distinct sur les durées.
- **`UNKNOWN` est exclu du calcul** : une exécution `UNKNOWN` signifie que la sonde n'a pas pu évaluer l'application. La compter comme une indisponibilité imputerait à l'application un problème de la sonde, et la compter comme disponible masquerait une absence de mesure. Le nombre d'exécutions `UNKNOWN` doit être suivi séparément, comme indicateur de **qualité de mesure**.

Exemple chiffré. Un contrôle s'exécute toutes les 5 minutes pendant 30 jours, soit 8 640 exécutions planifiées. On observe 8 590 exécutions `OK`, 12 `WARNING`, 26 `CRITICAL` et 12 `UNKNOWN`.

```texte
Exécutions évaluables  = 8 590 + 12 + 26          = 8 628
Disponibilité          = (8 590 + 12) / 8 628      = 99,70 %
Qualité de mesure      = 8 628 / 8 640             = 99,86 %
```

La durée d'indisponibilité correspondante peut s'estimer en multipliant le nombre d'exécutions `CRITICAL` par l'intervalle : 26 × 5 minutes, soit environ 2 heures et 10 minutes. Cette estimation est d'autant plus précise que l'intervalle est court.

Publiez toujours la disponibilité avec sa **couverture** : la part des échéances attendues qui ont donné une exécution évaluable (ici la qualité de mesure, 99,86 %). Une disponibilité de 99,70 % signifie « 99,70 % des exécutions évaluables ont réussi », et non « l'application a été disponible 99,70 % du temps ». Une disponibilité **temporelle** ne peut être déduite de ces chiffres que par une méthode explicitement convenue avec les parties prenantes (par exemple, attribution à chaque exécution de l'intervalle qui la suit, et traitement des périodes non couvertes).

Calcul dans Prometheus. Les requêtes suivantes calculent la même formule à partir du compteur `sva_check_runs_total` (§ 2.3.3 du guide d'intégration et d'observabilité). Elles respectent quatre règles :

- **Vue par sonde** : grouper par `job`, `instance` et `sva_check_id`. Deux sondes peuvent porter un contrôle de même identifiant ; grouper par `sva_check_id` seul mélangerait leurs mesures.
- **Vue de parc** : une agrégation distincte, explicitement nommée, additionne les exécutions de toutes les sondes pour un même identifiant de contrôle. Elle n'a de sens que si ces contrôles mesurent la même cible.
- **États valides** : les états évaluables sont sélectionnés explicitement (`sva_check_status=~"OK|WARNING|CRITICAL"`). Un filtre d'exclusion tel que `sva_check_status!="UNKNOWN"` sélectionnerait aussi une série dépourvue de label d'état.
- **Absence de numérateur et dénominateur nul** : un contrôle sans aucune exécution réussie n'a pas de série `OK` ; sans précaution, la division ne renvoie rien au lieu de 0 %. Le terme `or 0 * (…)` complète le numérateur par zéro lorsque des exécutions évaluables existent. À l'inverse, sans aucune exécution évaluable (dénominateur nul), la condition `> 0` fait disparaître le résultat : « aucune mesure » n'est ni 0 % ni 100 %.

```promql
# Disponibilité sur 30 jours, par sonde et par contrôle
(
    sum by (job, instance, sva_check_id)
    (increase(sva_check_runs_total{sva_check_status=~"OK|WARNING"}[30d]))
  or
    0 * sum by (job, instance, sva_check_id)
    (increase(sva_check_runs_total{sva_check_status=~"OK|WARNING|CRITICAL"}[30d]))
)
/
(
  sum by (job, instance, sva_check_id)
    (increase(sva_check_runs_total{sva_check_status=~"OK|WARNING|CRITICAL"}[30d])) > 0
)
```

PROMQL

```
# Taux d'échec sur 30 jours, par sonde et par contrôle (0 s'il n'y a eu aucun échec)
(
    sum by (job, instance, sva_check_id) (increase(sva_check_runs_total{sva_check_status="CRITICAL"}
    [30d]))
  or
    0 * sum by (job, instance, sva_check_id)
    (increase(sva_check_runs_total{sva_check_status=~"OK|WARNING|CRITICAL"}[30d]))
)
/
(
  sum by (job, instance, sva_check_id)
    (increase(sva_check_runs_total{sva_check_status=~"OK|WARNING|CRITICAL"}[30d])) > 0
)
```

PROMQL

```
# Disponibilité de parc sur 30 jours : toutes les sondes confondues, par contrôle
(
    sum by (sva_check_id) (increase(sva_check_runs_total{sva_check_status=~"OK|WARNING"}[30d]))
  or
    0 * sum by (sva_check_id)
    (increase(sva_check_runs_total{sva_check_status=~"OK|WARNING|CRITICAL"}[30d]))
)
/
(
  sum by (sva_check_id) (increase(sva_check_runs_total{sva_check_status=~"OK|WARNING|CRITICAL"}
    [30d])) > 0
)
```

PROMQL

```
# Contrôles sans aucune exécution évaluable sur 30 jours (à suivre séparément)
sum by (job, instance, sva_check_id) (increase(sva_check_runs_total[30d]))
  unless on (job, instance, sva_check_id)
(sum by (job, instance, sva_check_id)
    (increase(sva_check_runs_total{sva_check_status=~"OK|WARNING|CRITICAL"}[30d])) > 0)
```

Attention — La fonction `increase()` extrapole ses résultats et ne voit pas un contrôle créé, renommé ou dont la série est apparue en cours de période comme un contrôle de 30 jours : le nombre d'exécutions obtenu peut être fractionnaire et légèrement différent du décompte exact. Pour un rapport contractuel, préférez le décompte des exécutions par l'API (§ 7.10) ou les agrégats de la plateforme d'observabilité, et vérifiez ces requêtes sur des contrôles récents et des contrôles à échecs rares avant de les utiliser. En mode envoi OTLP, les labels `job` et `instance` sont dérivés de `service.name` et `service.instance.id` (§ 2.5.3 du guide d'intégration et d'observabilité).

La disponibilité mesurée par la sonde est une disponibilité **observée depuis un point du réseau, à intervalles réguliers**. Une interruption plus courte que l'intervalle entre deux exécutions n'est détectée que si elle coïncide avec une exécution : si elle tombe entièrement entre deux mesures, elle passe inaperçue. Seule une interruption plus longue que l'intervalle augmenté de la gigue (`jitter`) et des éventuels retards de planification (chapitre 6) recouvre forcément au moins une exécution ; une échéance sautée ou replanifiée, par exemple après une exécution trop longue ou pendant une maintenance, allonge d'autant cet écart.

Information — La détection d'une interruption plus courte que l'intervalle n'est pas garantie. Pour un engagement contractuel de niveau de service, définissez précisément la méthode de calcul (formule, traitement de `UNKNOWN` et des fenêtres de maintenance) avec les parties prenantes.

7.8 Historique, rétention et agrégats

La sonde conserve ses résultats selon trois niveaux de détail. Les durées par défaut sont modifiables par un administrateur (chapitre 6 du guide d'installation et d'administration).

Données	Rétention par défaut	Contenu
Exécutions détaillées	14 jours	Résultat complet de chaque exécution, étapes, mesures, erreurs
Agrégats horaires	400 jours	Par contrôle et par heure : nombre d'exécutions, nombre d'échecs, p50 et p95 de la durée
Artefacts	7 jours et 500 Mo	Captures, journaux de console, extraits réseau

Au-delà de 14 jours, le détail d'une exécution n'est plus disponible, mais les agrégats horaires permettent d'analyser les tendances sur plus d'un an : évolution de la disponibilité, dégradation progressive des temps de réponse.

