GUIDE DE DÉVELOPPEMENT DES SCÉNARIOS

SYNOVEX

Guide de développement des scénarios

Matériel : Raspberry Pi 5

Éditions : Essential, Professional, Enterprise

Public : Développeurs, intégrateurs, équipes de qualité logicielle, ingénieurs de supervision



SYNOVEX — SIREN 884 713 009

Table des matières

Informations sur le document 4

Chapitre 1 — Architecture des moteurs de scénarios 8
1.1 Composants de la sonde 8
1.2 Cycle de vie d'une exécution 11

Chapitre 2 — Premier scénario, de l'écriture à la mise en service 14

Chapitre 3 — Introduction à Puppeteer 17
3.1 Présentation de Puppeteer 17
3.2 Chromium et le protocole Chrome DevTools 17
3.3 Le navigateur headless 18
3.4 Navigation automatisée et simple requête HTTP 19
3.5 Cas d'utilisation en supervision 21
3.6 Avantages et limites 21
3.7 De CasperJS à Puppeteer 22
3.8 Prérequis techniques dans la sonde 24

Chapitre 4 — Construction d'un scénario 27
4.1 Deux modes d'écriture 27
4.2 Paramètres d'un contrôle navigateur 28
4.3 Scénario déclaratif 29
4.4 Concevoir un scénario avec l'enregistreur de Chrome 32
4.5 Scénario scripté : structure et contrat 33
4.6 Ouvrir une page et naviguer 36
4.7 Rechercher des éléments 37
4.8 Interagir avec la page 39
4.9 Authentification 40
4.10 Gérer l'attente et les délais 41
4.11 Assertions 43
4.12 Traitement des erreurs 44
4.13 Iframes, fenêtres, boîtes de dialogue et téléchargements 46
4.14 Fermeture et libération des ressources 47
4.15 Tester sur poste de travail avec le lanceur local 48
4.16 Déployer un scénario sur la sonde 51
4.17 Cycle de vie et versionnage des scripts 54

Chapitre 5 — Exemples progressifs 55
5.1 Présentation des exemples 55
5.2 Exemple 1 — Vérifier qu'une page web est accessible 58
5.3 Exemple 2 — Vérifier la présence d'un élément attendu 60
5.4 Exemple 3 — S'authentifier sur une application 62
5.5 Exemple 4 — Parcourir plusieurs pages 66
5.6 Exemple 5 — Vérifier le fonctionnement d'un moteur de recherche 69
5.7 Exemple 6 — Ajouter un produit au panier 71
5.8 Exemple 7 — Vérifier un parcours de commande en environnement de test 74
5.9 Exemple 8 — Détecter une erreur fonctionnelle 77
5.10 Synthèse des essais 82
5.11 Adapter les exemples à vos applications 82

Chapitre 6 — Bonnes pratiques des scénarios 84
6.1 Utiliser des sélecteurs robustes 84

6.2 Écrire des assertions explicites 85
6.3 Maîtriser les délais 85
6.4 Attendre des événements plutôt que temporiser 86
6.5 Gérer les erreurs sans les masquer 86
6.6 Limiter la consommation de mémoire et de processeur 87
6.7 Gérer les sessions et les comptes de test 87
6.8 Isoler les scénarios 88
6.9 Nettoyer les ressources et les données 88
6.10 Protéger les secrets 89
6.11 Maîtriser les ressources tierces 89
6.12 Identifier le trafic de la sonde 90
6.13 Captures d'écran, traces et données personnelles 90
6.14 Fiabilité dans la durée 91
6.15 Maintenir les scénarios au rythme des évolutions applicatives 92
6.16 Collaborer avec les équipes de développement 93
6.17 Liste de contrôle avant la mise en service d'un scénario 93

Chapitre 7 — Tests d'API et de bases de données 95
7.1 De la disponibilité technique à la réussite fonctionnelle 95
7.2 Superviser une API REST avec le contrôle http 95
7.3 Enchaîner des requêtes avec le contrôle api 98
7.4 Superviser un web service SOAP 102
7.5 Concevoir des contrôles fonctionnels robustes 104
7.6 Contrôles de bases de données (database) 104
7.7 Précautions et sécurité 113

Chapitre 8 — Sécurité des scripts et des identifiants 115
8.1 Scripts de scénarios 115
8.2 Secrets 116
8.3 Identifiants, secrets et comptes techniques 119

Chapitre 9 — Accès à l'API de la sonde 121
9.1 API REST : principes 121
9.2 Jetons d'API 127
9.3 Gestion des contrôles 128

Chapitre 10 — Analyse des résultats et diagnostic des scénarios 132
10.1 Lire un résultat d'exécution 132
10.2 Méthode d'analyse d'un échec 134
10.3 Scénarios navigateur (Puppeteer) 138

Chapitre 11 — Références 141
11.1 Référence de l'API REST 141
11.2 Types d'erreur 146

SYNOVEX

Guide de développement des scénarios

Sonde de supervision applicative autonome pour Raspberry Pi 5

Information	Valeur
Document	Guide de développement des scénarios — identifiant SYN-DOC-04
Révision documentaire	B
Public	Développeurs, intégrateurs, équipes de qualité logicielle, ingénieurs 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 est la référence des développeurs et des intégrateurs qui écrivent des scénarios de supervision pour SYNOVEX : architecture des moteurs de scénarios, Puppeteer et Chromium en mode headless, scénarios déclaratifs et scénarios scriptés, navigation, sélecteurs, interactions, authentification, attentes et délais, assertions, gestion des erreurs, libération des ressources, essai sur poste de travail et déploiement sur la sonde, exemples complets commentés, bonnes pratiques, tests d'API et de bases de données, sécurité des scripts et des identifiants.

Les exemples sont reproduits intégralement. Ils sont également fournis sous forme de fichiers dans l'archive `synovex-exemples.zip`, livrée avec les PDF de la bibliothèque documentaire et téléchargeable depuis la page d'accueil de la documentation en ligne. Une fois décompressée, l'archive crée le dossier `exemples/` : utilisez de préférence ces fichiers plutôt qu'une copie depuis le PDF.

Pour une première prise en main, suivez le chapitre 2, qui conduit de l'écriture d'un scénario à sa mise en service sur la sonde.

La bibliothèque documentaire SYNOVEX

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

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

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

Services en ligne. L'ensemble de la bibliothèque est également publié sur `https://docs-synovex.incitypark.fr`, avec une recherche et une navigation par profil. Ce site, le portail de support et le service de distribution des mises à jour de l'éditeur sont accessibles après l'annonce de leur mise en service. Avant cette annonce, ou si l'un de ces

services ne répond pas, utilisez le canal de contact prévu par votre contrat. Les adresses citées dans ce guide s'entendent sous cette réserve.

Chapitre 1 — Architecture des moteurs de scénarios

1.1 Composants de la sonde

La sonde réunit un matériel, un système d'exploitation et deux moteurs logiciels.

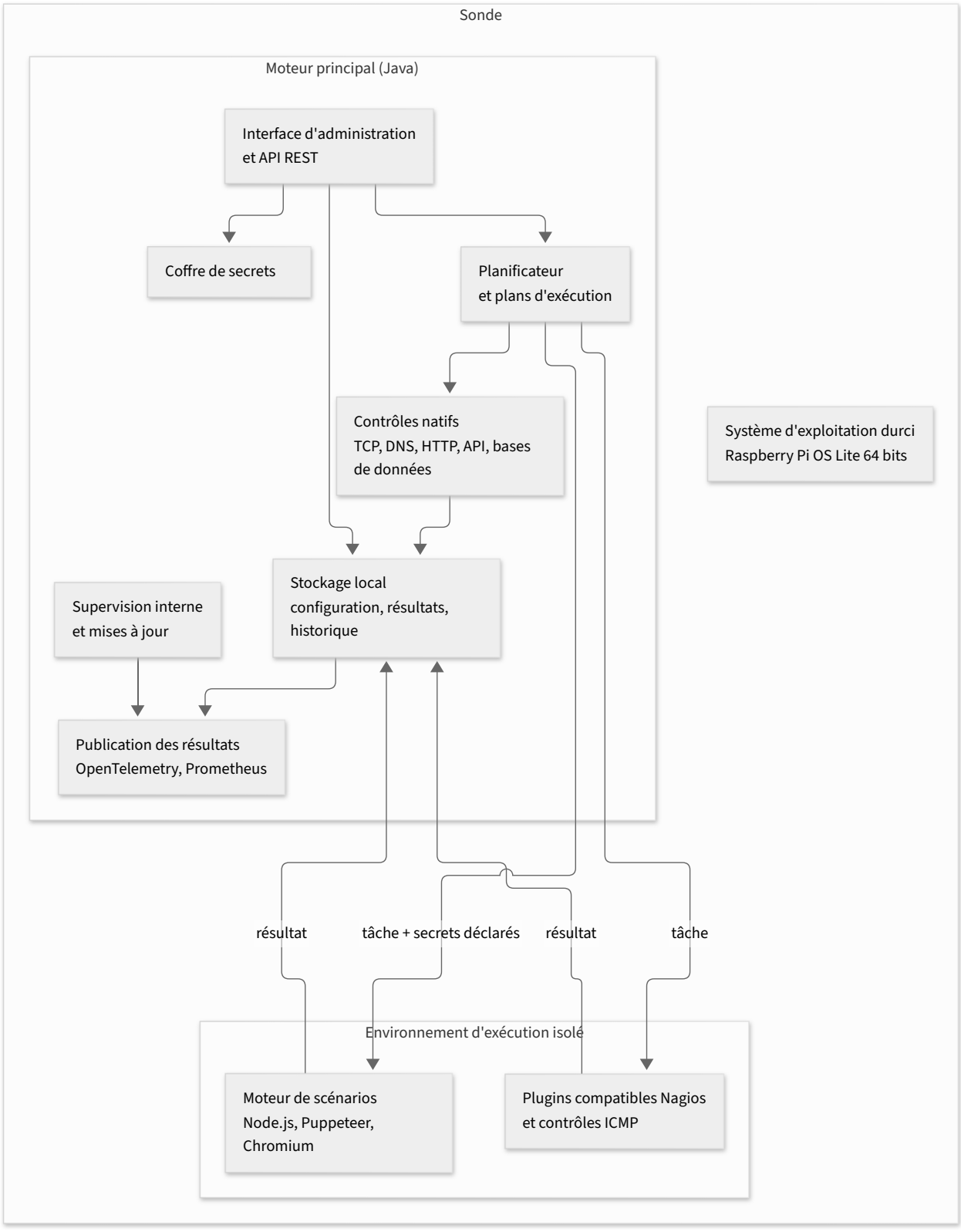


Schéma 1-1 — Composants de la sonde

1.1.1 Matériel et système

La sonde fonctionne sur un **Raspberry Pi 5**, ordinateur monocarte à processeur ARM 64 bits, avec une capacité mémoire adaptée à l'édition (chapitre 2 du guide d'installation et d'administration). Le système d'exploitation est **Raspberry Pi OS Lite 64 bits**, distribution dérivée de Debian maintenue par le fabricant de la carte, installé sans

interface graphique et durci : services inutiles retirés, accès distant SSH désactivé par défaut, comptes et droits limités au strict nécessaire. Le système est configuré pour limiter les écritures sur la carte microSD, dont la durée de vie dépend du volume écrit (chapitre 8 du guide d'installation et d'administration).

La carte microSD porte deux **emplacements système** (A et B), dont un seul est actif. Le système y est monté en **lecture seule** : il n'est pas modifié en fonctionnement normal, de sorte qu'une coupure électrique n'interrompt aucune écriture du système. Cette disposition ne protège pas contre une défaillance du support lui-même (§ 8.3 du guide d'installation et d'administration). Une mise à jour s'installe sur l'emplacement inactif, et la sonde revient automatiquement à l'emplacement précédent si la nouvelle version ne fonctionne pas correctement (§ 8.4.5 du guide d'installation et d'administration). Les données de la sonde sont stockées sur une partition séparée de la microSD. Un **SSD NVMe** facultatif, monté sur la carte M.2 HAT+ officielle, peut accueillir les données les plus volumineuses ; la sonde continue de démarrer sur la microSD (§ 8.3.4 du guide d'installation et d'administration).

1.1.2 Moteur principal

Le **moteur principal** est le cœur de la sonde. Développé en Java, il fonctionne en permanence et assure les fonctions suivantes.

Fonction	Description
Interface d'administration et API REST	Serveur HTTPS qui publie l'interface web et l'API documentée ; authentification des utilisateurs et des jetons, contrôle des droits, journal d'audit (chapitre 9). Le moteur d'authentification repose sur des fournisseurs interchangeables : la version actuelle utilise des comptes locaux, et l'ajout d'OpenID Connect est prévu dans une version ultérieure (§ 6.4 du guide d'installation et d'administration)
Planification	Calcul des échéances de chaque contrôle et de chaque plan d'exécution, répartition dans le temps pour éviter les pics, limitation du nombre d'exécutions simultanées (chapitre 6 du guide d'utilisation et de supervision)
Contrôles natifs	Exécution directe des contrôles `tcp`, `dns`, `http`, `api` et `database`. Les connexions aux bases de données utilisent les pilotes standard JDBC de chaque éditeur de base (chapitre 7)
Évaluation	Application des critères de réussite (assertions, seuils), des nouvelles tentatives et des règles de calcul de l'état (chapitre 3 du guide d'utilisation et de supervision)
Stockage local	Enregistrement de la configuration, des résultats, des étapes et des agrégats dans une base SQLite locale ; purge selon la durée de rétention
Coffre de secrets	Conservation chiffrée des mots de passe et jetons utilisés par les contrôles ; transmission limitée aux secrets déclarés par chaque contrôle
Publication	Production de métriques, traces et journaux au format OpenTelemetry ; export OTLP vers les destinations configurées ; endpoint Prometheus (chapitre 2 du guide d'intégration et d'observabilité)
Supervision interne	Surveillance de l'état de la sonde elle-même : mémoire, température, alimentation, espace disque, retards de planification, échecs d'export
Mises à jour	Vérification, téléchargement, contrôle de signature et installation des images système sur l'emplacement inactif, sans interrompre les contrôles ; canaux `security` (automatique par défaut) et `feature` (manuel par défaut) ; activation automatique dans une fenêtre de maintenance définie par le client, après préavis et drainage des exécutions, ou, par exception, à la demande d'un administrateur ou dès que possible pour un correctif critique si le client l'a choisi ; validation après redémarrage et retour arrière automatique (§ 8.4 du guide d'installation et d'administration)

1.1.3 Environnement d'exécution isolé

Certains contrôles exécutent du code qui ne provient pas de l'éditeur : scripts de scénarios écrits par vos équipes, plugins compatibles Nagios. Ils s'exécutent dans un **environnement isolé**, distinct du moteur principal :

- un compte système dédié, sans accès en lecture à la base de configuration ni au coffre de secrets ;

- des limites de mémoire, de processeur et de nombre de processus, qui empêchent un scénario défaillant d'épuiser les ressources de la sonde ;
- un processus neuf pour chaque exécution, arrêté de force si le délai maximal est dépassé ;
- la réception, pour chaque exécution, des seuls secrets que le contrôle a explicitement déclarés.

Le **moteur de scénarios** fait partie de cet environnement. Il repose sur **Node.js**, environnement d'exécution JavaScript, sur **Puppeteer**, bibliothèque de pilotage du navigateur, et sur **Chromium** en mode headless. Pour chaque exécution d'un scénario, il démarre un navigateur neuf avec un profil vierge conservé en mémoire, exécute les étapes, mesure leur durée, réalise une capture d'écran en cas d'échec, puis ferme le navigateur. Aucune session, aucun cookie ni aucune donnée de navigation n'est conservé d'une exécution à l'autre. La version de Chromium est fournie et validée par l'éditeur avec la version de Puppeteer correspondante (chapitre 3).

Les **plugins compatibles Nagios** et les contrôles **ICMP** (ping), qui reposent sur des utilitaires système, s'exécutent dans ce même environnement isolé, quelle que soit l'édition. Les plugins sont lancés directement, sans interpréteur de commandes, ce qui empêche toute injection de commande par leurs arguments.

> **Information** — Cette séparation explique pourquoi la création de scénarios scriptés et l'ajout de plugins personnalisés sont réservés aux administrateurs : ces opérations reviennent à exécuter du code sur le réseau où la sonde est installée. Le chapitre 9 détaille les rôles.

1.1.4 Répartition des types de contrôles

Type de contrôle	Composant d'exécution	Édition minimale
`icmp` (ping)	Environnement isolé, utilitaire système	Essential
`tcp`, `dns`, `http`	Moteur principal	Essential
`nagios`, plugins fournis	Environnement isolé	Essential
`nagios`, plugins personnalisés	Environnement isolé	Professional
`api`	Moteur principal	Professional
`database`	Moteur principal (JDBC)	Professional
`browser` (scénarios déclaratifs et scriptés)	Moteur de scénarios	Enterprise

Chaque famille d'exécution dispose de sa propre limite de simultanéité. Un parcours navigateur long ne retarde donc pas les contrôles réseau, qui s'exécutent en parallèle dans leur propre file. Par défaut, un seul parcours navigateur s'exécute à la fois, car Chromium est le composant le plus consommateur de mémoire. Les plans d'exécution permettent d'enchaîner plusieurs scénarios sans les faire se chevaucher (chapitre 6 du guide d'utilisation et de supervision).

1.2 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.

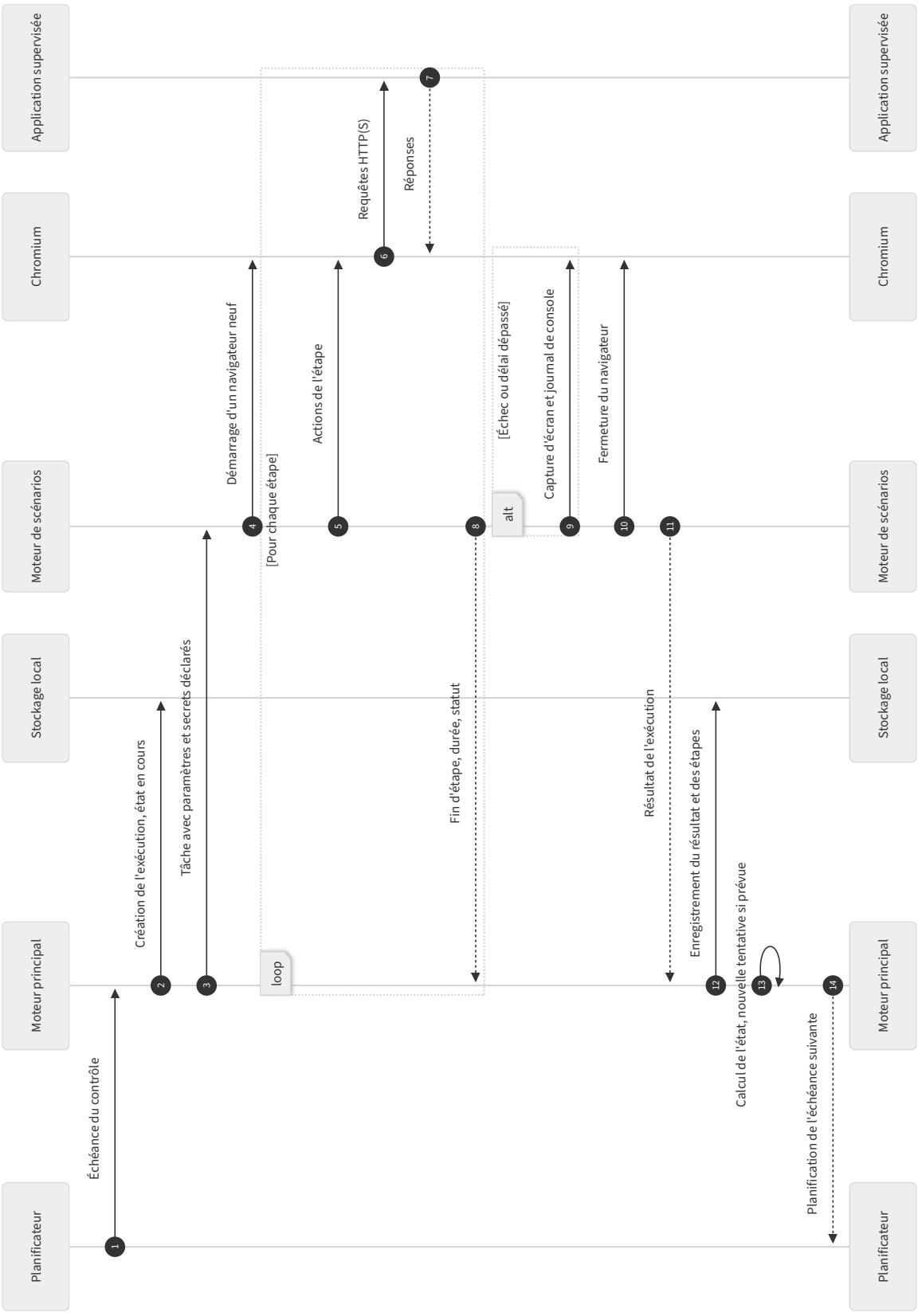


Schéma 1-2 — Cycle de vie d'une exécution (scénario navigateur)

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 du guide d'utilisation et de supervision détaille les notions d'état, de délai, de nouvelle tentative et de seuil utilisées à chaque étape.

Chapitre 2 — Premier scénario, de l'écriture à la mise en service

Ce chapitre conduit pas à pas de l'écriture d'un premier scénario à sa mise en service sur la sonde. Il enchaîne des procédures détaillées dans la suite du guide : suivez-le une première fois dans l'ordre, puis reportez-vous aux chapitres de référence pour les cas particuliers.

Le scénario retenu est l'exemple 3 (§ 5.4) : il ouvre la page de connexion de la **Boutique Démo**, une application **fictive** fournie avec les exemples, s'authentifie avec un compte de test, puis vérifie l'accès à l'espace client.

Objectif : écrire un scénario, le tester sur un poste de travail, le déployer sur la sonde, puis vérifier son fonctionnement.

Prérequis

- une sonde en service, d'édition **Enterprise**, seule édition qui exécute les scénarios navigateur ;
- un poste de travail Linux ou macOS avec Node.js 22.12 ou version ultérieure, `curl` et `openssl` ;
- le dossier d'exemples `exemples/puppeteer/` (§ 5.1.3) ;
- un jeton d'API de rôle `operator` pour un scénario déclaratif, ou `admin` pour un scénario scripté (§ 9.2) ;
- pour le déploiement, une application de test que la sonde peut joindre : sur la sonde, l'adresse `localhost` désigne la sonde elle-même, et non votre poste.

Étape 1 — Préparer le poste

1. Préparez l'accès sécurisé à l'API de la sonde : récupération et vérification du certificat, fichier d'en-tête contenant le jeton, variables `SONDE` et `CA`, fonction `sapi` utilisée par tous les exemples de ce guide. La procédure complète figure au § 9.1.3. Le jeton n'apparaît jamais dans la ligne de commande.
2. Installez les dépendances des exemples :

```bash
cd exemples/puppeteer
npm install
```

Vérification : la commande `sapi "$SONDE/api/v1/system"` renvoie les informations système de la sonde, sans erreur de certificat.

Étape 2 — Écrire le scénario

Partez du fichier `scenarios/03-authentification.mjs` pour un scénario scripté, ou `scenarios/03-authentification.json` pour sa version déclarative. Leur contenu est commenté au § 5.4. Pour votre propre application, choisissez le mode d'écriture selon le § 4.1 : le mode déclaratif suffit à un parcours simple et ne demande aucune programmation.

Un scénario scripté respecte le contrat décrit au § 4.5 : il exporte une fonction qui reçoit le contexte d'exécution, découpe le parcours en étapes nommées et vérifie chaque résultat attendu par une assertion explicite. Les identifiants ne figurent jamais dans le script : ils sont transmis comme secrets (§ 8.2).

Étape 3 — Tester sur le poste de travail

1. Dans un premier terminal, démarrez l'application de démonstration :

   ```bash
   npm run demo
   ```

2. Dans un second terminal, définissez les secrets dans des variables d'environnement, puis exécutez le scénario avec le lanceur local :

   ```bash
   export BOUTIQUE_USER='test-supervision@example.com'
   read -rs BOUTIQUE_PASSWORD && export BOUTIQUE_PASSWORD
   node lanceur-local.mjs scenarios/03-authentification.mjs \
     --param baseUrl=http://localhost:8080 \
     --secret BOUTIQUE_USER --secret BOUTIQUE_PASSWORD
   ```

   Le mot de passe du compte de test de l'application de démonstration est indiqué au § 4.15.2. La saisie est masquée : il n'apparaît ni à l'écran ni dans l'historique du shell.

3. Vérifiez le **cas d'échec** : saisissez un mot de passe erroné et contrôlez que le scénario échoue à la bonne étape, avec un message compréhensible.

Résultat attendu : le lanceur affiche un résultat JSON dont l'état est `OK`, et se termine avec le code 0. La lecture de ce résultat et ses écarts avec le format de la sonde sont décrits au § 4.15.3.

Étape 4 — Déployer sur la sonde

Le contrôle est toujours créé **désactivé**, puis validé par une exécution d'essai avant son activation.

- **Scénario déclaratif** : suivez le § 4.16.1. Le fichier JSON du contrôle est envoyé par `POST /api/v1/checks`, avec `"enabled": false` et l'adresse de votre application de test dans `params.baseUrl`.
- **Scénario scripté** : suivez le § 4.16.2. Le script est d'abord téléversé (`POST /api/v1/scripts`) ; la référence immuable renvoyée par la sonde est indiquée dans `browser.scriptRef` du contrôle.

Les secrets du contrôle sont créés au préalable par un administrateur, avec la liste des hôtes autorisés à les recevoir (§ 8.2).

Étape 5 — Vérifier et activer

1. Lancez une exécution d'essai :

   ```bash
   # Identifiant du contrôle : demo-connexion pour le scénario déclaratif de l'exemple,
   # ou l'identifiant choisi pour votre contrôle scripté (par exemple boutique-connexion)
   sapi -X POST -D - "$SONDE/api/v1/checks/demo-connexion/runs?dryRun=true"
   ```

   L'exécution d'essai réalise le scénario sans modifier l'état du contrôle et sans transmettre de télémétrie (§ 9.3.5). L'en-tête `Location` de la réponse indique l'adresse de l'exécution.

2. Consultez le résultat : état, durée de chaque étape, erreur éventuelle et capture d'écran (§ 10.1). En cas d'écart avec le test local, les causes les plus fréquentes sont listées au § 4.16.2, étape 4.

3. Si le résultat est conforme, activez le contrôle : `POST /api/v1/checks/{id}/enable`, avec l'identifiant de votre contrôle (`demo-connexion` dans l'exemple déclaratif).

Vérification : après la première échéance planifiée, le contrôle apparaît dans le tableau de bord avec l'état `OK`. Ses résultats et sa télémétrie suivent ensuite les règles des chapitre 10 et chapitre 2 du guide d'intégration et d'observabilité.

> **Attention** — L'application de démonstration n'a aucune sécurité et son compte de test est public. Ne l'exposez jamais sur un réseau de production ou ouvert. Les précautions à respecter pour la rendre joignable par une sonde de test figurent au § 5.1.3.

Chapitre 3 — Introduction à Puppeteer

Ce chapitre ouvre la partie consacrée aux scénarios navigateur. Il présente les technologies employées par la sonde pour simuler un utilisateur : Puppeteer, Chromium et le mode headless. Il explique ce qu'un navigateur automatisé apporte par rapport à une simple requête HTTP, dans quels cas l'utiliser, et quelles sont ses limites. Il s'adresse à tous les lecteurs. Les chapitres suivants détaillent ensuite la construction des scénarios (chapitre 4), des exemples progressifs (chapitre 5) et les bonnes pratiques (chapitre 6).

> **Information** — Les scénarios navigateur (contrôles de type `browser`) sont disponibles dans l'édition **Enterprise** (§ 3.8).

3.1 Présentation de Puppeteer

Puppeteer est une bibliothèque JavaScript, publiée en logiciel libre par l'équipe Chrome de Google, qui pilote un navigateur Chrome ou Chromium par programme. Elle permet d'ouvrir une page, de cliquer sur un bouton, de remplir un formulaire, d'attendre l'apparition d'un élément, de lire le contenu affiché ou de capturer l'écran, exactement comme le ferait un utilisateur, mais de façon automatique et reproductible.

Puppeteer s'exécute dans **Node.js**, l'environnement d'exécution JavaScript côté serveur. Un scénario Puppeteer est donc un programme JavaScript qui décrit, étape par étape, le parcours d'un utilisateur.

L'extrait suivant donne une idée de l'écriture. Il ouvre une page, saisit un terme de recherche, valide et attend les résultats :

```javascript
// Extrait illustratif : ouverture d'une page et recherche
await page.goto('https://boutique.example.com/recherche');
await page.locator('[data-testid=search-input]').fill('câble');
await page.locator('[data-testid=search-submit]').click();
await page.waitForSelector('[data-testid=search-result]');
```

Chaque instruction est précédée du mot-clé `await`. Les actions sur un navigateur prennent du temps (chargement, réponse du serveur, affichage) : `await` indique au programme d'attendre la fin de l'action avant de passer à la suivante. Le scénario se lit ainsi de haut en bas, dans l'ordre où l'utilisateur agit.

Dans la sonde, Puppeteer est intégré au **moteur de scénarios**. Ce composant isolé lance le navigateur, exécute vos scénarios, mesure chaque étape et produit le résultat (chapitre 1). Vous n'installez rien sur la sonde : vous écrivez le scénario, et la sonde se charge de l'exécuter.

3.2 Chromium et le protocole Chrome DevTools

3.2.1 Rôle de Chromium

Chromium est le projet de navigateur libre sur lequel reposent Google Chrome, Microsoft Edge et de nombreux autres navigateurs. Il comprend le moteur de rendu (Blink), qui interprète le HTML et les feuilles de style, et le moteur JavaScript (V8), qui exécute les scripts des pages.

Utiliser Chromium pour la supervision présente un intérêt majeur : la page est traitée **par le même moteur que celui de la majorité de vos utilisateurs**. Un dysfonctionnement provoqué par un script de la page, une ressource manquante ou une incompatibilité est donc observé dans des conditions proches de la réalité.

La sonde embarque une version de Chromium **fournie par l'éditeur**, associée à la version de Puppeteer correspondante. Vous n'avez pas à l'installer ni à la mettre à jour vous-même : elle fait partie de l'image système signée de la sonde et évolue avec les mises à jour du produit, toujours avec la version de Puppeteer correspondante. Les correctifs de sécurité de Chromium sont publiés sur le canal `security`, accessible sans contrat de maintenance pendant la période de support ; l'éditeur se fixe pour objectifs, non contractuels, de les publier sous 14 jours pour une vulnérabilité critique ou activement exploitée et sous 30 jours pour une vulnérabilité élevée (§ 8.4 du guide d'installation et d'administration).

3.2.2 Le protocole Chrome DevTools

Puppeteer ne simule pas les clics par des mouvements de souris à l'écran. Il dialogue avec le navigateur au moyen du **protocole Chrome DevTools** (CDP, *Chrome DevTools Protocol*), l'interface de pilotage qu'utilisent aussi les outils de développement intégrés à Chrome. Par ce canal, Puppeteer peut :

- ouvrir des pages et suivre leurs navigations ;
- interroger le document (DOM, *Document Object Model*, représentation en arbre de la page) ;
- envoyer des événements de souris et de clavier ;
- observer les requêtes réseau, leurs réponses et leurs durées ;
- recevoir les messages de la console JavaScript et les erreurs de la page ;
- capturer l'écran.

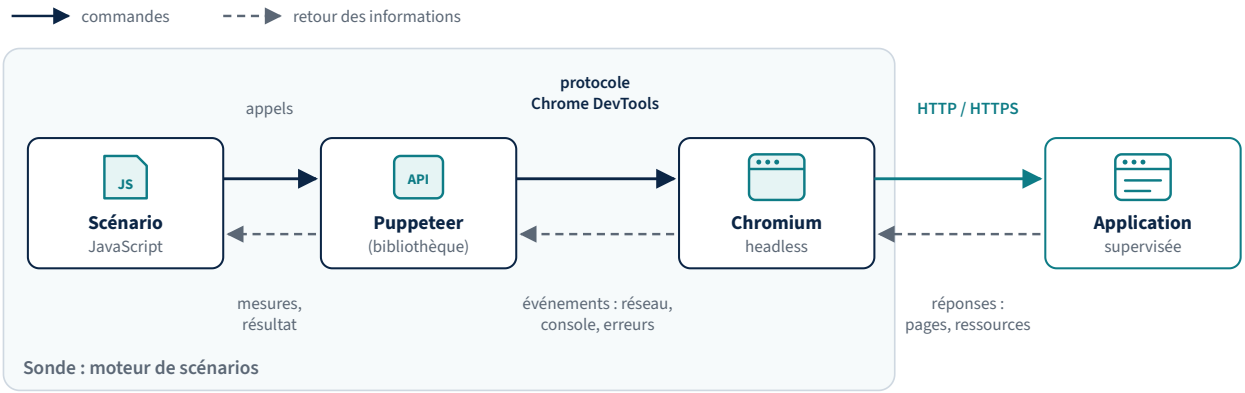


Figure 3-1 — Chaîne d'exécution d'un scénario : scénario, Puppeteer, protocole DevTools, Chromium, application supervisée

Cette architecture explique la richesse des informations disponibles en cas d'échec. La sonde ne sait pas seulement qu'une page n'a pas abouti : elle sait quelle étape a échoué, quelle requête a répondu en erreur et quel message la page a écrit dans sa console.

3.3 Le navigateur headless

3.3.1 Principe

Un navigateur **headless** (« sans tête ») fonctionne normalement, mais sans afficher de fenêtre. Il télécharge les pages, exécute leurs scripts, calcule leur mise en page et produit le même document qu'un navigateur classique ; simplement, rien n'est affiché à l'écran.

Ce mode convient à la sonde, qui ne possède ni écran ni interface graphique. Il consomme aussi moins de ressources qu'un navigateur affiché, puisqu'aucun environnement graphique n'est nécessaire.

Une page rendue en mode headless peut néanmoins faire l'objet d'une capture d'écran. C'est ainsi que la sonde illustre l'étape en échec d'un scénario (chapitre 10).

3.3.2 Mode headless et « headless shell »

Chromium propose deux variantes du fonctionnement headless :

Variante	Description	Usage
Mode headless du navigateur	Le navigateur complet, sans fenêtre. Comportement identique au navigateur affiché	Fidélité maximale au comportement des utilisateurs
Headless shell	Binaire allégé, spécialisé dans l'exécution sans fenêtre	Consommation réduite ; quelques différences de comportement possibles avec le navigateur complet

La sonde utilise le Chromium des dépôts Raspberry Pi OS, dans le **mode headless du navigateur complet** (« new headless »). Ce choix découle de la plateforme : lors de la rédaction de cette documentation, la documentation de Puppeteer sur la configuration système requise (https://pptr.dev/guides/system-requirements) ne mentionne pas *Chrome for Testing* ni le *headless shell* (`chrome-headless-shell`) pour Linux ARM64 ; la documentation de la version en développement (branche `next`) annonce *Chrome for Testing* pour certains systèmes Linux ARM64, la disponibilité réelle dépendant des artefacts publiés pour la version de Puppeteer et la distribution retenues, et Google distribue désormais une version stable de Chrome pour Linux ARM64 (https://blog.google/chromium/bringing-chrome-to-arm64-linux-devices/). Le choix du Chromium des dépôts Raspberry Pi OS est maintenu ; il est réexaminé pour chaque version épinglée de Puppeteer. Il n'a pas d'incidence sur l'écriture des scénarios.

3.3.3 Profil vierge à chaque exécution

À chaque exécution d'un scénario, le moteur de scénarios démarre un navigateur neuf avec un **profil vierge** : aucun cookie, aucune session, aucun cache, aucun historique. Le profil est conservé en mémoire, puis détruit à la fin de l'exécution.

Ce comportement a trois conséquences :

- chaque exécution reproduit la **première visite** d'un utilisateur. Un scénario qui a besoin d'être authentifié doit donc s'authentifier lui-même (§ 4.9) ;
- une exécution ne peut pas être perturbée par l'état laissé par la précédente (panier rempli, session expirée) ;
- aucune donnée de navigation, et donc aucun cookie de session d'un compte de test, ne subsiste sur la carte microSD.

3.4 Navigation automatisée et simple requête HTTP

La sonde propose aussi des contrôles `http` (chapitre 4 du guide d'utilisation et de supervision). Ceux-ci envoient une requête et analysent la réponse. Il est important de comprendre ce qui distingue les deux approches, pour choisir la bonne.

3.4.1 Ce que fait une requête HTTP

Un contrôle `http` envoie **une** requête vers **une** adresse. Il vérifie le code de réponse, les en-têtes, éventuellement le contenu du corps, et mesure les temps de résolution DNS, de connexion, de négociation TLS et de réponse. Il ne charge pas les ressources de la page (images, feuilles de style, scripts) et n'exécute aucun JavaScript.

3.4.2 Ce que fait un navigateur

Un navigateur charge le document, puis **toutes** les ressources qu'il référence : feuilles de style, scripts, images, polices, appels d'API effectués par la page, contenus de services tiers. Il exécute les scripts, qui peuvent eux-mêmes modifier la page, déclencher d'autres requêtes ou rediriger l'utilisateur. Il gère les cookies, les redirections d'authentification et les formulaires.

