



Veille Technologique

Technique

1er octobre 2026 – Fréquence : Bi-hebdomadaire (Mardi et Jeudi)

Synthèse & Sujet du Jour

Event-Driven Ansible : remédiation fermée et auto-guérison des plateformes OpenShift

Rulebooks source/rule/action, ansible.eda et Decision Environments, remédiation fermée Prometheus/Alertmanager sur OpenShift, et l'arrivée du serveur MCP dans Ansible Automation Platform 2.7 : l'automatisation réactive comme compétence d'ingénierie de plateforme.

1. Périmètre

Famille de veille : 2. Métiers, Compétences & Technologies (Indicateur Qualiopi 24).

Fréquence d'exécution : Bi-hebdomadaire (Mardi et Jeudi à 08h30).

Diffusion : [Radar Web Aperta](#) — [LinkedIn Aperta Scientia](#) — [X @ApertaScientia](#).

2. Sujet du Jour — Article Publié sur le Radar Web

Un playbook Ansible lancé à la main à 02h00 pendant un incident n'est pas de l'automatisation : c'est un script humain avec de la latence humaine. Le vrai changement de nature, c'est de transformer un signal — une alerte Prometheus, un événement Kubernetes, un message Kafka, un log journald — en décision et en action, sans réveiller personne. C'est exactement le périmètre d'**Event-Driven Ansible (EDA)** : le triptyque **source** → **rule** → **action**, matérialisé dans un **rulebook** versionné.

Avec l'arrivée d'**Ansible Automation Platform 2.7** et de son **serveur MCP**, EDA sort du registre du gadget de démonstration pour devenir une brique de plateforme : le plan de contrôle qui relie l'observabilité à la remédiation, et désormais un agent conversationnel au même plan de contrôle.

1. Le modèle source / rule / action

Un rulebook est un artefact YAML, versionnable et revu en pull request, dont la structure est volontairement minimale :

```
---
- name: Listen for events on a webhook
  hosts: localhost
  sources:
    - ansible.eda.webhook:
        host: 0.0.0.0
        port: 5000
  rules:
    - name: Say Hello
      condition: event.payload.message == "Ansible is super cool"
      action:
```



```
run_playbook:
  name: say-what.yml
```

Trois blocs, trois responsabilités :

- **sources** : une source d'événements fournie par la collection `ansible.eda` — `webhook`, `alertmanager`, `kafka`, `file`, `journald`, `pg_listen`, `range` pour les tests, plus les intégrations éditeurs (Splunk, ServiceNow, etc.). Chaque source expose un flux d'événements normalisés.
- **rules / condition** : le filtrage, exprimé dans le DSL de conditions d'EDA sur l'objet `event` (accès par attributs, opérateurs de comparaison et logiques, tests d'appartenance). C'est ici que vit la logique métier — et donc la dette de maintenance.
- **action** : `run_playbook`, `run_module`, `run_job_template` (déclenchement d'un Job Template du contrôleur AAP, donc réutilisation du RBAC et de l'inventaire existants), ou `debug/print_event` pour l'observation.

Point d'architecture important : **la condition est évaluée sur l'événement, pas sur l'état du système**. Un rulebook mal écrit rejoue donc la même remédiation en boucle sur une alerte qui oscille. La maîtrise du *blast radius* n'est pas optionnelle.

2. Du CLI au plan de contrôle : EDA Controller et Decision Environments

`ansible-rulebook` est un exécutable local, utile pour le prototypage et le débogage. En production, EDA est opéré par le **EDA Controller** (`ansible/eda-server`, Apache-2.0) intégré à Ansible Automation Platform : les rulebooks deviennent des **activations** (activer / désactiver / redémarrer), adossées à un projet Git et à un inventaire.

Le socle d'exécution est la **Decision Environment** : une image de conteneur (EE) qui embarque `ansible-rulebook`, Python et les collections requises. C'est le pendant, pour la décision, de l'Execution Environment pour l'exécution.

- **Reproductibilité** : la même DE buildée, versionnée et poussée dans un registre (Quay, Automation Hub) garantit que la décision prise en recette est identique à celle prise en production.
- **Cloisonnement** : une DE par domaine (réseau, sécurité, plateforme) limite les collections accessibles, donc la surface d'action d'une règle compromise.
- **Traçabilité** : chaque activation produit un historique exécutable — quel événement, quelle règle, quelle action, quel résultat. C'est la matière première de l'audit et du post-mortem.

Un rulebook ne se teste pas en production. La collection `ansible.eda` fournit des sources synthétiques (`range`) qui permettent de rejouer un flux d'événements en CI avant toute activation.

3. Remédiation fermée : Alertmanager comme déclencheur

Le cas d'usage canonique sur OpenShift est la boucle **Prometheus → Alertmanager → EDA → action sur le cluster** : le `PrometheusRule` déclenche, Alertmanager route et notifie via un `receiver` webhook, la source `ansible.eda.alertmanager` ingère la charge utile, la règle filtre sur le nom de l'alerte et les labels, l'action exécute le playbook de remédiation.

Un exemple d'implémentation de référence, documenté par le **Red Hat Developer Blog**, déploie une application Quarkus instrumentée, un `ServiceMonitor`, Alertmanager avec sa `AlertmanagerConfig`, Prometheus, et un Pod EDA portant le rulebook et le playbook d'action — le tout purement déclaratif en `oc apply`.

Ce qui fait la différence entre une démo et un système exploitable :

- **Idempotence** : la remédiation doit être sans effet si l'état est déjà correct. Un redémarrage de Pod en boucle est un incident, pas un remède.