La purge des données expirées s'effectue automatiquement, par petits lots, sans interrompre l'exécution des contrôles. Si l'espace de stockage vient à manquer, la sonde purge par anticipation les données les plus anciennes, en commençant par les artefacts, et le signale par une alerte interne (§ 7.9).

✓ **Conseil** — Pour conserver un historique détaillé au-delà de la rétention locale, exportez la télémétrie vers une plateforme d'observabilité (chapitre 2 du guide d'intégration et d'observabilité). C'est le moyen prévu pour l'archivage de longue durée et pour l'analyse croisée de plusieurs sondes. Augmenter fortement la rétention locale accroît l'occupation et l'usure de la carte microSD.

i **Information** — Si un SSD NVMe facultatif est installé et activé, l'historique, les agrégats et les artefacts y sont conservés plutôt que sur la microSD. Son fonctionnement et le comportement en cas de défaillance sont décrits au chapitre 3 (§ 3.12.2).

i **Information** — Les agrégats horaires sont consultables par l'API avec `GET /api/v1/checks/{id}/aggregates?from=…&to=…&step=1h`. Chaque élément de la réponse couvre une heure et indique son début (`start`), le nombre d'exécutions (`runs`), le nombre d'échecs (`failures`) et les durées médiane et au 95e centile (`p50DurationMs`, `p95DurationMs`). La réponse est paginée (`nextCursor`). Ils sont aussi présentés dans l'interface d'administration sous forme de tendances.

7.9 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.

7.9.1 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). 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)
`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

Composant	Rôle	États notables
`updates`	Mises à jour installées sur l'emplacement inactif et en attente d'activation (chapitre 8 du guide d'installation et d'administration)	`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 8 du guide d'installation et d'administration). Le champ facultatif `maintenanceEndsAt` est omis de l'exemple ci-dessus, qui reste volontairement minimal.

7.9.2 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
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

Indicateur	Métrique exportée	Signal d'alerte
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 (§ 8.4.12 du guide d'installation et d'administration)
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 du guide d'intégration et d'observabilité)

> **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 du guide d'intégration et d'observabilité. Les symptômes de dysfonctionnement de la sonde et leurs remèdes sont traités au chapitre 8.

7.10 Exporter les données

Plusieurs besoins justifient d'extraire les résultats de la sonde : rapport mensuel de disponibilité, analyse dans un tableur, archivage réglementaire.

Besoin	Moyen recommandé
Analyse continue, historique long, multi-sondes	Export de la télémétrie vers une plateforme d'observabilité (chapitre 2 du guide d'intégration et d'observabilité)
Extraction ponctuelle d'exécutions détaillées	API `GET /api/v1/runs`, avec pagination (§ 7.3.2)
Pièces justificatives d'un incident	Détail de l'exécution et artefacts (§ 7.3.3, § 7.3.4)
Sauvegarde de la configuration (pas des résultats)	Export de configuration (chapitre 6 du guide d'installation et d'administration et chapitre 8 du guide d'installation et d'administration)

L'exemple suivant extrait toutes les exécutions d'un contrôle sur une période et les convertit au format CSV. Il utilise `curl` et l'outil standard `jq`, à installer sur le poste d'administration. Il s'agit d'un exemple pédagogique, à adapter à vos besoins.

```bash
#!/usr/bin/env bash
# Extraction des exécutions d'un contrôle au format CSV (exemple pédagogique)
# Prérequis : curl, jq ; variables SONDE et CA définies et fichier d'en-tête
# "$HOME/.sva/entete-auth" créé selon le § 6.3.3 du guide d'installation et d'administration (jeton
    hors ligne de commande)
set -euo pipefail

CHECK="boutique-connexion"
FROM="2026-11-01T00:00:00Z"
TO="2026-12-01T00:00:00Z"
CURSOR=""

echo "id,checkId,status,startedAt,durationMs,errorType"
while : ; do
  URL="$SONDE/api/v1/runs?checkId=$CHECK&from=$FROM&to=$TO&limit=500"
  [ -n "$CURSOR" ] && URL="$URL&cursor=$CURSOR"
  PAGE=$(curl -sS --fail --cacert "$CA" \
              -H @"$HOME/.sva/entete-auth" "$URL")
  # Une ligne CSV par exécution
  echo "$PAGE" | jq -r '.items[] | [.id, .checkId, .status, .startedAt, .durationMs, (.error.type //
    "")] | @csv'
  # Page suivante, ou fin
  CURSOR=$(echo "$PAGE" | jq -r '.nextCursor // empty')
  [ -z "$CURSOR" ] && break
done
```

> **Attention** — Une extraction volumineuse sollicite la sonde. Planifiez-la en dehors des périodes chargées et limitez la période demandée. Rappelez-vous que les exécutions détaillées ne sont conservées que 14 jours par défaut : pour un rapport mensuel, extrayez les données régulièrement, ou utilisez une plateforme d'observabilité.

Le script fixe la borne de fin `TO` : l'extraction porte sur une période close. Elle ne donne le même résultat à chaque reprise que si `TO` précède suffisamment l'instant d'extraction pour qu'aucune exécution commencée avant `TO` ne soit encore en cours, et à la purge par la rétention près (§ 7.3.2) : lancez-la donc après la fin de la période, avec la marge indiquée au § 7.3.2. Si le script est interrompu plus de 24 heures, le curseur expire (`410 Gone`) : relancez alors l'extraction complète. La sonde ne fournit pas d'instantané : des exécutions purgées pendant la lecture n'apparaissent pas dans le fichier produit.

> **Information** — Le curseur renvoyé par l'API est une valeur opaque, encodée en base64url : il ne contient aucun caractère réservé dans une URL et se réinjecte tel quel, sans encodage. Ne cherchez ni à l'interpréter ni à le construire.

7.11 Synthèse

Question	Où trouver la réponse
Y a-t-il un problème en ce moment ?	Tableau de bord ; `GET /api/v1/checks/{id}/status`
Quel parcours a échoué, à quelle étape, pourquoi ?	Détail de l'exécution : `error.step`, `error.type`, `error.message`, artefacts

Question	Où trouver la réponse
L'échec est-il confirmé ?	Champ `attempt` ; exécution manuelle
Depuis quand ?	Date du changement d'état ; historique ; révision du contrôle
Est-ce local au site ?	Comparaison avec les autres contrôles et les autres sondes
L'application est-elle lente ?	Durées par étape, p95, phases HTTP
Quelle disponibilité sur le mois ?	Agrégats horaires ; plateforme d'observabilité ; formule du § 7.7.3
La sonde fonctionne-t-elle correctement ?	Santé de la sonde ; `GET /api/v1/system` ; métriques internes

Chapitre 8 — Diagnostic des contrôles

8.1 Méthode générale de diagnostic

8.1.1 Principe

Un incident de supervision peut avoir trois origines distinctes, qu'il faut séparer avant toute action :

1. **La ressource supervisée** : l'application, l'API, la base ou le service surveillé est réellement défaillant. C'est le cas nominal : la sonde remplit son rôle en le signalant.
2. **Le chemin entre la sonde et la ressource** : réseau, résolution DNS, pare-feu, proxy, certificat. La ressource fonctionne peut-être pour ses utilisateurs, mais pas depuis l'emplacement de la sonde.
3. **La sonde elle-même** : alimentation, stockage, mémoire, configuration du contrôle, script de scénario devenu obsolète.