3.4.3 Comparaison

Aspect	Contrôle `http`	Scénario navigateur
Nombre de requêtes	Une	Toutes celles de la page (souvent plusieurs dizaines)
JavaScript de la page	Non exécuté	Exécuté
Contenu généré dynamiquement	Invisible si produit par JavaScript	Visible
Interactions (clic, saisie)	Impossibles	Possibles
Parcours en plusieurs pages	Limité (contrôle `api` pour les API)	Natif
Services tiers (paiement, cartographie, authentification externe)	Non vérifiés	Vérifiés
Temps mesuré	Temps de réponse du serveur	Temps perçu par l'utilisateur, par étape
Consommation sur la sonde	Très faible	Élevée (un navigateur complet)
Fréquence raisonnable	Toutes les minutes	Toutes les 5 à 15 minutes
Sensibilité aux évolutions de l'interface	Faible	Réelle : les sélecteurs doivent être entretenus

3.4.4 Exemple : une page qui répond, mais qui ne fonctionne pas

Considérons une application dont la page de recherche est servie avec un code HTTP 200. Le formulaire de recherche appelle une API depuis le navigateur pour afficher les résultats. Si cette API est en panne, ou si le script de la page contient une erreur après une mise en production :

- le contrôle `http` sur la page de recherche reste `OK` : la page elle-même est bien servie ;
- le scénario navigateur échoue à l'étape « Vérifier les résultats », car aucun résultat n'apparaît, et il précise si l'API a répondu en erreur.

Les deux contrôles sont complémentaires. Le contrôle `http`, léger, détecte rapidement une indisponibilité franche. Le scénario, plus coûteux, détecte les défaillances du service rendu.

> ✓ **Conseil** — Pour une application critique, associez un contrôle `http` fréquent (toutes les minutes) sur la page d'accueil et un scénario navigateur moins fréquent (toutes les 5 à 10 minutes) sur le parcours principal. Vous obtenez une détection rapide des pannes franches et une vérification approfondie du service.

3.5 Cas d'utilisation en supervision

Les scénarios navigateur répondent aux situations dans lesquelles « le serveur répond » ne suffit pas à affirmer que « le service fonctionne » (chapitre 1 du guide d'utilisation et de supervision).

Cas d'usage	Ce que vérifie le scénario	Exemple du chapitre 5
Disponibilité perçue d'une page	La page se charge et affiche son contenu utile, pas seulement un code 200	Exemples 1 et 2
Authentification	Le compte de test peut se connecter et accède à son espace	Exemple 3
Refus d'authentification	Des identifiants invalides sont refusés avec le message attendu	Exemple 3 bis
Navigation	Les pages principales s'enchaînent, y compris les contenus intégrés (iframes)	Exemple 4
Recherche	La recherche renvoie des résultats pertinents dans un délai acceptable	Exemple 5
Parcours transactionnel	Un produit peut être ajouté au panier	Exemple 6
Tunnel de commande	Le tunnel est franchissable jusqu'à la confirmation, en environnement de test	Exemple 7
Cohérence fonctionnelle	Le résultat affiché est correct (quantité, total), au-delà du message de succès	Exemple 8
Performance perçue	Le temps de chaque étape reste dans les seuils définis	Tous (seuils de durée, chapitre 3 du guide d'utilisation et de supervision)
Services tiers	Le module de paiement, la cartographie ou le fournisseur d'identité répondent	Variantes des exemples 3 et 7

Le scénario mesure la durée de chaque étape. Il permet donc de répondre à trois questions : le parcours a-t-il échoué ? À quelle étape ? Pourquoi ? Les résultats sont détaillés au chapitre 10 et peuvent être transmis aux plateformes d'observabilité (chapitre 2 du guide d'intégration et d'observabilité).

3.6 Avantages et limites

3.6.1 Avantages

- **Fidélité.** Le scénario exerce l'application comme un utilisateur, avec le même moteur de navigateur que la plupart d'entre eux.
- **Détection des défaillances fonctionnelles.** Bouton inopérant, script en erreur, API appelée par la page en panne, service tiers défaillant : autant de pannes invisibles pour les contrôles techniques.
- **Diagnostic précis.** Étape en échec, type d'erreur, message, capture d'écran, erreurs de la console et requêtes en erreur.
- **Mesure du temps perçu.** Durée totale et durée par étape.
- **Proactivité.** Le scénario s'exécute en permanence, y compris la nuit et le week-end, lorsque peu d'utilisateurs sont connectés pour signaler un problème.

3.6.2 Limites

Limite	Explication	Conduite à tenir
Consommation de ressources	Un navigateur complet consomme beaucoup de mémoire et de processeur. Sur un Raspberry Pi 5, la sonde n'exécute qu'**un scénario à la fois** par défaut	Espacer les exécutions (5 minutes ou plus) ; regrouper les parcours dans des plans d'exécution (chapitre 6 du guide d'utilisation et de supervision) ; bloquer les ressources tierces inutiles (chapitre 6)
Authentification forte (MFA)	Un second facteur (code SMS, application d'authentification, clé physique) ne peut pas être fourni par un scénario	Prévoir avec les équipes sécurité un compte de test exempté de second facteur, limité en droits et en adresses d'origine
CAPTCHA et protections anti-robots	Ces dispositifs sont conçus pour bloquer les navigateurs automatisés	Demander une exemption pour l'adresse IP de la sonde ou pour le compte de test ; ne jamais chercher à contourner ces protections
Fragilité face aux évolutions de l'interface	Un sélecteur qui ne correspond plus à la page fait échouer le scénario, même si l'application fonctionne	Sélecteurs robustes (`data-testid`), coordination avec les équipes de développement (chapitre 6)
Effets de bord	Un scénario agit réellement sur l'application : commandes, messages, comptes verrouillés	Environnement de test, données de test, garde-fous et nettoyage (exemple 7, chapitre 5)
Volume de trafic	Chaque exécution charge toutes les ressources de la page et peut fausser les statistiques d'audience	Fréquence raisonnable ; identification du trafic de la sonde (§ 6.12)
Représentativité	La sonde mesure le service depuis son propre emplacement réseau, avec son propre matériel	Interpréter les temps comme une tendance ; comparer plusieurs sondes si nécessaire

> ! **Attention** — Un scénario navigateur n'est pas un test de charge. Il vérifie qu'un parcours fonctionne et mesure son temps de réponse. Il ne doit pas être planifié à haute fréquence pour « solliciter » une application.

3.7 De CasperJS à Puppeteer

Cette section s'adresse aux clients qui utilisaient la génération précédente de la solution, dont les scénarios web reposaient sur **CasperJS** et **PhantomJS**. Ces deux outils ne sont plus maintenus depuis plusieurs années, et PhantomJS repose sur un moteur WebKit figé, incapable d'exécuter correctement les applications web modernes. La nouvelle sonde n'exécute donc pas les scénarios CasperJS. Ceux-ci doivent être **réécrits**.

3.7.1 Ce qui ne change pas

Les **intentions** de vos scénarios historiques restent valables, de même que les principes de rédaction que vous appliquiez :

- simuler fidèlement l'utilisateur, par exemple en cliquant dans un champ avant d'y saisir du texte lorsque l'application réagit à ce clic ;
- attendre l'apparition d'un élément plutôt qu'imposer une pause fixe ;
- vérifier un résultat explicite (message, élément, texte) à chaque étape importante ;
- maintenir les scénarios à jour à chaque évolution de l'application.

Avant de réécrire un scénario, formalisez son intention : ce qu'il vérifie, le résultat attendu, le compte de test utilisé. Cette fiche sert ensuite à vérifier que le nouveau scénario détecte les mêmes anomalies.

3.7.2 Ce qui change

CasperJS empilait des étapes dans une file interne (`then`, `waitFor…`) et utilisait des fonctions de rappel de succès et d'échec. Puppeteer s'écrit de façon linéaire, avec `async` et `await` : chaque instruction attend la fin de la précédente. Les échecs sont des **exceptions**, que le moteur de scénarios intercepte pour marquer l'étape en échec. La gestion des erreurs devient ainsi beaucoup plus simple.

La sonde apporte en outre des changements d'organisation :

- le navigateur est lancé et fermé par le moteur de scénarios, pas par le script ;
- les captures d'écran sont réalisées automatiquement en cas d'échec ;
- les identifiants sont fournis par la sonde (secrets), jamais écrits dans le script ;
- les parcours simples peuvent être décrits sans code, en mode déclaratif.

3.7.3 Correspondance des concepts

Pratique CasperJS / PhantomJS	Équivalent avec la sonde et Puppeteer	Remarque
`casper.start(url)`, `casper.thenOpen(url)`	`await page.goto(url, { waitUntil: 'domcontentloaded' })`	Voir § 4.6
`casper.waitForSelector(sélecteur, succès, échec)`	`await page.waitForSelector(sélecteur, { visible: true })`, ou action sur un *locator*, qui attend automatiquement	Plus de fonctions de rappel : un échec lève une exception
`test.assertExists(sélecteur)`	`await expect.visible(page, sélecteur)`	Assertion fournie par le moteur de scénarios (§ 4.11)
`this.click(sélecteur)`	`await page.locator(sélecteur).click()`	Le *locator* attend que l'élément soit visible et cliquable
`this.sendKeys(sélecteur, texte)`	`await page.locator(sélecteur).fill(texte)`	Le champ est vidé puis rempli
`casper.withFrame(id, fonction)`	Obtenir l'objet `Frame` de l'iframe, puis agir sur ce cadre	Mode déclaratif : propriété `frame` de l'étape (§ 4.13)
`thenEvaluate` + `jQuery.Event('keyup')`	`await page.keyboard.press('Enter')`	Événements clavier natifs : la page n'a plus besoin d'embarquer jQuery
`this.capture(...)` dans le script	Supprimé	Capture automatique par le moteur en cas d'échec
`casper.wait(durée)`	À proscrire	Attendre un élément, une réponse ou une condition (§ 4.10)
Sélecteurs XPath absolus (`/html/body/table[1]/…`)	Attributs dédiés (`data-testid`), rôle d'accessibilité, texte ; XPath en dernier recours	Voir § 4.7 et chapitre 6
Extension Resurrectio (enregistrement d'actions)	Enregistreur des outils de développement de Chrome	Aide à la conception (§ 4.4)
Outil `scenarioRunner` sur poste de travail	Lanceur local fourni avec les exemples, puis exécution d'essai sur la sonde	§ 4.15 et § 4.16

Pratique CasperJS / PhantomJS	Équivalent avec la sonde et Puppeteer	Remarque
Identifiants écrits dans le script	Secrets déclarés par le contrôle	Chapitre 9 et § 4.9

3.7.4 Exemple de réécriture

Le guide historique décrivait un scénario d'authentification qui saisissait volontairement des identifiants invalides, puis vérifiait l'affichage du message « Erreur d'authentification ». Le code CasperJS ressemblait à ceci (extrait reconstitué, sélecteurs simplifiés) :

JAVASCRIPT

```
// Ancien style CasperJS — NE FONCTIONNE PAS sur la nouvelle sonde
casper.waitForSelector("input[name='username']",
  function success() {
    this.sendKeys("input[name='username']", "…");
  },
  function fail() {
    test.assertExists("input[name='username']");
  });
```

Le même parcours, réécrit pour la sonde, devient (version complète : exemple 3 bis du chapitre 5) :

JAVASCRIPT

```
// Nouveau style : linéaire, attentes intégrées, exceptions en cas d'échec
await step('Soumettre des identifiants invalides', async () => {
  await page.locator('[data-testid=login-email]').fill('compte-inexistant@example.com');
  await page.locator('[data-testid=login-password]').fill('mot-de-passe-volontairement-faux');
  await Promise.all([
    page.waitForNavigation({ waitUntil: 'domcontentloaded' }),
    page.locator('[data-testid=login-submit]').click(),
  ]);
});

await step("Vérifier le message d'erreur", async () => {
  await expect.text(page, '[data-testid=login-error]', "Erreur d'authentification");
});
```

La nouvelle version est plus courte, se lit dans l'ordre du parcours, et ses étapes nommées apparaissent dans les résultats de la sonde.

> **Information** — Il n'existe pas de convertisseur automatique fiable de CasperJS vers Puppeteer : les modèles d'exécution sont trop différents. La réécriture, guidée par la fiche d'intention de chaque scénario, est la démarche recommandée. Pendant la transition, faites fonctionner l'ancien et le nouveau scénario en parallèle et comparez leurs résultats avant de retirer l'ancien.

3.8 Prérequis techniques dans la sonde

3.8.1 Édition et matériel

Prérequis	Valeur
Édition	**Enterprise** (les scénarios déclaratifs et scriptés font partie de ce module)
Matériel	Raspberry Pi 5, **8 Go de mémoire vive minimum** (chapitre 2 du guide d'installation et d'administration)
Refroidissement	Refroidisseur actif ou boîtier ventilé obligatoire : l'exécution de Chromium sollicite fortement le processeur

Prérequis	Valeur
Stockage	Carte microSD fournie, suffisante pour un usage standard ; un SSD NVMe facultatif est utile pour des parcours fréquents, comme volume de données (§ 8.3.4 du guide d'installation et d'administration)

3.8.2 Composants logiciels embarqués

Composant	Version	Remarque
Node.js	Version LTS, indiquée dans les notes de version et par `GET /api/v1/system` (champ `versions`)	Exécute les scénarios scriptés
Puppeteer (`puppeteer-core`)	Indiquée dans les notes de version et par `GET /api/v1/system` (champ `versions`)	Version publiée par la sonde dans `GET /api/v1/system`
Chromium	Indiquée dans les notes de version et par `GET /api/v1/system` (champ `versions`)	Issu des dépôts Raspberry Pi OS, retenu par l'éditeur et intégré à l'image système signée ; mis à jour avec le produit (§ 3.3.2, chapitre 8 du guide d'installation et d'administration)
Modules JavaScript disponibles pour les scripts	Aucun	Un script ne peut ni importer ni installer de module : seuls les objets du contrat d'exécution sont disponibles (§ 4.5.1)

`puppeteer-core` est la variante de Puppeteer qui ne télécharge pas de navigateur et utilise celui qui lui est fourni. Elle est nécessaire sur le Raspberry Pi 5, dont l'architecture ARM 64 bits n'est pas couverte par les navigateurs que Puppeteer télécharge habituellement : lors de la rédaction de cette documentation, la documentation de Puppeteer ne liste pas *Chrome for Testing* pour Linux ARM64 (§ 3.3.2). La disponibilité effective des artefacts peut varier selon le canal et la version ; elle est vérifiée pour chaque version épinglée , et le choix du Chromium des dépôts Raspberry Pi OS reste la référence tant que cette vérification n'a pas conclu autrement. Cette particularité est transparente pour l'auteur de scénarios : l'API utilisée dans les scripts est la même.

> **Information** — Les exemples de cette documentation ont été écrits et testés sur poste de travail avec **Puppeteer 25.18.1** (Chrome for Testing 157), au moyen du lanceur local fourni avec les exemples. Ils n'utilisent que des fonctions stables de l'API Puppeteer : *locators*, `waitForSelector`, `waitForResponse`, `waitForFunction`, `waitForNavigation`, `keyboard`, et les sélecteurs `::-p-text()`, `::-p-aria()` et `::-p-xpath()`. Avant de déployer un scénario, vérifiez la version de Puppeteer de votre sonde dans `GET /api/v1/system` et effectuez une exécution d'essai (§ 4.16).

3.8.3 Capacité d'exécution

Le moteur de scénarios exécute par défaut **un seul scénario à la fois**. Sur une carte de 8 Go, cette limite peut être portée à deux après vérification de la consommation réelle de vos scénarios. Les scénarios dont l'échéance arrive pendant une exécution attendent leur tour. Le retard est mesuré et visible dans la supervision interne de la sonde (état de santé au chapitre 10, métriques de santé au chapitre 2 du guide d'intégration et d'observabilité).

Pour estimer la capacité nécessaire, multipliez la durée moyenne de chaque scénario par sa fréquence d'exécution horaire. Le total doit rester nettement inférieur à 60 minutes par heure et par emplacement d'exécution :

Scénario	Durée moyenne	Fréquence	Occupation par heure
Connexion	8 s	toutes les 5 min (12/h)	96 s
Recherche	6 s	toutes les 5 min (12/h)	72 s

Scénario	Durée moyenne	Fréquence	Occupation par heure
Tunnel de commande (test)	25 s	toutes les 15 min (4/h)	100 s
Total			**268 s, soit environ 4,5 min/h**

Dans cet exemple fictif, l'occupation reste faible et laisse une large marge pour les nouvelles tentatives et les exécutions manuelles. Le chapitre 6 du guide d'utilisation et de supervision détaille la planification et le regroupement des scénarios en plans d'exécution.

3.8.4 Prérequis réseau

Le navigateur de la sonde doit pouvoir joindre **toutes** les ressources des pages parcourues, et pas seulement l'adresse principale de l'application : réseaux de diffusion de contenu, services d'authentification, modules de paiement de test, API appelées par la page. Recensez ces destinations avec les équipes de l'application et autorisez-les dans le pare-feu et, le cas échéant, dans le proxy (chapitre 5 du guide d'installation et d'administration).

Une ressource bloquée par le pare-feu peut faire échouer un scénario, alors que l'application fonctionne pour les utilisateurs. À l'inverse, les ressources inutiles à la vérification (mesure d'audience, publicité) peuvent être volontairement bloquées par le scénario (§ 6.11).

3.8.5 Poste de développement

Pour écrire et mettre au point les scénarios, un poste de travail équipé de Node.js et de Puppeteer est recommandé. Le dossier d'exemples `exemples/puppeteer/`, livré dans l'archive `synovex-exemples.zip` avec la documentation, contient une application de démonstration fictive et un **lanceur local** qui reproduit le contrat d'exécution de la sonde. Leur utilisation est décrite au § 4.15.

Chapitre 4 — Construction d'un scénario

Ce chapitre explique comment construire un scénario navigateur, de sa conception à son déploiement sur la sonde. Il présente les deux modes d'écriture proposés par la sonde, le mode déclaratif et le mode scripté. Il détaille ensuite chaque aspect d'un scénario scripté : ouverture des pages, recherche d'éléments, interactions, authentification, attentes, assertions et traitement des erreurs. Il se termine par la mise au point sur poste de travail, le déploiement et le cycle de vie des scénarios.

Les exemples s'appuient sur la **Boutique Démo**, application web **fictive** fournie avec les exemples de la documentation (dossier `exemples/puppeteer/`). Les adresses `boutique.example.com` et le compte `test-supervision@example.com` sont également fictifs.

4.1 Deux modes d'écriture

Un scénario est un contrôle de type `browser` (chapitre 3 du guide d'utilisation et de supervision). Son contenu peut être décrit de deux manières.

	Scénario déclaratif	Scénario scripté
Forme	Liste d'étapes JSON (`browser.mode = "steps"`)	Module JavaScript Puppeteer (`browser.mode = "script"`)
Compétences requises	Aucune programmation ; connaissance des sélecteurs	JavaScript et API Puppeteer
Rôle nécessaire pour le créer ou en changer le code	`operator`	`admin` (le script est du code exécuté sur le réseau où la sonde est installée). Un `operator` ne peut ni définir ou retirer `browser.mode: script`, ni modifier `browser.scriptRef` : la sonde répond `403`. Il peut en revanche activer, désactiver, déclencher, ajuster par `PATCH` la planification ou les seuils, et supprimer le contrôle
Validation	Par schéma, à l'enregistrement : une erreur de structure est refusée immédiatement	À l'exécution d'essai
Logique conditionnelle, boucles, calculs	Non	Oui
Lecture de valeurs et comparaison entre étapes	Non	Oui
Attente d'une réponse d'API précise	Non	Oui
Garde-fous (vérifier l'environnement avant une action)	Limités	Complets
Maintenance	Très simple	Demande un développeur

4.1.1 Choisir le mode

Choisissez le **mode déclaratif** lorsque le parcours est une suite linéaire d'actions et de vérifications : ouvrir une page, cliquer, saisir, vérifier qu'un élément ou un texte apparaît. C'est le cas de la majorité des scénarios de disponibilité et d'authentification.

Choisissez le **mode scripté** lorsque le scénario doit :

- lire une valeur dans la page et la comparer à un résultat calculé (contrôle d'un total, d'une quantité) ;
- attendre la réponse d'une API appelée par la page ;
- adapter son comportement à ce qu'il trouve (fermer un bandeau s'il est présent) ;
- vérifier des conditions de sécurité avant une action irréversible (environnement de test) ;
- manipuler des fenêtres secondaires ou des boîtes de dialogue.

> ✓ **Conseil** — Commencez par un scénario déclaratif. Passez au mode scripté seulement lorsque le besoin l'exige. Un scénario déclaratif est plus simple à relire, à confier à une autre équipe et à maintenir.

4.2 Paramètres d'un contrôle navigateur

Quel que soit le mode, un scénario est un contrôle qui possède les paramètres communs décrits au chapitre 3 du guide d'utilisation et de supervision (planification, timeout, nouvelles tentatives, seuils, secrets, étiquettes) et un bloc `browser` spécifique.

JSON

```
{
  "id": "boutique-connexion",
  "name": "Boutique : connexion du compte de test",
  "type": "browser",
  "schedule": { "interval": "5m", "jitter": "30s" },
  "timeout": "60s",
  "retry": { "attempts": 1, "interval": "30s" },
  "thresholds": { "warningDuration": "8s" },
  "secrets": ["BOUTIQUE_USER", "BOUTIQUE_PASSWORD"],
  "browser": {
    "mode": "steps",
    "params": { "baseUrl": "https://boutique.example.com" },
    "viewport": { "width": 1366, "height": 768 },
    "captureOnFailure": true,
    "blockDomains": ["stats.example.net"],
    "userAgentSuffix": "SupervisionSonde",
    "steps": [
      { "name": "Ouvrir la page de connexion", "action": "goto", "url": "${param:baseUrl}/connexion"
    }
    ]
  }
}
```

Dans cet extrait, le tableau `steps` est abrégé : sa forme complète est présentée au § 4.3.

Propriété de `browser`	Rôle	Valeur par défaut
`mode`	`steps` (déclaratif) ou `script` (scripté)	Obligatoire
`steps`	Étapes du scénario déclaratif (50 au maximum)	Obligatoire en mode `steps`

Propriété de `browser`	Rôle	Valeur par défaut
`scriptRef`	Référence immuable du script téléversé (`nom@sha256:empreinte`)	Obligatoire en mode `script`
`params`	Paramètres non sensibles transmis au scénario (adresse de base, terme de recherche, identifiant de produit)	Aucun
`viewport`	Taille de la fenêtre simulée, en pixels	1366 × 768
`captureOnFailure`	Capture d'écran automatique en cas d'échec	`true`
`blockDomains`	Domaines dont les requêtes sont bloquées (mesure d'audience, publicité)	Aucun
`userAgentSuffix`	Suffixe ajouté à l'identification du navigateur, pour reconnaître le trafic de la sonde	Aucun

Les paramètres `params` rendent un même scénario réutilisable : l'adresse de base change entre la recette et la production, sans modification du scénario lui-même.

> ! **Avertissement** — Ne placez jamais un mot de passe, un jeton ou une donnée personnelle dans `params`. Ces valeurs sont visibles par tout utilisateur pouvant consulter le contrôle. Les informations sensibles sont des **secrets** (§ 4.9 et chapitre 9).

4.3 Scénario déclaratif

4.3.1 Structure

Un scénario déclaratif est une liste ordonnée d'étapes. Chaque étape possède un **nom**, qui apparaît dans les résultats, et une **action**. Les étapes sont exécutées dans l'ordre. La première étape en échec arrête le scénario : le contrôle passe à l'état `CRITICAL` et le résultat indique le nom de cette étape.

```json
"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 le formulaire", "action": "click", "selector": "[data-testid=login-submit]",
    "waitForNavigation": true },
  { "name": "Vérifier l'accès à l'espace client", "action": "assertVisible", "selector": "[data-
    testid=account-menu]" }
]
```

Le navigateur est ouvert avant la première étape et fermé après la dernière, par le moteur de scénarios. Vous n'avez rien à prévoir pour cela.

4.3.2 Propriétés d'une étape

Propriété	Rôle
`name`	Nom affiché dans les résultats et la télémétrie. 80 caractères au maximum. Choisissez un libellé **stable et explicite** (« Valider le formulaire », et non « Étape 4 »)
`action`	Action à réaliser (§ 4.3.3)

Propriété	Rôle
`selector`	Sélecteur de l'élément visé (§ 4.7)
`url`	Adresse à ouvrir (`goto`)
`value`	Valeur saisie, sélectionnée ou attendue
`key`	Touche du clavier (`press`)
`operator`	Mode de comparaison des assertions de texte, d'URL et de titre
`frame`	Sélecteur de l'iframe contenant l'élément (§ 4.13)
`timeout`	Délai maximal de l'étape (`5s` , `15s` …) ; à défaut, délai par défaut du moteur (10 s)
`waitForNavigation`	`true` : attendre le chargement de la nouvelle page provoqué par un clic ou une touche

4.3.3 Référence des actions

Actions de navigation et d'interaction

Action	Propriétés	Effet	Exemple
`goto`	`url`	Ouvre une adresse et attend que le document soit chargé	`{ "name": "Ouvrir l'accueil", "action": "goto", "url": "${param:baseUrl}/" }`
`click`	`selector` , `waitForNavigation`	Attend que l'élément soit visible et cliquable, puis clique	`{ "name": "Ouvrir le catalogue", "action": "click", "selector": "[data-testid=nav-catalogue]", "waitForNavigation": true }`
`fill`	`selector` , `value`	Vide le champ puis saisit la valeur	`{ "name": "Saisir le terme", "action": "fill", "selector": "[data-testid=search-input]", "value": "câble" }`
`select`	`selector` , `value`	Choisit l'option d'une liste déroulante, désignée par sa **valeur technique** (attribut `value` de l'option)	`{ "name": "Choisir le pays", "action": "select", "selector": "#pays", "value": "be" }`
`press`	`key` , `selector` (facultatif), `waitForNavigation`	Appuie sur une touche. Si un sélecteur est fourni, l'élément est d'abord cliqué pour recevoir le focus	`{ "name": "Valider par Entrée", "action": "press", "selector": "[data-testid=search-input]", "key": "Enter" }`
`hover`	`selector`	Survole un élément (menus déroulants)	`{ "name": "Déplier le menu", "action": "hover", "selector": "[data-testid=menu-compte]" }`
`waitForSelector`	`selector`	Attend qu'un élément devienne visible	`{ "name": "Attendre les résultats", "action": "waitForSelector", "selector": "[data-testid=search-result]" }`
`waitForNavigation`	—	Attend le chargement d'une nouvelle page	Voir l'attention ci-dessous

Les noms de touches de `press` sont ceux du clavier standard : `Enter` , `Tab` , `Escape` , `ArrowDown` , `Backspace` , `Digit7` (chiffre 7 de la rangée principale), `KeyA` , etc.

> **Attention** — L'action autonome `waitForNavigation` attend la **prochaine** navigation. Si le chargement de la page a déjà eu lieu au moment où l'étape commence, l'attente échoue au bout du délai. Préférez toujours la propriété `"waitForNavigation": true` sur l'étape `click` ou `press` qui déclenche le changement de page : l'attente est alors armée avant l'action.

Actions de vérification (assertions)

Action	Propriétés	Réussit si…
`assertVisible`	`selector`	L'élément est visible dans le délai imparti
`assertHidden`	`selector`	L'élément est absent ou masqué dans le délai imparti
`assertText`	`selector`, `value`, `operator`	Le texte de l'élément vérifie la comparaison
`assertUrl`	`value`, `operator`	L'adresse courante vérifie la comparaison
`assertTitle`	`value`, `operator`	Le titre de la page vérifie la comparaison

`operator`	Signification	Exemple de `value`
`contains` (par défaut)	Le texte contient la valeur	`"Bienvenue"`
`notContains`	Le texte ne contient pas la valeur	`"Erreur"`
`eq`	Le texte est exactement égal à la valeur (espaces de début et de fin ignorés)	`"1"`
`ne`	Le texte est différent de la valeur	`"0"`
`matches`	Le texte correspond à l'expression régulière	`"^[1-9][0-9]* résultat"`

4.3.4 Variables

Les valeurs `url` et `value` peuvent contenir des références, remplacées au moment de l'exécution :

Syntaxe	Source	Exemple
`${param:nom}`	Paramètre de `browser.params`	`"${param:baseUrl}/connexion"`
`${secret:NOM}`	Secret déclaré dans la liste `secrets` du contrôle	`"${secret:BOUTIQUE_PASSWORD}"`

Une référence vers un paramètre absent, ou vers un secret non déclaré dans `secrets`, est une erreur. Le contrôle est refusé à l'enregistrement, ou l'exécution échoue avec l'état `UNKNOWN`.

> **Conseil** — Utilisez `${secret:…}` uniquement dans les étapes `fill`. Une assertion qui compare un texte de la page à un secret risquerait de faire apparaître ce secret dans un message d'erreur. La sonde masque les valeurs de secrets connues dans les journaux, mais il est préférable de ne pas les exposer à ce risque.

4.3.5 Exemple complet

Le scénario suivant vérifie le moteur de recherche de la Boutique Démo. Il correspond au fichier `exemples/puppeteer/scenarios/05-recherche.json`.

```json
{
  "id": "demo-recherche",
  "name": "Boutique Démo : moteur de recherche",
  "type": "browser",
  "schedule": { "interval": "10m" },
  "timeout": "45s",
  "browser": {
    "mode": "steps",
    "params": { "baseUrl": "http://localhost:8080", "terme": "câble" },
    "steps": [
      { "name": "Ouvrir la page de recherche", "action": "goto", "url": "${param:baseUrl}/recherche"
    },
      { "name": "Saisir le terme recherché", "action": "fill", "selector": "[data-testid=search-
    input]", "value": "${param:terme}" },
      { "name": "Lancer la recherche", "action": "click", "selector": "[data-testid=search-submit]"
    },
      { "name": "Vérifier l'affichage des résultats", "action": "assertVisible", "selector": "[data-
    testid=search-result]" },
      { "name": "Vérifier le compteur de résultats", "action": "assertText", "selector": "[data-
    testid=search-count]", "operator": "matches", "value": "^[1-9][0-9]* résultat" }
    ]
  }
}
```

La recherche de cette application ne charge pas de nouvelle page : les résultats sont affichés par un script. L'étape « Lancer la recherche » n'a donc pas de `waitForNavigation`. L'étape suivante, `assertVisible`, attend l'apparition du premier résultat dans la limite de son délai.

4.3.6 Limites du mode déclaratif

Le mode déclaratif ne permet pas de :

- comparer deux valeurs lues dans la page, ou une valeur et un résultat calculé ;
- répéter une action ou choisir une action selon le contenu de la page ;
- attendre une réponse réseau précise ;
- manipuler une fenêtre secondaire ou une boîte de dialogue.

Lorsque l'un de ces besoins apparaît, convertissez le scénario en mode scripté. La conversion est directe : chaque étape déclarative devient un appel `step()` contenant l'instruction Puppeteer équivalente (§ 4.5).

4.4 Concevoir un scénario avec l'enregistreur de Chrome

Les outils de développement de Google Chrome comprennent un **enregistreur** (panneau *Recorder*) qui consigne les actions réalisées dans le navigateur : navigation, clics, saisies. L'enregistrement peut être exporté au format JSON ou sous forme de script Puppeteer. Cet outil succède, pour cet usage, à l'extension Resurrectio qu'utilisait la génération précédente.

4.4.1 Procédure

Objectif : obtenir rapidement la liste des actions et des sélecteurs d'un parcours.

Prérequis : Google Chrome sur un poste de travail ayant accès à l'application, idéalement en environnement de test.

Étapes

1. Ouvrez l'application dans Chrome, puis les outils de développement (touche F12).
2. Ouvrez le panneau **Recorder** (menu des outils de développement, *More tools*, puis *Recorder*, selon la version de Chrome).
3. Créez un nouvel enregistrement, nommez-le, puis démarrez-le.
4. Réalisez le parcours comme un utilisateur.
5. Arrêtez l'enregistrement.
6. Exportez-le au format **Puppeteer** (script JavaScript) ou **JSON**.

4.4.2 Adapter l'enregistrement

L'enregistrement est un **brouillon**, pas un scénario prêt à déployer. Il doit être retravaillé :

Point à reprendre	Raison
Supprimer le lancement et la fermeture du navigateur	Le moteur de scénarios les prend en charge
Regrouper les actions en étapes nommées (`step()` ou `name`)	Les résultats doivent indiquer l'étape métier en échec
Remplacer les sélecteurs générés par des sélecteurs stables (`data-testid`)	Les sélecteurs générés, positionnels ou dépendants du style, cassent à la moindre évolution (§ 4.7)
Supprimer les actions inutiles (clics de positionnement, défilements)	Elles allongent le scénario et le fragilisent
Remplacer les identifiants saisis par des références à des secrets	Les identifiants ne doivent jamais figurer dans un scénario
Ajouter les **assertions**	L'enregistreur consigne des actions, pas des vérifications. Un scénario sans assertion ne détecte presque rien
Supprimer les dépendances propres aux exports (bibliothèques d'exécution d'enregistrements)	Seul le contrat de la sonde est disponible dans le moteur

→ **Évolution prévue** — L'import direct d'un enregistrement JSON de l'enregistreur de Chrome sous forme de scénario déclaratif est prévu dans une version ultérieure, avec les importateurs du format historique et des projets SoapUI. Aucun importateur n'est proposé : l'enregistrement sert de base à une écriture manuelle.

4.5 Scénario scripté : structure et contrat

4.5.1 Le contrat du module

Un scénario scripté est un **module JavaScript** (extension `.mjs`). Il exporte par défaut une fonction asynchrone et ne peut importer aucun autre module (`import`). Le moteur de scénarios appelle cette fonction en lui transmettant un **objet de contexte**, qui contient tout ce dont le script dispose :

```javascript
export default async function scenario({ page, step, secrets, params, log, expect }) {
  // Le parcours est écrit ici, sous forme d'étapes.
}
```

Élément	Description
`page`	Objet `Page` de Puppeteer. Le navigateur est déjà lancé, avec un profil vierge. Le script **ne lance ni ne ferme** le navigateur
`step(nom, fonction)`	Déclare une étape mesurée. Le nom doit être **littéral et stable** (80 caractères au maximum, 50 étapes au maximum). Renvoie la valeur renvoyée par la fonction
`secrets`	Objet en lecture seule contenant **uniquement** les secrets déclarés par le contrôle
`params`	Paramètres non sensibles du contrôle (`browser.params`)
`log.info(message)`, `log.warn(message)`	Ajoute un message au journal de l'exécution. Les valeurs des secrets sont masquées
`expect`	Fonctions d'assertion (§ 4.11)

4.5.2 Structure type

JAVASCRIPT

```
// Scénario : connexion du compte de test à la boutique (application fictive).
// Paramètres : baseUrl. Secrets : BOUTIQUE_USER, BOUTIQUE_PASSWORD.

export default async function scenario({ page, step, secrets, params, expect }) {
  await step('Ouvrir la page de connexion', async () => {
    await page.goto(`${params.baseUrl}/connexion`, { waitUntil: 'domcontentloaded' });
    await expect.visible(page, '[data-testid=login-form]');
  });

  await step('Saisir les identifiants', async () => {
    await page.locator('[data-testid=login-email]').fill(secrets.BOUTIQUE_USER);
    await page.locator('[data-testid=login-password]').fill(secrets.BOUTIQUE_PASSWORD);
  });

  await step('Valider le formulaire', async () => {
    await Promise.all([
      page.waitForNavigation({ waitUntil: 'domcontentloaded' }),
      page.locator('[data-testid=login-submit]').click(),
    ]);
  });

  await step("Vérifier l'accès à l'espace client", async () => {
    await expect.visible(page, '[data-testid=account-menu]');
    expect.url(page, '/compte');
  });
}
```

Un scénario bien construit présente les caractéristiques suivantes :

- un commentaire d'en-tête indique l'application, les paramètres et les secrets attendus ;
- chaque étape correspond à une **intention métier** compréhensible par un exploitant ;
- chaque étape importante se termine par une **vérification** ;
- aucune valeur sensible et aucune adresse propre à un environnement ne figure dans le code.

4.5.3 Déroulement d'une exécution

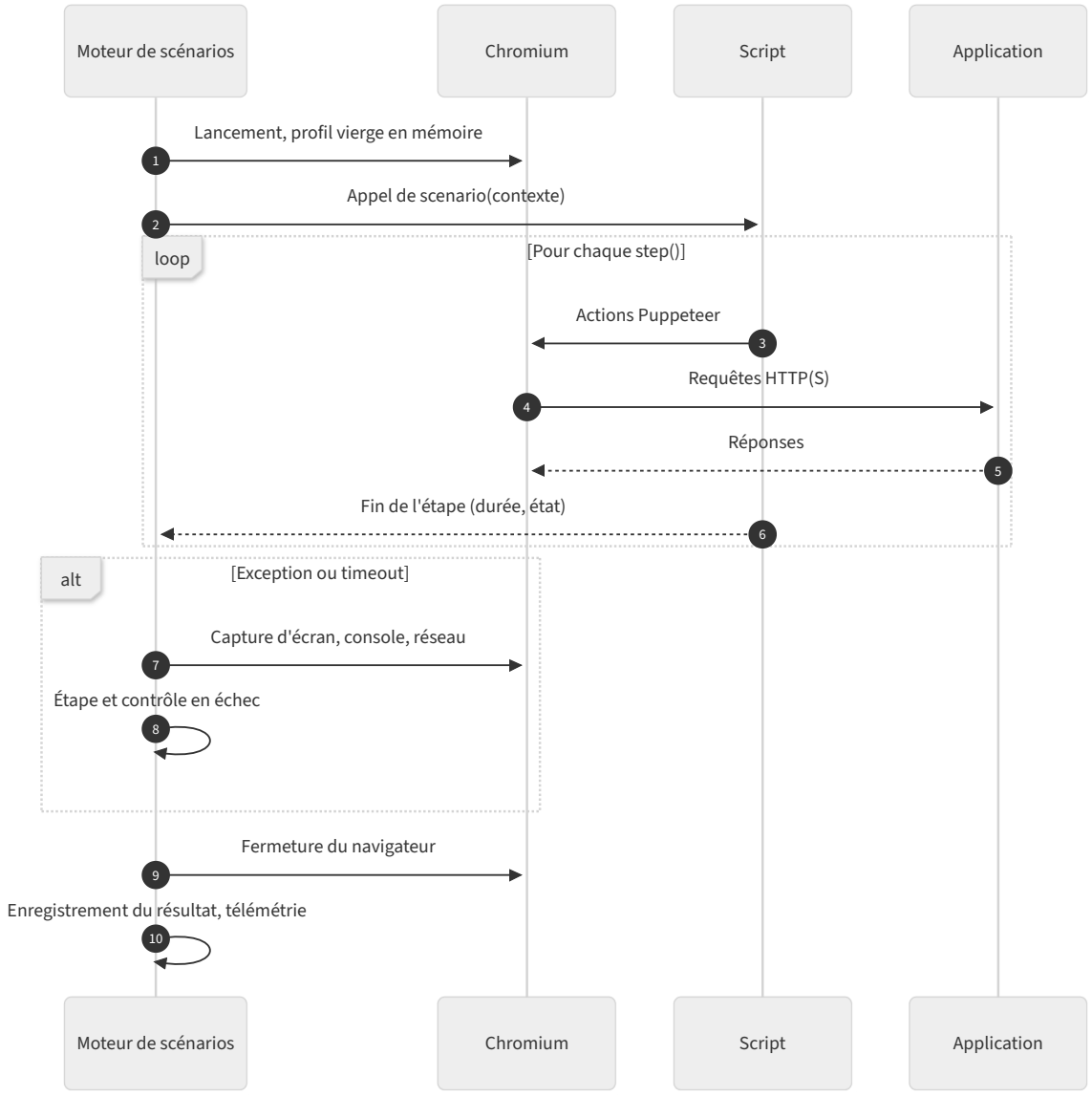


Schéma 4-1 — Déroulement de l'exécution d'un scénario scripté

4.5.4 Le code en dehors des étapes

Le code placé hors de tout `step()` s'exécute normalement, mais n'est pas mesuré. Réservez-le aux préparations sans action sur la page : lecture des paramètres, déclaration de fonctions utilitaires, abonnement à des événements.

```javascript
export default async function scenario({ page, step, params }) {
  // Préparation hors étape : valeurs par défaut des paramètres
  const produit = params.produit || 'P100';

  // Fonction utilitaire réutilisée par plusieurs étapes
  const cliquerEtAttendre = (selecteur) =>
    Promise.all([
      page.waitForNavigation({ waitUntil: 'domcontentloaded' }),
      page.locator(selecteur).click(),
    ]);

  await step('Ouvrir le catalogue', async () => {
    await page.goto(`${params.baseUrl}/`, { waitUntil: 'domcontentloaded' });
    await cliquerEtAttendre('[data-testid=nav-catalogue]');
  });
  // …
}
```

> **Attention** — Le nom d'une étape doit rester **identique d'une exécution à l'autre**. N'y insérez pas de valeur variable (date, quantité, identifiant). Les noms d'étapes servent à agréger les mesures dans le temps et dans les plateformes d'observabilité. Un nom variable créerait une nouvelle série de mesures à chaque exécution.

4.6 Ouvrir une page et naviguer

4.6.1 Ouvrir une adresse

```javascript
const reponse = await page.goto(`${params.baseUrl}/`, { waitUntil: 'domcontentloaded' });
expect.ok(reponse && reponse.ok(), `Code HTTP inattendu : ${reponse?.status()}`);
```

`page.goto()` renvoie la réponse HTTP du document principal, ce qui permet de vérifier son code. L'option `waitUntil` détermine le moment où la navigation est considérée comme terminée :

Valeur	Terminé lorsque…	Usage recommandé
`domcontentloaded`	Le document HTML est analysé, avant le chargement des images	**Valeur recommandée**, à compléter par l'attente de l'élément utile
`load`	La page et toutes ses ressources (images, feuilles de style) sont chargées	Mesure du chargement complet ; sensible aux ressources tierces lentes
`networkidle2`	Au plus deux connexions réseau actives pendant 500 ms	Pages à chargement progressif ; peut ne jamais aboutir sur une page qui interroge régulièrement le serveur
`networkidle0`	Aucune connexion réseau active pendant 500 ms	Déconseillé en supervision : les pages modernes maintiennent souvent des connexions

La pratique la plus robuste consiste à attendre `domcontentloaded`, puis l'élément qui prouve que la page est **utilisable** (formulaire, liste de résultats, menu). On mesure ainsi le temps utile pour l'utilisateur, sans dépendre des ressources secondaires.

4.6.2 Naviguer par un clic

Lorsqu'un clic provoque le chargement d'une nouvelle page, l'attente de la navigation doit être **armée avant le clic**. Sinon, une page très rapide peut être chargée avant que l'attente ne commence, et celle-ci échoue alors au bout du délai.

```javascript
// Bonne pratique : attente et clic lancés ensemble
await Promise.all([
  page.waitForNavigation({ waitUntil: 'domcontentloaded' }),
  page.locator('[data-testid=nav-catalogue]').click(),
]);
```

`Promise.all` lance les deux opérations simultanément et attend qu'elles soient toutes deux terminées.

4.6.3 Pages à navigation interne

Dans une application monopage, construite avec un framework JavaScript, un clic modifie souvent le contenu et l'adresse **sans charger de nouvelle page**. `waitForNavigation` ne convient pas dans ce cas : attendez l'élément caractéristique de la nouvelle vue.

```javascript
await page.locator('[data-testid=nav-commandes]').click();
await page.waitForSelector('[data-testid=orders-table]', { visible: true });
```

4.6.4 Autres opérations de navigation

Opération	Instruction
Revenir à la page précédente	`await page.goBack({ waitUntil: 'domcontentloaded' })`
Recharger la page	`await page.reload({ waitUntil: 'domcontentloaded' })`
Lire l'adresse courante	`page.url()`
Lire le titre	`await page.title()`

4.7 Rechercher des éléments

4.7.1 Les sélecteurs

Un **sélecteur** désigne un ou plusieurs éléments de la page. Puppeteer accepte les sélecteurs CSS standard, ainsi que des sélecteurs étendus propres à Puppeteer, préfixés par `::-p-`.

Type	Exemple	Robustesse	Commentaire
Attribut dédié aux tests	`[data-testid=login-submit]`	**Très bonne**	Recommandé : attribut posé volontairement par les développeurs, indépendant du style et des textes
Identifiant	`#login-submit`	Bonne	Si l'identifiant est stable (pas généré automatiquement)
Rôle et nom d'accessibilité	`::-p-aria(Valider la commande)`	Bonne	Désigne l'élément par son nom accessible (libellé, `aria-label`)
Texte visible	`::-p-text(Ajouter au panier)`	Moyenne	Lisible ; dépend des libellés et de la langue

Type	Exemple	Robustesse	Commentaire
Attribut fonctionnel	`input[name=email]`, `form[action="/connexion"]`	Moyenne	Correct si l'attribut a un sens fonctionnel
Classe CSS	`.btn-primary`	Faible	Les classes servent au style et changent souvent
Position	`div > ul > li:nth-child(3) > a`	**Très faible**	Casse au moindre changement de mise en page
XPath	`::-p-xpath(//button[contains(., "Valider")])`	Variable	En dernier recours ; jamais de chemin absolu du type `/html/body/div[2]/…`

> ✓ **Conseil** — Demandez aux équipes de développement d'ajouter des attributs `data-testid` aux éléments utilisés par vos scénarios (§ 6.16). C'est la mesure qui réduit le plus les faux positifs liés aux évolutions de l'interface.

Les sélecteurs se testent directement dans la console des outils de développement de Chrome. `document.querySelectorAll('[data-testid=search-result]')` renvoie les éléments correspondants. Vérifiez qu'un sélecteur désigne **un seul** élément lorsque vous voulez agir dessus.

4.7.2 Locators et handles

Puppeteer propose deux façons d'accéder aux éléments.

Les *locators* (`page.locator(sélecteur)`) décrivent **comment trouver** un élément. L'élément est recherché au moment de l'action, et l'action attend automatiquement qu'il soit présent, visible, stable (sans animation en cours) et, pour un clic ou une saisie, activé. C'est la méthode recommandée pour agir sur la page.

```javascript
await page.locator('[data-testid=add-to-cart]').click();
await page.locator('[data-testid=quantity]').fill('2');
```

Les *handles* (`page.$()`, `page.$$()`, `page.waitForSelector()`) renvoient une **référence** à un élément trouvé à un instant donné. Ils servent surtout à lire des informations.

```javascript
// Lire le texte d'un élément
const total = await page.$eval('[data-testid=cart-total]', (el) => el.textContent.trim());

// Compter des éléments
const lignes = await page.$$('[data-testid=cart-line]');

// Lire un attribut technique plutôt qu'un texte formaté
const prix = Number(await page.$eval('[data-testid=product-price]', (el) => el.dataset.value));
```

`page.$eval(sélecteur, fonction)` exécute la fonction **dans la page**, sur le premier élément correspondant, et renvoie le résultat au scénario. `page.$$eval` fait de même sur la liste de tous les éléments correspondants. Si aucun élément ne correspond, `page.$eval` lève une erreur. Assurez-vous donc de la présence de l'élément au préalable, par exemple avec `expect.visible`.

> ℹ **Information** — Le code passé à `$eval`, `$$eval` ou `waitForFunction` s'exécute dans le navigateur, et non dans le scénario. Il n'a pas accès aux variables du scénario, sauf si elles sont passées explicitement en argument (§ 4.10.4).

4.8 Interagir avec la page

4.8.1 Cliquer

```javascript
await page.locator('[data-testid=add-to-cart]').click();
```

Le clic est réalisé au centre de l'élément, après son défilement dans la zone visible. Puppeteer clique à cette position : si un autre élément recouvre la cible, par exemple un bandeau de consentement aux cookies, c'est cet élément qui reçoit le clic. Le bouton visé n'est alors pas actionné, et c'est la vérification de l'étape suivante qui échoue, avec un diagnostic moins clair. Fermez donc d'abord le bandeau (§ 4.8.6).

4.8.2 Saisir du texte

```javascript
// fill() vide le champ, puis saisit la valeur
await page.locator('[data-testid=login-email]').fill(secrets.BOUTIQUE_USER);
```

Certains champs réagissent à chaque frappe : autocomplétion, masque de saisie. `fill()` peut alors ne pas déclencher le comportement attendu. Dans ce cas, simulez la frappe touche par touche :

```javascript
await page.locator('[data-testid=search-input]').click();
await page.keyboard.type('câble', { delay: 50 }); // 50 ms entre deux frappes
```

4.8.3 Listes, cases à cocher et boutons radio

```javascript
// Liste déroulante : sélection par la valeur technique de l'option
await page.select('#pays', 'be');

// Case à cocher ou bouton radio : un clic
await page.locator('[data-testid=accept-cgv]').click();
await page.locator('[data-testid=delivery-pickup]').click();

// Vérifier l'état d'une case
const cochee = await page.$eval('[data-testid=accept-cgv]', (el) => el.checked);
expect.ok(cochee, 'La case des conditions générales devrait être cochée');
```

4.8.4 Clavier

```javascript
await page.keyboard.press('Enter');      // Valider
await page.keyboard.press('Tab');        // Champ suivant
await page.keyboard.press('Escape');     // Fermer une fenêtre modale
await page.keyboard.press('ArrowDown');  // Parcourir une liste de suggestions
```

Ces événements sont générés par le navigateur, comme s'ils provenaient d'un vrai clavier. Il n'est plus nécessaire de recourir à des bibliothèques de la page, comme jQuery, pour simuler une touche, ce qu'imposait l'ancienne génération.

4.8.5 Survol

```javascript
// Déplier un menu qui s'ouvre au survol
await page.locator('[data-testid=menu-compte]').hover();
await page.locator('[data-testid=menu-commandes]').click();
```

4.8.6 Éléments facultatifs : bandeaux et fenêtres d'information

Un bandeau de consentement aux cookies ou une fenêtre promotionnelle peut apparaître lors de la première visite, ce qui est toujours le cas avec le profil vierge de la sonde. S'il est présent, il faut le fermer. Son absence ne doit pas faire échouer le scénario :

```javascript
await step('Fermer le bandeau de consentement', async () => {
  // Attente courte : le bandeau est facultatif
  const bandeau = await page
    .waitForSelector('[data-testid=cookie-refuse]', { visible: true, timeout: 3000 })
    .catch(() => null);
  if (bandeau) await bandeau.click();
});
```

> ✓ **Conseil** — Lorsque c'est possible, préférez **refuser** les cookies non essentiels plutôt que les accepter. La sonde ne fausse ainsi pas les statistiques d'audience, et le parcours se rapproche de celui d'un utilisateur prudent.

4.9 Authentification

4.9.1 Formulaire de connexion

Le cas le plus courant est un formulaire avec identifiant et mot de passe. Les valeurs proviennent toujours de l'objet `secrets` :

```javascript
await step('Se connecter', async () => {
  await page.goto(`${params.baseUrl}/connexion`, { waitUntil: 'domcontentloaded' });
  await page.locator('[data-testid=login-email]').fill(secrets.BOUTIQUE_USER);
  await page.locator('[data-testid=login-password]').fill(secrets.BOUTIQUE_PASSWORD);
  await Promise.all([
    page.waitForNavigation({ waitUntil: 'domcontentloaded' }),
    page.locator('[data-testid=login-submit]').click(),
  ]);
  // Indispensable : prouver que la connexion a réussi
  await expect.visible(page, '[data-testid=account-menu]');
});
```

La vérification finale est essentielle. De nombreuses applications répondent avec un code HTTP 200 **même lorsque l'authentification échoue**, en réaffichant le formulaire accompagné d'un message d'erreur. Sans vérification de l'espace authentifié, le scénario conclurait à tort au succès.

Les secrets utilisés sont créés par un administrateur et **déclarés** dans le contrôle (liste `secrets`). Leur création et leur rotation sont décrites au chapitre 9.

4.9.2 Authentification HTTP de base

Certaines applications internes demandent une authentification par le navigateur (fenêtre de connexion du navigateur, mécanisme HTTP *Basic*). Elle se configure avant la navigation :

```javascript
await page.authenticate({ username: secrets.INTRANET_USER, password: secrets.INTRANET_PASSWORD });
await page.goto(`${params.baseUrl}/`, { waitUntil: 'domcontentloaded' });
```

4.9.3 Fournisseurs d'identité et authentification unique

Avec une authentification unique (SSO, *Single Sign-On*), l'utilisateur est redirigé vers un fournisseur d'identité, puis renvoyé vers l'application. Le scénario suit le même chemin :

1. ouverture de l'application ;
2. attente du formulaire du fournisseur d'identité, sur un autre domaine ;
3. saisie des identifiants du compte de test ;
4. attente du retour vers l'application et vérification de l'espace authentifié.

Le navigateur de la sonde doit pouvoir joindre le fournisseur d'identité (chapitre 5 du guide d'installation et d'administration). Nommez distinctement l'étape de connexion au fournisseur : si celui-ci est indisponible, le résultat le montrera clairement.

4.9.4 Second facteur d'authentification

Un scénario ne peut pas recevoir un code SMS, valider une notification sur un téléphone ou utiliser une clé physique. Pour superviser une application protégée par un second facteur, convenez avec les équipes sécurité d'un **compte de test dédié**, exempté du second facteur. Encadrez-le par des mesures compensatoires : droits minimaux, restriction aux adresses IP de la sonde, mot de passe long renouvelé régulièrement, surveillance des connexions.

> **Avertissement** — Ne cherchez jamais à contourner un second facteur ou un CAPTCHA, par exemple avec un service de résolution automatique. Outre le risque juridique et contractuel, cela affaiblirait la sécurité de l'application que vous supervisez.

4.9.5 Sessions

Le profil vierge impose une authentification à **chaque exécution** (§ 3.3.3). C'est voulu : la connexion fait partie du service à vérifier, et une session conservée masquerait une panne du mécanisme d'authentification. Si plusieurs scénarios d'un même plan d'exécution nécessitent une session, chacun se connecte. Le § 6.7 détaille la gestion des sessions et des comptes de test.

4.10 Gérer l'attente et les délais

La gestion des attentes est la principale source de fiabilité ou de fragilité d'un scénario. Une application web n'est jamais instantanée : chaque action déclenche des chargements, des appels réseau et des mises à jour de l'affichage. Le scénario doit attendre **le bon événement**, ni trop tôt ni trop longtemps.

4.10.1 Principe : attendre un événement, jamais une durée

À faire	À proscrire
Attendre qu'un élément apparaisse	Attendre 5 secondes « par sécurité »

À faire	À proscrire
Attendre la réponse d'une API	Supposer que la réponse est arrivée
Attendre qu'une condition soit vraie dans la page	Recommencer l'action en boucle

Une pause fixe est **toujours** mauvaise. Trop courte, elle provoque des échecs aléatoires lorsque l'application est un peu lente. Trop longue, elle allonge chaque exécution, fausse les mesures de durée et occupe inutilement le moteur de scénarios.

> **Information** — L'ancienne fonction de pause de Puppeteer (`page.waitForTimeout`) a été supprimée des versions récentes. N'écrivez pas de pause avec `setTimeout` non plus : utilisez l'une des attentes ci-dessous.

4.10.2 Les attentes disponibles

Besoin	Instruction
Agir sur un élément dès qu'il est prêt	`page.locator(sélecteur).click()` , `.fill()` , `.hover()` : attente intégrée
Attendre qu'un élément soit présent sans agir	`page.locator(sélecteur).wait()`
Attendre qu'un élément soit visible	`page.waitForSelector(sélecteur, { visible: true })`
Attendre qu'un élément disparaisse (indicateur de chargement)	`page.waitForSelector(sélecteur, { hidden: true })`
Attendre une nouvelle page	`page.waitForNavigation({ waitUntil: 'domcontentloaded' })` , armée avant le clic
Attendre la réponse d'une requête	`page.waitForResponse(condition)` , armée avant l'action
Attendre une condition calculée dans la page	`page.waitForFunction(fonction, options, …arguments)`

4.10.3 Attendre une réponse réseau

Lorsqu'une action déclenche un appel d'API, attendre la réponse permet de vérifier son code et de mesurer précisément le temps du service.

```javascript
const [reponse] = await Promise.all([
  page.waitForResponse((r) => r.url().includes('/api/produits') && r.request().method() === 'GET'),
  page.locator('[data-testid=search-submit]').click(),
]);
expect.equal(reponse.status(), 200, "Code HTTP de l'API de recherche");
```

La condition passée à `waitForResponse` doit être **précise**, par exemple le chemin et la méthode. Une condition trop large risque de retenir une autre requête de la page.

4.10.4 Attendre une condition

`waitForFunction` évalue une fonction dans la page jusqu'à ce qu'elle renvoie une valeur vraie. Les valeurs du scénario sont transmises en arguments après les options :

```javascript
// Attendre que le compteur du panier ait augmenté de 1
await page.waitForFunction(
  (avant) => Number(document.querySelector('[data-testid=cart-count]').textContent) === avant + 1,
  { timeout: 10000 },
  articlesAvant,
);
```

4.10.5 Hiérarchie des délais

Trois niveaux de délais s'appliquent à un scénario :

Niveau	Défini par	Valeur par défaut	Effet du dépassement
Timeout du contrôle	`timeout` du contrôle	Selon le contrôle (exemple : `60s`)	Le moteur interrompt l'exécution ; état `CRITICAL`, `error.type = timeout`
Délai de navigation	Moteur de scénarios ; option `timeout` de `goto` et `waitForNavigation`	30 s	Exception, `error.type = navigation_error`
Délai d'attente d'un élément ou d'une condition	Moteur ; option `timeout` des attentes ; `locator().setTimeout(ms)`	10 s	Exception, `error.type = element_not_found` ou `assertion_failed`

Règles pratiques :

- fixez le timeout du contrôle à **deux à trois fois la durée normale** du parcours (§ 3.4 du guide d'utilisation et de supervision), et au moins 30 secondes ;
- n'augmentez le délai d'une attente que pour une opération connue pour être lente (génération d'un document, recherche complexe), et seulement pour cette attente ;
- ne cherchez pas à masquer une lenteur par un délai long : utilisez plutôt les seuils de durée (`thresholds.warningDuration`), qui signalent la lenteur par l'état `WARNING` sans faire échouer le scénario (chapitre 3 du guide d'utilisation et de supervision).

```javascript
// Délai spécifique à une opération lente connue
await page.locator('[data-testid=export-ready]').setTimeout(30000).wait();
```

4.11 Assertions

Une **assertion** est une vérification explicite. Si elle n'est pas satisfaite, elle lève une exception et l'étape échoue avec un message lisible. Un scénario sans assertion ne vérifie presque rien : il prouve seulement que les éléments cliqués existaient.

4.11.1 Fonctions fournies

Fonction	Échoue si…	Exemple
`expect.visible(cible, sélecteur, { timeout })`	L'élément n'est pas visible dans le délai	`await expect.visible(page, '[data-testid=account-menu]')`
`expect.hidden(cible, sélecteur, { timeout })`	L'élément reste visible au-delà du délai	`await expect.hidden(page, '[data-testid=loading]')`
`expect.text(cible, sélecteur, attendu, { timeout, exact })`	Le texte ne contient pas `attendu` (ou ne lui est pas égal si `exact: true`) dans le délai	`await expect.text(page, '[data-testid=welcome]', 'Bienvenue')`
`expect.url(cible, motif)`	L'adresse courante ne contient pas le motif (chaîne) ou ne lui correspond pas (expression régulière)	`expect.url(page, /\/compte$/)`
`expect.ok(condition, message)`	La condition est fausse	`expect.ok(lignes.length > 0, 'Panier vide')`
`expect.equal(réel, attendu, message)`	Les deux valeurs ne sont pas strictement égales	`expect.equal(total, 179, 'Total du panier')`

Les fonctions `visible`, `hidden` et `text` sont asynchrones, car elles attendent : elles doivent être précédées de `await`. `url`, `ok` et `equal` vérifient immédiatement. Le paramètre `cible` est généralement `page`, mais peut aussi être un cadre (`Frame`) obtenu à partir d'une iframe (§ 4.13.1).

4.11.2 Écrire de bonnes assertions

- **Vérifiez le résultat, pas l'action.** Après un ajout au panier, vérifiez le contenu du panier, pas seulement l'affichage du message « Produit ajouté ».
- **Rédigez des messages utiles.** Le message apparaît dans les résultats et dans les alertes. « Total du panier : attendu 179, obtenu 89,5 » permet un diagnostic immédiat, contrairement à « Erreur ».
- **Comparez des valeurs techniques.** Lisez un attribut (`data-value="89.50"`) plutôt qu'un texte formaté (« 89,50 € »), qui dépend de la langue et de la mise en forme.
- **Tolérez ce qui peut légitimement varier.** Un nombre de résultats de recherche peut évoluer avec le catalogue : vérifiez un minimum plutôt qu'une valeur exacte.

```javascript
// Assertion avec un message de diagnostic explicite
const totalAffiche = Number(await page.$eval('[data-testid=cart-total]', (el) => el.dataset.value));
const totalAttendu = Math.round(prixUnitaire * quantite * 100) / 100;
expect.equal(totalAffiche, totalAttendu, 'Total du panier');
```

4.12 Traitement des erreurs

4.12.1 Le principe : laisser l'exception remonter

Dans un scénario, une erreur est une **exception**. Toute exception non interceptée dans un `step()` :

1. marque l'étape en échec ;
2. arrête le scénario ;
3. déclenche la collecte des éléments de diagnostic (capture d'écran, console, requêtes en erreur) ;
4. détermine l'état du contrôle et le type d'erreur.

Ce comportement convient dans la très grande majorité des cas. **N'interceptez pas les exceptions pour les ignorer** : un `try/catch` qui avale une erreur transforme une panne en succès apparent.

```javascript
// À NE PAS FAIRE : l'échec est masqué, le scénario conclut à tort au succès
try {
  await expect.visible(page, '[data-testid=account-menu]');
} catch (e) {
  // ignoré
}
```

4.12.2 Cas légitimes d'interception

Interceptez une erreur seulement lorsque l'échec est **normal** et prévu par le parcours, par exemple un élément facultatif (§ 4.8.6) :

```javascript
const bandeau = await page
  .waitForSelector('[data-testid=promo-close]', { visible: true, timeout: 3000 })
  .catch(() => null); // l'absence du bandeau n'est pas une erreur
if (bandeau) await bandeau.click();
```

Pour **enrichir** un message d'erreur, interceptez l'exception, puis levez-en une nouvelle :

```javascript
try {
  await page.locator('[data-testid=pay-test-card]').click();
} catch (e) {
  throw new Error(`Module de paiement de test indisponible : ${e.message}`);
}
```

4.12.3 Types d'erreur et états

Le moteur de scénarios classe chaque échec dans la liste fermée des types d'erreur de la sonde, dont la référence complète figure au chapitre 10 (§ 10.2.2). Pour les scénarios, les plus fréquents sont :

Type (`error.type`)	Situation typique	État
`assertion_failed`	Une assertion `expect` n'est pas satisfaite	`CRITICAL`
`element_not_found`	Un élément attendu par une action n'est jamais apparu	`CRITICAL`
`navigation_error`	La page n'a pas pu être chargée (erreur réseau, délai de navigation)	`CRITICAL`
`dns_error`, `connection_refused`, `tls_error`	Erreur réseau identifiée lors d'une navigation	`CRITICAL`
`timeout`	Le timeout global du contrôle est dépassé	`CRITICAL`
`script_error`	Erreur de programmation dans le script (variable non définie, fonction inexistante)	`CRITICAL`
`browser_crash`	Le navigateur s'est arrêté de façon inattendue	`UNKNOWN`
`secret_missing`	Un secret déclaré dans `secrets` est absent de la sonde : le scénario ne démarre pas	`UNKNOWN`

L'état `UNKNOWN` signale un problème de la sonde ou de l'outil d'exécution, et non de l'application supervisée. Il ne doit pas être confondu avec une panne applicative (chapitre 10).

4.12.4 Journaliser utilement

`log.info()` et `log.warn()` ajoutent des informations au journal de l'exécution, consultable avec le résultat. Utilisez-les pour les valeurs qui faciliteront le diagnostic : nombre de résultats, numéro de commande de test, valeur lue.

```javascript
log.info(`${resultats.length} résultat(s) pour « ${terme} »`);
```

Ne journalisez jamais une donnée personnelle ni une valeur sensible. Les valeurs des secrets déclarés sont masquées automatiquement, mais une donnée affichée par l'application (adresse, numéro de client) ne l'est pas.

4.13 Iframes, fenêtres, boîtes de dialogue et téléchargements

4.13.1 Iframes

Une **iframe** est une page intégrée dans une autre : formulaire de paiement, module de contact, carte, contenu d'un service tiers. Ses éléments appartiennent à un document distinct. Un sélecteur exécuté sur la page principale ne les trouve pas.

En mode scripté, on obtient d'abord le cadre (`Frame`), puis on travaille dans ce cadre :

```javascript
// Récupérer le cadre de l'iframe identifiée par #widget-contact
const cadre = await (await page.waitForSelector('#widget-contact')).contentFrame();
await expect.visible(cadre, '[data-testid=contact-form]');
await cadre.locator('[data-testid=contact-message]').fill('Message de test');
```

En mode déclaratif, la propriété `frame` de l'étape indique le sélecteur de l'iframe :

```json
{ "name": "Vérifier le formulaire de contact", "action": "assertVisible", "frame": "#widget-
    contact", "selector": "[data-testid=contact-form]" }
```

> **Information** — Les iframes de paiement des prestataires spécialisés sont souvent conçues pour résister à l'automatisation, pour des raisons de sécurité. En environnement de test, utilisez le mode de paiement simulé fourni par le prestataire ou par votre application (chapitre 5, exemple 7).

4.13.2 Nouvelles fenêtres et onglets

Un lien ouvert dans un nouvel onglet (`target="_blank"`) ou une fenêtre ouverte par un script crée une nouvelle page. On l'intercepte avec l'événement `popup`, armé avant le clic :

```javascript
const [fenetre] = await Promise.all([
  new Promise((resoudre) => page.once('popup', resoudre)),
  page.locator('[data-testid=open-terms]').click(),
]);
await fenetre.waitForSelector('[data-testid=terms-content]', { visible: true });
await fenetre.close(); // la fenêtre secondaire peut être fermée par le script
```

La page principale (`page`) reste ouverte et utilisable. Le moteur de scénarios ferme toutes les fenêtres en fin d'exécution.

4.13.3 Boîtes de dialogue

Les boîtes de dialogue JavaScript (`alert`, `confirm`, `prompt`) bloquent la page tant qu'elles ne sont pas fermées. Préparez leur traitement avant l'action qui les déclenche :

```javascript
// Accepter la prochaine boîte de confirmation
page.once('dialog', (dialogue) => dialogue.accept());
await page.locator('[data-testid=cart-clear]').click();
```

Pour refuser, utilisez `dialogue.dismiss()`. `dialogue.message()` renvoie le texte affiché, qui peut être vérifié.

4.13.4 Téléchargements

Le téléchargement de fichiers par le navigateur n'est pas un usage prévu des scénarios de la sonde : le moteur de scénarios **refuse les téléchargements**, afin que les fichiers ne s'accumulent pas sur la carte microSD. Pour vérifier qu'un document est disponible (facture de test, export), vérifiez plutôt la **réponse** du serveur :

```javascript
const [reponse] = await Promise.all([
  page.waitForResponse((r) => r.url().includes('/factures/') && r.request().method() === 'GET'),
  page.locator('[data-testid=invoice-download]').click(),
]);
expect.equal(reponse.status(), 200, 'Téléchargement de la facture de test');
expect.ok((reponse.headers()['content-type'] || '').includes('application/pdf'), 'Type de document
    inattendu');
```

4.14 Fermeture et libération des ressources

Avec la sonde, le **moteur de scénarios** est responsable du cycle de vie du navigateur :

- il lance un navigateur neuf avant l'appel du script ;
- il le ferme après le script, que l'exécution ait réussi, échoué ou dépassé son timeout ;
- en cas de blocage, il arrête de force le navigateur et tous ses processus ;
- il détruit le profil temporaire (cookies, cache, stockage local).

Le script ne doit donc **ni lancer ni fermer** le navigateur. N'appelez pas `browser.close()` ni `page.close()` sur la page principale. Les fenêtres secondaires que vous ouvrez peuvent être fermées par le script (§ 4.13.2), mais ce n'est pas obligatoire.

Cette répartition simplifie l'écriture : contrairement à un script Puppeteer autonome, un scénario n'a pas besoin de blocs `try/finally` pour garantir la fermeture du navigateur. Elle garantit aussi qu'aucun processus Chromium orphelin ne subsiste sur la sonde, ce qui est essentiel sur un appareil disposant d'une mémoire limitée.

> **Attention** — Un scénario Puppeteer trouvé sur Internet ou produit par un outil d'enregistrement commence généralement par `puppeteer.launch()` et se termine par `browser.close()`. Ces lignes doivent être **supprimées**, et le reste du code placé dans la fonction exportée, organisé en étapes.

4.15 Tester sur poste de travail avec le lanceur local

4.15.1 Présentation

Le dossier `exemples/puppeteer/` contient un **lanceur local** (`lanceur-local.mjs`). Cet outil pédagogique reproduit sur un poste de travail le contrat d'exécution de la sonde : objet de contexte, étapes, assertions, timeout global, capture en cas d'échec, résultat JSON. Il permet de mettre au point un scénario rapidement, en observant le navigateur, avant de le déployer.

> **Information** — Le lanceur local est **distinct du produit**. Il utilise Puppeteer standard et le navigateur téléchargé par Puppeteer sur le poste. Il n'offre ni l'isolation, ni le chiffrement des secrets, ni la planification, ni la télémétrie de la sonde, et sa classification des erreurs est une approximation. La validation finale se fait toujours par une exécution d'essai sur la sonde (§ 4.16).

4.15.2 Procédure

Objectif : exécuter un scénario sur un poste de travail et lire son résultat.

Prérequis : Node.js 22.12 ou version ultérieure (minimum exigé par Puppeteer 25) ; accès à l'application à tester (ou à l'application de démonstration fournie).

Étapes

1. Installez les dépendances des exemples :

```bash
cd exemples/puppeteer
npm install
```

2. Si vous utilisez l'application de démonstration, démarrez-la dans un premier terminal :

```bash
npm run demo
```

3. Dans un second terminal, définissez les secrets dans des variables d'environnement, jamais sur la ligne de commande :

```bash
# Identifiant du compte de test fictif de l'application de démonstration
export BOUTIQUE_USER='test-supervision@example.com'
# Saisie masquée du mot de passe (Demo-2026! pour l'application de démonstration) :
# il n'apparaît ni à l'écran ni dans l'historique du shell
read -rs BOUTIQUE_PASSWORD && export BOUTIQUE_PASSWORD
```

4. Exécutez le scénario en indiquant ses paramètres et le **nom** des secrets à transmettre :

```bash
node lanceur-local.mjs scenarios/03-authentification.mjs \
  --param baseUrl=http://localhost:8080 \
  --secret BOUTIQUE_USER --secret BOUTIQUE_PASSWORD
```

5. Pour observer le navigateur pendant la mise au point, ajoutez `--visible`.

Résultat attendu : le lanceur affiche le résultat au format JSON et se termine avec le code 0 si l'état est `OK`.

Option	Rôle
`--param nom=valeur`	Paramètre non sensible (répétable)
`--secret NOM`	Secret lu dans la variable d'environnement `NOM` (répétable)
`--timeout 60s`	Délai global du scénario (60 s par défaut), hors lancement et fermeture du navigateur (voir ci-dessous)
`--avertissement 8s`	Durée au-delà de laquelle un succès devient `WARNING`
`--bloquer domaine`	Bloque les requêtes vers un domaine (répétable)
`--visible`	Affiche le navigateur
`--sortie dossier`	Dossier des artefacts (par défaut `resultats/`)

Le lanceur accepte aussi un **contrôle déclaratif** complet au format JSON. Les paramètres et la liste des secrets sont alors lus dans le fichier :

```bash
node lanceur-local.mjs scenarios/03-authentification.json
```

Phases et délais du lanceur. Une exécution comprend trois phases, bornées séparément :

Phase	Contenu	Borne
1. Lancement	Démarrage du navigateur, puis ouverture et préparation de la page (taille d'affichage, délais par défaut, écouteurs de diagnostic)	Démarrage : délai de lancement de Puppeteer, 30 s. Préparation : délai propre de 30 s ; en cas d'erreur ou de dépassement, le navigateur est fermé et le lanceur s'arrête avec le code 3 (`UNKNOWN`)
2. Scénario	Exécution de la fonction du scénario, étape par étape	Délai global (`--timeout`, ou `timeout` du contrôle déclaratif)
3. Fermeture	Capture d'écran en cas d'échec, puis fermeture du navigateur	5 s pour la capture, 5 s pour la fermeture, puis arrêt forcé du processus

Le délai global ne couvre donc que la phase 2, qui est aussi la seule mesurée par `durationMs` : la durée totale de la commande peut le dépasser de la durée du lancement et de la fermeture. Les minuteries des délais sont annulées dès qu'elles deviennent inutiles.

> **Information** — Le délai global est mis en œuvre par une mise en concurrence (`Promise.race`) entre le scénario et une minuterie. Cette construction **n'annule pas** les opérations déjà entamées par le scénario : lorsque le délai expire, le lanceur cesse d'attendre, refuse toute nouvelle étape, puis ferme le navigateur, et c'est cette fermeture qui interrompt les opérations encore en cours. Le moteur de scénarios de la sonde applique ses propres délais (chapitre 3 du guide d'utilisation et de supervision).

4.15.3 Lire le résultat

```json
{
  "id": "01M7TPKDNETWDPE3DP1EJTTDK8",
  "checkId": "03-authentification",
  "trigger": "dryRun",
  "status": "OK",
  "startedAt": "2026-11-19T20:05:08.783Z",
  "durationMs": 344,
  "error": null,
  "steps": [
    { "index": 0, "name": "Ouvrir la page de connexion", "status": "OK", "startedOffsetMs": 1,
    "durationMs": 151 },
    { "index": 1, "name": "Saisir les identifiants", "status": "OK", "startedOffsetMs": 152,
    "durationMs": 89 },
    { "index": 2, "name": "Valider le formulaire", "status": "OK", "startedOffsetMs": 241,
    "durationMs": 94 },
    { "index": 3, "name": "Vérifier l'accès à l'espace client", "status": "OK", "startedOffsetMs":
    335, "durationMs": 9 }
  ],
  "artifacts": [],
  "log": []
}
```

Ce résultat est au **format du lanceur local**. Il reprend la structure des résultats de la sonde (chapitre 10), mais n'est pas conforme au schéma `run-result` du produit. Ses écarts sont les suivants :

Écart	Format du lanceur local	Résultat de la sonde
`log`	Présent : journal des avertissements collectés pendant l'exécution	Absent : le schéma n'admet pas de champ supplémentaire ; les journaux sont des artefacts (`kind: console`)
`checkRevision`	Absent : le lanceur ne gère pas de révisions de configuration	Obligatoire
`checkId`	Nom du fichier, ou `id` du contrôle déclaratif	Identifiant du contrôle sur la sonde
`attempt` , `planRunId` , `measurements` , `trace`	Absents	Présents selon le contexte d'exécution
`trigger`	Toujours `dryRun`	Selon le déclenchement
`error.type`	Classification approchée	Classification du moteur de scénarios

Un outil qui valide les résultats selon le schéma de la sonde (intégration, tests d'un consommateur de l'API) doit donc utiliser l'exemple complet du § 10.1, et non la sortie du lanceur local. En cas d'échec, `error` indique le type, l'étape et le message. Une capture d'écran `failure.png` et le journal `console.json` sont enregistrés dans `resultats/<identifiant>/`. Le code de sortie suit la convention Nagios : 0 `OK` , 1 `WARNING` , 2 `CRITICAL` , 3 `UNKNOWN` .