L'état `UNKNOWN` est réservé aux situations où la sonde n'a pas pu évaluer le contrôle (troisième cas). Un état `CRITICAL` signale en principe un problème de la ressource ou du chemin réseau (premier et deuxième cas). Cette distinction, décrite au chapitre 3, est le premier indice à exploiter.

8.1.2 Démarche recommandée

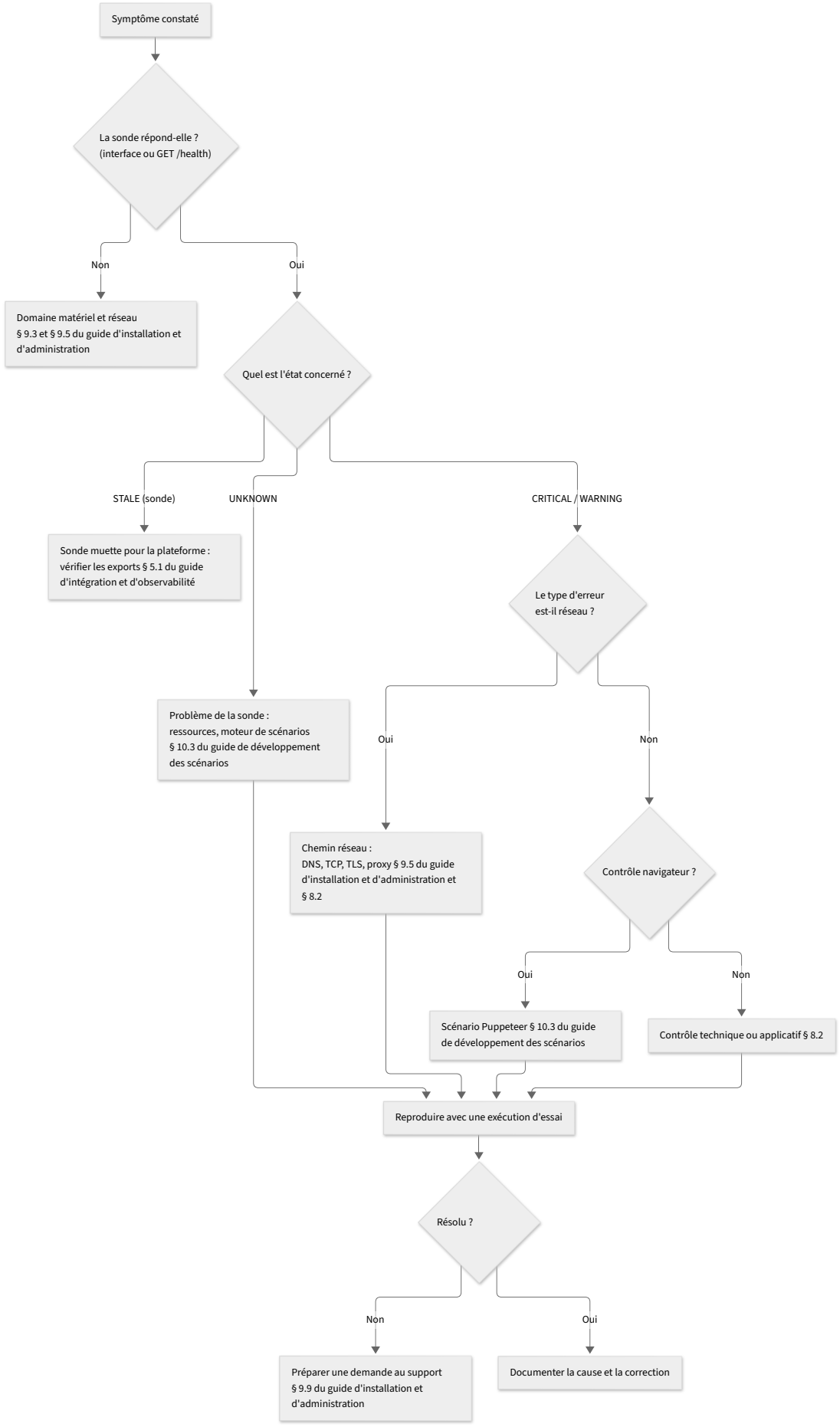


Schéma 8-1 — Démarche de diagnostic

En pratique :

1. **Qualifier le symptôme** : quel contrôle, depuis quand, sur quelle sonde, avec quel état et quel type d'erreur (`error.type`).
2. **Consulter le détail de la dernière exécution en échec** : étape concernée, message, capture d'écran et journal de console pour un scénario navigateur (voir chapitre 7).
3. **Comparer avec l'historique** : un échec isolé, un bagotage ou une rupture nette n'appellent pas la même analyse.
4. **Vérifier si plusieurs contrôles sont touchés en même temps**. Si tous les contrôles d'une même sonde échouent simultanément, la cause se trouve probablement côté sonde ou réseau local, et non dans les applications supervisées.
5. **Reproduire** avec une exécution d'essai (`POST /api/v1/checks/{id}/runs?dryRun=true`). Elle n'enregistre pas d'état et n'exporte rien, ce qui évite de fausser l'historique pendant le diagnostic.
6. **Corriger, puis vérifier** par une nouvelle exécution d'essai avant de réactiver le contrôle ou d'attendre la prochaine échéance.

> ✓ **Conseil** — Notez systématiquement l'heure UTC de l'exécution en échec et son identifiant (`runId`). Ces deux informations permettent de retrouver l'événement dans les journaux, les traces OpenTelemetry et les plateformes d'observabilité.

8.2 Contrôles techniques et applicatifs

8.2.1 Tableau des symptômes

Symptôme	Causes possibles	Vérifications	Actions correctives
`timeout`	Ressource lente ou surchargée ; flux filtré sans rejet explicite (le paquet est ignoré) ; timeout du contrôle trop court ; sonde saturée	Durées des exécutions précédentes ; `sva.agent.scheduler.lag` ; essai `curl -v` ou contrôle `tcp`	Corriger le flux réseau ; adapter `timeout` à un niveau réaliste (chapitre 3) ; réduire la charge de la sonde
`connection_refused`	Service arrêté ; mauvais port ; pare-feu qui rejette activement	Contrôle `tcp` sur l'hôte et le port	Vérifier la cible et le port ; vérifier l'état du service
`dns_error`	Nom inexistant, faute de frappe, DNS injoignable, enregistrement supprimé	Contrôle `dns` ; `getent hosts`	Corriger le nom ou la configuration DNS
`tls_error`	Certificat émis par une autorité non reconnue (autorité interne) ; nom du certificat ne correspondant pas à l'URL ; protocole TLS obsolète côté serveur ; proxy d'inspection TLS	`curl -v` (section TLS) ; détail de l'erreur dans l'exécution	Importer l'autorité de certification interne et la référencer dans le contrôle (`tls.caRef`, chapitre 4) ; utiliser le nom exact du certificat. Ne désactivez la vérification (`tls.verify: false`) qu'en dernier recours, et uniquement en environnement de test