La durée mesurée commence après le lancement du navigateur et s'arrête à la fin du scénario, avant la fermeture. Elle reflète donc le temps de l'application, et non celui du démarrage ou de la fermeture de Chromium.

> ✓ **Conseil** — Testez systématiquement le **cas d'échec** de votre scénario : modifiez temporairement une valeur attendue, ou utilisez un mot de passe erroné dans votre variable d'environnement, et vérifiez que le scénario échoue à la bonne étape avec un message compréhensible. Un scénario qui ne sait pas échouer ne protège de rien.

4.16 Déployer un scénario sur la sonde

4.16.1 Scénario déclaratif

Objectif : mettre en service un scénario déclaratif.

Prérequis : rôle `operator` ou `admin` ; édition Enterprise ; secrets nécessaires créés par un administrateur (chapitre 9) ; flux réseau ouverts vers l'application et ses ressources (chapitre 5 du guide d'installation et d'administration).

Étapes

1. Créez le contrôle désactivé, par l'interface d'administration ou par l'API :

   ```bash
   # Fonction sapi définie au § 9.1.3 (certificat vérifié, jeton lu dans un fichier)
   sapi -X POST -H 'Content-Type: application/json' \
     --data @demo-connexion.json "$SONDE/api/v1/checks"
   ```

   Le fichier `demo-connexion.json` reprend l'exemple `scenarios/03-authentification.json`. Avant l'envoi, adaptez `params.baseUrl` : dans l'exemple, `http://localhost:8080` désigne le poste de test, alors que sur la sonde, `localhost` désigne la sonde elle-même.

   Pour créer le contrôle désactivé, positionnez `"enabled": false` dans le fichier. La sonde valide la structure et refuse un contrôle invalide avec un message d'erreur détaillé (chapitre 9).

2. Lancez une exécution d'essai :

   ```bash
   sapi -X POST -D - "$SONDE/api/v1/checks/demo-connexion/runs?dryRun=true"
   ```

   L'exécution d'essai réalise le scénario sans modifier l'état du contrôle et sans transmettre de télémétrie. La réponse indique l'adresse de suivi de l'exécution.

3. Consultez le résultat de l'essai (étapes, durée, erreur éventuelle, capture d'écran) dans l'interface ou par `GET /api/v1/runs/{runId}` (chapitre 10).

4. Si le résultat est conforme, activez le contrôle : `POST /api/v1/checks/demo-connexion/enable`.

Vérification : après la première échéance planifiée, le contrôle apparaît avec l'état `OK` dans le tableau de bord.

4.16.2 Scénario scripté

Objectif : mettre en service un scénario scripté.

Prérequis : rôle `admin` ; édition Enterprise ; script testé sur poste de travail (§ 4.15) ; secrets créés.

Étapes

1. **Téléversez le script** par l'interface d'administration ou par `POST /api/v1/scripts`. Chaque requête envoie un seul module `.mjs`, avec le type de contenu `text/javascript`, dans la limite de 1 Mo. La sonde renvoie une **référence immuable** de la forme `nom@sha256:empreinte`, par exemple `boutique-connexion@sha256:4f1c9e2ab7d3` (empreinte abrégée ici ; la référence réelle comporte 64 caractères hexadécimaux).

2. **Créez le contrôle** de type `browser`, en mode `script`, désactivé, en indiquant cette référence dans

`browser.scriptRef` :

```json
{
  "id": "boutique-connexion",
  "name": "Boutique : connexion du compte de test",
  "type": "browser",
  "enabled": false,
  "schedule": { "interval": "5m", "jitter": "30s" },
  "timeout": "60s",
  "secrets": ["BOUTIQUE_USER", "BOUTIQUE_PASSWORD"],
  "browser": {
    "mode": "script",
    "scriptRef": "boutique-connexion@sha256:4f1c9e2ab7d3",
    "params": { "baseUrl": "https://boutique.example.com" },
    "captureOnFailure": true
  }
}
```

3. **Lancez une exécution d'essai** (`?dryRun=true`), comme au § 4.16.1.
4. **Analysez le résultat.** Un écart entre le poste de travail et la sonde provient le plus souvent :
   - d'une ressource que la sonde ne peut pas joindre (pare-feu, proxy, résolution DNS) ;
   - d'une version différente de Puppeteer ou de Chromium (§ 3.8.2) ;
   - de performances plus modestes, qui nécessitent un délai adapté.
5. **Activez le contrôle**, puis ajoutez-le si nécessaire à un plan d'exécution (chapitre 6 du guide d'utilisation et de supervision).

Vérification : le contrôle s'exécute selon sa planification ; les résultats et la télémétrie apparaissent (chapitre 10 et chapitre 2 du guide d'intégration et d'observabilité).

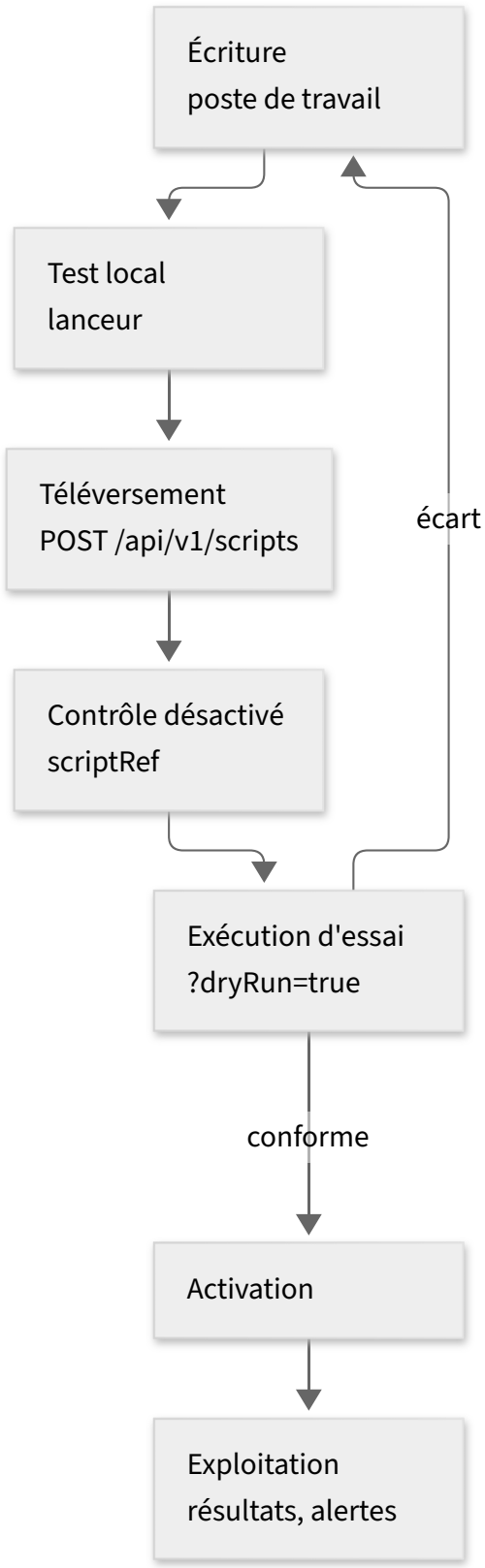


Schéma 4-2 — Cycle de mise en service d'un scénario scripté

4.17 Cycle de vie et versionnage des scripts

4.17.1 Références immuables

Chaque téléversement d'un script crée une **nouvelle version**, identifiée par l'empreinte de son contenu. Un contrôle désigne toujours une version précise (`scriptRef`). Ce mécanisme apporte trois garanties :

- **aucun changement implicite** : téléverser une nouvelle version ne modifie aucun contrôle existant ;
- **traçabilité** : chaque résultat est rattaché à la révision du contrôle, et donc à la version exacte du script exécuté ;
- **retour arrière immédiat** : il suffit de remettre l'ancienne référence dans le contrôle.

4.17.2 Procédure de mise à jour d'un scénario scripté

1. Modifiez le script dans votre gestionnaire de versions (Git recommandé).
2. Testez-le sur poste de travail, y compris le cas d'échec (§ 4.15).
3. Téléversez la nouvelle version et notez la référence renvoyée.
4. Pour un changement important, créez un **contrôle temporaire** désactivé utilisant la nouvelle référence, et lancez une exécution d'essai.
5. Mettez à jour `browser.scriptRef` dans le contrôle de production (`PATCH /api/v1/checks/{id}`). Cette modification requiert le rôle `admin` (§ 4.1).
6. Surveillez les exécutions suivantes. En cas de régression, restaurez la référence précédente.

> ✓ **Conseil** — Inscrivez la référence `nom@sha256:empreinte` renvoyée par la sonde dans le message de version de votre gestionnaire de code. Le lien entre le code source et ce qui s'exécute sur la sonde reste ainsi vérifiable à tout moment.

4.17.3 Coordination avec les mises en production

Une évolution de l'application supervisée peut rendre un scénario obsolète : nouveau libellé, nouvelle étape dans le tunnel, sélecteur supprimé. Le § 6.15 décrit l'organisation recommandée pour maintenir les scénarios au rythme des mises en production, et la manière de suspendre temporairement un scénario pendant une intervention planifiée.

4.17.4 Retrait d'un scénario

Lorsqu'un parcours n'existe plus :

1. désactivez le contrôle et retirez-le des plans d'exécution ;
2. supprimez le contrôle lorsque son historique n'est plus utile (l'historique déjà enregistré est conservé selon la rétention, § 3.12 du guide d'utilisation et de supervision) ;
3. supprimez les secrets qui ne sont plus utilisés par aucun contrôle ;
4. informez les équipes qui reçoivent les alertes ou les tableaux de bord associés.

Chapitre 5 — Exemples progressifs

Ce chapitre présente huit scénarios complets, du plus simple au plus élaboré. Chacun répond à un besoin de supervision courant et met en œuvre une notion nouvelle. Pour chaque exemple sont présentés l'objectif, les prérequis, le code commenté, les explications, la version déclarative lorsqu'elle est possible, le résultat attendu, les variantes utiles et les erreurs courantes.

Tous les exemples ont été exécutés et vérifiés sur un poste de travail, avec le lanceur local décrit au § 4.15, et *Chrome for Testing*. La sonde utilise, elle, le Chromium des dépôts Raspberry Pi OS : avant de déployer un exemple, effectuez une exécution d'essai sur votre sonde (§ 4.16).

5.1 Présentation des exemples

5.1.1 L'application de démonstration

Les exemples ciblent la **Boutique Démo**, une application web **fictive** fournie dans le dossier `exemples/puppeteer/application-demo/`. Elle s'exécute sur votre poste de travail (`http://localhost:8080`) et reproduit les fonctions d'un site marchand simple :

Fonction	Adresse	Particularité utile aux exemples
Accueil	`/`	Produits à la une ; bandeau « environnement de test »
Catalogue	`/catalogue`	Liste des produits
Fiche produit	`/produit/P100` … `/produit/P400`	Ajout au panier par appel d'API, avec un délai simulé
Recherche	`/recherche`	Résultats affichés par JavaScript après un appel d'API
Connexion	`/connexion`	Répond HTTP 200 même en cas d'échec d'authentification
Espace client	`/compte`	Accessible après authentification
Aide	`/aide`	Formulaire de contact dans une iframe
Panier	`/panier`	Lignes, quantités, total
Commande de test	`/commande`	Tunnel en trois étapes, paiement **simulé**, annulation possible

Le compte de test est **fictif** : `test-supervision@example.com`, mot de passe `Demo-2026!`. Tous les éléments utiles portent un attribut `data-testid`, comme le recommande le chapitre 6.

Pour les besoins de l'exemple 8, l'application peut simuler deux **pannes fonctionnelles**. Elles s'activent par un paramètre d'adresse réservé à la démonstration :

Paramètre	Panne simulée
`?panne=panier`	L'ajout au panier affiche « Produit ajouté » mais n'enregistre rien
`?panne=prix`	Le total du panier ignore la quantité



Figure 5-1 — Page d'accueil de la Boutique Démo avec le bandeau « Environnement de test »

5.1.2 Vue d'ensemble

Exemple	Objectif	Notion introduite	Déclaratif	Fichiers
1	Page accessible	`goto`, code HTTP, titre	Oui	`01-page-accessible.*`
2	Élément présent	Attente et vérification d'un contenu	Oui	`02-element-present.*`
3	Authentification	Secrets, formulaire, navigation par clic	Oui	`03-authentification.*`
3 bis	Refus d'authentification	Vérifier un échec attendu	Oui	`03b-authentification-refusee.*`
4	Plusieurs pages	Enchaînement d'étapes, iframe	Oui	`04-navigation-pages.*`
5	Moteur de recherche	Attente d'une réponse d'API, pertinence	Partiel	`05-recherche.*`
6	Ajout au panier	Attente d'une condition, contrôle d'état	Oui	`06-ajout-panier.*`
7	Commande de test	Garde-fous, parcours long, nettoyage	Non	`07-parcours-commande.mjs`
8	Erreur fonctionnelle	Calcul du résultat attendu, indices techniques	Non	`08-detection-erreur-fonctionnelle.mjs`

5.1.3 Exécuter les exemples

Les fichiers des exemples sont livrés dans l'archive `synovex-exemples.zip`, fournie avec les PDF de la documentation et téléchargeable depuis la page d'accueil de la documentation en ligne ; décompressée, elle crée le dossier `exemples/`. La procédure complète est décrite au § 4.15, et dans le fichier `exemples/puppeteer/README.md`. En résumé :

```bash
cd exemples/puppeteer
npm install
npm run demo                        # premier terminal : application fictive
export BOUTIQUE_USER='test-supervision@example.com'
read -rs BOUTIQUE_PASSWORD && export BOUTIQUE_PASSWORD   # saisie masquée : Demo-2026! (compte
    fictif)
node lanceur-local.mjs scenarios/01-page-accessible.mjs --param baseUrl=http://localhost:8080
```

> **Information** — Par défaut, l'application de démonstration n'écoute que sur la **boucle locale** (`127.0.0.1`) : elle n'est joignable que depuis le poste qui l'exécute, même si celui-ci est raccordé à un réseau partagé (salle de formation, réseau d'entreprise). C'est le mode à utiliser pour tous les exemples de ce chapitre.

Mise à disposition volontaire sur un réseau de recette. Pour qu'une sonde de test exécute les exemples contre la Boutique Démo, celle-ci doit être joignable depuis le réseau. Cette exposition est une opération distincte et volontaire :

1. Obtenez l'accord du responsable du réseau : utilisez uniquement un réseau de recette ou de formation autorisé, jamais un réseau de production ni un réseau ouvert (Wi-Fi public, invités).
2. Relevez l'adresse IP du poste sur ce réseau.
3. Démarrez l'application en indiquant cette adresse dans la variable d'environnement `HOTE` (exemple fictif : `HOTE=192.0.2.80 npm run demo`). Le serveur affiche alors un avertissement rappelant que l'application est exposée.
4. Limitez si possible les accès au port de l'application (8080 par défaut) à l'adresse de la sonde de test, par le pare-feu du poste.
5. Dans les scénarios, remplacez `http://localhost:8080` par `http://<adresse-du-poste>:8080` (paramètre `baseUrl`).
6. Arrêtez l'application (Ctrl+C) dès la fin des essais.

> **Avertissement** — La Boutique Démo n'a aucune sécurité : son compte de test et son mot de passe sont publiés dans cette documentation, ses échanges ne sont pas chiffrés et elle ne journalise pas les accès. Ne l'exposez jamais sur toutes les interfaces du poste (`HOTE=0.0.0.0`) ni sur un réseau que vous ne maîtrisez pas.

Pour déployer un exemple sur une sonde, remplacez `baseUrl` par l'adresse de votre application, adaptez les sélecteurs, puis suivez le § 4.16.

> **Information** — Les blocs « Résultat attendu » de ce chapitre sont des **extraits (projection)** de la sortie du lanceur local : ils ne reprennent que les champs utiles à l'explication (`checkId` , `status` , `durationMs` , `error` , `steps` , `artifacts`). La sortie complète du lanceur et ses écarts avec le format de la sonde sont décrits au § 4.15.3 ; l'exemple de résultat intégral de la sonde figure au § 10.1.

> **Avertissement** — Ces scénarios sont écrits pour l'application de démonstration. Ne les exécutez jamais tels quels contre une application réelle, et jamais contre une application de production sans l'accord de ses responsables. Les exemples 6 et 7 modifient des données (panier, commande).

5.2 Exemple 1 — Vérifier qu'une page web est accessible

5.2.1 Objectif

Vérifier que la page d'accueil se charge dans un navigateur, répond avec un code HTTP de succès et provient bien de l'application attendue.

C'est le scénario le plus simple. Il va déjà au-delà d'un contrôle `http` : la page est traitée par un vrai navigateur, et la vérification du titre écarte les fausses réussites, comme une page de maintenance ou la page d'erreur d'un proxy servies avec un code 200.

5.2.2 Prérequis

- Application de démonstration démarrée.
- Paramètre `baseUrl`.

5.2.3 Scénario scripté

Fichier `exemples/puppeteer/scenarios/01-page-accessible.mjs` :

```javascript
// Exemple 1 — Vérifier qu'une page web est accessible.
// Application cible : « Boutique Démo » (fictive). Paramètre : baseUrl.
// Contrat : le moteur fournit page, step, params et expect ; il lance et ferme le navigateur.

export default async function scenario({ page, step, params, expect }) {
  await step("Ouvrir la page d'accueil", async () => {
    // page.goto() renvoie la réponse HTTP du document principal.
    const reponse = await page.goto(`${params.baseUrl}/`, { waitUntil: 'domcontentloaded' });
    // Une page accessible doit répondre avec un code 2xx.
    expect.ok(reponse && reponse.ok(), `Code HTTP inattendu : ${reponse ? reponse.status() : 'aucune
    réponse'}`);
  });

  await step('Vérifier le titre de la page', async () => {
    // Le titre confirme que la bonne application a répondu (et non une page d'erreur d'un proxy).
    const titre = await page.title();
    expect.ok(titre.includes('Boutique Démo'), `Titre inattendu : « ${titre} »`);
  });
}
```

5.2.4 Explications

- `page.goto()` renvoie la **réponse HTTP du document principal**. `reponse.ok()` est vrai pour un code compris entre 200 et 299. La réponse peut être nulle dans de rares cas (navigation vers la même adresse avec une ancre) : le test `reponse && …` couvre ce cas.
- `waitUntil: 'domcontentloaded'` considère la page chargée dès que le document est analysé, sans attendre toutes les images. C'est la valeur recommandée (§ 4.6.1).
- Les deux étapes apparaissent séparément dans le résultat. Une page lente se distingue ainsi d'une page erronée.

5.2.5 Version déclarative

Fichier `exemples/puppeteer/scenarios/01-page-accessible.json` :

```json
{
  "id": "demo-accueil-accessible",
  "name": "Boutique Démo : page d'accueil accessible",
  "type": "browser",
  "schedule": { "interval": "5m", "jitter": "30s" },
  "timeout": "30s",
  "browser": {
    "mode": "steps",
    "params": { "baseUrl": "http://localhost:8080" },
    "steps": [
      { "name": "Ouvrir la page d'accueil", "action": "goto", "url": "${param:baseUrl}/" },
      { "name": "Vérifier le titre de la page", "action": "assertTitle", "operator": "contains",
    "value": "Boutique Démo" }
    ]
  }
}
```

En mode déclaratif, l'action `goto` échoue si la page ne peut pas être chargée (erreur réseau, délai dépassé). La vérification explicite du code HTTP n'est pas disponible dans ce mode : l'assertion sur le titre la remplace utilement.

5.2.6 Résultat attendu

```json
{
  "checkId": "01-page-accessible",
  "status": "OK",
  "durationMs": 76,
  "error": null,
  "steps": [
    { "index": 0, "name": "Ouvrir la page d'accueil", "status": "OK", "durationMs": 71 },
    { "index": 1, "name": "Vérifier le titre de la page", "status": "OK", "durationMs": 5 }
  ]
}
```

Les durées dépendent de l'environnement. Elles sont ici très courtes, car l'application fonctionne sur le poste de travail lui-même.

5.2.7 Variantes

- **Seuil de performance** : ajoutez `"thresholds": { "warningDuration": "3s" }` au contrôle pour obtenir un état `WARNING` lorsque la page devient lente.
- **Page interne** : la même structure vérifie toute page importante (page de connexion, page de statut publique).

5.2.8 Erreurs courantes

Symptôme	Cause probable	Correction
`navigation_error` sur l'étape 1	Adresse erronée ; application arrêtée ; flux bloqué par le pare-feu	Vérifier `baseUrl` et les flux (chapitre 5 du guide d'installation et d'administration)
`connection_refused`	Aucun service n'écoute sur l'adresse et le port indiqués	Vérifier le port et l'état de l'application

Symptôme	Cause probable	Correction
`tls_error`	Certificat expiré, ou émis par une autorité interne non reconnue par la sonde	Renouveler le certificat ; importer l'autorité interne (chapitre 5 du guide d'installation et d'administration)
Titre inattendu	Page de maintenance, page d'un proxy, redirection vers un portail captif	Examiner la capture d'écran (chapitre 10)

5.3 Exemple 2 — Vérifier la présence d'un élément attendu

5.3.1 Objectif

Vérifier que la page affiche son **contenu utile** : le message d'accueil et au moins un produit à la une. Une page peut se charger correctement alors que la zone principale reste vide, par exemple lorsque le service qui fournit les produits est en panne.

5.3.2 Prérequis

Identiques à l'exemple 1.

5.3.3 Scénario scripté

Fichier `exemples/puppeteer/scenarios/02-element-present.mjs` :

JAVASCRIPT

```
// Exemple 2 — Vérifier la présence d'un élément attendu.
// On ne se contente pas d'une réponse HTTP : on vérifie que le contenu utile est affiché.

export default async function scenario({ page, step, params, expect }) {
  await step("Ouvrir la page d'accueil", async () => {
    await page.goto(`${params.baseUrl}/`, { waitUntil: 'domcontentloaded' });
  });

  await step("Vérifier le message d'accueil", async () => {
    // Attente explicite de l'élément, puis contrôle de son texte.
    await expect.text(page, '[data-testid=welcome]', 'Bienvenue');
  });

  await step('Vérifier la présence des produits à la une', async () => {
    // expect.visible() attend qu'au moins un produit soit visible :
    // il garantit à lui seul la présence d'un produit à la une.
    await expect.visible(page, '[data-testid=featured-product]');
  });
}
```

5.3.4 Explications

- `expect.text()` **attend** que l'élément existe et que son texte contienne la valeur attendue, dans la limite du délai par défaut. Si le contenu est affiché par un script après le chargement, l'assertion patiente sans pause fixe.
- `expect.visible()` attend qu'au moins un élément correspondant soit **visible**. Un élément présent dans le document mais masqué ne suffit pas.
- Pour exiger **plusieurs** éléments, comptez-les après cette attente : `page.$$()` renvoie la liste des éléments correspondants, et `expect.ok(produits.length >= 3, …)` vérifie la quantité minimale. Un seuil de 1 serait redondant : `expect.visible()` échouerait avant.

5.3.5 Version déclarative

Fichier `exemples/puppeteer/scenarios/02-element-present.json` :

```json
{
  "id": "demo-accueil-contenu",
  "name": "Boutique Démo : contenu de la page d'accueil",
  "type": "browser",
  "schedule": { "interval": "5m" },
  "timeout": "30s",
  "browser": {
    "mode": "steps",
    "params": { "baseUrl": "http://localhost:8080" },
    "steps": [
      { "name": "Ouvrir la page d'accueil", "action": "goto", "url": "${param:baseUrl}/" },
      { "name": "Vérifier le message d'accueil", "action": "assertText", "selector": "[data-
    testid=welcome]", "operator": "contains", "value": "Bienvenue" },
      { "name": "Vérifier la présence des produits à la une", "action": "assertVisible", "selector":
    "[data-testid=featured-product]" }
    ]
  }
}
```

5.3.6 Résultat attendu

État `OK`, trois étapes `OK`. Si la zone des produits est vide, l'étape « Vérifier la présence des produits à la une » échoue avec un message de la forme :

```json
{
  "status": "CRITICAL",
  "error": {
    "type": "assertion_failed",
    "step": "Vérifier la présence des produits à la une",
    "message": "Élément [data-testid=featured-product] non visible après 10000 ms"
  }
}
```

5.3.7 Variantes

- **Absence d'un message d'erreur** : `await expect.hidden(page, '.message-erreur-global')` vérifie qu'aucun bandeau d'incident n'est affiché.
- **Contenu dynamique** : pour un élément dont le texte varie (date, prix), vérifiez sa présence et sa forme plutôt que sa valeur exacte, par exemple avec une expression régulière en mode déclaratif (`"operator": "matches"`).

5.3.8 Erreurs courantes

Symptôme	Cause probable	Correction
Échec après une mise en production, alors que la page semble correcte	Le sélecteur ne correspond plus (attribut renommé)	Mettre à jour le sélecteur ; convenir d'attributs `data-testid` stables (chapitre 6)
Échec aléatoire	Contenu affiché tardivement, au-delà du délai	Vérifier la lenteur réelle ; ajuster le délai de cette seule assertion
Le scénario réussit alors que la page est vide	Sélecteur trop large, qui trouve un autre élément	Rendre le sélecteur spécifique ; vérifier un texte

5.4 Exemple 3 — S'authentifier sur une application

5.4.1 Objectif

Vérifier que le compte de test peut se connecter et accéder à son espace client. C'est l'un des scénarios les plus utiles : une authentification défaillante rend l'application inutilisable pour tous les utilisateurs, alors que les pages publiques restent accessibles et que les contrôles techniques restent au vert.

5.4.2 Prérequis

- Paramètre `baseUrl`.
- Secrets `BOUTIQUE_USER` et `BOUTIQUE_PASSWORD`, créés sur la sonde par un administrateur et déclarés dans le contrôle (chapitre 9). Sur poste de travail, ce sont des variables d'environnement.
- Un **compte de test dédié**, aux droits minimaux, convenu avec les responsables de l'application (§ 6.7).

5.4.3 Scénario scripté

Fichier `exemples/puppeteer/scenarios/03-authentification.mjs` :

JAVASCRIPT

```
// Exemple 3 — S'authentifier sur une application avec un compte de test.
// Secrets déclarés par le contrôle : BOUTIQUE_USER, BOUTIQUE_PASSWORD.
// Les valeurs ne figurent jamais dans le script : elles sont fournies par le moteur.

export default async function scenario({ page, step, secrets, params, expect }) {
  await step('Ouvrir la page de connexion', async () => {
    await page.goto(`${params.baseUrl}/connexion`, { waitUntil: 'domcontentloaded' });
    await expect.visible(page, '[data-testid=login-form]');
  });

  await step('Saisir les identifiants', async () => {
    // locator().fill() attend que le champ soit disponible, le vide, puis saisit la valeur.
    await page.locator('[data-testid=login-email]').fill(secrets.BOUTIQUE_USER);
    await page.locator('[data-testid=login-password]').fill(secrets.BOUTIQUE_PASSWORD);
  });

  await step('Valider le formulaire', async () => {
    // Le clic déclenche une navigation : on l'attend en même temps que le clic,
    // pour ne pas manquer un chargement très rapide.
    await Promise.all([
      page.waitForNavigation({ waitUntil: 'domcontentloaded' }),
      page.locator('[data-testid=login-submit]').click(),
    ]);
  });

  await step("Vérifier l'accès à l'espace client", async () => {
    // L'application répond HTTP 200 même en cas d'échec d'authentification :
    // seule la présence de l'espace client prouve la réussite.
    await expect.visible(page, '[data-testid=account-menu]');
    expect.url(page, '/compte');
    await expect.text(page, '[data-testid=account-name]', 'Bonjour');
  });
}
```

5.4.4 Explications

- **Aucun identifiant dans le code.** `secrets.BOUTIQUE_USER` et `secrets.BOUTIQUE_PASSWORD` sont fournis par le moteur de scénarios, uniquement parce qu'ils sont déclarés dans le contrôle. Leurs valeurs sont masquées dans les journaux.

- **Navigation armée avant le clic.** `Promise.all([page.waitForNavigation(…), …click()])` garantit que le chargement de la page suivante ne sera pas manqué (§ 4.6.2).
- **Vérification de la réussite.** L'application de démonstration, comme beaucoup d'applications réelles, répond HTTP 200 même lorsque l'authentification échoue. Seule la présence de l'espace client (`account-menu`), associée à l'adresse `/compte`, prouve la connexion.

5.4.5 Version déclarative

Fichier `exemples/puppeteer/scenarios/03-authentification.json` :

JSON

```
{
  "id": "demo-connexion",
  "name": "Boutique Démo : connexion du compte de test",
  "type": "browser",
  "schedule": { "interval": "5m", "jitter": "30s" },
  "timeout": "45s",
  "retry": { "attempts": 1, "interval": "30s" },
  "secrets": ["BOUTIQUE_USER", "BOUTIQUE_PASSWORD"],
  "browser": {
    "mode": "steps",
    "params": { "baseUrl": "http://localhost:8080" },
    "captureOnFailure": true,
    "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 le formulaire", "action": "click", "selector": "[data-testid=login-
    submit]", "waitForNavigation": true },
      { "name": "Vérifier l'accès à l'espace client", "action": "assertVisible", "selector": "[data-
    testid=account-menu]" },
      { "name": "Vérifier l'URL de l'espace client", "action": "assertUrl", "operator": "contains",
    "value": "/compte" }
    ]
  }
}
```

La propriété `retry` du contrôle autorise une nouvelle tentative après 30 secondes avant de conclure à l'état `CRITICAL`. Un incident ponctuel, comme une perte de paquets réseau, ne déclenche ainsi pas d'alerte (chapitre 3 du guide d'utilisation et de supervision).

5.4.6 Résultat attendu

Résultat obtenu avec le lanceur local :

```json
{
  "checkId": "03-authentification",
  "status": "OK",
  "durationMs": 344,
  "error": null,
  "steps": [
    { "index": 0, "name": "Ouvrir la page de connexion", "status": "OK", "startedOffsetMs": 1,
    "durationMs": 151 },
    { "index": 1, "name": "Saisir les identifiants", "status": "OK", "startedOffsetMs": 152,
    "durationMs": 89 },
    { "index": 2, "name": "Valider le formulaire", "status": "OK", "startedOffsetMs": 241,
    "durationMs": 94 },
    { "index": 3, "name": "Vérifier l'accès à l'espace client", "status": "OK", "startedOffsetMs":
    335, "durationMs": 9 }
  ]
}
```

Avec un mot de passe erroné, le scénario échoue à la dernière étape :

```json
{
  "status": "CRITICAL",
  "error": {
    "type": "assertion_failed",
    "step": "Vérifier l'accès à l'espace client",
    "message": "Élément [data-testid=account-menu] non visible après 10000 ms"
  }
}
```

5.4.7 Variantes

- **Authentification unique (SSO)** : ajoutez une étape « Se connecter au fournisseur d'identité » entre l'ouverture de l'application et la vérification de l'espace client (§ 4.9.3).
- **Déconnexion** : ajoutez une étape finale qui clique sur « Se déconnecter » et vérifie le retour à la page publique. Le mécanisme de déconnexion est ainsi vérifié, et la session est libérée côté serveur.

5.4.8 Erreurs courantes

Symptôme	Cause probable	Correction
`UNKNOWN` avant toute étape	Secret non déclaré dans `secrets` (mode scripté)	Créer le secret et l'ajouter à la liste `secrets` (chapitre 9)
Échec à « Vérifier l'accès à l'espace client »	Mot de passe expiré ou modifié ; compte verrouillé	Vérifier le compte avec les responsables ; mettre à jour le secret
Échecs répétés puis verrouillage du compte	Les nouvelles tentatives multiplient les échecs d'authentification	Limiter `retry.attempts` ; convenir d'une politique de verrouillage adaptée au compte de test
Échec sur une autre étape après une mise en production	Formulaire modifié	Mettre à jour les sélecteurs

5.4.9 Exemple complémentaire 3 bis — Vérifier le refus d'une authentification invalide

Ce scénario reprend l'intention d'un scénario historique de la génération précédente : soumettre volontairement des identifiants **invalides** et vérifier que l'application les refuse en affichant son message d'erreur. Il complète l'exemple 3. Il vérifie que le mécanisme d'authentification n'accepte pas n'importe quel identifiant, et que la page d'erreur fonctionne.

Fichier `exemples/puppeteer/scenarios/03b-authentification-refusee.mjs` :

JAVASCRIPT

```
// Exemple 3 bis — Vérifier qu'une authentification invalide est correctement refusée.
// Reprend l'intention d'un scénario historique : on soumet volontairement des
// identifiants erronés et l'on vérifie que l'application affiche son message d'erreur.
// Le compte utilisé est fictif et n'existe pas : aucun secret n'est nécessaire.
// Attention : sur une application réelle, des échecs répétés peuvent verrouiller un
// compte ou alerter l'équipe sécurité ; utilisez un identifiant inexistant convenu avec elle.

export default async function scenario({ page, step, params, expect }) {
  await step('Ouvrir la page de connexion', async () => {
    await page.goto(`${params.baseUrl}/connexion`, { waitUntil: 'domcontentloaded' });
  });

  await step('Soumettre des identifiants invalides', async () => {
    await page.locator('[data-testid=login-email]').fill('compte-inexistant@example.com');
    await page.locator('[data-testid=login-password]').fill('mot-de-passe-volontairement-faux');
    await Promise.all([
      page.waitForNavigation({ waitUntil: 'domcontentloaded' }),
      page.locator('[data-testid=login-submit]').click(),
    ]);
  });

  await step("Vérifier le message d'erreur", async () => {
    // Le résultat attendu est un refus : le message doit apparaître…
    await expect.text(page, '[data-testid=login-error]', "Erreur d'authentification");
    // … et l'espace client ne doit pas être accessible.
    await expect.hidden(page, '[data-testid=account-menu]');
  });
}
```

Version déclarative :

Fichier `exemples/puppeteer/scenarios/03b-authentification-refusee.json` :

```json
{
  "id": "demo-connexion-refusee",
  "name": "Boutique Démo : refus d'une authentification invalide",
  "type": "browser",
  "schedule": { "interval": "15m" },
  "timeout": "30s",
  "browser": {
    "mode": "steps",
    "params": { "baseUrl": "http://localhost:8080" },
    "steps": [
      { "name": "Ouvrir la page de connexion", "action": "goto", "url": "${param:baseUrl}/connexion"
    },
      { "name": "Saisir un identifiant inexistant", "action": "fill", "selector": "[data-
    testid=login-email]", "value": "compte-inexistant@example.com" },
      { "name": "Saisir un mot de passe erroné", "action": "fill", "selector": "[data-testid=login-
    password]", "value": "mot-de-passe-volontairement-faux" },
      { "name": "Valider le formulaire", "action": "click", "selector": "[data-testid=login-
    submit]", "waitForNavigation": true },
      { "name": "Vérifier le message d'erreur", "action": "assertText", "selector": "[data-
    testid=login-error]", "operator": "contains", "value": "Erreur d'authentification" },
      { "name": "Vérifier l'absence d'accès", "action": "assertHidden", "selector": "[data-
    testid=account-menu]" }
    ]
  }
}
```

Résultat attendu : état `OK`, car le refus est le comportement attendu. Si l'application acceptait l'identifiant inexistant, ou n'affichait plus son message d'erreur, le scénario passerait à l'état `CRITICAL`.

> ! **Attention** — Sur une application réelle, des échecs d'authentification répétés peuvent déclencher des alertes de sécurité ou verrouiller un compte existant. Convenez avec l'équipe sécurité d'un identifiant **inexistant** réservé à ce scénario, et d'une fréquence modérée (toutes les 15 minutes ou plus).

5.5 Exemple 4 — Parcourir plusieurs pages

5.5.1 Objectif

Vérifier que les pages principales s'enchaînent : accueil, catalogue, fiche produit, aide. Le formulaire de contact, intégré dans une **iframe**, doit aussi être disponible. Chaque page constitue une étape : le résultat indique laquelle est lente ou en échec.

5.5.2 Prérequis

- Paramètre `baseUrl`, et paramètre facultatif `produit` (par défaut `P200`).

5.5.3 Scénario scripté

Fichier `exemples/puppeteer/scenarios/04-navigation-pages.mjs` :

```javascript
// Exemple 4 — Parcourir plusieurs pages, y compris un contenu placé dans une iframe.
// Chaque page visitée constitue une étape : le résultat indique laquelle est lente ou en échec.

export default async function scenario({ page, step, params, expect }) {
  // Clic sur un lien suivi de l'attente de la nouvelle page.
```

```
  const cliquerEtAttendre = (selecteur) =>
    Promise.all([
      page.waitForNavigation({ waitUntil: 'domcontentloaded' }),
      page.locator(selecteur).click(),
    ]);

  await step("Ouvrir la page d'accueil", async () => {
    await page.goto(`${params.baseUrl}/`, { waitUntil: 'domcontentloaded' });
    await expect.visible(page, '[data-testid=welcome]');
  });

  await step('Ouvrir le catalogue', async () => {
    await cliquerEtAttendre('[data-testid=nav-catalogue]');
    await expect.text(page, '[data-testid=page-title]', 'Catalogue', { exact: true });
    const articles = await page.$$('[data-testid=catalog-item]');
    expect.ok(articles.length > 0, 'Le catalogue est vide');
  });

  await step('Ouvrir une fiche produit', async () => {
    // Le produit visé est un paramètre : le scénario reste valable si le catalogue évolue.
    await cliquerEtAttendre(`[data-testid=product-link-${params.produit || 'P200'}]`);
    await expect.visible(page, '[data-testid=product-name]');
    await expect.visible(page, '[data-testid=add-to-cart]');
  });

  await step("Ouvrir la page d'aide", async () => {
    await cliquerEtAttendre('[data-testid=nav-aide]');
    await expect.visible(page, '[data-testid=contact-frame]');
  });

  await step('Vérifier le formulaire de contact (iframe)', async () => {
    // Les éléments d'une iframe appartiennent à un autre document :
    // on obtient d'abord l'objet Frame, puis on travaille dans ce cadre.
    const cadre = await (await page.waitForSelector('#widget-contact')).contentFrame();
    await expect.visible(cadre, '[data-testid=contact-form]');
    await expect.visible(cadre, '[data-testid=contact-submit]');
    // Aucun message n'est envoyé : la supervision ne doit pas créer de faux contacts.
  });
}
```

5.5.4 Explications

- **Fonction utilitaire.** `cliquerEtAttendre()` évite de répéter le motif « attente de navigation et clic » à chaque page. Elle est déclarée hors des étapes, car elle n'agit pas sur la page.
- **Navigation par les liens**, et non par des adresses directes. Le scénario vérifie ainsi aussi que les liens de navigation fonctionnent, comme le ferait un utilisateur.
- **Iframe.** Le formulaire de contact appartient à un autre document. `contentFrame()` donne accès à ce document, sur lequel les assertions s'appliquent normalement (§ 4.13.1).
- **Aucun envoi.** Le scénario vérifie la présence du formulaire sans l'envoyer. La supervision ne doit pas créer de faux messages dans les outils de support.

5.5.5 Version déclarative

Fichier `exemples/puppeteer/scenarios/04-navigation-pages.json` :

```json
{
  "id": "demo-navigation",
  "name": "Boutique Démo : navigation entre les pages principales",
  "type": "browser",
  "schedule": { "interval": "10m" },
  "timeout": "60s",
  "thresholds": { "warningDuration": "15s" },
  "browser": {
    "mode": "steps",
    "params": { "baseUrl": "http://localhost:8080" },
    "steps": [
      { "name": "Ouvrir la page d'accueil", "action": "goto", "url": "${param:baseUrl}/" },
      { "name": "Ouvrir le catalogue", "action": "click", "selector": "[data-testid=nav-catalogue]",
    "waitForNavigation": true },
      { "name": "Vérifier le catalogue", "action": "assertVisible", "selector": "[data-
    testid=catalog-item]" },
      { "name": "Ouvrir une fiche produit", "action": "click", "selector": "[data-testid=product-
    link-P200]", "waitForNavigation": true },
      { "name": "Vérifier la fiche produit", "action": "assertVisible", "selector": "[data-
    testid=add-to-cart]" },
      { "name": "Ouvrir la page d'aide", "action": "click", "selector": "[data-testid=nav-aide]",
    "waitForNavigation": true },
      { "name": "Vérifier le formulaire de contact (iframe)", "action": "assertVisible", "frame":
    "#widget-contact", "selector": "[data-testid=contact-form]" }
    ]
  }
}
```

La propriété `frame` de la dernière étape désigne l'iframe dans laquelle l'élément doit être recherché.

5.5.6 Résultat attendu

État `OK`, cinq étapes (sept en version déclarative). Le seuil `warningDuration` de 15 secondes, défini dans la version déclarative, signale par un état `WARNING` un parcours anormalement lent.

5.5.7 Variantes

- **Pages critiques métier** : remplacez le parcours par les pages qui comptent pour vos utilisateurs (conditions générales, page de contact, foire aux questions).
- **Vérification du menu** : ajoutez des assertions sur la présence de chaque entrée du menu principal.

5.5.8 Erreurs courantes

Symptôme	Cause probable	Correction
`navigation_error` sur un clic	Le lien n'entraîne pas de chargement de page (application monopage)	Remplacer `waitForNavigation` par l'attente d'un élément de la nouvelle vue (§ 4.6.3)
`element_not_found` dans l'iframe	Sélecteur recherché dans la page principale au lieu du cadre ; iframe chargée lentement	Utiliser le cadre (`contentFrame`) ; attendre l'élément dans le cadre
Iframe d'un domaine tiers inaccessible	Flux réseau non autorisé depuis la sonde	Autoriser le domaine de l'iframe (chapitre 5 du guide d'installation et d'administration)

5.6 Exemple 5 — Vérifier le fonctionnement d'un moteur de recherche

5.6.1 Objectif

Vérifier que la recherche renvoie des résultats **pertinents** : un nombre minimal de résultats, qui contiennent tous le terme recherché. La recherche de l'application est exécutée par un appel d'API depuis la page : le scénario attend la réponse de cette API et contrôle son code.

5.6.2 Prérequis

- Paramètres `baseUrl`, `terme` (par défaut « câble ») et `minimum` (par défaut 1).
- Un terme dont les résultats sont **stables** dans le temps, convenu avec les responsables du catalogue.

5.6.3 Scénario scripté

Fichier `exemples/puppeteer/scenarios/05-recherche.mjs` :

JAVASCRIPT

```
// Exemple 5 — Vérifier le fonctionnement d'un moteur de recherche.
// La recherche de l'application est exécutée en JavaScript par un appel d'API :
// on attend la réponse de cette API, puis l'affichage des résultats.
// Paramètres : baseUrl, terme (défaut « câble »), minimum (défaut 1).

export default async function scenario({ page, step, params, expect, log }) {
  const terme = params.terme || 'câble';
  const minimum = Number(params.minimum || 1);

  await step('Ouvrir la page de recherche', async () => {
    await page.goto(`${params.baseUrl}/recherche`, { waitUntil: 'domcontentloaded' });
    await expect.visible(page, '[data-testid=search-input]');
  });

  await step('Lancer la recherche', async () => {
    await page.locator('[data-testid=search-input]').fill(terme);
    // On arme l'attente de la réponse AVANT le clic, puis on clique.
    const [reponse] = await Promise.all([
      page.waitForResponse((r) => r.url().includes('/api/produits') && r.request().method() ===
    'GET'),
      page.locator('[data-testid=search-submit]').click(),
    ]);
    expect.equal(reponse.status(), 200, "Code HTTP de l'API de recherche");
  });

  await step('Vérifier les résultats', async () => {
    // Le compteur n'apparaît qu'une fois les résultats affichés.
    await expect.visible(page, '[data-testid=search-count]');
    const resultats = await page.$$eval('[data-testid=search-result]', (els) => els.map((e) =>
    e.textContent.trim()));
    log.info(`${resultats.length} résultat(s) pour « ${terme} »`);
    expect.ok(resultats.length >= minimum, `${resultats.length} résultat(s) pour « ${terme} »,
    ${minimum} attendu(s) au minimum`);
    // Pertinence : chaque résultat doit contenir le terme recherché.
    const horsSujet = resultats.filter((t) => !t.toLowerCase().includes(terme.toLowerCase()));
    expect.ok(horsSujet.length === 0, `Résultats sans rapport avec « ${terme} » :
    ${horsSujet.join(', ')}`);
  });
}
```

5.6.4 Explications

- **Attente d'une réponse d'API.** `page.waitForResponse()` est armée **avant** le clic, dans un `Promise.all`. La condition est précise : chemin `/api/produits` et méthode `GET`. Le code de la réponse est vérifié explicitement. Si l'API répond en erreur, le message l'indique, ce qui oriente immédiatement le diagnostic vers le service de recherche plutôt que vers l'interface.
- **Lecture groupée.** `page.$$eval()` lit en une seule opération le texte de tous les résultats.
- **Pertinence.** Vérifier qu'il existe des résultats ne suffit pas : un moteur défaillant peut renvoyer tout le catalogue. Le scénario vérifie que chaque résultat contient le terme.
- **Journalisation.** Le nombre de résultats est journalisé. En cas d'anomalie, cette information est disponible avec le résultat.

5.6.5 Version déclarative

Fichier `exemples/puppeteer/scenarios/05-recherche.json` :

```json
{
  "id": "demo-recherche",
  "name": "Boutique Démo : moteur de recherche",
  "type": "browser",
  "schedule": { "interval": "10m" },
  "timeout": "45s",
  "browser": {
    "mode": "steps",
    "params": { "baseUrl": "http://localhost:8080", "terme": "câble" },
    "steps": [
      { "name": "Ouvrir la page de recherche", "action": "goto", "url": "${param:baseUrl}/recherche"
    },
      { "name": "Saisir le terme recherché", "action": "fill", "selector": "[data-testid=search-
    input]", "value": "${param:terme}" },
      { "name": "Lancer la recherche", "action": "click", "selector": "[data-testid=search-submit]"
    },
      { "name": "Vérifier l'affichage des résultats", "action": "assertVisible", "selector": "[data-
    testid=search-result]" },
      { "name": "Vérifier le compteur de résultats", "action": "assertText", "selector": "[data-
    testid=search-count]", "operator": "matches", "value": "^[1-9][0-9]* résultat" }
    ]
  }
}
```

La version déclarative vérifie l'affichage des résultats et la forme du compteur. Elle ne peut ni contrôler la réponse de l'API, ni vérifier la pertinence de chaque résultat : c'est un exemple typique de besoin qui justifie le mode scripté.

5.6.6 Résultat attendu

État `OK`. Le journal de l'exécution contient :

```json
"log": [ { "niveau": "INFO", "message": "2 résultat(s) pour « câble »" } ]
```

Avec un terme sans résultat (`--param terme=introuvable`), le scénario échoue à l'étape attendue :

```json
{
  "status": "CRITICAL",
  "error": {
    "type": "assertion_failed",
    "step": "Vérifier les résultats",
    "message": "0 résultat(s) pour « introuvable », 1 attendu(s) au minimum"
  }
}
```

5.6.7 Variantes

- **Temps de réponse du moteur** : ajoutez un seuil `warningDuration` au contrôle, ou mesurez la durée entre le clic et la réponse et journalisez-la.
- **Validation par la touche Entrée** : remplacez le clic par `page.keyboard.press('Enter')` dans le `Promise.all`, pour vérifier le comportement le plus courant chez les utilisateurs.
- **Suggestions** : vérifiez la liste de suggestions qui s'affiche pendant la frappe, en saisissant avec `page.keyboard.type()` (§ 4.8.2).

5.6.8 Erreurs courantes

Symptôme	Cause probable	Correction
`timeout` ou `element_not_found` à « Lancer la recherche »	La condition de `waitForResponse` ne correspond à aucune requête (chemin d'API différent)	Identifier la requête dans l'onglet *Network* des outils de développement
Faux échec après une évolution du catalogue	Le terme choisi n'a plus de résultats	Convenir d'un terme et d'un produit de référence pérennes
Résultats « hors sujet »	Recherche floue ou synonymes volontaires	Adapter le critère de pertinence

5.7 Exemple 6 — Ajouter un produit au panier

5.7.1 Objectif

Vérifier que l'ajout d'un produit au panier fonctionne **réellement** : l'API répond, le compteur augmente et le produit figure dans le panier. Cet exemple reprend la situation qui a motivé la supervision applicative (chapitre 1 du guide d'utilisation et de supervision) : un site qui s'affiche correctement, mais sur lequel il est impossible d'ajouter un produit au panier.

5.7.2 Prérequis

- Paramètres `baseUrl` et `produit` (par défaut `P100`).
- Un **produit de test** convenu avec le métier : toujours disponible, sans incidence sur les stocks réels ni sur les statistiques de vente.

5.7.3 Scénario scripté

Fichier `exemples/puppeteer/scenarios/06-ajout-panier.mjs` :

```javascript
// Exemple 6 — Ajouter un produit au panier.
```

```
// Le navigateur démarre avec un profil vierge : le panier est donc vide au départ.
// Paramètres : baseUrl, produit (défaut P100).

export default async function scenario({ page, step, params, expect }) {
  const produit = params.produit || 'P100';
  let articlesAvant = 0;

  await step('Ouvrir la fiche produit', async () => {
    await page.goto(`${params.baseUrl}/produit/${produit}`, { waitUntil: 'domcontentloaded' });
    await expect.visible(page, '[data-testid=add-to-cart]');
    articlesAvant = Number(await page.$eval('[data-testid=cart-count]', (el) => el.textContent));
  });

  await step('Ajouter le produit au panier', async () => {
    // Le bouton déclenche un appel POST à l'API du panier ; on attend sa réponse.
    const [reponse] = await Promise.all([
      page.waitForResponse((r) => r.url().includes('/api/panier') && r.request().method() ===
    'POST'),
      page.locator('[data-testid=add-to-cart]').click(),
    ]);
    expect.equal(reponse.status(), 200, "Code HTTP de l'API du panier");
    await expect.visible(page, '[data-testid=add-to-cart-message]');
  });

  await step('Vérifier le compteur du panier', async () => {
    // Attente d'une condition calculée dans la page plutôt que d'un délai fixe.
    await page.waitForFunction(
      (avant) => Number(document.querySelector('[data-testid=cart-count]').textContent) === avant +
    1,
      {},
      articlesAvant,
    );
  });

  await step('Vérifier le contenu du panier', async () => {
    await page.goto(`${params.baseUrl}/panier`, { waitUntil: 'domcontentloaded' });
    await expect.visible(page, `[data-testid=cart-line][data-product-id=${produit}]`);
  });
}
```

5.7.4 Explications

- **Lecture de l'état initial.** Le nombre d'articles est lu avant l'ajout. Le profil du navigateur est vierge, et le panier est donc vide en principe, mais le scénario ne fait pas cette hypothèse.
- **Attente d'une condition.** `page.waitForFunction()` attend que le compteur ait augmenté de un. La valeur `articlesAvant` est transmise en argument, car le code de la fonction s'exécute dans la page (§ 4.10.4).
- **Vérification du résultat, pas du message.** Le message « Produit ajouté » est vérifié, puis le panier est ouvert pour s'assurer que la ligne du produit existe. Seule cette dernière vérification prouve que l'ajout a été enregistré.

5.7.5 Version déclarative

Fichier `exemples/puppeteer/scenarios/06-ajout-panier.json` :

```json
{
  "id": "demo-ajout-panier",
  "name": "Boutique Démo : ajout d'un produit au panier",
  "type": "browser",
  "schedule": { "interval": "10m" },
  "timeout": "45s",
  "browser": {
    "mode": "steps",
    "params": { "baseUrl": "http://localhost:8080" },
    "steps": [
      { "name": "Ouvrir la fiche produit", "action": "goto", "url": "${param:baseUrl}/produit/P100"
    },
      { "name": "Ajouter le produit au panier", "action": "click", "selector": "[data-testid=add-to-
    cart]" },
      { "name": "Vérifier la confirmation d'ajout", "action": "assertText", "selector": "[data-
    testid=add-to-cart-message]", "operator": "contains", "value": "ajouté" },
      { "name": "Vérifier le compteur du panier", "action": "assertText", "selector": "[data-
    testid=cart-count]", "operator": "eq", "value": "1" },
      { "name": "Ouvrir le panier", "action": "goto", "url": "${param:baseUrl}/panier" },
      { "name": "Vérifier le contenu du panier", "action": "assertVisible", "selector": "[data-
    testid=cart-line][data-product-id=P100]" }
    ]
  }
}
```

5.7.6 Résultat attendu

État `OK`, quatre étapes (six en version déclarative).

5.7.7 Précautions

- Le panier créé est associé à une session **anonyme**, détruite à la fin de l'exécution avec le profil du navigateur. Selon l'application, ces paniers abandonnés peuvent toutefois s'accumuler côté serveur ou apparaître dans les statistiques. Signalez leur origine aux équipes métier.
- Sur une application réelle, identifiez le trafic de la sonde (§ 6.12) pour qu'il soit exclu des indicateurs commerciaux.

5.7.8 Erreurs courantes

Symptôme	Cause probable	Correction
Échec à « Vérifier le compteur du panier » alors que le message s'affiche	Ajout non enregistré : véritable anomalie fonctionnelle	Escalader vers l'équipe applicative (voir exemple 8)
Échec intermittent du compteur	Le compteur est mis à jour par un autre mécanisme (rechargement)	Vérifier directement le contenu du panier
Produit indisponible	Le produit de test a été retiré ou est en rupture	Convenir d'un produit de test permanent

5.8 Exemple 7 — Vérifier un parcours de commande en environnement de test

5.8.1 Objectif

Vérifier le tunnel de commande de bout en bout : connexion, panier, adresse, livraison, récapitulatif et confirmation. La commande de test créée est ensuite **annulée**. C'est le scénario le plus complet, et aussi le plus sensible, puisqu'il crée des données dans l'application.

> **Avertissement** — Un scénario qui passe commande **agit réellement** sur l'application. Ne l'exécutez **jamais** contre un environnement de production sans un accord formel des responsables métier et techniques, un moyen de paiement de test fourni par le prestataire de paiement, un produit et un compte de test identifiés, et une procédure d'annulation éprouvée. **Ne déclenchez jamais de paiement réel.** En cas de doute, limitez le scénario au récapitulatif, sans confirmation (§ 5.8.7).

5.8.2 Prérequis

- Environnement de **test** (recette) de l'application, identifiable de façon fiable.
- Paramètres `baseUrl`, `environnementAttendu` (`recette`), `autoriserCommande` (`oui`) et `produit`.
- Secrets `BOUTIQUE_USER` et `BOUTIQUE_PASSWORD` d'un compte de test dédié.
- Mode de paiement **simulé** sur l'environnement visé.
- Moyen d'**annuler** les commandes de test, dans le parcours ou par un traitement côté serveur convenu avec les équipes.

5.8.3 Scénario scripté

Fichier `exemples/puppeteer/scenarios/07-parcours-commande.mjs` :

JAVASCRIPT

```
// Exemple 7 — Vérifier un parcours de commande, EN ENVIRONNEMENT DE TEST UNIQUEMENT.
//
// Ce scénario crée une commande. Il ne doit JAMAIS être exécuté contre un site de
// production sans un accord formel du métier, un moyen de paiement de test et une
// procédure d'annulation. Deux garde-fous indépendants protègent son exécution :
//   1. le paramètre « autoriserCommande » doit valoir explicitement « oui » ;
//   2. la page doit déclarer l'environnement attendu (ici « recette ») ;
// faute de quoi le scénario s'arrête AVANT toute action irréversible.
// La commande de test créée est ensuite annulée (nettoyage).
//
// Paramètres : baseUrl, environnementAttendu (« recette »), autoriserCommande (« oui »), produit.
// Secrets : BOUTIQUE_USER, BOUTIQUE_PASSWORD (compte de test dédié).

export default async function scenario({ page, step, secrets, params, expect, log }) {
  const produit = params.produit || 'P100';
  const cliquerEtAttendre = (selecteur) =>
    Promise.all([page.waitForNavigation({ waitUntil: 'domcontentloaded' }),
    page.locator(selecteur).click()]);

  await step("Contrôler l'autorisation et l'environnement", async () => {
    // Garde-fou n° 1 : autorisation explicite portée par la configuration du contrôle.
    expect.equal(params.autoriserCommande, 'oui', 'Commande de test non autorisée par la
    configuration');
    await page.goto(`${params.baseUrl}/`, { waitUntil: 'domcontentloaded' });
    // Garde-fou n° 2 : l'application doit s'identifier comme environnement de test.
    const environnement = await page.$eval('meta[name=environnement]', (m) => m.content).catch(() =>
    null);
    expect.equal(environnement, params.environnementAttendu, "Environnement de l'application");
```

```
    await expect.visible(page, '[data-testid=env-banner]');
  });

  await step('Se connecter avec le compte de test', async () => {
    await page.goto(`${params.baseUrl}/connexion`, { waitUntil: 'domcontentloaded' });
    await page.locator('[data-testid=login-email]').fill(secrets.BOUTIQUE_USER);
    await page.locator('[data-testid=login-password]').fill(secrets.BOUTIQUE_PASSWORD);
    await cliquerEtAttendre('[data-testid=login-submit]');
    await expect.visible(page, '[data-testid=account-menu]');
  });

  await step('Constituer le panier', async () => {
    await page.goto(`${params.baseUrl}/produit/${produit}`, { waitUntil: 'domcontentloaded' });
    await Promise.all([
      page.waitForResponse((r) => r.url().includes('/api/panier') && r.request().method() ===
    'POST'),
      page.locator('[data-testid=add-to-cart]').click(),
    ]);
    await page.goto(`${params.baseUrl}/panier`, { waitUntil: 'domcontentloaded' });
    await expect.visible(page, `[data-testid=cart-line][data-product-id=${produit}]`);
  });

  await step('Renseigner l\'adresse', async () => {
    await cliquerEtAttendre('[data-testid=checkout-start]');
    await expect.visible(page, '[data-testid=checkout-address]');
    // Les champs sont préremplis par le compte de test ; on valide l'étape.
    await cliquerEtAttendre('[data-testid=checkout-next]');
  });

  await step('Choisir le mode de livraison', async () => {
    await expect.visible(page, '[data-testid=checkout-delivery]');
    await page.locator('[data-testid=delivery-pickup]').click();
    await cliquerEtAttendre('[data-testid=checkout-next]');
  });

  await step('Vérifier le récapitulatif', async () => {
    await expect.visible(page, '[data-testid=checkout-summary]');
    // Dernier contrôle avant l'action irréversible : le paiement doit être simulé.
    await expect.text(page, '[data-testid=payment-mode]', 'simulé');
  });

  let numero;
  await step('Confirmer la commande de test', async () => {
    await cliquerEtAttendre('[data-testid=order-confirm]');
    await expect.visible(page, '[data-testid=order-confirmation]');
    numero = await page.$eval('[data-testid=order-number]', (el) => el.textContent.trim());
    // Un numéro de commande de test doit respecter le format convenu.
    expect.ok(/^TEST-\d+$/.test(numero), `Numéro de commande inattendu : ${numero}`);
    log.info(`Commande de test créée : ${numero}`);
  });

  await step('Annuler la commande de test', async () => {
    // Nettoyage : la supervision ne doit pas laisser de commandes en attente de traitement.
    await cliquerEtAttendre('[data-testid=order-cancel]');
    await expect.text(page, '[data-testid=order-status]', 'annulee', { exact: true });
    log.info(`Commande de test annulée : ${numero}`);
  });
}
```

5.8.4 Explications

Deux garde-fous indépendants protègent l'exécution. Ils sont vérifiés **avant toute action** :

1. **Autorisation explicite** : le paramètre `autoriserCommande` doit valoir exactement « oui ». Un contrôle copié d'un autre environnement, ou mal configuré, s'arrête immédiatement.

2. **Identification de l'environnement** : l'application déclare son environnement dans une balise `<meta name="environnement">`. Le scénario compare cette valeur à `environnementAttendu`. Si l'adresse `baseUrl` désigne par erreur la production, la valeur lue ne correspond pas, et le scénario s'arrête sans rien créer.

Un troisième contrôle intervient juste avant l'action irréversible : l'étape « Vérifier le récapitulatif » s'assure que le mode de paiement affiché est **simulé**.

Nettoyage. La commande créée est annulée dans la dernière étape. Son numéro est journalisé, pour être retrouvé si l'annulation échoue.

> ! **Attention** — Ce nettoyage par le scénario **n'est pas garanti** : si l'exécution est interrompue entre la confirmation et l'annulation, la commande de test reste enregistrée, et aucune exécution suivante ne l'annulera.

Une interruption peut provenir d'un timeout, d'un redémarrage de la sonde, d'une coupure réseau ou d'un arrêt du navigateur ; chaque lancement crée sa propre commande. Pour ce parcours :

- prévoyez un **nettoyage indépendant côté environnement de recette**, convenu avec ses responsables : par exemple, une purge planifiée des commandes dont le numéro commence par `TEST-` et qui ont été créées par le compte de test dédié, quel que soit leur statut ;
- configurez le contrôle **sans nouvelle tentative** (`"retry": { "attempts": 0 }`, § 3.5 du guide d'utilisation et de supervision) : après un échec ambigu, par exemple un timeout survenu après la confirmation, une nouvelle tentative créerait une seconde commande, alors que la première a peut-être été enregistrée : le numéro étant attribué par l'application à chaque création, rien ne lui permet de reconnaître la répétition (§ 3.5.2 du guide d'utilisation et de supervision) ;
- surveillez le nombre de commandes `TEST-…` non annulées (ligne « Commandes de test non annulées » du § 5.8.8).

> ✓ **Conseil** — Demandez aux équipes de développement d'exposer l'environnement de manière fiable : balise `meta`, en-tête HTTP ou point d'information dédié. Un garde-fou fondé sur l'adresse seule est fragile : une adresse de recette peut être redirigée, ou recopiée par erreur dans un contrôle de production.

5.8.5 Résultat attendu

État `OK`, huit étapes. Le journal contient le numéro de la commande de test créée, puis annulée :

```json
"log": [
  { "niveau": "INFO", "message": "Commande de test créée : TEST-00001" },
  { "niveau": "INFO", "message": "Commande de test annulée : TEST-00001" }
]
```

Si l'environnement ne correspond pas, ici avec `environnementAttendu=production`, le scénario s'arrête à la première étape, **sans aucune action sur l'application** :

```json
{
  "status": "CRITICAL",
  "error": {
    "type": "assertion_failed",
    "step": "Contrôler l'autorisation et l'environnement",
    "message": "Environnement de l'application : attendu « production », obtenu « recette »"
  }
}
```

5.8.6 Ordonnancement

Ce scénario est long et sollicite plusieurs services. Planifiez-le à une fréquence modérée, par exemple toutes les 15 à 30 minutes, et de préférence dans un plan d'exécution qui commence par les scénarios de connexion et d'ajout au panier avec l'option `stopOnFailure` (chapitre 6 du guide d'utilisation et de supervision). Si la connexion échoue, il est inutile de tenter la commande : l'alerte porte alors directement sur la cause.

5.8.7 Variantes

- **Parcours sans confirmation** : arrêtez le scénario après « Vérifier le récapitulatif ». Le tunnel est vérifié jusqu'au paiement, sans créer de commande. C'est la variante recommandée lorsqu'aucun environnement de test n'est disponible.
- **Paiement de test d'un prestataire** : utilisez les moyens de paiement de test documentés par votre prestataire, exclusivement sur l'environnement de test associé.

5.8.8 Erreurs courantes

Symptôme	Cause probable	Correction
Échec au premier contrôle	Paramètres absents ou incorrects ; balise d'environnement absente	Vérifier la configuration ; convenir d'un marqueur d'environnement
Échec à « Annuler la commande de test »	Bouton d'annulation absent pour ce statut de commande	Annuler manuellement la commande journalisée ; prévenir les équipes métier
Commandes de test non annulées qui s'accumulent	Scénario interrompu entre la confirmation et l'annulation (timeout, incident)	Mettre en place une purge côté serveur des commandes `TEST-…` ; surveiller leur nombre
Le scénario dépasse son timeout	Parcours long sur un Raspberry Pi 5	Porter le timeout du contrôle à deux à trois fois la durée normale (§ 3.4 du guide d'utilisation et de supervision)

5.9 Exemple 8 — Détecter une erreur fonctionnelle

5.9.1 Objectif

Détecter une anomalie **invisible pour une supervision technique** : toutes les réponses HTTP valent 200, la page affiche « Produit ajouté au panier », et pourtant le résultat métier est faux. Le panier peut être vide, ou son total incohérent. Le scénario **recalcule** le résultat attendu et le compare à ce que l'application affiche. Il vérifie aussi l'absence d'erreurs JavaScript et de réponses serveur en erreur pendant le parcours.

5.9.2 Prérequis

- Paramètres `baseUrl`, `produit` (par défaut `P300`) et `quantite` (par défaut 2).
- Paramètre `panne` : **réservé à la démonstration**, il active une panne simulée de l'application fictive. Il n'a pas d'équivalent dans une application réelle.

5.9.3 Scénario scripté

Fichier `exemples/puppeteer/scenarios/08-detection-erreur-fonctionnelle.mjs` :

```javascript
// Exemple 8 — Détecter une erreur fonctionnelle invisible pour une supervision technique.
//
// Toutes les réponses HTTP de ce parcours valent 200, et l'application affiche
```

```
// « Produit ajouté au panier ». Pourtant, le résultat métier peut être faux :
// panier vide, ou total incohérent. Ce scénario recalcule le résultat attendu et
// le compare à ce que l'application affiche.
//
// Pour la démonstration, l'application fictive simule deux pannes :
//   --param panne=panier  : l'ajout est confirmé mais rien n'est enregistré ;
//   --param panne=prix    : le total du panier ignore la quantité.
// Sans paramètre « panne », le scénario doit réussir.
//
// Paramètres : baseUrl, produit (défaut P300), quantite (défaut 2), panne (démonstration).

export default async function scenario({ page, step, params, expect, log }) {
  const produit = params.produit || 'P300';
  const quantite = Number(params.quantite || 2);
  const suffixe = params.panne ? `?panne=${encodeURIComponent(params.panne)}` : '';

  // Indices techniques collectés pendant tout le parcours.
  const erreursJavaScript = [];
  const reponsesEnErreur = [];
  page.on('pageerror', (e) => erreursJavaScript.push(e.message));
  page.on('response', (r) => { if (r.status() >= 500) reponsesEnErreur.push(`${r.status()}
    ${r.url()}`); });

  let prixUnitaire;

  await step('Ouvrir la fiche produit', async () => {
    await page.goto(`${params.baseUrl}/produit/${produit}${suffixe}`, { waitUntil:
    'domcontentloaded' });
    // Le prix de référence est lu dans un attribut technique stable, pas dans le texte formaté.
    prixUnitaire = Number(await page.$eval('[data-testid=product-price]', (el) =>
    el.dataset.value));
    expect.ok(prixUnitaire > 0, `Prix unitaire invalide : ${prixUnitaire}`);
  });

  // Nom d'étape littéral et stable : il ne doit pas dépendre d'une valeur variable.
  await step('Ajouter le produit au panier', async () => {
    // fill() remplace le contenu existant du champ quantité.
    await page.locator('[data-testid=quantity]').fill(String(quantite));
    await Promise.all([
      page.waitForResponse((r) => r.url().includes('/api/panier') && r.request().method() ===
    'POST'),
      page.locator('[data-testid=add-to-cart]').click(),
    ]);
    await expect.text(page, '[data-testid=add-to-cart-message]', 'ajouté');
  });

  await step('Vérifier le panier', async () => {
    await page.goto(`${params.baseUrl}/panier${suffixe}`, { waitUntil: 'domcontentloaded' });
    // Le message de succès ne suffit pas : la ligne doit exister avec la bonne quantité.
    await expect.visible(page, `[data-testid=cart-line][data-product-id=${produit}]`);
    const quantiteAffichee = Number(
      await page.$eval(`[data-testid=cart-line][data-product-id=${produit}] [data-testid=cart-line-
    qty]`, (el) => el.textContent),
    );
    expect.equal(quantiteAffichee, quantite, 'Quantité dans le panier');
  });

  await step('Vérifier la cohérence du total', async () => {
    const totalAffiche = Number(await page.$eval('[data-testid=cart-total]', (el) =>
    el.dataset.value));
    const totalAttendu = Math.round(prixUnitaire * quantite * 100) / 100;
    log.info(`Total attendu ${totalAttendu}, total affiché ${totalAffiche}`);
    expect.equal(totalAffiche, totalAttendu, 'Total du panier');
  });

  await step('Vérifier l\'absence d\'erreurs techniques', async () => {
```

```
    expect.ok(erreursJavaScript.length === 0, `Erreurs JavaScript : ${erreursJavaScript.join(' |
    ')}`);
    expect.ok(reponsesEnErreur.length === 0, `Réponses serveur en erreur : ${reponsesEnErreur.join('
    | ')}`);
  });
}
```

5.9.4 Explications

- **Valeur de référence.** Le prix unitaire est lu dans un attribut technique (`data-value`), et non dans le texte affiché (« 89,50 € »), dont le format dépend de la langue.
- **Calcul indépendant.** Le total attendu est calculé par le scénario (prix × quantité), puis comparé au total affiché. C'est cette comparaison qui révèle l'anomalie.
- **Indices techniques.** Le scénario s'abonne, avant toute navigation, aux événements `pageerror` (erreurs JavaScript non gérées) et `response` (réponses HTTP). Il vérifie en dernière étape qu'aucune erreur n'a été collectée. Le moteur de scénarios collecte aussi ces informations pour le diagnostic. Ici, le scénario en fait un **critère d'échec** explicite.
- **Nom d'étape stable.** L'étape s'appelle « Ajouter le produit au panier », et non « Ajouter 2 produits » : la quantité est un paramètre, et le nom ne doit pas varier (§ 4.5.4).

5.9.5 Résultats attendus

Sans panne simulée, l'état est `OK`, avec cinq étapes réussies.

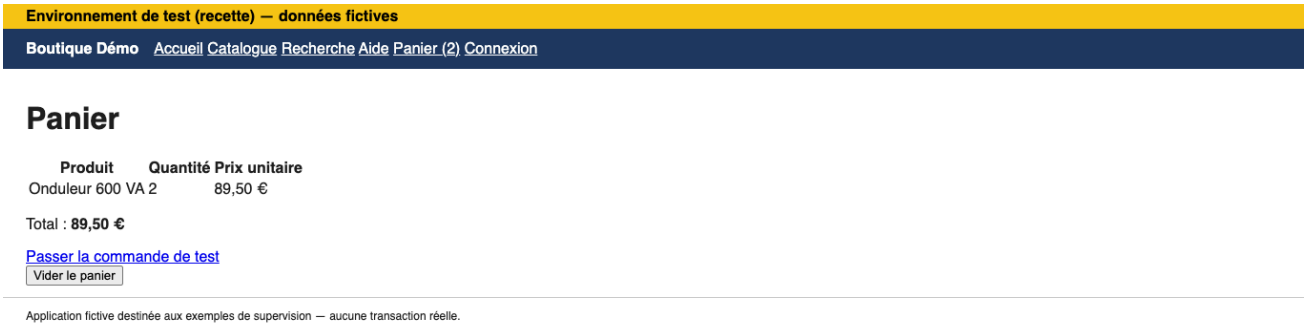


Figure 5-2 — Capture produite par l'exemple 8 avec la panne « prix » de la Boutique Démo : le total du panier est incohérent

Avec la panne `prix` (`--param panne=prix`), le total ignore la quantité. Le scénario échoue à la bonne étape, avec un message qui donne immédiatement la nature du problème. Résultat obtenu avec le lanceur local :

```json
{
  "checkId": "08-detection-erreur-fonctionnelle",
  "status": "CRITICAL",
  "durationMs": 600,
  "error": {
    "type": "assertion_failed",
    "step": "Vérifier la cohérence du total",
    "message": "Total du panier : attendu « 179 », obtenu « 89.5 »"
  },
  "steps": [
    { "index": 0, "name": "Ouvrir la fiche produit", "status": "OK", "durationMs": 74 },
    { "index": 1, "name": "Ajouter le produit au panier", "status": "OK", "durationMs": 507 },
    { "index": 2, "name": "Vérifier le panier", "status": "OK", "durationMs": 17 },
    { "index": 3, "name": "Vérifier la cohérence du total", "status": "CRITICAL", "durationMs": 2 }
  ],
  "artifacts": [
    { "name": "failure.png", "kind": "screenshot", "bytes": 39086 },
    { "name": "console.json", "kind": "console", "bytes": 226 }
  ]
}
```

Avec la panne `panier` (`--param panne=panier`), l'application confirme l'ajout mais n'enregistre rien. Le scénario échoue à l'étape « Vérifier le panier » :

```json
{
  "status": "CRITICAL",
  "error": {
    "type": "assertion_failed",
    "step": "Vérifier le panier",
    "message": "Élément [data-testid=cart-line][data-product-id=P300] non visible après 10000 ms"
  }
}
```

Dans les deux cas, toutes les réponses HTTP valaient 200 : aucun contrôle technique n'aurait détecté ces anomalies.

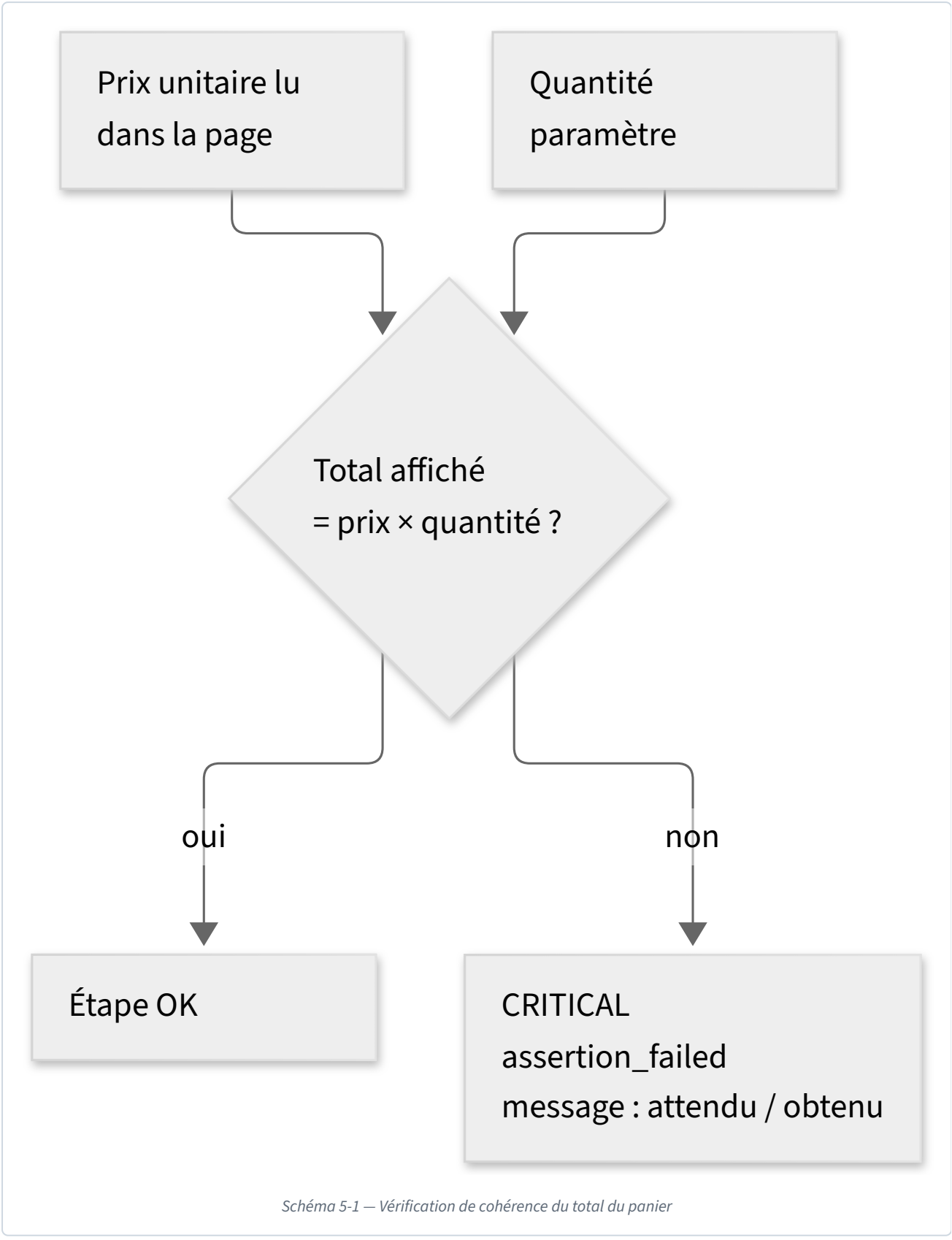


Schéma 5-1 — Vérification de cohérence du total du panier

5.9.6 Variantes

- **Cohérence entre pages** : comparez le total du panier au montant du récapitulatif de commande.