Symptôme	Causes possibles	Vérifications	Actions correctives
`certificate_expired`	Certificat du serveur supervisé expiré	Date d'expiration dans le détail de l'exécution ; métrique `sva.tls.certificate.expiry`	Renouveler le certificat du serveur ; configurer `certificateExpiryWarningDays` pour être averti à l'avance
`http_status`	Code de réponse différent de l'attendu : redirection vers une page de connexion, 401, 403, 404, 5xx	Code reçu, en-tête `Location`	Corriger l'URL ou l'assertion ; ajouter l'authentification ; activer ou désactiver `followRedirects` selon le besoin
`assertion_failed` sur un contrôle HTTP ou API	Contenu modifié par une évolution de l'application ; assertion trop stricte ; réponse d'erreur au format attendu	Corps de la réponse, chemin JSONPath ou XPath	Mettre à jour l'assertion ; préférer des critères stables (chapitre 5)
`plugin_error` (état `UNKNOWN`)	Le plugin n'a pas pu s'exécuter jusqu'à son terme, ou a renvoyé un code retour en dehors de 0 à 3 : plugin introuvable ou non autorisé, non exécutable, arguments invalides, plugin interrompu	Sortie du plugin dans le détail de l'exécution	Vérifier le nom du plugin et ses arguments (chapitre 4) ; pour un plugin personnalisé, vérifier qu'il a été déposé et qu'il est exécutable
Plugin Nagios en état `UNKNOWN` en permanence	Dépendance manquante dans l'environnement isolé ; besoin de privilèges (certains plugins exigent des droits élevés)	Sortie du plugin	Utiliser un contrôle natif équivalent (`icmp`, `tcp`, `dns`, `http`) lorsqu'il existe ; les plugins exigeant des privilèges ne sont pas pris en charge
`database_error`	Hôte ou port injoignable ; nom de base erroné ; pilote incompatible avec la version du serveur ; requête invalide	Message d'erreur ; contrôle `tcp` sur le port du SGBD (5432, 3306, 1433, 1521…)	Corriger les paramètres ; vérifier les versions de SGBD prises en charge, indiquées dans les notes de version
`auth_failed`	Mot de passe changé ou expiré ; compte verrouillé après trop d'échecs ; compte désactivé	Date de dernière rotation du secret ; état du compte côté application	Mettre à jour le secret (chapitre 6 du guide d'installation et d'administration) ; demander un compte de test dont le mot de passe n'expire pas ou dont la rotation est planifiée

Symptôme	Causes possibles	Vérifications	Actions correctives
Règle de base de données en échec alors que les données semblent correctes	Portée de la règle mal choisie (`allRows` , `anyRow` , `firstRow`) ; type de valeur (texte comparé à un nombre) ; nom de colonne en majuscules ou minuscules selon le SGBD	Résultat brut de la requête dans le détail de l'exécution	Ajuster la portée, le type ou l'alias de colonne (chapitre 5)
Résultats incohérents d'une exécution à l'autre	Répartition de charge vers des serveurs hétérogènes ; cache ; données de test modifiées par d'autres utilisateurs ; dépendance à l'heure	Historique, attributs des requêtes, heure des échecs	Stabiliser les données de test ; superviser chaque serveur individuellement si nécessaire ; utiliser des assertions moins dépendantes du contexte
Bagotage (alternance rapide OK / CRITICAL)	Ressource réellement instable ; seuil de performance trop proche des valeurs habituelles ; réseau intermittent	Distribution des durées ; corrélation avec l'heure ou la charge	Configurer des nouvelles tentatives (`retry.attempts` et `retry.interval`) pour confirmer un échec ; ajuster `warningDuration` ; signaler l'instabilité aux équipes concernées
État `STALE` sur un contrôle	Le contrôle n'a produit aucun résultat depuis deux intervalles : sonde saturée, plan d'exécution bloqué par un contrôle plus long que prévu. Un contrôle n'est jamais `STALE` en dehors de la fenêtre d'activité de son plan, ni pendant l'état de maintenance de la sonde et l'intervalle qui suit la reprise	`sva.agent.scheduler.lag` ; durée du plan (`maxDuration`) ; `activeWindows`	Réduire la charge ou la fréquence ; ajuster les timeouts du plan ; vérifier les fenêtres d'activité (chapitre 6)
Contrôle jamais exécuté	Contrôle désactivé ; `schedule: null` sans rattachement à un plan ; plan désactivé ; édition ne comprenant pas ce type de contrôle	État du contrôle et des plans ; édition dans `GET /api/v1/system`	Activer le contrôle ; le rattacher à un plan ; vérifier l'édition
Erreur `resource_exhausted`	Mémoire ou nombre de processus limité atteint dans l'environnement isolé	`free -h` ; métriques de la sonde	Réduire la concurrence ; espacer les exécutions lourdes ; vérifier la RAM de la carte par rapport à l'édition

8.2.2 Distinguer un problème de la sonde d'un problème de la ressource

Lorsqu'un contrôle échoue, posez-vous les questions suivantes, dans l'ordre :

1. **La ressource est-elle accessible depuis un poste situé dans le même réseau que la sonde ?** Si non, le problème n'est pas propre à la sonde.
2. **D'autres contrôles de la même sonde échouent-ils au même moment ?** Si tous échouent, examinez la sonde et son réseau (§ 9.3 et § 9.5 du guide d'installation et d'administration).
3. **Une autre sonde, sur un autre site, supervise-t-elle la même ressource ?** Si elle la voit en `OK`, le problème est lié au chemin réseau du premier site.
4. **L'exécution d'essai reproduit-elle l'échec ?** Si non, l'échec était probablement transitoire. Les nouvelles tentatives confirmeront ou non le diagnostic.

8.3 Types d'erreur et actions de premier niveau

Le tableau suivant reprend la liste fermée des types d'erreur que la sonde associe à une exécution (champ `error.type`). Le § 9.1 en donne une description complète.

Type d'erreur	État habituel	Origine probable	Première action
`timeout`	`CRITICAL`	Ressource lente ou flux filtré	Vérifier la ressource et le flux ; ajuster le timeout
`dns_error`	`CRITICAL`	Nom ou DNS	Contrôle `dns` ; configuration DNS
`connection_refused`	`CRITICAL`	Service arrêté, mauvais port	Contrôle `tcp`
`connection_reset`	`CRITICAL`	Coupure par le serveur, un pare-feu ou un équipement intermédiaire	Journaux du serveur et des équipements réseau
`tls_error`	`CRITICAL`	Autorité, nom ou protocole TLS	Autorité de certification interne, nom exact
`certificate_expired`	`CRITICAL`	Certificat serveur expiré	Renouvellement du certificat
`http_status`	`CRITICAL`	Code HTTP inattendu	Code, redirections, authentification
`assertion_failed`	`CRITICAL`	Contenu différent de l'attendu	Comparer réel et attendu
`element_not_found`	`CRITICAL`	Sélecteur obsolète, élément tardif	Capture d'écran ; sélecteur
`navigation_error`	`CRITICAL`	Navigation impossible	Réseau ; URL
`script_error`	`CRITICAL`	Erreur dans un scénario scripté	Corriger et tester le script
`browser_crash`	`UNKNOWN`	Chromium défaillant	Mémoire, redémarrage
`plugin_error`	`UNKNOWN`	Plugin Nagios qui n'a pas pu s'exécuter jusqu'à son terme, ou code retour en dehors de 0 à 3	Sortie du plugin, nom et arguments
`secret_missing`	`UNKNOWN`	Secret référencé par le contrôle mais absent de la sonde (non créé, supprimé, mal orthographié, configuration importée sans ses secrets)	Un administrateur crée le secret, avec ses hôtes autorisés (§ 6.8 du guide d'installation et d'administration)
`database_error`	`CRITICAL`	Connexion ou requête SQL	Paramètres, pilote, requête
`no_data`	`UNKNOWN` (selon `onEmpty`)	Requête `database` sans aucune ligne renvoyée	Réécrire la requête pour qu'elle renvoie toujours une ligne (chapitre 5)