- **Règles métier** : frais de port offerts au-delà d'un montant, remise appliquée pour un code promotionnel de test, taxes correctement calculées.
- **Indicateurs métier** : pour des règles portant sur des volumes, par exemple le nombre de commandes de la journée, préférez un contrôle de base de données (chapitre 7), plus léger qu'un scénario navigateur.

5.9.7 Erreurs courantes

Symptôme	Cause probable	Correction
Échec sur l'arrondi	Comparaison de nombres décimaux sans arrondi	Arrondir au centime avant comparaison (comme dans l'exemple)
Échec à « Vérifier l'absence d'erreurs techniques » sans impact visible	Erreur JavaScript d'un script tiers (mesure d'audience)	Bloquer le domaine tiers (`blockDomains`) ou filtrer les erreurs par origine
Différence de format	Lecture d'un texte formaté au lieu d'une valeur technique	Lire un attribut `data-*` convenu avec les développeurs

5.10 Synthèse des essais

Les exemples ont été exécutés sur poste de travail avec le lanceur local, Puppeteer 25.18.1 et Chrome for Testing 157, contre l'application de démonstration. Ces essais portent sur l'environnement du poste de travail ; sur la sonde, qui utilise le Chromium des dépôts Raspberry Pi OS, chaque scénario se vérifie par une exécution d'essai (§ 4.16). Les cas d'échec ont été provoqués volontairement pour vérifier que chaque scénario échoue à la bonne étape, avec un message exploitable.

Exécution	État obtenu	Étape en échec	Type d'erreur
Exemples 1 à 8, versions scriptées	`OK`	—	—
Exemples 1 à 6, versions déclaratives (dont 3 bis)	`OK`	—	—
Exemple 5, terme sans résultat	`CRITICAL`	Vérifier les résultats	`assertion_failed`
Exemple 3, mot de passe erroné	`CRITICAL`	Vérifier l'accès à l'espace client	`assertion_failed`
Exemple 3, secret absent	`UNKNOWN`	Avant la première étape	`secret_missing`
Exemple 7, environnement attendu « production »	`CRITICAL`	Contrôler l'autorisation et l'environnement	`assertion_failed`
Exemple 8, panne `prix`	`CRITICAL`	Vérifier la cohérence du total	`assertion_failed`
Exemple 8, panne `panier`	`CRITICAL`	Vérifier le panier	`assertion_failed`
Exemple 4, timeout global de 300 ms	`CRITICAL`	Étape en cours à l'expiration	`timeout`
Exemple 1, application arrêtée	`CRITICAL`	Ouvrir la page d'accueil	`connection_refused`

5.11 Adapter les exemples à vos applications

Pour transformer un exemple en scénario de supervision de votre application :

1. **Formalisez l'intention** : ce que le scénario doit prouver, l'étape la plus critique, le résultat attendu, le compte de test et les données de test.

2. **Choisissez le mode** : déclaratif si le parcours est linéaire, scripté sinon (§ 4.1.1).
3. **Identifiez les sélecteurs** dans les outils de développement de Chrome. Demandez des attributs `data-testid` pour les éléments critiques (§ 6.1).
4. **Remplacez les valeurs propres à l'environnement** par des paramètres (`baseUrl`, produit, terme) et les identifiants par des secrets.
5. **Ajoutez des assertions** à chaque étape importante, avec des messages explicites.
6. **Testez sur poste de travail**, en succès **et** en échec.
7. **Déployez** sur la sonde, désactivé, puis lancez une exécution d'essai (§ 4.16).
8. **Définissez la planification et les seuils** : fréquence, timeout, nouvelles tentatives, `warningDuration`.
9. **Documentez** le scénario : intention, responsables de l'application, compte de test, procédure en cas d'alerte.

Le chapitre 6 rassemble les bonnes pratiques qui garantissent la fiabilité de ces scénarios dans la durée.

Chapitre 6 — Bonnes pratiques des scénarios

Un scénario de supervision n'a de valeur que s'il est **fiable** : il doit échouer lorsque le service est dégradé, et seulement dans ce cas. Un scénario qui produit de fausses alertes finit par être ignoré, et un scénario qui ne détecte rien donne une fausse assurance. Ce chapitre rassemble les pratiques qui garantissent cette fiabilité dans la durée, sur le matériel contraint qu'est un Raspberry Pi 5, et dans le respect de la sécurité des applications supervisées.

Chaque section se termine par une courte liste de règles. Le § 6.17 les reprend sous forme de liste de contrôle, à appliquer avant toute mise en service.

6.1 Utiliser des sélecteurs robustes

Les sélecteurs sont la première cause de **faux positifs**, c'est-à-dire d'alertes déclenchées alors que l'application fonctionne. Lorsqu'une page évolue (nouvelle mise en forme, réorganisation d'un formulaire, changement de libellé), un sélecteur fragile cesse de correspondre et le scénario échoue.

L'ordre de préférence est le suivant (§ 4.7.1) :

1. **attribut dédié aux tests** : `[data-testid=add-to-cart]` ;
2. **identifiant stable** : `#login-submit`, s'il n'est pas généré automatiquement ;
3. **nom d'accessibilité** : `::-p-aria(Ajouter au panier)` ;
4. **texte visible** : `::-p-text(Ajouter au panier)`, en sachant qu'il dépend des libellés et de la langue ;
5. **attribut fonctionnel** : `input[name=email]` ;
6. **XPath relatif**, en dernier recours.

À proscrire :

- les sélecteurs **positionnels** (`div:nth-child(3) > ul > li > a`) et les XPath **absolus** (`/html/body/div[2]/table[1]/tbody/tr[3]/td[1]/a`) ;
- les **classes de style** (`.btn-blue`), qui changent à chaque refonte graphique ;
- les **identifiants générés** par les frameworks (`#ember1234`, `#mui-17`), qui changent d'une version à l'autre, voire d'un chargement à l'autre.

> **Information** — La génération précédente recommandait d'utiliser systématiquement des chemins XPath obtenus par « Copy XPath ». Cette recommandation est abandonnée : ces chemins absolus décrivent la position d'un élément dans la page, et cassent à la moindre évolution de la structure.

Un bon sélecteur est aussi **unique**. S'il correspond à plusieurs éléments, l'action porte sur le premier, qui n'est pas forcément celui que vous visez. Vérifiez-le dans la console des outils de développement :

```javascript
// Dans la console de Chrome : nombre d'éléments correspondant au sélecteur
document.querySelectorAll('[data-testid=add-to-cart]').length
```

Règles

- Privilégier `data-testid` et le demander aux équipes de développement (§ 6.16).
- Ne jamais utiliser de sélecteur positionnel ni de XPath absolu.

- Vérifier l'unicité de chaque sélecteur d'action.

6.2 Écrire des assertions explicites

Une action réussie ne prouve pas que le service fonctionne. Un clic sur « Se connecter » réussit même lorsque l'authentification échoue. C'est l'**assertion** qui suit qui fait du scénario un contrôle.

Principes :

- **une assertion à chaque étape importante**, qui vérifie l'effet attendu de l'étape ;
- **vérifier le résultat métier**, pas l'accusé de réception : le contenu du panier plutôt que le message « Produit ajouté » (chapitre 5, exemple 8) ;
- **des messages qui permettent le diagnostic** : « Total du panier : attendu 179, obtenu 89,5 » plutôt que « Erreur » ;
- **comparer des valeurs techniques** (attributs `data-*`) plutôt que des textes formatés, qui dépendent de la langue, de la devise et de la mise en forme ;
- **tolérer ce qui varie légitimement** : un nombre minimal de résultats plutôt qu'un nombre exact, une expression régulière plutôt qu'un texte complet.

JAVASCRIPT

```
// Faible : prouve seulement que le bouton existait
await page.locator('[data-testid=add-to-cart]').click();

// Robuste : prouve que le produit est réellement dans le panier
await page.goto(`${params.baseUrl}/panier`, { waitUntil: 'domcontentloaded' });
await expect.visible(page, `[data-testid=cart-line][data-product-id=${produit}]`);
```

Règles

- Terminer chaque étape importante par une assertion.
- Vérifier le résultat métier, pas seulement l'affichage d'un message.
- Rédiger des messages d'échec qui indiquent la valeur attendue et la valeur obtenue.

6.3 Maîtriser les délais

Trois niveaux de délais s'appliquent : le timeout global du contrôle, le délai de navigation et le délai d'attente des éléments (§ 4.10.5). Les réglages recommandés sont les suivants.

Réglage	Recommandation
Timeout du contrôle (`timeout`)	Deux fois la durée normale du parcours, et au moins 30 secondes
Délai d'une attente particulière	Laisser la valeur par défaut. L'augmenter seulement pour une opération connue pour être lente, et seulement pour cette attente
Seuil d'avertissement (`thresholds.warningDuration`)	Environ 1,5 fois la durée habituelle, mesurée sur plusieurs jours
Nouvelles tentatives (`retry`)	Une tentative après 30 à 60 secondes pour les parcours importants

Le délai ne doit pas servir à **masquer une lenteur**. Un parcours qui prend habituellement 6 secondes et qui en prend soudain 25 révèle une dégradation. Avec un timeout de 60 secondes et un seuil d'avertissement à 9 secondes, cette dégradation est signalée par l'état `WARNING`, sans fausse alerte `CRITICAL`.

Pensez aussi aux performances de la sonde. Un Raspberry Pi 5 exécute un navigateur plus lentement qu'un poste de travail récent. Mesurez la durée des scénarios **sur la sonde**, avec des exécutions d'essai, avant de fixer les seuils.

Règles

- Fixer les seuils à partir de durées mesurées sur la sonde, pas sur le poste de travail.
- Signaler la lenteur par `warningDuration`, et réserver l'échec aux dysfonctionnements.
- Ne jamais allonger les délais globalement pour faire disparaître des échecs intermittents sans en comprendre la cause.

6.4 Attendre des événements plutôt que temporiser

Une pause fixe est la cause la plus fréquente des scénarios instables (§ 4.10.1). Remplacez-la systématiquement par l'attente de l'événement qui marque la fin de l'opération :

Situation	Au lieu d'une pause, attendre…
Après un clic qui charge une page	La navigation, armée avant le clic (`Promise.all`)
Après un clic qui affiche un contenu	L'élément de ce contenu (`waitForSelector`, `expect.visible`)
Après une action qui appelle une API	La réponse de cette API (`waitForResponse`)
Pendant un indicateur de chargement	La disparition de l'indicateur (`{ hidden: true }`)
Après une mise à jour d'un compteur	La condition (`waitForFunction`)

Le mode déclaratif applique ce principe automatiquement : chaque action attend son élément, et chaque assertion attend sa condition.

Règles

- Aucune pause fixe dans un scénario.
- Armer les attentes de navigation et de réponse **avant** l'action qui les déclenche.

6.5 Gérer les erreurs sans les masquer

Le moteur de scénarios intercepte les exceptions, marque l'étape en échec, collecte les éléments de diagnostic et détermine l'état du contrôle (§ 4.12). Le script n'a donc presque jamais besoin de `try/catch`.

- **Ne jamais intercepter une erreur pour l'ignorer.** Un échec masqué devient un succès apparent, ce qui est le pire résultat pour un contrôle de supervision.
- **Intercepter uniquement l'absence normale d'un élément facultatif**, comme un bandeau de consentement ou une fenêtre promotionnelle (§ 4.8.6), avec un délai court.
- **Enrichir un message** si nécessaire, en levant une nouvelle erreur plus explicite.
- **Tester le cas d'échec** de chaque scénario avant sa mise en service (§ 4.15.3).

Règles

- Laisser remonter les exceptions.
- N'intercepter que les éléments facultatifs, avec un délai court.
- Vérifier que le scénario échoue à la bonne étape lorsque l'application est défaillante.

6.6 Limiter la consommation de mémoire et de processeur

Sur un Raspberry Pi 5, le navigateur est le composant le plus coûteux de la sonde. La mémoire disponible détermine le nombre de scénarios exécutables simultanément (un par défaut) et la stabilité de l'ensemble. Les pratiques suivantes réduisent la consommation de chaque exécution :

Pratique	Effet
Bloquer les domaines tiers inutiles à la vérification (`blockDomains`) : mesure d'audience, publicité, widgets sociaux, cartes	Moins de requêtes, moins de scripts exécutés, moins de mémoire ; parcours plus rapide et plus stable (§ 6.11)
Garder la taille de fenêtre par défaut (`viewport`) ; ne pas simuler un écran 4K	Moins de mémoire pour le rendu et les captures
Limiter le nombre de pages parcourues à ce qui est nécessaire pour l'intention du scénario	Exécution plus courte
Fermer les fenêtres secondaires ouvertes par le scénario dès qu'elles ne sont plus utiles	Libère la mémoire pendant l'exécution
Ne pas accumuler de grandes quantités de données dans le script (contenus de pages entiers, listes de milliers d'éléments)	Mémoire du processus Node.js
Espacer les exécutions : toutes les 5 à 15 minutes plutôt que toutes les minutes	Moins de charge cumulée, plus de marge pour les nouvelles tentatives
Regrouper les scénarios d'une même application dans un plan d'exécution	Évite les exécutions concurrentes et les échéances qui se chevauchent (chapitre 6 du guide d'utilisation et de supervision)

Surveillez les indicateurs de santé de la sonde : mémoire, température, retard du planificateur (chapitre 10). Un retard croissant signale que la capacité d'exécution est saturée.

Règles

- Bloquer les ressources tierces non nécessaires.
- Espacer les exécutions des scénarios.
- Surveiller le retard du planificateur et la mémoire de la sonde.

6.7 Gérer les sessions et les comptes de test

Chaque exécution démarre avec un profil vierge (§ 3.3.3). Chaque scénario qui a besoin d'une session doit donc s'authentifier. Ce choix est délibéré : la connexion fait partie du service vérifié, et aucune session ne subsiste sur la sonde.

Le **compte de test** utilisé doit être :

- **dédié** à la supervision, nommé de façon explicite (exemple fictif : `test-supervision@example.com`) et jamais partagé avec des personnes ;
- **limité en droits** au strict nécessaire au parcours : pas de droits d'administration, pas d'accès à des données réelles de clients ;
- **exempté du second facteur** si nécessaire, uniquement avec l'accord de l'équipe sécurité et des mesures compensatoires (restriction aux adresses de la sonde, mot de passe long) ;
- **connu des équipes** : support, sécurité et métier doivent savoir que ses connexions régulières sont normales ;

- **protégé contre le verrouillage** : une politique de verrouillage après plusieurs échecs peut bloquer la supervision. Convenez d'un seuil adapté et limitez les nouvelles tentatives.

Lorsque plusieurs sondes supervisent la même application depuis des sites différents, prévoyez un compte de test **par sonde**. Une connexion de l'une ne fermera pas la session de l'autre, et l'origine d'une anomalie sera identifiable.

Règles

- Un compte de test dédié, aux droits minimaux, connu des équipes.
- Une politique de verrouillage et des nouvelles tentatives compatibles.
- Un compte par sonde lorsque plusieurs sondes visent la même application.

6.8 Isoler les scénarios

Un scénario doit pouvoir s'exécuter **seul**, à tout moment, quel que soit le résultat des autres scénarios et des exécutions précédentes.

- **Pas de dépendance à l'état laissé par un autre scénario.** Un scénario de commande ne doit pas supposer que le scénario d'ajout au panier a rempli le panier : il constitue lui-même son panier (chapitre 5, exemple 7).
- **Pas de dépendance à l'ordre d'exécution.** Les plans d'exécution enchaînent des scénarios pour éviter des tests inutiles (`stopOnFailure`), mais chaque scénario reste autonome (chapitre 6 du guide d'utilisation et de supervision).
- **Des données de test indépendantes.** Si deux scénarios modifient des données, ils utilisent des données distinctes (produit, compte), afin de ne pas interférer.
- **Pas d'état stocké entre deux exécutions.** Le moteur ne conserve rien d'une exécution à l'autre, et le script ne doit pas chercher à contourner ce principe.

Règles

- Chaque scénario constitue lui-même les données dont il a besoin.
- Aucun scénario ne dépend du résultat d'un autre.

6.9 Nettoyer les ressources et les données

Le moteur de scénarios nettoie tout ce qui concerne la sonde : navigateur, processus, profil, cache (§ 4.14). Le script n'a pas à s'en préoccuper.

En revanche, les **données créées dans l'application supervisée** relèvent de la responsabilité du scénario et des équipes qui l'exploitent :

- **annuler ou supprimer** ce que le scénario crée (commande de test, réservation, message), dans le scénario lui-même lorsque c'est possible (chapitre 5, exemple 7) ;
- **journaliser l'identifiant** de toute donnée créée, pour pouvoir la retrouver si le nettoyage échoue ;
- **prévoir un nettoyage côté serveur** des données de test résiduelles, par exemple une purge périodique des commandes `TEST-…` mise en place par l'équipe applicative ;
- **préférer les parcours sans effet de bord** lorsqu'ils suffisent : arrêter le tunnel de commande au récapitulatif vérifie l'essentiel sans rien créer.

Règles

- Ne jamais fermer le navigateur dans le script : le moteur s'en charge.
- Annuler ce que le scénario crée, et journaliser les identifiants créés.

- Prévoir une purge côté serveur des données de test.

6.10 Protéger les secrets

Les identifiants des comptes de test donnent accès à vos applications. Ils sont traités comme des secrets (chapitre 9) :

- **jamais dans le code** d'un script ni dans un scénario déclaratif : toujours `secrets.NOM` ou `${secret:NOM}` ;
- **jamais dans les paramètres** (`params`), qui sont visibles par tous les utilisateurs pouvant consulter le contrôle ;
- **jamais dans les journaux** : n'écrivez pas `log.info(secrets.BOUTIQUE_PASSWORD)`. La sonde masque les valeurs connues des secrets, mais ce masquage est une protection de dernier recours, pas une permission ;
- **jamais dans un message d'assertion** : ne comparez pas un texte de la page à un secret ;
- **déclarés au plus juste** : un contrôle ne déclare que les secrets dont il a besoin. Le moteur ne lui transmet que ceux-là ;
- **renouvelés régulièrement**, selon la politique de l'entreprise, et immédiatement en cas de doute. Le remplacement d'un secret sur la sonde ne nécessite aucune modification des scénarios.

Sur poste de travail, le lanceur local lit les secrets dans des variables d'environnement. Ne les écrivez pas dans des fichiers versionnés, et ne les passez pas en clair sur la ligne de commande.

Règles

- Aucune valeur sensible dans le code, les paramètres ou les journaux.
- Déclarer uniquement les secrets nécessaires.
- Renouveler les secrets selon la politique de l'entreprise.

6.11 Maîtriser les ressources tierces

Une page moderne charge de nombreuses ressources de **domaines tiers** : mesure d'audience, publicité, réseaux sociaux, discussion en ligne, tests A/B. Elles sont rarement utiles à la vérification du parcours et posent trois problèmes :

- elles ralentissent le scénario et faussent la mesure de performance de **votre** application ;
- elles consomment de la mémoire sur la sonde ;
- leurs propres défaillances (script en erreur, service lent) peuvent faire échouer un scénario alors que votre application fonctionne.

La propriété `blockDomains` du contrôle bloque les requêtes vers les domaines indiqués et leurs sous-domaines :

```json
"browser": {
  "mode": "script",
  "scriptRef": "boutique-panier@sha256:9a7e31c4d2b8",
  "blockDomains": ["stats.example.net", "ads.example.org", "chat.example.com"]
}
```

À l'inverse, **ne bloquez pas** les services tiers qui font partie du service rendu : fournisseur d'identité, module de paiement, réseau de diffusion des ressources de l'application, cartographie d'une page de recherche de magasins. Leur défaillance est une défaillance réelle du service, que le scénario doit détecter.

Règles

- Bloquer les domaines de mesure d'audience, de publicité et de widgets non essentiels.

- Ne jamais bloquer un service tiers qui participe au parcours vérifié.

6.12 Identifier le trafic de la sonde

Le trafic de la sonde doit être reconnaissable par les équipes de l'application, pour trois raisons :

- **statistiques** : les exécutions régulières ne doivent pas fausser l'audience ni les indicateurs commerciaux ;
- **sécurité** : les connexions répétées du compte de test ne doivent pas être prises pour une attaque ;
- **diagnostic** : les journaux de l'application doivent permettre de retrouver les requêtes d'une exécution.

Moyens disponibles :

Moyen	Mise en œuvre
Suffixe d'identification du navigateur	Propriété `userAgentSuffix` du contrôle (par exemple `SupervisionSonde`)
Adresse IP de la sonde	Communiquée aux équipes ; adresse fixe recommandée (chapitre 5 du guide d'installation et d'administration)
Compte de test dédié	Exclu des statistiques par son identifiant
Refus des cookies non essentiels	Le bandeau de consentement est refusé par le scénario (§ 4.8.6)

Règles

- Rendre le trafic de la sonde identifiable, et en informer les équipes de l'application.
- Faire exclure ce trafic des statistiques d'audience et des indicateurs métier.

6.13 Captures d'écran, traces et données personnelles

En cas d'échec, la sonde enregistre automatiquement une capture d'écran et les éléments de diagnostic (console, requêtes en erreur). Ces **artefacts** sont précieux pour comprendre un incident (chapitre 10). Ils peuvent aussi contenir des informations sensibles.

- **Ne réalisez pas de capture manuelle** dans les scripts : la capture automatique en cas d'échec suffit, et elle est gérée avec une rétention et un quota (chapitre 8 du guide d'installation et d'administration).
- **Utilisez des données fictives.** Avec un compte de test sans données réelles, les captures ne montrent aucune donnée personnelle.
- **Évitez les pages sensibles** lorsqu'elles ne sont pas nécessaires au parcours : données bancaires, documents d'identité, informations de santé.
- **Limitez l'accès** aux résultats détaillés aux personnes qui en ont besoin, à l'aide des rôles (chapitre 9).
- **Conservez les artefacts le temps nécessaire**, sans plus. La rétention par défaut est courte (7 jours).

> ! **Attention** — Les champs de mot de passe sont masqués à l'écran par le navigateur, et donc sur les captures. En revanche, une page peut afficher en clair une donnée personnelle, comme une adresse ou un numéro de client. Un compte de test sans données réelles est la meilleure protection.

Les mesures ci-dessus réduisent le risque sans le supprimer. Gardez à l'esprit les limites suivantes :

Élément	Ce qui est protégé	Ce qui ne l'est pas
Masquage automatique	Les **valeurs connues** des secrets déclarés par le contrôle sont remplacées dans les journaux, la console du navigateur et les messages d'erreur	Une donnée personnelle affichée par la page ; un secret **réaffiché** par l'application sous une autre forme (tronqué, encodé, reformaté), ou un secret qui n'est pas déclaré dans le contrôle. Une capture d'écran n'est jamais masquée
Capture d'écran automatique	Elle n'est produite qu'en cas d'échec	Elle montre la page telle qu'elle était affichée au moment de l'échec, données comprises
Journal de console	Messages d'erreur de la page, secrets connus masqués	Les messages écrits par l'application elle-même peuvent contenir des identifiants, des adresses électroniques ou des jetons de session
Rôles	Seuls les rôles autorisés consultent et téléchargent les artefacts (chapitre 9)	Un artefact téléchargé, puis copié dans un ticket ou une messagerie, échappe à la rétention et aux rôles de la sonde

En conséquence :

- **Désactivez la capture** pour un contrôle qui traverse des pages sensibles qu'il ne peut pas éviter : `"captureOnFailure": false` dans la section `browser` (§ 4.2). Le diagnostic s'appuie alors sur l'étape en échec, le message d'erreur et le journal de console ; compensez par des noms d'étapes et des messages d'assertion précis (§ 6.2).
- **N'écrivez pas de donnée de la page dans les messages** du script (`log.info`, messages d'assertion) lorsqu'elle peut être personnelle : préférez un comptage ou un indicateur (« 3 lignes dans le panier ») à la valeur elle-même.
- **Respectez la rétention** : ne conservez pas de copie locale des artefacts au-delà de l'analyse de l'incident.
- **Contrôlez tout artefact avant de le transmettre**, en particulier au support : ouvrez la capture et le journal, puis retirez ou masquez tout élément sensible. Il en va de même pour l'archive de diagnostic, qui contient des journaux et peut contenir des captures (§ 9.9 du guide d'installation et d'administration). Le canal de transfert vers le support et la durée de conservation par l'éditeur sont définis dans sa politique de support, selon les conditions de votre contrat, et indiqués sur le portail du support (`https://support-synovex.incitypark.fr`).

Règles

- Pas de capture manuelle dans les scripts.
- Comptes et données de test exclusivement.
- Accès aux artefacts limité par les rôles ; rétention courte.
- Capture désactivée (`captureOnFailure: false`) sur les parcours qui traversent des pages sensibles.
- Contrôle du contenu de tout artefact ou archive avant transmission.

6.14 Fiabilité dans la durée

Un scénario est fiable lorsqu'il produit le même résultat dans les mêmes conditions. Les échecs **intermittents**, qui disparaissent à l'exécution suivante sans intervention, minent la confiance des équipes. Ils doivent être traités comme de vrais défauts du scénario ou de l'application.

Cause d'instabilité	Remède
Pause fixe ou attente mal armée	Attendre l'événement adéquat, armé avant l'action (§ 6.4)
Élément facultatif (bandeau, fenêtre promotionnelle) qui recouvre un bouton	Fermer l'élément s'il est présent (§ 4.8.6)

Cause d'instabilité	Remède
Données de test qui changent (produit retiré, terme sans résultat)	Données de référence convenues et pérennes
Service tiers instable	Bloquer s'il n'est pas essentiel ; sinon, l'échec est légitime
Lenteur ponctuelle du réseau	Une nouvelle tentative (`retry.attempts: 1`)
Sonde saturée	Espacer les exécutions, regrouper en plans d'exécution (§ 6.6)

Les **nouvelles tentatives** absorbent les incidents réellement ponctuels. Elles ne doivent pas servir à masquer un scénario instable. Si un scénario n'atteint l'état `OK` qu'après une nouvelle tentative de façon régulière, analysez la cause.

Suivez dans le temps, pour chaque scénario, le taux d'exécutions en échec et le nombre de nouvelles tentatives, à partir de l'historique de la sonde ou de la télémétrie exportée (chapitre 10 et chapitre 2 du guide d'intégration et d'observabilité). Un scénario sain présente un taux d'échec nul en dehors des incidents réels.

Règles

- Traiter chaque échec intermittent comme un défaut à corriger.
- Limiter les nouvelles tentatives à une ou deux.
- Suivre le taux d'échec de chaque scénario.

6.15 Maintenir les scénarios au rythme des évolutions applicatives

Les applications évoluent : nouveaux libellés, formulaires réorganisés, étapes ajoutées au tunnel. Les scénarios doivent suivre. La génération précédente le rappelait déjà : les scripts doivent être maintenus à jour à chaque évolution de l'application web.

Organisation recommandée :

1. **Désigner un responsable** pour chaque scénario, côté exploitation ou côté équipe applicative.
2. **Intégrer les scénarios au cycle de mise en production.** Toute évolution touchant un parcours supervisé prévoit la mise à jour du scénario correspondant.
3. **Tester le scénario sur l'environnement de recette** avant la mise en production, avec une sonde ou le lanceur local.
4. **Pendant une intervention planifiée**, désactiver les scénarios concernés (§ 6.8 du guide d'utilisation et de supervision) plutôt que de laisser des alertes être ignorées, et penser à les réactiver ensuite.
5. **Après la mise en production**, lancer une exécution d'essai, puis réactiver.
6. **Versionner** les scénarios scriptés dans un gestionnaire de code, et déployer par références immuables (§ 4.17).

> ✓ **Conseil** — Un échec `element_not_found` ou `assertion_failed` qui apparaît juste après une mise en production signale le plus souvent une évolution de la page, et non une panne. Comparez la capture d'écran au comportement attendu avant de déclarer un incident (chapitre 10).

Règles

- Un responsable par scénario.
- Mise à jour des scénarios intégrée au cycle de mise en production.
- Suspension planifiée pendant les interventions.

6.16 Collaborer avec les équipes de développement

Les scénarios les plus fiables sont conçus **avec** les équipes qui développent l'application. Quelques aménagements, peu coûteux pour elles, améliorent considérablement la supervision :

Demande	Bénéfice
Ajouter des attributs `data-testid` stables aux éléments des parcours critiques	Sélecteurs robustes, beaucoup moins de faux positifs
Exposer des valeurs techniques (`data-value="89.50"`) à côté des valeurs formatées	Assertions de calcul fiables
Fournir un compte de test, des produits de test et un moyen de paiement simulé	Parcours transactionnels sans risque
Exposer de façon fiable l'environnement (balise `meta` , en-tête)	Garde-fous des scénarios transactionnels
Purger côté serveur les données de test	Pas d'accumulation de données parasites
Prévenir des évolutions touchant les parcours supervisés	Scénarios mis à jour avant la mise en production
Exclure le trafic de la sonde des statistiques	Indicateurs métier non faussés
Instrumenter l'application avec OpenTelemetry	Corrélation possible entre les étapes du scénario et les traces côté serveur (chapitre 2 du guide d'intégration et d'observabilité)

Ces attributs et aménagements bénéficient aussi aux tests automatisés des équipes de développement. Ils constituent généralement une demande bien accueillie.

Règles

- Associer les équipes de développement à la conception des scénarios critiques.
- Formaliser les données de test et les attributs `data-testid` dans un document partagé.

6.17 Liste de contrôle avant la mise en service d'un scénario

#	Vérification	Référence
1	L'intention du scénario est documentée (ce qu'il prouve, résultat attendu, responsable)	§ 6.15
2	Le mode retenu est le plus simple possible (déclaratif si suffisant)	§ 4.1.1
3	Les sélecteurs sont stables (`data-testid` de préférence), uniques, sans position ni XPath absolu	§ 6.1
4	Chaque étape importante se termine par une assertion portant sur le résultat métier	§ 6.2
5	Les messages d'assertion indiquent la valeur attendue et la valeur obtenue	§ 6.2
6	Aucune pause fixe ; les attentes de navigation et de réponse sont armées avant l'action	§ 6.4
7	Aucun `try/catch` qui masque une erreur	§ 6.5
8	Les noms d'étapes sont littéraux, stables et explicites	§ 4.5.4
9	Aucune valeur sensible dans le code, les paramètres ou les journaux ; secrets déclarés au plus juste	§ 6.10
10	Le compte de test est dédié, limité en droits et connu des équipes	§ 6.7
11	Les données créées sont annulées ou purgées ; aucune action irréversible hors environnement de test	§ 6.9

#	Vérification	Référence
12	Les domaines tiers non essentiels sont bloqués	§ 6.11
13	Le trafic de la sonde est identifiable et exclu des statistiques	§ 6.12
14	Le scénario a été testé en succès **et** en échec sur poste de travail	§ 4.15
15	Une exécution d'essai sur la sonde est conforme ; la durée mesurée sur la sonde est connue	§ 4.16
16	Timeout, seuil d'avertissement et nouvelles tentatives sont fixés à partir de cette durée	§ 6.3
17	La fréquence est compatible avec la capacité de la sonde (un scénario à la fois par défaut)	§ 3.8.3
18	Les équipes destinataires des alertes savent comment réagir	Chapitre 10

Chapitre 7 — Tests d'API et de bases de données

7.1 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
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 (§ 7.6.8)	`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 du guide d'utilisation et de supervision) ou un dysfonctionnement purement fonctionnel.

7.2 Superviser une API REST avec le contrôle `http`

7.2.1 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 § 4.6 du guide d'utilisation et de supervision, 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**.

7.2.2 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.

7.2.3 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

Assertion	Défaillance détectée
`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

7.2.4 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 (`\"`).

7.2.5 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}$`, identifiant de type `^CMD-[0-9]+$`

Besoin	Opérateur	Exemple
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.

7.3 Enchaîner des requêtes avec le contrôle `api`

7.3.1 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 du guide d'utilisation et de supervision), 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 du guide d'utilisation et de supervision) se déclare pour l'ensemble du contrôle, dans `api.proxy`, et non requête par requête. De même, le paramètre commun `ipFamily` (§ 3.1.3 du guide d'utilisation et de supervision) s'applique à toutes les requêtes du contrôle ; les cibles IPv6 suivent les mêmes règles que pour le contrôle `http` (§ 4.1.3 du guide d'utilisation et de supervision).

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.

7.3.2 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	—

Champ	Obligatoire	Description	Défaut
`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.

7.3.3 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 (§ 8.2.1). 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 (§ 8.2.1). 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.

7.3.4 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.

7.3.5 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 9)
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

Symptôme	Cause probable	Action
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 (§ 8.2)
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

7.4 Superviser un web service SOAP

7.4.1 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.

7.4.2 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

7.4.3 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 du guide d'utilisation et de supervision) 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.

7.5 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 (§ 7.7)
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 10)
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.

7.6 Contrôles de bases de données (`database`)

7.6.1 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 du guide d'utilisation et de supervision).

7.6.2 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	—
`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.

7.6.3 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 du guide d'utilisation et de supervision, et § 4.1.3 du guide d'utilisation et de supervision)
`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 (§ 7.6.4)
`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 (§ 8.2.1)
`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` (§ 7.6.6). Défaut : `UNKNOWN`

Paramètre	Obligatoire	Description
`database.exportValues`	Non	Colonnes publiées comme indicateurs métier (10 au maximum, § 5.7 du guide d'utilisation et de supervision)

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`, § 7.6.4)	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 (§ 7.6.5), 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 (§ 7.6.5).

> ! **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.

7.6.4 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.

7.6.5 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, § 7.6.3).

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.

7.6.6 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`)
`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`

7.6.7 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 du guide d'utilisation et de supervision.

7.6.8 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`.

7.6.9 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.

7.6.10 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

7.6.11 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 9), 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.

7.7 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.

7.7.1 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.

7.7.2 Protéger les identifiants

- Stockez tous les mots de passe, clés d'API et secrets clients comme **secrets** de la sonde (chapitre 9). 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 (§ 8.2.1) : 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** (§ 7.6.4), et renouvelez leurs secrets selon la politique de l'entreprise.

7.7.3 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.

7.7.4 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.

Chapitre 8 — Sécurité des scripts et des identifiants

8.1 Scripts de scénarios

Les scénarios **scriptés** (édition Enterprise) sont des modules JavaScript Puppeteer, décrits au chapitre 4. Le script est téléversé séparément du contrôle, puis référencé par celui-ci.

8.1.1 Téléverser un script

Le téléversement d'un script se fait par l'interface d'administration ou par `POST /api/v1/scripts` (rôle `admin`, édition Enterprise). La procédure complète de mise en service d'un scénario scripté, du téléversement à l'activation du contrôle, figure au chapitre 4 (§ 4.16.2). Les règles propres à l'administration sont les suivantes :

- chaque requête transmet un seul module `.mjs`, avec le type de contenu `text/javascript`, dans la limite de 1 Mo. Les scénarios déclaratifs ne passent pas par cette ressource : leurs étapes figurent dans la définition du contrôle (`POST /api/v1/checks`, type `application/json`) ;
- la sonde valide le script, calcule son empreinte SHA-256 et renvoie une **référence** de la forme `nom@sha256:empreinte`, par exemple `boutique-connexion@sha256:4f1c9e2ab7d3` (empreinte abrégée dans les exemples de cette documentation : une référence réelle comporte les 64 caractères hexadécimaux de l'empreinte), à indiquer dans la propriété `browser.scriptRef` du contrôle ;
- le téléversement d'un script et toute création, modification ou suppression de `browser.scriptRef` dans un contrôle sont réservés au rôle `admin` : la même requête émise par un `operator` est refusée avec `403` (§ 6.4.1 du guide d'installation et d'administration et § 9.3.1).

Vérification : `GET /api/v1/scripts/{name}` liste les versions téléversées.

8.1.2 Pourquoi une référence par empreinte

La référence désigne **une version précise et immuable** du script. Téléverser une nouvelle version ne modifie donc pas les contrôles existants : le changement n'intervient qu'au moment où vous mettez à jour `scriptRef` dans le contrôle. Ce mécanisme apporte trois garanties :

- aucune modification involontaire d'un scénario en production ;
- la possibilité de revenir à la version précédente en rétablissant l'ancienne référence ;
- la traçabilité : le journal d'audit indique qui a téléversé quelle version, et quand.

> ✓ **Conseil** — Conservez vos scripts dans un système de gestion de versions (Git), et faites figurer l'empreinte renvoyée par la sonde dans le message de version. Le lien entre le code source et ce qui s'exécute sur la sonde reste ainsi vérifiable.

8.2 Secrets

8.2.1 Principe

Un **secret** est une valeur sensible dont un contrôle a besoin : mot de passe d'un compte de test, jeton d'une API, mot de passe d'une base de données. La sonde stocke les secrets **chiffrés** et applique quatre règles :

1. **Écriture seule** : un secret peut être créé, remplacé ou supprimé, mais sa valeur ne peut jamais être relue, ni par l'interface ni par l'API, quel que soit le rôle.
2. **Déclaration explicite** : un contrôle ne reçoit que les secrets qu'il déclare dans sa propriété `secrets`.
3. **Masquage** : la valeur d'un secret est masquée dans les messages d'erreur, les journaux, les sorties de plugins et la console du navigateur enregistrée.
4. **Destinations autorisées** : chaque secret porte une liste **obligatoire** d'hôtes autorisés, `allowedHosts` (noms d'hôtes ou motifs de noms, avec un port éventuel), fixée par un `admin`. La sonde n'injecte un secret que dans une requête, une connexion ou une page dont l'hôte figure dans cette liste.

Cette quatrième règle protège contre le détournement d'un secret : un contrôle non scripté modifié pour envoyer le secret vers un autre serveur ne le reçoit pas ; le cas des scénarios scriptés est précisé ci-dessous. Elle s'applique ainsi :

Situation	Comportement de la sonde
L'hôte de la requête, de la connexion ou de la page figure dans `allowedHosts`	Le secret est injecté
L'hôte n'y figure pas	Le secret n'est pas injecté ; l'exécution échoue
Redirection HTTP vers un **autre hôte**	Les en-têtes porteurs d'un secret (par exemple `Authorization`) ne sont **pas retransmis** à la destination de la redirection. Une redirection étant une nouvelle requête, la règle d'injection s'applique à sa destination pour **tout élément porteur d'un secret** (en-tête, corps renvoyé lors d'une redirection `307` ou `308`, paramètre d'adresse), et aucun n'est transmis à un hôte absent de la liste
Hôte inscrit avec un port (par exemple `db.example.internal:5432`)	Le secret n'est injecté que vers ce port
Changement de destination (URL ou hôte modifié dans le contrôle, navigation vers une autre page)	La vérification porte sur la destination de chaque requête, connexion ou page au moment de l'injection : un contrôle modifié pour viser un hôte absent de la liste ne reçoit pas le secret
Scénario scripté (`browser.mode=script`)	Le script reçoit la valeur dans l'objet `secrets` (chapitre 4) : la sonde ne peut pas garantir à elle seule que le code ne la transmet pas ailleurs. C'est l'une des raisons pour lesquelles ces scénarios sont réservés au rôle `admin`
Un `operator` crée ou modifie un contrôle	Il peut référencer un secret existant ; il ne peut ni lire sa valeur ni modifier `allowedHosts`
Import d'une configuration (§ 6.12.3 du guide d'installation et d'administration)	Les références aux secrets sont **revérifiées** : secret existant, déclaration dans le contrôle, hôtes autorisés
Import d'une archive antérieure à `allowedHosts` (secret sans liste)	Le secret est importé mais reste **inutilisable**, c'est-à-dire jamais injecté, tant qu'un `admin` n'a pas défini sa liste ; le compte rendu d'import le signale

Les scénarios scriptés restent réservés au rôle `admin` (§ 6.4.1 du guide d'installation et d'administration).

Les noms de secrets s'écrivent en majuscules, chiffres et tirets bas, et commencent par une lettre (`BOUTIQUE_PASSWORD`, `API_STOCK_TOKEN`).

8.2.2 Créer ou remplacer un secret

Objectif : enregistrer le mot de passe du compte de test de la boutique.

Prérequis : rôle `admin`.

Étapes :

1. Dans l'interface d'administration, ouvrez la gestion des secrets et créez le secret `BOUTIQUE_PASSWORD`, ou utilisez l'API, `PUT /api/v1/secrets/{name}`, avec un corps de la forme `{"value": …, "allowedHosts": [ … ]}`. La liste `allowedHosts` est obligatoire et contient au moins un hôte : ici, le seul site de la boutique de test (nom fictif). Pour une **création**, la requête porte l'en-tête `If-None-Match: *` (§ 9.1.5) : si un secret de même nom existe déjà, la sonde répond `412 Precondition Failed` au lieu de l'écraser. Exemple, la valeur étant lue sans affichage :

   ```bash
   read -rs VAL && printf '%s' "$VAL" \
     | jq -Rs '{value: ., allowedHosts: ["recette.boutique.example.com"]}' \
     | sapi -X PUT -H 'Content-Type: application/json' -H 'If-None-Match: *' \
         --data @- "$SONDE/api/v1/secrets/BOUTIQUE_PASSWORD"; unset VAL
   ```

2. Vérifiez sa présence : `GET /api/v1/secrets` (rôle `admin`) liste les noms et les métadonnées des secrets, jamais leurs valeurs. Ces métadonnées comprennent la date de rotation et la liste `allowedHosts` de chaque secret. Pour un seul secret, `GET /api/v1/secrets/{name}` (rôle `admin`) renvoie ses métadonnées, dont `allowedHosts`, et son `ETag`, nécessaires à sa rotation (§ 8.2.4) ; cette lecture ne renvoie jamais la valeur du secret.

Corps de requête correspondant (valeur fictive) :

```json
{ "value": "********", "allowedHosts": ["recette.boutique.example.com"] }
```

Résultat attendu : le secret est disponible pour les contrôles qui le déclarent, et uniquement vers les hôtes autorisés.

8.2.3 Utiliser un secret dans un contrôle

Un secret s'utilise en deux temps :

1. il est **déclaré** dans la liste `secrets` du contrôle ;
2. il est **référencé** à l'endroit voulu par la syntaxe `${secret:NOM}`. Dans un scénario scripté, il est lu dans l'objet `secrets` (chapitre 4).

```json
{
  "id": "api-stock",
  "name": "API de stock : article témoin",
  "type": "http",
  "schedule": { "interval": "2m" },
  "timeout": "10s",
  "secrets": ["API_STOCK_TOKEN"],
  "http": {
    "method": "GET",
    "url": "https://api.example.com/v1/stock/ARTICLE-TEMOIN",
    "headers": { "Authorization": "Bearer ${secret:API_STOCK_TOKEN}" },
    "assertions": [ { "kind": "status", "operator": "eq", "value": 200 } ]
  }
}
```

Dans cet exemple, le secret `API_STOCK_TOKEN` doit avoir été créé avec `"allowedHosts": ["api.example.com"]` : l'hôte de l'URL du contrôle figure dans la liste, le secret est donc injecté dans l'en-tête `Authorization`.

Un contrôle qui référence un secret non déclaré, ou un secret inexistant, est refusé à la création (`422`). La suppression d'un secret encore référencé par un contrôle est refusée (`409`).

8.2.4 Rotation

Remplacer la valeur d'un secret suffit : tous les contrôles qui l'utilisent emploient la nouvelle valeur dès leur prochaine exécution. Le remplacement utilise la même requête qu'à la création (§ 8.2.2), mais avec l'en-tête `If-Match` portant l'`ETag` du secret, et non `If-None-Match: *` (§ 9.1.5). Chaque `PUT` exige en outre la liste `allowedHosts` en même temps que la valeur. Lisez donc d'abord le secret par `GET /api/v1/secrets/{name}` (rôle `admin`) : la réponse contient ses métadonnées, dont la liste `allowedHosts`, et l'en-tête `ETag`, mais jamais sa valeur. Renvoyez la liste à l'identique, sauf modification voulue, pour ne pas restreindre ni étendre par mégarde les destinations autorisées. La date de rotation est enregistrée et visible dans la liste des secrets.

Exemple de rotation, la nouvelle valeur étant lue sans affichage (nom de secret fictif) :

```bash
# 1. Lire les métadonnées et l'ETag du secret (la valeur n'est jamais renvoyée)
ENTETES=$(mktemp)
META=$(sapi -D "$ENTETES" "$SONDE/api/v1/secrets/BOUTIQUE_PASSWORD")
ETAG=$(awk 'tolower($1)=="etag:" {print $2}' "$ENTETES" | tr -d '\r'); rm -f "$ENTETES"

# 2. Remplacer la valeur en renvoyant à l'identique la liste allowedHosts relue
read -rs VAL && printf '%s' "$VAL" \
  | jq -Rs --argjson hotes "$(jq -c '.allowedHosts' <<<"$META")" \
      '{value: ., allowedHosts: $hotes}' \
  | sapi -X PUT -H 'Content-Type: application/json' -H "If-Match: $ETAG" \
      --data @- "$SONDE/api/v1/secrets/BOUTIQUE_PASSWORD"; unset VAL
```

Si un autre administrateur a modifié le secret entre la lecture et le remplacement, la sonde répond `412 Precondition Failed` : relisez le secret et recommencez.

Procédure de rotation sans interruption d'un mot de passe de compte de test :

1. Prévenez les équipes concernées et choisissez un moment hors période critique.
2. Modifiez le mot de passe du compte de test dans l'application supervisée.
3. Remplacez immédiatement la valeur du secret sur la sonde (exemple ci-dessus).
4. Lancez une exécution immédiate des contrôles concernés (§ 9.3.5) pour confirmer le bon fonctionnement.

Entre les étapes 2 et 3, une exécution peut échouer avec le type d'erreur `auth_failed`. Si l'application le permet, désactivez temporairement les contrôles concernés pendant l'opération.

8.2.5 Supprimer un secret

`DELETE /api/v1/secrets/{name}` supprime un secret, avec l'en-tête `If-Match` portant l'`ETag` renvoyé par `GET /api/v1/secrets/{name}` (§ 9.1.5). La suppression est refusée (`409 Conflict`) tant qu'un contrôle le référence.

> **Avertissement** — Utilisez des **comptes de test dédiés** à la supervision, dotés des droits strictement nécessaires, et jamais des comptes nominatifs ou des comptes à privilèges. Pour les bases de données, privilégiez un compte en **lecture seule**. Une sonde est un équipement installé dans votre réseau : limiter ce que ses identifiants permettent limite les conséquences d'une compromission.

8.3 Identifiants, secrets et comptes techniques

8.3.1 Ce que la sonde doit connaître

Pour superviser vos applications en profondeur, la sonde a besoin d'identifiants : comptes de test pour se connecter aux applications web, comptes de lecture pour interroger les bases de données, jetons pour appeler les API, clés d'API des plateformes d'observabilité. Ces informations sont des **secrets**.

La sonde protège les secrets de la manière suivante :

Mesure	Effet
Stockage chiffré	Les secrets sont chiffrés sur la microSD
Écriture seule	Un secret enregistré ne peut jamais être relu, ni par l'interface ni par l'API ; il peut seulement être remplacé ou supprimé
Déclaration par contrôle	Chaque contrôle déclare les secrets qu'il utilise ; seuls ceux-ci lui sont transmis lors de l'exécution
Référencement	Les configurations ne contiennent jamais la valeur d'un secret, mais une référence de la forme `${secret:NOM}`
Masquage	Les valeurs des secrets sont masquées dans les journaux et les messages d'erreur
Exclusion des exports	Les exports de configuration excluent les secrets, ou les chiffrent par une phrase de passe

La procédure de création, de rotation et de suppression des secrets est décrite au chapitre 9.

> ! **Attention** — Sans composant matériel de sécurité, la clé qui protège les secrets réside sur la même microSD que les secrets eux-mêmes. Le chiffrement protège contre la copie de la base de données, mais **pas contre le vol de la microSD entière**. La sécurité physique de la sonde (§ 5.11 du guide d'installation et d'administration) et le moindre privilège des comptes (§ 8.3.2) restent donc essentiels.

> ! **Attention** — Les captures d'écran produites en cas d'échec d'un scénario (chapitre 10) peuvent afficher des données présentes à l'écran. Les champs de mot de passe sont masqués par le navigateur, mais une page peut réafficher une donnée sensible. Utilisez des comptes de test sans données réelles.

8.3.2 Principe du moindre privilège

Le **principe du moindre privilège** consiste à n'accorder à un compte que les droits strictement nécessaires à sa fonction. Appliqué à la sonde, il limite les conséquences d'une fuite d'identifiants ou d'une erreur de configuration.

Usage	Recommandation
Comptes de test applicatifs (scénarios navigateur, API)	Comptes **dédiés à la supervision**, nominativement identifiables (par exemple `test-supervision@example.com`), sans droits d'administration, sans accès à des données réelles, sans moyen de paiement réel
Comptes de bases de données	Compte **en lecture seule**, limité aux tables ou vues nécessaires ; idéalement, une vue dédiée qui expose uniquement les indicateurs à contrôler
Jetons d'API des applications	Portée minimale (lecture seule), durée de validité limitée
Clés d'API des plateformes d'observabilité	Clés limitées à l'**ingestion** des données (par exemple, pour Dynatrace, un jeton disposant uniquement des droits d'ingestion des signaux exportés)
Comptes d'accès à la sonde	Rôle minimal adapté à chaque usage (§ 5.9.1 du guide d'installation et d'administration)

Exemple de compte en lecture seule pour un contrôle de base de données PostgreSQL (fictif) :

```sql
-- Compte de supervision en lecture seule, limité à une vue dédiée (exemple PostgreSQL)
CREATE ROLE supervision_ro LOGIN PASSWORD '********';
CREATE VIEW reporting.v_supervision_commandes AS
  SELECT count(*) AS nb_commandes_jour
  FROM ventes.commandes
  WHERE created_at >= current_date;
GRANT USAGE ON SCHEMA reporting TO supervision_ro;
GRANT SELECT ON reporting.v_supervision_commandes TO supervision_ro;
```

> **Information** — Exposer une vue plutôt que les tables évite de donner à la sonde l'accès aux données détaillées : elle ne lit que l'indicateur agrégé dont elle a besoin. Les contrôles de bases de données sont décrits au chapitre 7.

8.3.3 Comptes techniques dédiés

Créez des comptes techniques distincts pour chaque usage et chaque sonde lorsque c'est possible :

- un compte par application supervisée, plutôt qu'un compte partagé entre toutes ;
- un compte de proxy dédié à la sonde ;
- une clé d'API d'ingestion par plateforme et par site.

Vous pourrez ainsi révoquer un accès sans interrompre les autres, et identifier précisément l'origine des connexions dans les journaux de vos applications.

> **Conseil** — Informez les équipes applicatives de l'existence des comptes de test de la sonde. Elles éviteront de les désactiver lors d'une revue des comptes, et pourront exclure leur activité des statistiques métier.

8.3.4 Rotation des secrets

Renouvelez les secrets selon la politique de votre organisation, et immédiatement en cas de suspicion de compromission. La sonde conserve la date de dernière rotation de chaque secret (chapitre 9). Pour une rotation sans interruption de la supervision :

1. Créez le nouveau mot de passe dans l'application cible (ou un second jeton).
2. Remplacez la valeur du secret dans la sonde.
3. Déclenchez une exécution immédiate des contrôles concernés et vérifiez leur état.
4. Désactivez l'ancien mot de passe ou jeton dans l'application cible.

Chapitre 9 — Accès à l'API de la sonde

9.1 API REST : principes

9.1.1 Base et format

Élément	Règle
Adresse de base	`https://<adresse-de-la-sonde>/api/v1`
Version	Indiquée dans le chemin (`v1`). Une future version majeure coexistera au moins 12 mois avec la précédente
Format	JSON encodé en UTF-8 (`Content-Type: application/json`)
Dates	RFC 3339, en UTC. Exemple : `2025-11-19T09:15:02Z`
Durées dans les résultats	En millisecondes, dans des champs suffixés `Ms` (`durationMs`)
Durées dans la configuration	Chaînes `500ms` , `30s` , `5m` , `1h`
Identifiants	Choisis par vous pour les objets de configuration (`portail-accueil`) ; générés par la sonde (ULID) pour les exécutions
Description de l'API	Document OpenAPI 3.1 disponible à l'adresse `GET /api/v1/openapi.json`

Le document OpenAPI décrit toutes les ressources, leurs paramètres et leurs schémas. Il peut être importé dans un outil de test d'API ou servir à générer un client dans le langage de votre choix.

Deux adresses se situent hors de l'arborescence `/api/v1` :

Adresse	Usage	Authentification
`GET /health`	Vérifier que la sonde répond (vivacité). Ne renvoie aucune information sensible. Réponse `{"status": "OK"}` (code `200`) ; pendant l'**état de maintenance** qui ouvre l'activation d'une mise à jour, `{"status": "MAINTENANCE"}` (code `503`) (§ 8.4.9 du guide d'installation et d'administration)	Aucune
`GET /metrics`	Métriques au format Prometheus (chapitre 2 du guide d'intégration et d'observabilité)	Jeton de rôle `metrics`

9.1.2 Authentification

Chaque appel à l'API, hors `GET /api/v1/system/certificate` (téléchargement du certificat de la sonde, sans authentification), doit présenter un **jeton d'API** dans l'en-tête HTTP `Authorization` :

```http
GET /api/v1/checks HTTP/1.1
Host: sonde-paris-01.example.internal
Authorization: Bearer <jeton>
Accept: application/json
```

Le jeton détermine l'identité de l'appelant et son rôle (§ 6.4 du guide d'installation et d'administration). Un appel sans jeton, ou avec un jeton invalide, expiré ou révoqué, reçoit la réponse `401 Unauthorized` . Un appel authentifié mais non autorisé pour le rôle du jeton reçoit `403 Forbidden` .

> → **Évolution prévue** — L'authentification par certificat client (mTLS) n'est pas disponible actuellement : elle est prévue dans une version ultérieure, avec la console centrale (§ 6.14 du guide d'installation et d'administration).

9.1.3 Préparer son poste : certificat et jeton

Les exemples de ce chapitre utilisent l'outil en ligne de commande `curl`, disponible sur Linux, macOS et Windows. Ils suivent deux règles de sécurité :

- **le certificat de la sonde est toujours vérifié**, grâce à l'option `--cacert`. L'option `-k` (`--insecure`) désactive cette vérification et ne doit pas être utilisée ;
- **le jeton n'apparaît jamais dans la ligne de commande**, où il serait visible dans l'historique du shell et la liste des processus. Il est lu depuis un fichier protégé.

Objectif : préparer un poste Linux ou macOS pour appeler l'API de façon sûre.

Prérequis : `curl` et `openssl` installés ; un jeton d'API (§ 9.2) ; l'empreinte SHA-256 du certificat de la sonde.

Étapes :

1. Créez un répertoire de travail accessible à vous seul :

   ```bash
   mkdir -p "$HOME/.sva" && chmod 700 "$HOME/.sva"
   ```

2. Si la sonde utilise son certificat autosigné, récupérez-le, soit par l'adresse `GET /api/v1/system/certificate` (§ 4.4.4 du guide d'installation et d'administration), soit avec `openssl` :

   ```bash
   openssl s_client -connect sonde-paris-01.example.internal:443 \
     -servername sonde-paris-01.example.internal </dev/null 2>/dev/null \
     | openssl x509 -outform PEM > "$HOME/.sva/sonde-paris-01.pem"
   ```

3. Affichez son empreinte et **comparez-la** avec celle relevée lors de la mise en service :

   ```bash
   openssl x509 -in "$HOME/.sva/sonde-paris-01.pem" -noout -fingerprint -sha256
   ```

   Si les empreintes diffèrent, supprimez le fichier et n'allez pas plus loin.

   Si la sonde utilise un certificat émis par l'autorité de certification de l'entreprise, utilisez à la place le certificat de cette autorité.

4. Créez le fichier d'en-tête d'authentification `entete-auth` une fois, puis recréez-le de la même manière à chaque rotation du jeton. Le répertoire `$HOME/.sva` est d'abord rendu privé ; un éventuel ancien fichier est supprimé, car un masque de permissions (`umask`) ne s'applique qu'à la création et ne resserre pas les droits d'un fichier existant ; le nouveau fichier est créé vide en mode `600` **avant** que le jeton n'y soit écrit. La commande `read -rs`, propre à bash et à zsh, lit le jeton au clavier sans l'afficher ; les parenthèses limitent le masque et la variable à cette seule opération :

   ```bash
   (
     umask 077
     mkdir -p "$HOME/.sva" && chmod 700 "$HOME/.sva" &&
     rm -f "$HOME/.sva/entete-auth" &&
     : > "$HOME/.sva/entete-auth" && chmod 600 "$HOME/.sva/entete-auth" &&
     IFS= read -rs SONDE_TOKEN &&
     printf 'Authorization: Bearer %s\n' "$SONDE_TOKEN" > "$HOME/.sva/entete-auth"
   )
   ```

Si le jeton vous a été remis dans un fichier, par exemple `$HOME/.sva/jeton`, la forme suivante est équivalente (`printf` est une commande interne du shell : le jeton ne figure dans les arguments d'aucun processus) :

```bash
(
  umask 077
  mkdir -p "$HOME/.sva" && chmod 700 "$HOME/.sva" &&
  rm -f "$HOME/.sva/entete-auth" &&
  : > "$HOME/.sva/entete-auth" && chmod 600 "$HOME/.sva/entete-auth" &&
  printf 'Authorization: Bearer %s\n' "$(cat "$HOME/.sva/jeton")" > "$HOME/.sva/entete-auth"
)
```

Supprimez ensuite le fichier `jeton` s'il n'a plus d'usage.

5. Définissez des variables de confort pour la session :

```bash
SONDE=https://sonde-paris-01.example.internal
CA="$HOME/.sva/sonde-paris-01.pem"
```

> **Information** — Pourquoi cette méthode protège le jeton :
>
> - le répertoire `$HOME/.sva` est ramené aux droits `700` et l'ancien fichier `entete-auth` éventuel est supprimé, car `umask` ne modifie pas les droits d'un fichier existant ; le nouveau fichier naît vide avec les droits `600` (lecture et écriture pour vous seul), et le jeton n'y est écrit qu'ensuite : il n'existe aucun instant où le jeton serait lisible par un autre compte. Chaque commande est enchaînée par `&&` : si l'une échoue, le jeton n'est pas écrit ;
> - `read` et `printf` sont des **commandes internes** du shell (bash ou zsh ; l'option `-s` de `read` n'existe pas dans le shell POSIX `sh`) : elles ne créent aucun processus, et le jeton n'apparaît donc ni dans la liste des processus (`ps`), ni dans l'historique du shell. Dans la seconde forme, le processus `cat` ne reçoit que le **chemin** du fichier, jamais le jeton. Vous pouvez le vérifier avec `type printf`, qui doit répondre `printf is a shell builtin`. N'appelez pas `/usr/bin/printf` ou `env printf`, qui seraient des programmes externes ;
> - `curl -H @"$HOME/.sva/entete-auth"` lit ensuite l'en-tête dans le fichier : le jeton ne figure pas dans les arguments de `curl`.
>
> N'activez pas la trace du shell (`set -x`) pendant ces opérations : elle afficherait le jeton. Pour renouveler un jeton (§ 9.2), recréez simplement le fichier selon l'étape 4.

Vérification : interrogez l'état de la sonde.

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

Résultat attendu : une réponse JSON décrivant la sonde (version, édition, ressources, état des composants). Sa forme est définie par le schéma `system.schema.json`, et ses champs sont commentés au chapitre 10 (§ 7.9 du guide d'utilisation et de supervision).

> **Information** — `-H @fichier` demande à `curl` de lire les en-têtes dans un fichier. L'option `--fail-with-body` fait échouer la commande sur une erreur HTTP tout en affichant le détail renvoyé par la sonde. Ces options requièrent une version récente de `curl` : 7.55 ou ultérieure pour `-H @fichier`, 7.76 ou ultérieure pour `--fail-with-body`.

Dans la suite du chapitre, les commandes supposent ces variables définies. Pour alléger les exemples, nous utilisons la fonction suivante :

```bash
# Appel authentifié à l'API de la sonde, certificat vérifié
sapi() {
  curl --fail-with-body -sS --cacert "$CA" -H @"$HOME/.sva/entete-auth" \
       -H 'Accept: application/json' "$@"
}
```

L'outil `jq`, cité dans plusieurs exemples, met en forme et filtre les réponses JSON. Son usage est facultatif.

9.1.4 Codes de réponse et erreurs

Code	Signification	Cas typique
`200 OK`	Succès avec contenu	Lecture, modification
`201 Created`	Objet créé ; l'en-tête `Location` indique son adresse	Création d'un contrôle
`202 Accepted`	Action acceptée et lancée en arrière-plan ; l'en-tête `Location` indique où suivre son avancement	Exécution manuelle, test d'une destination d'export
`204 No Content`	Succès sans contenu	Suppression, activation
`304 Not Modified`	Ressource inchangée depuis la version que vous détenez	Lecture conditionnelle
`400 Bad Request`	Requête mal formée	JSON invalide
`401 Unauthorized`	Jeton absent, invalide, expiré ou révoqué	—
`403 Forbidden`	Rôle insuffisant, fonction non incluse dans l'édition, ou mot de passe actuel erroné lors du changement de son propre mot de passe. Ces cas se distinguent par le champ `type` de l'erreur : `/api/v1/problems/edition-feature-unavailable` pour une fonction absente de l'édition, `/api/v1/problems/current-password-invalid` pour un mot de passe actuel erroné (§ 6.4.2 du guide d'installation et d'administration)	Rôle `operator` qui téléverse un script ; contrôle `browser` créé sur une sonde Professional ; `PUT /api/v1/users/me/password` avec un mot de passe actuel erroné
`404 Not Found`	Objet inexistant	Identifiant erroné
`405 Method Not Allowed`	Méthode non admise par la ressource	`PUT /api/v1/settings/license`, section en lecture seule
`409 Conflict`	Opération incompatible avec l'état actuel	Création d'un identifiant déjà utilisé ; suppression d'un secret référencé par un contrôle
`410 Gone`	Ressource qui n'est plus disponible	Curseur de pagination expiré, plus de 24 heures après sa délivrance (§ 9.1.6)
`412 Precondition Failed`	Condition de l'en-tête `If-Match` ou `If-None-Match` non satisfaite (§ 9.1.5)	Objet modifié depuis votre dernière lecture ; création d'un secret qui existe déjà
`422 Unprocessable Content`	Contenu valide en JSON, mais non conforme au schéma ou aux règles	Champ obligatoire manquant, durée invalide

Code	Signification	Cas typique
`428 Precondition Required`	Ni `If-Match` ni `If-None-Match` dans une requête qui en exige un (§ 9.1.5)	`PUT` ou `DELETE` sans en-tête conditionnel
`429 Too Many Requests`	Limite de débit atteinte	Boucle d'automatisation trop rapide
`5xx`	Erreur interne de la sonde	À signaler au support si elle persiste

Les erreurs sont décrites au format **Problem Details** (RFC 9457), type de contenu `application/problem+json`. Exemple de refus de validation (fictif) :

```json
{
  "type": "/api/v1/problems/validation",
  "title": "Configuration invalide",
  "status": 422,
  "detail": "Le contrôle 'portail-accueil' ne respecte pas le schéma.",
  "errors": [
    { "pointer": "/schedule", "detail": "Indiquer 'interval' ou 'cron', mais pas les deux." },
    { "pointer": "/timeout", "detail": "Format de durée invalide : '15 secondes'." }
  ]
}
```

Le champ `type` est une adresse relative de la forme `/api/v1/problems/{code}`, servie par la sonde. Il permet à un outil de distinguer les causes d'un même code HTTP : un refus `403` lié à l'édition porte ainsi le type `/api/v1/problems/edition-feature-unavailable`, et un refus pour mot de passe actuel erroné le type `/api/v1/problems/current-password-invalid`, et non celui d'un rôle insuffisant.

9.1.5 Modifications concurrentes : ETag et If-Match

Plusieurs administrateurs, ou un administrateur et un outil d'automatisation, peuvent modifier le même objet. Pour éviter qu'une modification n'en écrase silencieusement une autre, la sonde applique un **contrôle de concurrence optimiste** :

1. Chaque lecture d'un objet de configuration renvoie un en-tête `ETag`, qui identifie sa version. Pour un secret ou un compte, cette lecture est la lecture individuelle : `GET /api/v1/secrets/{name}` (§ 8.2.4) ou `GET /api/v1/users/{id}` (§ 6.4.2 du guide d'installation et d'administration), réservées au rôle `admin`.
2. Toute modification (`PUT`, `PATCH`) ou suppression (`DELETE`) d'un objet existant doit présenter cette valeur dans l'en-tête `If-Match`.
3. Une ressource **créée** par `PUT`, comme un secret (`PUT /api/v1/secrets/{name}`, § 8.2.2), n'a pas encore d'`ETag` : la création présente l'en-tête `If-None-Match: *`, qui signifie « uniquement si la ressource n'existe pas ». Si elle existe déjà, la sonde répond `412 Precondition Failed` : il s'agit alors d'un remplacement, qui exige `If-Match`.
4. Une requête `PUT`, `PATCH` ou `DELETE` qui ne présente ni `If-Match` ni `If-None-Match` est refusée avec le code `428 Precondition Required`.
5. Si l'objet a changé entre-temps, la sonde refuse l'opération avec le code `412 Precondition Failed`. Relisez alors l'objet, réappliquez votre modification sur la version actuelle et renvoyez-la.

Portée. Cette règle s'applique aux ressources de configuration : contrôles, plans, scripts, destinations d'export, secrets, comptes et sections de paramètres (`/api/v1/settings/{section}`). Les requêtes d'action et d'import en écriture seule, qui ne modifient pas une ressource relue au préalable (`PUT /api/v1/system/maintenance`, `PUT /api/v1/system/certificate`, `PUT /api/v1/system/dot1x/credential`), n'en relèvent pas.

Exception. Le changement de son propre mot de passe, `PUT /api/v1/users/me/password` (§ 6.4.2 du guide d'installation et d'administration), n'exige ni `If-Match` ni `If-None-Match` : la preuve demandée est le mot de passe actuel, et non la version du compte. Cette requête ne reçoit donc jamais le code `428`.

```bash
# Lecture : afficher l'ETag renvoyé
sapi -D - -o /dev/null "$SONDE/api/v1/checks/portail-accueil" | grep -i '^etag'
```

9.1.6 Pagination et filtres

Les listes potentiellement longues, comme l'historique des exécutions, sont **paginées par curseur**. La réponse contient les éléments et, s'il en reste, un curseur vers la page suivante :

```json
{
  "items": [ { "id": "01M7SJ4Q8R2V6X9Z3B5D7F1H0K", "checkId": "portail-accueil", "status": "OK" } ],
  "nextCursor": "eyJpZCI6IjAxSkRCNUs4UjJYN1E5TjNWNFQ2VzhZMFpBIn0"
}
```

Pour obtenir la page suivante, renvoyez la même requête avec le paramètre `cursor`. Le paramètre `limit` fixe la taille de page (100 par défaut, 1 000 au maximum). Le curseur est une valeur opaque, encodée en base64url : il se transmet tel quel, sans encodage supplémentaire.

Cohérence de la pagination. Pour l'historique des exécutions, les règles suivantes s'appliquent :

Règle	Conséquence pour vous
Les éléments sont rangés selon un **ordre total** : date de début (`startedAt`), puis identifiant (`id`) pour départager deux exécutions de même date	Aucun élément n'est lu deux fois ni sauté du fait d'un ex æquo
Le curseur encode la **dernière position lue** et reste valide **24 heures**	Au-delà, la sonde répond `410 Gone` : relancez la lecture depuis la première page
Les exécutions enregistrées pendant la lecture n'apparaissent dans les pages suivantes que si leur date de début les place après la dernière position lue ; sinon, elles manquent à la lecture en cours	Pour un **export reproductible**, fixez toujours une borne `to` dans le passé : les exécutions commencées après elle sont exclues. Une exécution commencée avant `to` mais enregistrée plus tard reste dans la période : deux lectures successives ne donnent le même ensemble que si `to` est assez ancienne pour qu'aucune exécution de la période ne soit encore en cours (§ 7.3.2 du guide d'utilisation et de supervision), et sous réserve de la règle suivante
La sonde ne fournit **pas d'instantané** : les exécutions purgées par la rétention (§ 3.12 du guide d'utilisation et de supervision) pendant la lecture disparaissent des pages suivantes	Pour un export complet, choisissez une borne `from` nettement plus récente que la limite de rétention, ou lisez les périodes anciennes en priorité

```bash
# Exécutions en échec d'un contrôle sur une période donnée
sapi "$SONDE/api/v1/runs?checkId=portail-accueil&status=CRITICAL&from=2025-11-18T00:00:00Z&to=2025-
    11-19T00:00:00Z&limit=50"
```

9.2 Jetons d'API

9.2.1 Principe

Un **jeton d'API** est une chaîne aléatoire qui authentifie un outil ou un script auprès de l'API. Il possède :

- un **rôle** qui limite ses droits (§ 6.4.1 du guide d'installation et d'administration) ;
- un **nom** descriptif, qui indique son usage ;
- une **date d'expiration** facultative ;
- un **propriétaire**, l'utilisateur qui l'a créé.

La sonde ne conserve qu'une **empreinte** du jeton. La valeur en clair n'est affichée **qu'une seule fois**, au moment de la création : notez-la immédiatement dans un coffre-fort de mots de passe ou dans le gestionnaire de secrets de l'outil qui l'utilisera.

Un jeton est **immuable** : son nom, son rôle et sa date d'expiration ne se modifient plus après sa création, et l'API ne propose ni `PUT` ni `PATCH` sur les jetons. Pour changer l'un d'eux, ou pour renouveler un jeton qui arrive à expiration, créez un nouveau jeton, mettez à jour l'outil qui l'utilise, puis révoquez l'ancien (§ 9.2.4).

9.2.2 Créer un jeton

Objectif : créer un jeton en lecture seule pour le serveur Prometheus.

Prérequis : rôle `admin`.

Étapes :

1. Dans l'interface d'administration, ouvrez la gestion des jetons, ou utilisez `POST /api/v1/tokens`.
2. Indiquez un nom explicite (`prometheus-prod`), le rôle `metrics` et une date d'expiration conforme à votre politique de sécurité.
3. Copiez la valeur affichée et enregistrez-la dans l'outil destinataire.

Exemple par l'API (champs `name`, `role` et `expiresAt`) :

```bash
sapi -X POST -H 'Content-Type: application/json' \
  -d '{"name": "prometheus-prod", "role": "metrics", "expiresAt": "2026-11-30T00:00:00Z"}' \
  "$SONDE/api/v1/tokens"
```

Résultat attendu : la réponse contient la valeur du jeton, visible cette seule fois. Les champs de la requête et de la réponse sont décrits par le schéma `token.schema.json`.

Vérification : le jeton apparaît dans `GET /api/v1/tokens`, sans sa valeur.

9.2.3 Choisir le rôle et la durée de vie

Usage	Rôle	Expiration conseillée
Prometheus (collecte `/metrics`)	`metrics`	12 mois, renouvellement planifié
Nagios, Centreon (`check_sva`)	`metrics`	12 mois
Tableau de bord interne en lecture	`viewer`	12 mois
Outil d'automatisation créant des contrôles	`operator`	3 à 6 mois
Outil d'automatisation complet (déploiement de scripts, secrets)	`admin`	Courte ; à réserver à des outils de confiance

9.2.4 Révoquer un jeton

Révoquez immédiatement un jeton par `DELETE /api/v1/tokens/{id}` ou depuis l'interface lorsque :

- l'outil qui l'utilisait est retiré ;
- le jeton a pu être exposé (fichier partagé par erreur, dépôt de code, capture d'écran) ;
- son propriétaire quitte l'équipe.

La révocation est effective immédiatement : les appels suivants reçoivent `401 Unauthorized`.

> **Avertissement** — Ne stockez jamais un jeton d'API dans un dépôt de code source, un fichier de configuration lisible par tous ou un ticket. Utilisez le gestionnaire de secrets de votre outil d'automatisation (coffre Ansible, gestionnaire de secrets de votre chaîne d'intégration continue, coffre-fort d'entreprise).

9.3 Gestion des contrôles

Cette section décrit les opérations sur les contrôles. Les options propres à chaque type sont détaillées :

- au chapitre 4 du guide d'utilisation et de supervision pour `icmp`, `tcp`, `dns`, `http` et `nagios` ;
- au chapitre 7 pour `api` et `database` ;
- au chapitre 4 pour `browser`.

9.3.1 Opérations disponibles

Opération	Méthode et chemin	Rôle minimal
Lister les contrôles	`GET /api/v1/checks` (filtres `type`, `enabled`, `tag`, `status`)	`viewer`
Lire un contrôle	`GET /api/v1/checks/{id}`	`viewer`
Créer	`POST /api/v1/checks`	`operator` ; `admin` pour un scénario scripté ou un contrôle qui utilise un plugin personnalisé
Remplacer	`PUT /api/v1/checks/{id}`	`operator` pour un contrôle non scripté ; `admin` si le contrôle existant ou la nouvelle définition est un scénario scripté ou utilise un plugin personnalisé, même si son code reste inchangé (un `operator` ajuste la planification ou les seuils d'un tel contrôle par `PATCH`)
Modifier partiellement	`PATCH /api/v1/checks/{id}` (JSON Merge Patch)	`operator` ; `admin` si la modification crée, change ou retire `browser.mode: script`, `browser.scriptRef` ou un plugin personnalisé
Supprimer	`DELETE /api/v1/checks/{id}`	`operator`
Activer, désactiver	`POST /api/v1/checks/{id}/enable`, `POST /api/v1/checks/{id}/disable`	`operator`
Exécuter immédiatement	`POST /api/v1/checks/{id}/runs`	`operator`
Exécution d'essai	`POST /api/v1/checks/{id}/runs?dryRun=true`	`operator`
État courant	`GET /api/v1/checks/{id}/status`	`metrics`

> **Attention** — Un `operator` peut ajuster la planification, les seuils ou les étiquettes d'un scénario scripté par `PATCH`, mais toute requête de sa part qui touche au code exécuté (passage en mode `script`, changement ou retrait de `scriptRef`, ajout, changement ou retrait d'un plugin personnalisé) est refusée avec `403`. Le choix du code exécuté relève toujours d'un `admin` (§ 6.4.1 du guide d'installation et d'administration).