Type d'erreur	État habituel	Origine probable	Première action
`auth_failed`	`CRITICAL`	Identifiants refusés	Secret, état du compte
`resource_exhausted`	`UNKNOWN`	Sonde à court de ressources	Concurrence, RAM
`runner_crash`	`UNKNOWN`	Moteur de scénarios interrompu	Redémarrage automatique ; si répété, support
`agent_restart`	`UNKNOWN`	Moteur principal redémarré pendant l'exécution	Cause du redémarrage (mise à jour, coupure)
`maintenance`	`UNKNOWN`	Exécution interrompue par le drainage au début de l'activation d'une mise à jour	Aucune si ponctuel ; sinon ajuster `updates.drainTimeout` ou les fenêtres (§ 9.7 du guide d'installation et d'administration)
`unknown`	`UNKNOWN`	Erreur non classée	Message ; support si répété

Chapitre 9 — Références

9.1 Types d'erreur

Chaque exécution en échec porte un type d'erreur issu de la liste fermée ci-dessous (champ `error.type`). Cette liste reprend le tableau de référence du chapitre 7 (§ 7.5.2), qui détaille en outre la première vérification à effectuer pour chaque type. Le message détaillé, propre à chaque cas, figure dans le détail de l'exécution.

Type	Signification	Cause typique	État	Renvoi
`timeout`	Le délai maximal de l'exécution ou d'une attente a été dépassé	Ressource lente, flux filtré sans rejet	`CRITICAL`	§ 8.2
`dns_error`	Le nom n'a pas pu être résolu	Nom erroné, DNS injoignable	`CRITICAL`	§ 9.5 du guide d'installation et d'administration
`connection_refused`	La connexion a été refusée	Service arrêté, mauvais port	`CRITICAL`	§ 8.2
`connection_reset`	La connexion a été interrompue par le serveur ou un équipement intermédiaire	Pare-feu, surcharge, plantage du service	`CRITICAL`	§ 8.2
`tls_error`	La négociation TLS a échoué	Autorité inconnue, nom de certificat incorrect, protocole obsolète	`CRITICAL`	§ 8.2
`certificate_expired`	Le certificat du serveur est expiré	Renouvellement oublié	`CRITICAL`	§ 8.2
`http_status`	Le code de réponse HTTP n'est pas celui attendu	Erreur serveur, redirection, authentification	`CRITICAL`	§ 8.2
`assertion_failed`	Une assertion ou une règle n'est pas satisfaite	Contenu modifié, incident fonctionnel, valeur métier hors seuil	`CRITICAL` (`WARNING` pour une règle de sévérité `warning`)	§ 8.2, § 10.3 du guide de développement des scénarios
`element_not_found`	Un élément attendu n'a pas été trouvé dans la page	Évolution de la page, sélecteur fragile	`CRITICAL`	§ 10.3 du guide de développement des scénarios
`navigation_error`	Le navigateur n'a pas pu charger une page	Réseau, URL, certificat	`CRITICAL`	§ 10.3 du guide de développement des scénarios
`script_error`	Le scénario scripté a levé une erreur de programmation	Erreur dans le script	`CRITICAL`	§ 10.3 du guide de développement des scénarios
`browser_crash`	Chromium s'est arrêté anormalement	Mémoire insuffisante, page trop lourde	`UNKNOWN`	§ 10.3 du guide de développement des scénarios

Type	Signification	Cause typique	État	Renvoi
`plugin_error`	Le plugin Nagios n'a pas pu s'exécuter jusqu'à son terme, ou a renvoyé un code retour en dehors de 0 à 3	Plugin absent ou non exécutable, arguments invalides, plugin interrompu	`UNKNOWN`	§ 8.2
`database_error`	La connexion ou la requête SQL a échoué	Paramètres, pilote, requête	`CRITICAL`	§ 8.2
`no_data`	Un contrôle `database` n'a renvoyé aucune ligne ; les règles ne peuvent pas être évaluées	Requête sans fonction d'agrégation, données absentes	`UNKNOWN` (par défaut, selon `onEmpty`)	§ 8.2, chapitre 5
`auth_failed`	L'authentification a été refusée	Mot de passe changé, compte verrouillé	`CRITICAL`	§ 8.2
`secret_missing`	Un secret référencé par le contrôle est absent de la sonde	Secret non créé, supprimé ou mal orthographié dans le contrôle ; configuration importée sans ses secrets	`UNKNOWN`	§ 6.8.2 et § 6.12.3 du guide d'installation et d'administration
`resource_exhausted`	La sonde a manqué de ressources	Mémoire, concurrence trop élevée	`UNKNOWN`	§ 8.2, § 10.3 du guide de développement des scénarios
`runner_crash`	Le moteur de scénarios s'est interrompu pendant l'exécution	Incident logiciel	`UNKNOWN`	§ 10.3 du guide de développement des scénarios
`agent_restart`	Le moteur principal a redémarré pendant l'exécution	Mise à jour, coupure, redémarrage	`UNKNOWN`	Chapitre 8 du guide d'installation et d'administration
`maintenance`	L'exécution a été interrompue à l'expiration du délai de drainage, au début de l'activation d'une mise à jour	Exécution plus longue que `updates.drainTimeout`	`UNKNOWN`	Chapitre 8 du guide d'installation et d'administration
`unknown`	Erreur non classée	Divers	`UNKNOWN`	§ 9.9 du guide d'installation et d'administration

Information — L'état indiqué est l'état habituel. Le classement définitif d'un type d'erreur en `CRITICAL` ou en `UNKNOWN` est décrit au chapitre 3. Les textes exacts des messages d'erreur sont propres à chaque version du logiciel et figurent dans ses notes de version ; le type d'erreur est l'information à exploiter dans vos règles.

9.2 États et codes retour Nagios

9.2.1 États de la sonde

État	Signification	Valeur de `sva.check.status`	Code retour de `check_sva`
`OK`	Contrôle réussi, dans les seuils	0	0

État	Signification	Valeur de `sva.check.status`	Code retour de `check_sva`
`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`

9.2.2 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;;
```

9.3 Glossaire

Les termes sont classés par ordre alphabétique. Les termes en police `code` désignent des valeurs ou des noms techniques tels qu'ils apparaissent dans l'API et la configuration.

Ce glossaire ne retient que les termes employés dans ce guide. Le glossaire complet figure dans la référence technique et API.

Terme	Définition
802.1X	Norme de contrôle d'accès au réseau : l'équipement doit s'authentifier auprès du commutateur pour obtenir l'accès au port. La sonde la prend en charge sur sa prise Ethernet avec la méthode EAP-TLS (authentification par certificat), configurée dans la section `dot1x`. L'exception par adresse MAC (MAB) reste possible en alternative. Voir chapitre 5 du guide d'installation et d'administration

Terme	Définition
A/B (emplacements système)	Organisation de la microSD en deux emplacements système, l'un actif, l'autre inactif, en lecture seule. Une mise à jour s'installe sur l'emplacement inactif ; la sonde redémarre une fois sur ce nouvel emplacement et revient automatiquement à l'emplacement précédent si elle n'est pas saine dans les 10 minutes. Voir chapitre 8 du guide d'installation et d'administration
API REST	Interface de programmation accessible en HTTPS, qui permet de piloter la sonde par des requêtes au format JSON. Base : `/api/v1`. Voir chapitre 6 du guide d'installation et d'administration et § 3.3 de la référence technique et API
APM (*Application Performance Monitoring*)	Supervision des performances applicatives à partir du trafic réel, au moyen d'agents installés sur les serveurs. Complémentaire de la supervision synthétique. Voir chapitre 1
Artefact	Fichier produit par une exécution, principalement en cas d'échec : capture d'écran, journal de console du navigateur, extrait réseau. Conservé sur la sonde avec un quota
Assertion	Condition vérifiée par un contrôle ou un scénario (code HTTP attendu, texte présent, valeur d'un champ JSON). Une assertion non vérifiée fait échouer l'exécution avec le type d'erreur `assertion_failed`
Attribut	Information associée à une métrique, une trace ou un log (par exemple, l'identifiant du contrôle). Le nombre de valeurs distinctes d'un attribut détermine la cardinalité
Autorité de certification (AC)	Organisme ou service qui signe des certificats TLS. Une entreprise dispose souvent de sa propre autorité interne, que la sonde doit connaître pour vérifier les certificats des applications internes
Bagotage (*flapping*)	Alternance rapide d'un contrôle entre réussite et échec. Il traduit souvent un problème intermittent ou un seuil mal réglé. Voir chapitres 3 et 7
Bridage (*throttling*)	Réduction automatique de la fréquence du processeur du Raspberry Pi en cas de surchauffe ou de sous-tension. Signalé par `vcgencmd get_throttled` et la métrique `sva.hw.throttled`
Cardinalité	Nombre de combinaisons distinctes de valeurs d'attributs pour une métrique. Une cardinalité élevée augmente le coût et la charge des plateformes d'observabilité
Centile (percentile)	Valeur en dessous de laquelle se situe un pourcentage donné des mesures. Le 95e centile (p95) d'une durée est la durée que 95 % des exécutions ne dépassent pas
Centreon	Solution de supervision compatible avec les plugins Nagios. Elle peut interroger la sonde avec le plugin `check_sva`
Certificat autosigné	Certificat TLS signé par la sonde elle-même, généré au premier démarrage. Le navigateur affiche un avertissement tant qu'il n'est pas reconnu. On vérifie alors son empreinte
Certificat d'édition	Fichier signé par l'éditeur, inscrit sur la carte microSD lors de sa préparation et lié à l'identifiant matériel de cette carte. Il détermine l'édition de la sonde, est vérifié localement et n'expire pas. Voir § 4.5 du guide d'installation et d'administration
`check_sva`	Plugin Nagios qui interroge l'état d'un contrôle de la sonde depuis Nagios ou Centreon, sous forme de contrôle actif. Voir chapitre 2 du guide d'intégration et d'observabilité
Chromium	Navigateur web libre, base de Google Chrome. La sonde l'utilise sans interface graphique (mode headless) pour exécuter les scénarios. Il est présent sur la carte dans toutes les éditions, puisqu'elles partagent le même système, mais il est désactivé par le certificat d'édition hors édition Enterprise
Classe d'exécution	Catégorie de contrôles partageant une limite de concurrence : contrôles légers, plugins et bases de données, navigateur. Voir chapitre 3
Collecte (*scrape*)	Interrogation périodique de l'adresse `/metrics` de la sonde par Prometheus (mode *pull*). Voir chapitre 2 du guide d'intégration et d'observabilité
Compte de test	Compte applicatif dédié à la supervision, sans données réelles ni droits superflus, utilisé par les scénarios et contrôles

Terme	Définition
Concurrence	Nombre d'exécutions menées simultanément. Elle est limitée par classe d'exécution pour préserver les ressources de la sonde
Console centrale	Outil facultatif de gestion de plusieurs sondes, hébergé par l'éditeur ou chez le client. Évolution prévue
Contrôle	Unité de supervision configurée sur la sonde : un type (`http`, `browser`…), une cible, une planification et des critères de réussite
Contrôle actif / passif	Dans l'écosystème Nagios, un contrôle actif est lancé par le serveur de supervision ; un contrôle passif lui est transmis par un tiers. Le plugin `check_sva` est un contrôle actif
Cron (expression)	Syntaxe de planification en cinq champs (minute, heure, jour du mois, mois, jour de la semaine), par exemple `*/15 8-20 * * *`
`data-testid`	Attribut HTML ajouté par les équipes de développement pour désigner de façon stable un élément utilisé par les tests et les scénarios. C'est le sélecteur à privilégier. Voir chapitre 6 du guide de développement des scénarios
Datadog	Plateforme d'observabilité en mode SaaS. Elle reçoit les données de la sonde via OTLP. Voir chapitre 2 du guide d'intégration et d'observabilité
Destination d'export	Plateforme qui reçoit la télémétrie de la sonde : backend OTLP, Datadog, Dynatrace, Grafana, Prometheus. Objet `exporter` de l'API
Disponibilité	Part des exécutions réussies (`OK` ou `WARNING`) parmi les exécutions évaluées sur une période ; les exécutions `UNKNOWN` en sont exclues. Voir chapitre 7
DNS	Système de résolution des noms de domaine en adresses IP
Données de performance (*perfdata*)	Mesures chiffrées renvoyées par un plugin Nagios après le caractère `|` de sa sortie
Drainage	Phase qui ouvre l'activation d'une mise à jour : la sonde ne démarre plus aucune exécution et laisse celles en cours se terminer, dans la limite du délai `updates.drainTimeout` (5 minutes par défaut, 30 minutes au maximum). Au-delà, elles sont interrompues en `UNKNOWN` avec le type d'erreur `maintenance`. Voir chapitre 8 du guide d'installation et d'administration
Dynatrace	Plateforme d'observabilité. Elle reçoit les données de la sonde via son API OTLP. Voir chapitre 2 du guide d'intégration et d'observabilité
Édition	Niveau fonctionnel de la sonde (Essential, Professional, Enterprise), déterminé par le certificat d'édition inscrit sur la carte (voir **Certificat d'édition**). L'image système est commune à toutes les éditions
Empreinte (SHA-256)	Valeur calculée qui identifie de manière unique un certificat ou un fichier. On la compare pour vérifier l'authenticité du certificat de la sonde