9.3.2 Créer un contrôle

Objectif : créer un contrôle HTTP vérifiant la page d'accueil d'un portail.

Prérequis : rôle `operator` ; édition Essential ou supérieure ; la cible est joignable depuis la sonde (§ 3.2.2 du guide d'utilisation et de supervision).

Étapes :

1. Rédigez la définition du contrôle dans un fichier `portail-accueil.json` :

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

   Le contrôle est créé **désactivé** (`"enabled": false`) pour pouvoir être testé avant d'entrer en service.

2. Envoyez-le à la sonde :

   ```bash
   sapi -X POST -H 'Content-Type: application/json' \
     --data @portail-accueil.json "$SONDE/api/v1/checks"
   ```

3. Lancez une exécution d'essai (§ 9.3.5) et vérifiez son résultat.
4. Activez le contrôle (§ 9.3.4).

Résultat attendu : réponse `201 Created` ; le contrôle apparaît dans la liste et dans l'interface.

En cas de problème : une réponse `422` détaille les champs non conformes (§ 9.1.4) ; une réponse `409` indique que l'identifiant existe déjà ; une réponse `403` signale un rôle insuffisant ou, avec le type d'erreur `/api/v1/problems/edition-feature-unavailable`, un type de contrôle non inclus dans votre édition (§ 9.1.4).

Dans l'interface d'administration, la même opération s'effectue par un formulaire. Celui-ci applique les mêmes règles de validation que l'API (§ 9.1.4).

9.3.3 Modifier un contrôle

Deux méthodes sont possibles :

- `PUT` remplace la définition complète. C'est la méthode à privilégier lorsque la définition de référence est conservée dans un fichier, par exemple dans un dépôt de configuration ;
- `PATCH` ne modifie que les champs fournis (format JSON Merge Patch, RFC 7396). C'est la méthode la plus pratique pour un ajustement ponctuel.

Exemple : porter la fréquence à 2 minutes, en respectant le contrôle de concurrence (§ 9.1.5).

```bash
# 1. Lire l'ETag courant
ETAG=$(sapi -D - -o /dev/null "$SONDE/api/v1/checks/portail-accueil" \
       | awk 'tolower($1)=="etag:" {print $2}' | tr -d '\r')

# 2. Appliquer la modification partielle
sapi -X PATCH -H 'Content-Type: application/merge-patch+json' \
  -H "If-Match: $ETAG" \
  -d '{"schedule": {"interval": "2m", "jitter": "15s"}}' \
  "$SONDE/api/v1/checks/portail-accueil"
```

Chaque modification incrémente la **révision** du contrôle. Le numéro de révision est enregistré dans chaque résultat d'exécution : vous savez ainsi quelle version de la configuration a produit un résultat donné.

> **Information** — Avec JSON Merge Patch, les objets sont fusionnés champ par champ, mais un **tableau** fourni remplace intégralement le tableau existant. Pour modifier une seule assertion d'un contrôle HTTP, renvoyez donc la liste complète des assertions. Une propriété fixée à `null` est **supprimée** : `{"schedule": null}` retire ainsi la planification propre d'un contrôle, qui ne s'exécute plus que par un plan d'exécution ou manuellement (chapitre 6 du guide d'utilisation et de supervision).

9.3.4 Activer et désactiver

Désactiver un contrôle suspend ses exécutions planifiées sans le supprimer, et conserve son historique. C'est la méthode à utiliser pendant une intervention planifiée sur l'application supervisée, pour éviter des alertes inutiles.

```bash
sapi -X POST "$SONDE/api/v1/checks/portail-accueil/disable"
sapi -X POST "$SONDE/api/v1/checks/portail-accueil/enable"
```

> **Conseil** — Notez dans votre procédure de changement de **réactiver** les contrôles à la fin de l'intervention. Un contrôle oublié en état désactivé ne supervise plus rien. La liste `GET /api/v1/checks?enabled=false` permet de les repérer.

> **Évolution prévue** — Des fenêtres de maintenance planifiées, suspendant automatiquement des contrôles sur une période donnée, sont prévues dans une version ultérieure (§ 6.8 du guide d'utilisation et de supervision).

9.3.5 Exécution immédiate et exécution d'essai

	Exécution immédiate	Exécution d'essai (`dryRun`)
Appel	`POST /api/v1/checks/{id}/runs`	`POST /api/v1/checks/{id}/runs?dryRun=true`

	Exécution immédiate	Exécution d'essai (`dryRun`)
Contrôle désactivé	Autorisé, par exemple pour mettre au point un contrôle avant de l'activer	Autorisé
Effet sur l'état courant	Oui	Non
Export vers les plateformes	Oui	Non
Conservation du résultat	Dans l'historique	Consultable, marqué `dryRun`
Usage	Confirmer un incident ou la résolution d'un incident, rejouer un contrôle	Mettre au point un nouveau contrôle ou une modification sans perturber la supervision

L'exécution étant asynchrone, la sonde répond `202 Accepted` avec un en-tête `Location` vers la ressource d'exécution. Il suffit de suivre cette ressource jusqu'à la fin de l'exécution :

```bash
# Lancer une exécution d'essai et récupérer l'adresse de suivi
LOC=$(sapi -X POST -D - -o /dev/null \
      "$SONDE/api/v1/checks/portail-accueil/runs?dryRun=true" \
      | awk 'tolower($1)=="location:" {print $2}' | tr -d '\r')

# Consulter le résultat (répéter jusqu'à la fin de l'exécution)
sapi "$SONDE$LOC" | jq '{status, durationMs, error}'
```

L'en-tête `Location` de la réponse `202 Accepted` désigne la ressource de l'exécution, de la forme `/api/v1/runs/{runId}`.

> ✓ **Conseil** — Réalisez systématiquement une exécution d'essai après toute création ou modification significative d'un contrôle. Elle valide en une opération la configuration, l'accès réseau à la cible et les assertions.

9.3.6 Supprimer un contrôle

```bash
ETAG=$(sapi -D - -o /dev/null "$SONDE/api/v1/checks/portail-accueil" \
       | awk 'tolower($1)=="etag:" {print $2}' | tr -d '\r')
sapi -X DELETE -H "If-Match: $ETAG" "$SONDE/api/v1/checks/portail-accueil"
```

La suppression est définitive pour la configuration. L'historique des exécutions reste consultable jusqu'à l'échéance de sa durée de rétention. La suppression d'un contrôle référencé par un plan d'exécution est refusée (`409 Conflict`) ; la réponse liste les plans concernés. Retirez d'abord le contrôle de ces plans.

> ! **Attention** — Recréer ultérieurement un contrôle avec le même identifiant le rattache, dans les plateformes d'observabilité, à l'historique de l'ancien contrôle. Si la nouvelle définition vérifie autre chose, choisissez un nouvel identifiant.

Chapitre 10 — Analyse des résultats et diagnostic des scénarios

10.1 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) en échec, sur une boutique en ligne fictive ; c'est le même que celui du § 3.7.2 du guide d'utilisation et de supervision.

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	—
`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 du guide d'utilisation et de supervision)

Champ	Signification	Utilité pour l'analyse
`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 du guide d'utilisation et de supervision)
`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 (§ 10.2.2)	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 (§ 10.2.4)
`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 du guide d'utilisation et de supervision pour les contrôles réseau et techniques, chapitre 7 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 du guide d'utilisation et de supervision)

10.2 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.

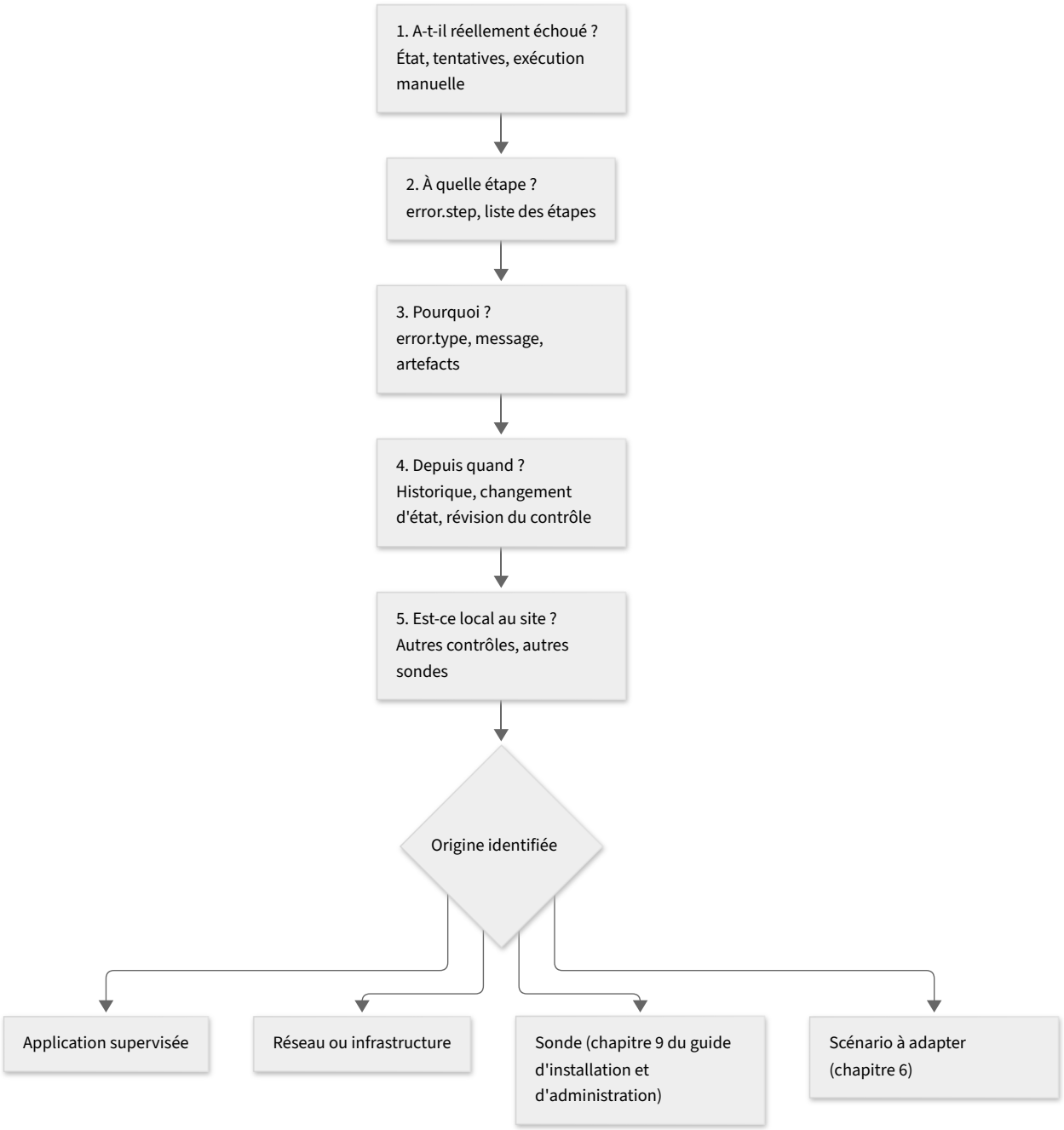


Schéma 10-1 — Méthode d'analyse d'un échec en cinq questions

10.2.1 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 du guide d'utilisation et de supervision). Reportez-vous au chapitre 9 du guide d'installation et d'administration.

- **`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 du guide d'utilisation et de supervision et chapitre 9 du guide d'installation et d'administration).

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 du guide d'utilisation et de supervision). En cas de doute, lancez une exécution manuelle (§ 7.3.5 du guide d'utilisation et de supervision).

10.2.2 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 § 3.8.2 du guide d'utilisation et de supervision.

`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
`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

`error.type`	Signification	État	Origine la plus probable	Première vérification
`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 (§ 9.3.5)
`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 9)
`browser_crash`	Le navigateur Chromium s'est arrêté anormalement	`UNKNOWN`	Mémoire insuffisante sur la sonde, page très lourde	Santé de la sonde (§ 7.9 du guide d'utilisation et de supervision), chapitre 9 du guide d'installation et d'administration
`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 du guide d'utilisation et de supervision)
`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 7)
`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` (§ 8.2)
`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 9 du guide d'installation et d'administration
`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

`error.type`	Signification	État	Origine la plus probable	Première vérification
`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 explique comment écrire des scénarios robustes à ces évolutions.

10.2.3 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 du guide d'utilisation et de supervision) ?
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 9) 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.

10.2.4 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
Tous les contrôles de la sonde échouent ou passent en `UNKNOWN`	Problème de la sonde ou de son raccordement réseau (chapitre 9 du guide d'installation et d'administration)
Seuls les contrôles vers des destinations externes (Internet) échouent	Problème de sortie Internet du site : proxy, pare-feu, lien opérateur

Observation	Interprétation probable
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).

10.3 Scénarios navigateur (Puppeteer)

Les scénarios navigateur sont les contrôles les plus riches, et donc les plus exposés aux évolutions des applications. Le chapitre 6 décrit les bonnes pratiques qui préviennent la plupart des incidents ci-dessous.

10.3.1 Tableau des symptômes

Symptôme	Causes possibles	Vérifications	Actions correctives
Tous les scénarios en `UNKNOWN` avec `browser_crash`	Chromium ne démarre pas : mémoire insuffisante, mise à jour incomplète, environnement isolé défaillant	`sva.browser.launches` ; `free -h` ; `journalctl -u sva-runner` ; dernière mise à jour appliquée	Redémarrer la sonde ; vérifier la RAM (8 Go pour l'édition Enterprise) ; si le problème suit une mise à jour, contacter le support
`browser_crash` occasionnel sur un scénario précis	Page très lourde (vidéos, cartes, nombreux scripts tiers) ; fuite de mémoire de la page	Taille et contenu de la page ; durée de l'exécution	Bloquer les domaines tiers inutiles (`blockDomains`) ; réduire la taille de la fenêtre (`viewport`) ; ne pas exécuter deux scénarios lourds en parallèle
`resource_exhausted` sur les scénarios	Concurrence navigateur trop élevée pour la RAM ; scénarios trop longs	Concurrence configurée ; métriques mémoire	Revenir à une seule exécution navigateur simultanée ; regrouper les scénarios dans un plan d'exécution séquentiel
`element_not_found`	Sélecteur devenu invalide après une évolution de l'interface ; élément affiché plus tard que prévu ; élément situé dans une iframe ; page différente (bannière de consentement, page de maintenance)	Capture d'écran de l'échec ; comparaison avec la page actuelle dans un navigateur	Mettre à jour le sélecteur en privilégiant un attribut stable (`data-testid` , rôle, texte) ; traiter la bannière de consentement en début de scénario ; préciser la propriété `frame`
`timeout` sur une attente d'élément	Application lente ; attente d'un événement qui ne se produit jamais (navigation attendue alors que la page se met à jour sans changer d'URL)	Durée des étapes précédentes ; capture	Remplacer l'attente de navigation par l'attente d'un élément ; ajuster le timeout de l'étape

Symptôme	Causes possibles	Vérifications	Actions correctives
`navigation_error`	URL injoignable depuis la sonde ; erreur DNS ou TLS dans le navigateur ; redirection vers un domaine non autorisé par le pare-feu ou le proxy de l'entreprise	Message d'erreur (code `net::ERR_…` renvoyé par Chromium)	Traiter comme un incident réseau (§ 9.5 du guide d'installation et d'administration, § 8.2 du guide d'utilisation et de supervision) ; faire autoriser la destination dans le pare-feu ou le proxy de l'entreprise (chapitre 5 du guide d'installation et d'administration)
`script_error`	Erreur de programmation dans un scénario scripté (variable non définie, syntaxe) ; module non autorisé	Message et ligne de l'erreur	Corriger le script ; tester sur poste de travail avec le lanceur local (chapitre 4) puis par une exécution d'essai
`assertion_failed` sur un texte	Libellé modifié ; espaces ou casse différents ; texte traduit selon la langue du navigateur	Capture ; texte réel de l'élément	Assouplir l'assertion (contient plutôt qu'égal) ; fixer la langue attendue
Le scénario fonctionne sur le poste de développement mais pas sur la sonde	Résolution DNS ou proxy différents ; autorité de certification interne absente de la sonde ; écran de taille différente ; langue ou fuseau horaire différents ; poste déjà authentifié (cookies), alors que la sonde démarre avec un profil vierge ; version de Chromium différente	Comparer le contexte : réseau, `viewport`, langue, cookies	Tester sur le poste avec un profil vierge (lanceur local) ; aligner `viewport` ; ajouter l'étape d'authentification ; importer l'autorité de certification interne
Le scénario échoue sur une page de vérification anti-robot, un CAPTCHA ou une double authentification	Protection anti-automatisation de l'application ; politique de sécurité imposant une authentification multifacteur	Capture d'écran	Ne pas chercher à contourner ces protections. Demander aux équipes de l'application une exemption pour l'adresse IP de la sonde ou un compte de test dédié sans authentification multifacteur, dans un environnement prévu à cet effet
Les mesures de durée varient fortement	Scripts tiers (publicité, statistiques) ; cache vide à chaque exécution (comportement attendu : profil vierge) ; charge de la sonde	Durée par étape ; requêtes réseau	Bloquer les domaines tiers non pertinents ; ne comparer que des exécutions de même nature
Captures d'écran absentes	Option `captureOnFailure` désactivée ; quota d'artefacts atteint ; échec survenu avant l'ouverture de la page	Paramètres du contrôle ; `sva.storage.usage` (zone `artifacts`)	Réactiver l'option ; ajuster la rétention des artefacts (chapitre 9)
Scénario en `CRITICAL` à cause d'une étape intermédiaire alors que l'objectif est atteint	Étape de vérification trop stricte ou devenue inutile	Détail des étapes	Réviser le scénario avec l'équipe applicative

10.3.2 Lire un échec de scénario

Pour un scénario en échec, le détail de l'exécution fournit quatre éléments complémentaires :

1. **L'étape en échec**, nommée par le scénario : elle situe le problème dans le parcours.
2. **Le type d'erreur** : il indique la nature du problème (élément absent, délai dépassé, assertion non vérifiée).
3. **Le message** : il précise le sélecteur, la valeur attendue ou l'erreur de navigation.
4. **La capture d'écran et le journal de console** : ils montrent ce que voyait réellement le navigateur et les erreurs JavaScript de la page.

> ✓ **Conseil** — Une capture d'écran qui montre une page d'erreur de l'application (« Service indisponible », erreur 500) signifie que le scénario a correctement détecté un incident applicatif. Une capture qui montre une page normale mais un élément introuvable oriente plutôt vers un scénario à mettre à jour.

Chapitre 11 — Références

11.1 Référence de l'API REST

Base : `https://<sonde>/api/v1`. Authentification : `Authorization: Bearer <jeton>` ou session de l'interface. Dans les exemples de la documentation, le jeton n'apparaît jamais dans la ligne de commande : l'en-tête est lu dans un fichier protégé, avec `curl -H @"$HOME/.sva/entete-auth"` (§ 9.1.3). L'authentification par certificat client (mTLS) est reportée à une version ultérieure, avec la console centrale. Description complète au format OpenAPI 3.1 : `GET /api/v1/openapi.json`. Voir chapitre 9.

11.1.1 Conventions

Sujet	Règle
Format	JSON UTF-8 ; dates RFC 3339 en UTC ; durées en millisecondes dans les champs suffixés `Ms` (`durationMs`, `p50DurationMs`…) des réponses
Erreurs	`application/problem+json` (RFC 9457) : `type` (adresse relative `/api/v1/problems/{code}`), `title`, `status`, `detail`, `errors[]`. `400` pour un JSON mal formé ; `422` pour une erreur de validation ; `403` pour un rôle insuffisant, avec le type `/api/v1/problems/edition-feature-unavailable` pour une fonction absente de l'édition, ou avec le type `/api/v1/problems/current-password-invalid` pour un mot de passe actuel erroné (`PUT /api/v1/users/me/password`) ; `405` pour une méthode non admise (section `license` en lecture seule)
Concurrence	`ETag` sur les ressources de configuration, renvoyé par leur lecture (pour un secret ou un compte : `GET /api/v1/secrets/{name}` ou `GET /api/v1/users/{id}`) ; `If-Match` obligatoire en `PUT`, `PATCH`, `DELETE` sur une ressource existante ; `If-None-Match: *` pour une création par `PUT` (secret) ; réponse `428` si la requête ne présente ni `If-Match` ni `If-None-Match`, `412` en cas de conflit de version ou de création d'une ressource qui existe déjà. Les requêtes d'action et d'import en écriture seule (`PUT /api/v1/system/maintenance`, `/system/certificate`, `/system/dot1x/credential`) ne portent pas sur une ressource de configuration et n'en relèvent pas. Exception : `PUT /api/v1/users/me/password` n'exige ni `If-Match` ni `If-None-Match`, la preuve étant le mot de passe actuel (§ 9.1.5)
Pagination	`?limit=…&cursor=…` ; 100 éléments par défaut, 1 000 au maximum ; curseur opaque en base64url ; réponse `{ "items": [...], "nextCursor": "…" }`. Ordre total (`startedAt`, `id`) ; curseur valide 24 heures (au-delà : `410 Gone`) ; pour un export reproductible, fixez une borne `to` ; les éléments purgés par la rétention pendant la lecture disparaissent des pages suivantes, sans instantané (chapitre 10)
Actions longues	Réponse `202 Accepted` et en-tête `Location` vers la ressource de suivi
Paramètres de chemin	En anglais : `{id}` pour les contrôles, plans, utilisateurs, jetons et destinations d'export ; `{name}` pour les secrets, scripts, plugins et artefacts ; `{runId}` pour les exécutions ; `{section}` pour les sections de paramètres

11.1.2 Ressources

Méthode et chemin	Rôle minimal	Description
`GET /health`	aucun	Vivacité du moteur principal : `OK` (code `200`) ; `MAINTENANCE` (code `503`) pendant l'état de maintenance qui ouvre l'activation d'une mise à jour
`GET /metrics`	`metrics`	Exposition des métriques au format Prometheus

Méthode et chemin	Rôle minimal	Description
`GET /api/v1/system`	`viewer`	Version, édition et état du certificat d'édition (`editionCertificateStatus` : mode restreint et sa cause), ressources (dont `resources.memoryTotalBytes`), état des modules et des exports, états de santé (dont `storage`, `network`, `support`, `updates`), date de fin de la période de support (`supportEndsAt`) et du justificatif de maintenance (`maintenanceEndsAt`). `supportEndsAt` est lisible avec le rôle `viewer`
`GET /api/v1/checks`	`viewer`	Liste des contrôles (filtres `type`, `enabled`, `tag`, `status`)
`POST /api/v1/checks`	`operator` ; `admin` pour un scénario scripté ou un plugin personnalisé	Création d'un contrôle ; un `operator` qui crée un scénario scripté ou un contrôle utilisant un plugin personnalisé reçoit `403`
`GET /api/v1/checks/{id}`	`viewer`	Détail d'un contrôle
`PUT /api/v1/checks/{id}`	`operator` pour un contrôle non scripté ; `admin` pour un scénario scripté ou un contrôle utilisant un plugin personnalisé	Remplacement complet ; un `operator` qui crée, modifie ou retire `browser.mode: script`, `browser.scriptRef` ou un plugin personnalisé reçoit `403` (§ 9.3.1)
`PATCH /api/v1/checks/{id}`	`operator` ; `admin` pour toute modification de `browser.mode: script`, de `browser.scriptRef` ou d'un plugin personnalisé	Modification partielle (JSON Merge Patch) ; un `operator` qui crée, modifie ou retire `browser.mode: script`, `browser.scriptRef` ou un plugin personnalisé reçoit `403`
`DELETE /api/v1/checks/{id}`	`operator`	Suppression (historique conservé selon la rétention) ; refusée avec `409` et la liste des plans concernés si un plan référence le contrôle
`POST /api/v1/checks/{id}/enable`	`operator`	Activation
`POST /api/v1/checks/{id}/disable`	`operator`	Désactivation
`POST /api/v1/checks/{id}/runs`	`operator`	Exécution immédiate ; `?dryRun=true` pour une exécution d'essai
`GET /api/v1/checks/{id}/status`	`metrics`	Dernier état consolidé
`GET /api/v1/checks/{id}/aggregates?from=&to=&step=1h`	`viewer`	Agrégats horaires d'un contrôle
`GET /api/v1/runs`	`viewer`	Historique (filtres `checkId`, `planRunId`, `status`, `from`, `to`)
`GET /api/v1/runs/{runId}`	`viewer`	Détail d'une exécution
`GET /api/v1/runs/{runId}/artifacts/{name}`	`viewer`	Téléchargement d'un artefact
`GET /api/v1/plans`, `GET /api/v1/plans/{id}`	`viewer`	Plans d'exécution

Méthode et chemin	Rôle minimal	Description
`POST`, `PUT`, `PATCH`, `DELETE /api/v1/plans[/{id}]`	`operator`	Création, modification, suppression d'un plan
`POST /api/v1/plans/{id}/enable` · `/disable`	`operator`	Activation, désactivation d'un plan
`POST /api/v1/plans/{id}/runs`	`operator`	Exécution immédiate d'un plan
`POST /api/v1/scripts`	`admin`	Téléversement d'un script (renvoie une référence `nom@sha256:…`)
`GET /api/v1/scripts[/{name}]`	`viewer`	Liste des scripts et de leurs versions
`GET /api/v1/secrets`	`admin`	Noms et métadonnées des secrets (jamais les valeurs)
`GET /api/v1/secrets/{name}`	`admin`	Métadonnées d'un secret, dont `allowedHosts`, et en-tête `ETag`, à renvoyer dans `If-Match` lors d'une rotation ou d'une suppression ; la valeur du secret n'est jamais renvoyée
`PUT /api/v1/secrets/{name}`	`admin`	Création ou rotation d'un secret (écriture seule), corps `{"value": …, "allowedHosts": […]}` : la liste des hôtes autorisés est obligatoire, et la sonde n'injecte le secret que vers ces hôtes (§ 8.2). Création : en-tête `If-None-Match: *` (`412` si le secret existe déjà) ; rotation : en-tête `If-Match` avec l'`ETag` du secret. Lors d'une rotation, renvoyez la liste `allowedHosts` relue, avec l'`ETag`, par `GET /api/v1/secrets/{name}` (§ 8.2.4)
`DELETE /api/v1/secrets/{name}`	`admin`	Suppression (refusée avec `409` si un contrôle le référence)
`GET`, `POST`, `PUT`, `DELETE /api/v1/exporters[/{id}]`	`admin`	Destinations d'export
`POST /api/v1/exporters/{id}/test`	`admin`	Test d'une destination
`GET`, `PUT /api/v1/settings/{section}`	`admin`	Paramètres : `network`, `time`, `proxy`, `retention`, `security`, `trust`, `logging`, `updates`, `ssh`, `dot1x`, `license` (lecture seule : `PUT` refusé avec `405`). Le certificat de la sonde se gère par `POST /api/v1/system/certificate/csr` (demande générée sur la sonde) et `PUT /api/v1/system/certificate` (import)
`POST /api/v1/settings/network/apply`	`admin`	Application d'une configuration réseau ; renvoie un identifiant de confirmation
`POST /api/v1/settings/network/confirm`	`admin`	Confirmation, depuis la nouvelle adresse, sous 120 secondes ; sans confirmation, l'ancienne configuration est rétablie
`POST /api/v1/settings/ssh/lock`	`admin`	Verrouillage définitif de SSH (corps `{"confirm": "<nom d'hôte>"}`) ; irréversible jusqu'à une réinitialisation usine
`GET`, `POST /api/v1/users`	`admin`	Liste et création des comptes de l'interface (§ 6.4.2 du guide d'installation et d'administration)

Méthode et chemin	Rôle minimal	Description
`GET /api/v1/users/{id}`	`admin`	Lecture d'un compte (identifiant, identifiant de connexion, rôle ; jamais le mot de passe ni son empreinte) et en-tête `ETag`, à renvoyer dans `If-Match` pour `PATCH /api/v1/users/{id}`
`PATCH /api/v1/users/{id}`	`admin`	Modification du rôle (`role`) ou du mot de passe (`password`) d'un compte, avec `If-Match` ; l'identifiant de connexion ne se modifie pas. Un changement de rôle ferme immédiatement les sessions de l'interface du compte ; un changement de mot de passe les ferme également, sauf la session depuis laquelle la requête est émise si elle appartient au compte. Les jetons d'API du compte ne sont pas modifiés et conservent le rôle fixé à leur création (§ 6.4.2 du guide d'installation et d'administration)
`DELETE /api/v1/users/{id}`	`admin`	Suppression d'un compte ; ses sessions et ses jetons sont révoqués immédiatement
`PUT /api/v1/users/me/password`	tout utilisateur authentifié	Changement de son propre mot de passe : corps `{"currentPassword": …, "newPassword": …}`. Sans `If-Match` (exception à la règle de concurrence). Mot de passe actuel erroné : `403` avec le type `/api/v1/problems/current-password-invalid`. Après le changement, les autres sessions de l'interface du compte sont fermées, la session courante est conservée, et les jetons d'API du compte ne sont pas modifiés
`GET`, `POST /api/v1/tokens`	`admin`	Liste et création des jetons d'API ; un jeton est immuable (ni `PUT` ni `PATCH`) : pour le modifier, créez-en un nouveau et révoquez l'ancien
`DELETE /api/v1/tokens/{id}`	`admin`	Révocation d'un jeton
`GET /api/v1/audit-events`	`admin`	Journal d'audit (filtres `from` , `to` , `actor` , `action`)
`GET /api/v1/config/export`	`admin`	Export de la configuration (archive JSON, sans section `network` , sans comptes ni secrets)
`POST /api/v1/config/export`	`admin`	Export avec la phrase de passe dans l'en-tête `X-Config-Passphrase` , comme pour l'import : comptes (empreintes) et secrets inclus, chiffrés en AES-256-GCM
`POST /api/v1/config/import`	`admin`	Import de la configuration ; `?dryRun=true` pour valider ; identité de la sonde restaurée par défaut, paramètre `?newIdentity=true` pour cloner ; phrase de passe dans l'en-tête `X-Config-Passphrase` ; compte rendu JSON
`GET /api/v1/events`	`viewer`	Flux d'événements en temps réel (Server-Sent Events)

Méthode et chemin	Rôle minimal	Description
`GET /api/v1/updates`	`admin`	État des mises à jour : version de chaque emplacement système, emplacement actif, mises à jour disponibles, `supportEndsAt`, `maintenanceEndsAt` ; état de la mise à jour en attente (`state`), activation annoncée (`scheduledActivation` : identifiant `activationId` , numéro d'annonce `sequence` , cas `kind` , version, canal, caractère critique, heure prévue, fin prévue `windowEnd` , `noticeReduced`), report en cours (`postponedUntil`) ; image préparée dans l'emplacement inactif (`staged` : version, canal, caractère critique) ; dernière tentative (`lastResult` : identifiant d'activation, version, résultat, dates)
`POST /api/v1/updates/apply`	`admin`	Activation de la mise à jour installée sur l'emplacement inactif : immédiate, ou programmée avec le corps facultatif `{"scheduledAt": "<date RFC 3339>"}` . Drainage, redémarrage, retour automatique à l'emplacement précédent si la sonde n'est pas saine dans les 10 minutes. Sans `scheduledAt` , activation immédiate et sans préavis ; date passée ou à moins d'une minute : `422` ; date antérieure à l'expiration du préavis : acceptée, réponse avec `noticeReduced: true` . La réponse porte l'identifiant `activationId` , à comparer ensuite avec `lastResult.activationId` (§ 8.4.2 et § 8.4.12 du guide d'installation et d'administration)
`POST /api/v1/updates/postpone`	`admin`	Report de l'activation prévue, corps `{"until": "<date RFC 3339>"}` : 30 jours au plus, 7 jours pour un correctif critique ; journalisé
`POST /api/v1/updates/upload`	`admin`	Import d'une image de mise à jour signée (mise à jour hors ligne)
`POST /api/v1/system/reboot`	`admin`	Redémarrage propre
`POST /api/v1/system/shutdown`	`admin`	Arrêt propre
`POST /api/v1/system/factory-reset`	`admin`	Réinitialisation usine (corps `{"confirm": "<nom d'hôte>"}`)
`PUT /api/v1/system/maintenance`	`admin`	Import du justificatif de maintenance (fichier signé)
`POST /api/v1/system/recovery-code`	`admin`	Génère un nouveau code de récupération (mot de passe de l'administrateur exigé) ; affiché une seule fois, l'ancien est invalidé (§ 4.4.6 du guide d'installation et d'administration)
`GET /api/v1/plugins`	`operator`	Liste des plugins personnalisés déposés (Professional, Enterprise)
`POST /api/v1/plugins?name=<nom>` , `DELETE /api/v1/plugins/{name}`	`admin`	Dépôt d'un plugin personnalisé (fichier exécutable, 1 Mo au maximum) ; retrait refusé (`409`) s'il est utilisé par un contrôle (§ 4.7.10 du guide d'utilisation et de supervision)
`POST /api/v1/system/storage/ssd`	`admin`	Activation du SSD NVMe (corps `{"confirm": "<nom d'hôte>"}`) : formatage, migration des données, redémarrage

Méthode et chemin	Rôle minimal	Description
`POST /api/v1/system/dot1x/csr`	`admin`	Génération sur la sonde de la clé et de la demande de certificat client 802.1X (méthode recommandée)
`PUT /api/v1/system/dot1x/credential`	`admin`	Installation du certificat client signé (PEM) émis pour la dernière CSR, ou import d'un PKCS#12 protégé ; écriture seule
`POST /api/v1/system/diagnostics`	`admin`	Archive de diagnostic : version, santé, configuration sans secrets, journaux récents
`GET /api/v1/system/certificate`	aucun	Certificat TLS de la sonde (PEM) ; vérifier son empreinte avant usage
`GET /api/v1/openapi.json`	tout utilisateur authentifié	Description OpenAPI

11.1.3 Rôles

Rôle	Droits
`admin`	Tous les droits : comptes, jetons, secrets, réseau (dont 802.1X), SSH, mises à jour (politique, activation, report), justificatif de maintenance, stockage SSD, scénarios scriptés, plugins personnalisés
`operator`	Création et modification des contrôles déclaratifs et des plans, activation, désactivation, déclenchement, consultation. Pas de code exécutable ; référence d'un secret existant possible, sans le lire, le créer, le modifier ni changer ses hôtes autorisés : toute requête qui crée, modifie ou retire `browser.mode: script`, `browser.scriptRef` ou un plugin personnalisé est refusée avec `403`
`viewer`	Lecture seule des états, résultats et artefacts
`metrics`	`/metrics` et `GET /api/v1/checks/{id}/status` uniquement

11.2 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 10 (§ 10.2.2), 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 du guide d'utilisation et de supervision
`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 du guide d'utilisation et de supervision
`connection_reset`	La connexion a été interrompue par le serveur ou un équipement intermédiaire	Pare-feu, surcharge, plantage du service	`CRITICAL`	§ 8.2 du guide d'utilisation et de supervision
`tls_error`	La négociation TLS a échoué	Autorité inconnue, nom de certificat incorrect, protocole obsolète	`CRITICAL`	§ 8.2 du guide d'utilisation et de supervision

Type	Signification	Cause typique	État	Renvoi
`certificate_expired`	Le certificat du serveur est expiré	Renouvellement oublié	`CRITICAL`	§ 8.2 du guide d'utilisation et de supervision
`http_status`	Le code de réponse HTTP n'est pas celui attendu	Erreur serveur, redirection, authentification	`CRITICAL`	§ 8.2 du guide d'utilisation et de supervision
`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 du guide d'utilisation et de supervision, § 10.3
`element_not_found`	Un élément attendu n'a pas été trouvé dans la page	Évolution de la page, sélecteur fragile	`CRITICAL`	§ 10.3
`navigation_error`	Le navigateur n'a pas pu charger une page	Réseau, URL, certificat	`CRITICAL`	§ 10.3
`script_error`	Le scénario scripté a levé une erreur de programmation	Erreur dans le script	`CRITICAL`	§ 10.3
`browser_crash`	Chromium s'est arrêté anormalement	Mémoire insuffisante, page trop lourde	`UNKNOWN`	§ 10.3
`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 du guide d'utilisation et de supervision
`database_error`	La connexion ou la requête SQL a échoué	Paramètres, pilote, requête	`CRITICAL`	§ 8.2 du guide d'utilisation et de supervision
`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 du guide d'utilisation et de supervision, chapitre 7
`auth_failed`	L'authentification a été refusée	Mot de passe changé, compte verrouillé	`CRITICAL`	§ 8.2 du guide d'utilisation et de supervision
`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`	§ 8.2.2 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 du guide d'utilisation et de supervision, § 10.3
`runner_crash`	Le moteur de scénarios s'est interrompu pendant l'exécution	Incident logiciel	`UNKNOWN`	§ 10.3
`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

Type	Signification	Cause typique	État	Renvoi
`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 du guide d'utilisation et de supervision. 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.