Endpoint	Adresse (URL) d'un service auquel on envoie des requêtes, par exemple l'endpoint OTLP d'une plateforme
Environnement isolé	Contexte d'exécution séparé (compte système distinct, limites de ressources) dans lequel la sonde exécute les scénarios et les plugins, sans accès à sa base ni aux secrets non déclarés
ETag	Identifiant de version d'une ressource de l'API. Il est renvoyé dans l'en-tête `If-Match` pour éviter d'écraser une modification concurrente
Étape	Subdivision nommée et mesurée d'un scénario ou d'une suite de requêtes API. Elle permet de localiser un échec
État	Résultat synthétique d'un contrôle : `OK`, `WARNING`, `CRITICAL`, `UNKNOWN` ou `STALE`. Voir § 9.2

Terme	Définition
État de maintenance	État de la sonde pendant l'activation d'une mise à jour, du début du drainage jusqu'au retour en service après le redémarrage (événement `maintenance.ended`) : `GET /health` répond `MAINTENANCE` (code 503), `check_sva` renvoie 3 avec le texte `MAINTENANCE`, la métrique `sva.agent.maintenance` vaut 1. Il ne suffit pas à neutraliser les alertes, car la sonde est ensuite injoignable pendant son redémarrage. À ne pas confondre avec le justificatif de maintenance. Voir chapitre 2 du guide d'intégration et d'observabilité et chapitre 8 du guide d'installation et d'administration
Exécution	Occurrence de lancement d'un contrôle, avec son état, sa durée, ses étapes et ses éventuels artefacts. Identifiée par un `runId`
Exécution d'essai (*dry run*)	Exécution déclenchée manuellement qui n'enregistre pas d'état et n'exporte rien. Utilisée pour tester un contrôle
Faux positif, faux négatif	Un faux positif est une alerte émise alors que la ressource fonctionne ; un faux négatif, l'absence d'alerte alors qu'elle est défaillante. Voir chapitre 7
Fenêtre d'activité	Plage horaire pendant laquelle un plan d'exécution est actif (`activeWindows`), par exemple les heures ouvrées
Fenêtre de maintenance	Plage récurrente (jours, heure de début, durée), définie par le client dans la section `updates` (`windows`, 7 au plus), pendant laquelle une mise à jour déjà installée peut être activée, c'est-à-dire la sonde redémarrée sur la nouvelle version. Aucune fenêtre n'est définie par défaut ; sans fenêtre, aucune activation automatique n'a lieu. Voir chapitre 8 du guide d'installation et d'administration
Gigue (*jitter*)	Décalage aléatoire ajouté au démarrage des exécutions pour éviter qu'elles ne partent toutes au même instant
Grafana	Outil de visualisation de données, souvent associé à Prometheus, Mimir, Tempo et Loki
Headless	Mode de fonctionnement d'un navigateur sans fenêtre affichée. Les pages sont chargées et exécutées normalement
Histogramme	Type de métrique qui répartit des mesures (durées) dans des intervalles, et permet de calculer des centiles
HTTP / HTTPS	Protocole du Web ; HTTPS en est la version chiffrée par TLS
ICMP	Protocole de contrôle du réseau, utilisé notamment par la commande `ping`. Son équivalent en IPv6, ICMPv6, assure aussi la découverte des voisins (NDP)
Indicateur métier	Valeur fonctionnelle mesurée par la sonde (nombre de commandes du jour, par exemple), exportée comme métrique `sva.indicator.value`
Interface d'administration	Interface web servie par la sonde en HTTPS, pour la configurer et consulter ses résultats
IPv6	Version 6 du protocole IP. La sonde fonctionne en double pile : IPv4 obligatoire, IPv6 activable (désactivé par défaut), par autoconfiguration (SLAAC) ou adresse statique. Le paramètre `ipFamily` d'un contrôle choisit la famille d'adresses. Voir chapitre 5 du guide d'installation et d'administration
JDBC	Interface standard de connexion aux bases de données depuis Java, utilisée par le moteur principal pour les contrôles `database`
Jeton d'API	Chaîne secrète qui authentifie un outil ou un utilisateur auprès de l'API, avec un rôle déterminé. Elle n'est affichée qu'une fois, à sa création
Journal d'audit	Registre des opérations d'administration : connexions, modifications, téléversements, mises à jour, activations de SSH, récupérations du compte administrateur
JSON	Format texte de données structurées, utilisé par l'API et les définitions de contrôles
JSONPath	Langage d'expression permettant de désigner une valeur dans un document JSON, par exemple `$.available`

Terme	Définition
Justificatif de maintenance	Fichier signé par l'éditeur, distinct du certificat d'édition, importé hors ligne lorsque le client souscrit le contrat de maintenance facultatif. Il porte une date de fin et donne accès aux nouvelles versions fonctionnelles. Son échéance n'affecte aucune fonction acquise ni l'accès aux correctifs de sécurité. Voir chapitre 8 du guide d'installation et d'administration
Licence	Droit d'utilisation perpétuel du logiciel intégré, attaché à la carte microSD livrée. Il se matérialise par le certificat d'édition : aucune clé n'est à saisir, aucun renouvellement n'est nécessaire. À ne pas confondre avec le justificatif de maintenance
Log (journal)	Enregistrement horodaté d'un événement. Les logs d'exécution de la sonde peuvent être exportés via OpenTelemetry. Voir chapitre 2 du guide d'intégration et d'observabilité
Métrique	Mesure numérique horodatée (durée, état, nombre d'exécutions) transmise aux plateformes d'observabilité
Mode réduit (SSD)	Fonctionnement d'une sonde dont le SSD NVMe activé est absent ou défaillant : les contrôles continuent, l'historique local est conservé sur la microSD et l'état de santé `storage` passe en `CRITICAL`. À ne pas prolonger, car l'usure de la microSD s'accélère. Distinct du **mode restreint**. Voir § 8.3.4 du guide d'installation et d'administration
Moteur de scénarios	Composant de la sonde fondé sur Node.js, Puppeteer et Chromium, qui exécute les scénarios navigateur dans l'environnement isolé
Moteur principal	Composant Java de la sonde : planification, API, interface d'administration, contrôles techniques et applicatifs, stockage, export
MTTD, MTTR	Temps moyen de détection (*Mean Time To Detect*) et temps moyen de rétablissement (*Mean Time To Restore*) d'un incident. La supervision synthétique vise d'abord à réduire le MTTD. Voir chapitre 1
Nagios	Logiciel de supervision de référence, dont le format de plugins (codes retour 0 à 3) est largement repris
Node.js	Environnement d'exécution JavaScript utilisé par le moteur de scénarios
Nouvelle tentative (*retry*)	Réexécution automatique d'un contrôle après un échec, avant de confirmer l'état `CRITICAL`
NTP	Protocole de synchronisation de l'heure, indispensable à la sonde
OAuth 2.0	Protocole d'autorisation qui délivre des jetons d'accès aux API. Le flux *client credentials* permet à un compte technique d'obtenir un jeton. Voir chapitre 5
Observabilité	Capacité à comprendre l'état d'un système à partir des données qu'il émet : métriques, traces et logs
Onduleur	Alimentation secourue qui protège la sonde des coupures et micro-coupures électriques. Recommandé. Voir chapitre 8 du guide d'installation et d'administration
OpenAPI	Format standard de description d'une API REST. La sonde publie la description de son API à l'adresse `/api/v1/openapi.json`. Voir chapitre 6 du guide d'installation et d'administration
OpenTelemetry	Standard ouvert de production et de transport des métriques, traces et logs
OTLP	Protocole de transport d'OpenTelemetry, en HTTP ou gRPC
Partition de démarrage	Partition de la microSD au format FAT32, étiquetée `SVA-BOOT`, lisible depuis tout poste de travail. On y dépose `provision.yaml` (et, le cas échéant, les fichiers 802.1X qu'il désigne), `recovery.txt` et `factory-reset.txt` ; la sonde y écrit `setup-code.txt`, `provision-result.txt` et `factory-reset-challenge.txt`. Avec le mécanisme A/B, d'autres partitions de la carte peuvent apparaître sur le poste : elles ne doivent pas être modifiées. La partition `SVA-BOOT` reste sur la microSD, même lorsqu'un SSD est installé
PEM	Format texte d'un certificat ou d'une clé, encodé en Base64 entre des lignes `-----BEGIN …-----` et `-----END …-----`. Format attendu pour les certificats d'autorité et pour le certificat 802.1X signé importé sur la sonde. Voir chapitre 5 du guide d'installation et d'administration

Terme	Définition
Période de support	Période pendant laquelle l'éditeur fournit les correctifs de sécurité d'une version majeure, sans contrat de maintenance : au moins 5 ans après sa mise sur le marché. Sa date de fin (mois et année) est indiquée à l'achat et exposée par l'API (`supportEndsAt`). Au-delà, la sonde continue de fonctionner, sans nouveau correctif. Voir chapitre 8 du guide d'installation et d'administration
Plan d'exécution (*playlist*)	Liste ordonnée de contrôles exécutés en séquence selon une planification commune. Voir chapitre 6
Plugin Nagios	Programme respectant la convention Nagios (code retour, sortie texte, données de performance), exécuté par un contrôle `nagios`
Préavis	Délai minimal entre l'annonce d'une activation de mise à jour et son début (`updates.notice`, 24 heures par défaut, 7 jours au maximum). Une activation qui ne peut respecter le préavis passe à la fenêtre suivante. Voir chapitre 8 du guide d'installation et d'administration
Procédure stockée	Programme enregistré dans une base de données, exécutable par un contrôle `database` (champ `procedure`). Voir chapitre 5
Prometheus	Système libre de collecte et de stockage de métriques. Il interroge la sonde sur `/metrics` ou reçoit ses données via OTLP
PromQL	Langage de requête de Prometheus, utilisé pour les tableaux de bord et les règles d'alerte. Voir chapitre 2 du guide d'intégration et d'observabilité
Proxy	Serveur intermédiaire par lequel passent les connexions sortantes vers Internet
Puppeteer	Bibliothèque JavaScript qui pilote Chromium par programme. Elle est utilisée par les scénarios scriptés
Rétention	Durée de conservation des données sur la sonde (exécutions, agrégats, artefacts, audit)
Rôle	Ensemble de droits attribué à un compte ou un jeton : `admin`, `operator`, `viewer`, `metrics`
RUM (*Real User Monitoring*)	Mesure de l'expérience des utilisateurs réels, collectée dans leur navigateur. Complémentaire de la supervision synthétique. Voir chapitre 1
SaaS	Logiciel fourni sous forme de service hébergé par son éditeur (Datadog, Dynatrace, Grafana Cloud, par exemple)
Scénario	Contrôle de type `browser` qui reproduit un parcours utilisateur dans un navigateur
Scénario déclaratif	Scénario décrit par une liste d'étapes JSON, sans code
Scénario scripté	Scénario écrit en JavaScript avec Puppeteer, réservé au rôle `admin`
Secret	Valeur sensible (mot de passe, jeton, clé) stockée chiffrée par la sonde, jamais relisible, transmise uniquement aux contrôles qui la déclarent
Sélecteur	Expression qui désigne un élément d'une page web (sélecteur CSS, texte, rôle d'accessibilité, XPath)
Server-Sent Events (SSE)	Flux d'événements unidirectionnel sur HTTP. Utilisé par `GET /api/v1/events` pour suivre les exécutions en temps réel. Voir chapitre 6 du guide d'installation et d'administration et chapitre 7
Seuil de performance	Durée au-delà de laquelle une exécution réussie passe en `WARNING` (`warningDuration`) ou en `CRITICAL` (`criticalDuration`)
SLA, SLO	Engagement de niveau de service (*Service Level Agreement*) et objectif de niveau de service (*Service Level Objective*), par exemple un taux de réussite d'un parcours sur un mois. Voir chapitre 1
SOAP	Protocole d'échange de messages XML utilisé par certains web services. Il se supervise par un contrôle `http` en `POST` XML. Voir chapitre 5
Sonde	Appareil de supervision constitué d'un Raspberry Pi 5 et du logiciel préinstallé sur la carte microSD
Sous-tension	Alimentation électrique insuffisante, détectée par le Raspberry Pi. Cause fréquente d'instabilité et de corruption

Terme	Définition
SSD NVMe	Disque à mémoire flash raccordé en PCIe. Sur la sonde, un SSD M.2 2230 ou 2242 monté sur la carte M.2 HAT+ officielle peut, en option, accueillir les données (historique, artefacts, journaux). Le système et le démarrage restent sur la microSD. Voir chapitres 2, 3 et 8 du guide d'installation et d'administration
SSH	Protocole d'accès à distance en ligne de commande. Désactivé par défaut sur la sonde ; un administrateur peut l'activer pour une durée limitée (4 heures par défaut, 72 heures au maximum) ou le verrouiller définitivement. Voir chapitre 5 du guide d'installation et d'administration
`STALE`	État d'un contrôle actif dont aucune exécution ne s'est terminée depuis plus de deux intervalles de planification. Il qualifie le contrôle, et non une exécution. Voir chapitre 3 et § 9.2
Supervision applicative	Vérification du bon fonctionnement des applications du point de vue de leurs utilisateurs et des règles métier, au-delà de la disponibilité des serveurs
Supervision synthétique	Supervision par simulation de requêtes ou de parcours utilisateurs, exécutés à intervalles réguliers depuis un point d'observation. Lorsqu'elle s'appuie sur plusieurs sondes installées dans des environnements différents (réseau interne, sites distants, clients, partenaires, régions), on parle de supervision synthétique distribuée. Voir § 1.6.1
`sva` (préfixe)	Préfixe technique des composants et des identifiants du logiciel de la sonde, issu de sa désignation technique interne « SVA ». On le retrouve dans le nom de la partition de démarrage de la microSD (`SVA-BOOT`), des services (`sva-agent` , `sva-runner`), du nom d'hôte par défaut (`sva-XXXXXX`), du plugin Nagios (`check_sva`), des métriques (`sva.*` , écrites `sva_*` au format Prometheus) et du répertoire de travail utilisé par les exemples sur le poste d'administration (`$HOME/.sva` , § 6.3.3 du guide d'installation et d'administration). Ces identifiants sont stables et indépendants du nom commercial SYNOVEX (chapitre 2)
SYNOVEX	Nom commercial de la gamme de sondes décrite dans cette documentation (éditions SYNOVEX Essential, Professional et Enterprise). Les identifiants techniques (`sva` , `sva-agent` , `sva-runner` , `check_sva`) proviennent de la désignation technique interne « SVA » et en sont indépendants
TCP	Protocole de transport orienté connexion, utilisé par la plupart des services réseau
Timeout (délai maximal)	Durée au-delà de laquelle une exécution ou une attente est interrompue et considérée en échec
TLS	Protocole de chiffrement des communications, utilisé par HTTPS
Trace	Représentation d'une exécution sous forme d'arbre de spans, transmise via OpenTelemetry
Type d'erreur	Catégorie normalisée de la cause d'un échec (`timeout` , `dns_error` …). Voir § 9.1
ULID	Format d'identifiant unique et triable chronologiquement, utilisé pour les exécutions (`runId`)
UTC	Temps universel coordonné. La sonde horodate ses données en UTC ; l'affichage peut utiliser le fuseau horaire local
XPath	Langage de désignation d'éléments dans un document XML ou HTML. Utilisable dans les assertions et, en dernier recours, comme sélecteur