- **Fenêtre et anti-rebond** : ne pas réagir à une alerte **firing** sur un symptôme transitoire ; exiger une durée minimale ou un **for** : dans la règle Prometheus.
- **Escalade graduée** : auto-correction pour les cas connus et sûrs, création de ticket ou notification sinon. EDA n'est pas « tout automatiser », c'est automatiser ce qui est décidable.
- **RBAC minimal** : le ServiceAccount qui porte la remédiation ne doit pouvoir que ce qu'il doit pouvoir (**drain** d'un nœud n'est pas la suppression d'un PVC).

4. Cas d'usage plateforme qui passent l'épreuve de la production

- **Cycle de vie des nœuds** : cordonner et drainer un nœud détecté dégradé par ses métriques, avant que le kubelet ne le fasse pour vous, avec éviction respectant les PDB applicatives.
- **Certificats et secrets** : détection d'un certificat proche de l'expiration, rotation et reload des composants concernés — un classique qui ne devrait jamais dépendre d'un humain.
- **Élasticité** : mise à l'échelle automatique guidée par un signal métier, y compris sur les charges virtualisées (le Red Hat Developer Blog documente le scaling CPU automatique de machines OpenShift Virtualization piloté par EDA).
- **Sécurité (SOAR)** : isolement d'un workload suspect sur détection, quarantaine, enrichissement et ouverture de ticket. Le rapprochement entre détection et confinement reste la principale source de réduction du temps d'exposition.
- **AIOps** : chaînage des signaux d'observabilité et d'ITSM (Instana, Splunk, ServiceNow) vers une remédiation outillée — le sujet des solution guides publiés pour AAP 2.7.

5. AAP 2.7 : EDA à l'ère agentique

Ansible Automation Platform 2.7 change l'ergonomie et l'intégration d'EDA plus qu'il n'en change le cœur :

- **Serveur MCP pour AAP (Technology Preview)** : les agents IA externes (Claude, Cursor, etc.) interrogent l'état des jobs, collectent des faits et lancent des workflows d'automatisation en langage naturel. Le plan de contrôle devient une capacité callable par un agent — et donc un objet de sécurité à part entière.
- **Automation intelligent assistant avec « bring your own knowledge »** : le pipeline RAG de l'assistant peut être étendu avec la documentation interne de l'organisation (politiques, runbooks, terminologie), ce qui rapproche l'assistant du contexte réel d'exploitation.
- **OIDC identity provider pour HashiCorp Vault (Technology Preview)** : AAP émet des JWT à courte durée de vie, scopés au job, à l'utilisateur et à l'organisation, en remplacement des comptes de service à longue durée de vie. C'est exactement le modèle qu'il faut pour qu'un rulebook accède à un secret sans secret stocké.
- **Execution Environment builder & content catalog** : l'environnement de décision se construit dans l'UI, à partir de collections découvertes automatiquement — la barrière d'entrée à EDA baisse, le risque de dérive augmente d'autant.
- **Ansible development workspaces** : environnement de développement navigateur standardisé (VS Code, `ansible-core`, `ansible-creator`, `ansible-lint`, Molecule).

La lecture d'ingénieur : donner à un agent LLM la capacité de *lancer* une remédiation n'a de sens que si les garde-fous — RBAC, revue de rulebook, DE immuable, idempotence — sont déjà en place. L'agent amplifie la qualité du socle, il ne la remplace pas.

6. Ce qu'il faut enseigner : EDA comme discipline d'exploitation



Event-Driven Ansible n'est pas un produit à ajouter au catalogue : c'est la jonction entre l'observabilité, la gestion de configuration et la sécurité opérationnelle. La compétence qui manque sur le marché n'est pas « écrire un rulebook », c'est **concevoir une boucle de décision bornée** : quel événement, quel seuil, quelle action, quel retour arrière, quel journal.

3. Impact sur les Cursus Aperta Scientia

- **Cursus AS200 (DevOps Platform Engineer — 399h)** : ce sujet alimente directement l'automatisation d'infrastructure et l'exploitation à l'échelle — écriture de rulebooks EDA et d'activations côté **DO374 / AAP**, intégration à l'observabilité Prometheus/Alertmanager et aux politiques de sécurité, exploitation et durcissement multi-cluster côté **DO380**, et maîtrise des environnements d'exécution et de décision (Execution / Decision Environments) comme artefacts de chaîne de livraison. Préparation aux certifications **EX188**, **EX280** et **EX288**.
- **Cursus AS300 (AI Platform Engineer — 399h)** : transposition du modèle au plan de contrôle IA — remédiation réactive sur les **InferenceService** KServe et les **ServingRuntime** vLLM (éviction d'un GPU en erreur, réduction de charge sur un ServingRuntime dégradé, redémarrage contrôlé d'un pipeline MLOps), et interaction avec le **serveur MCP** d'AAP comme point d'intégration entre agents et automatisation. L'auto-guérison d'une plateforme d'inférence est une compétence d'exploitation, pas une option.

4. Sources

- **Red Hat Developer Blog — Event-Driven Ansible rulebook for automation* :
developers.redhat.com/articles/2024/04/12/event-driven-ansible-rulebook-automation
- **Red Hat Developer Blog — OpenShift application monitoring with Event-Driven Ansible & Alertmanager* :
developers.redhat.com/articles/2024/01/08/openshift-application-monitoring-event-driven-ansible-alertmanager
- **Red Hat Developer Blog — 5 ways to leverage Event-Driven Ansible for your platform* :
developers.redhat.com/articles/2024/10/25/5-ways-leverage-event-driven-ansible-your-platform
- **Red Hat Developer Blog — Implement remediation strategies with Event-Driven Ansible* :
developers.redhat.com/articles/2025/01/09/implement-remediation-strategies-event-driven-ansible
- **Red Hat Developer Blog — Leveraging Ansible Event-Driven Automation for Automatic CPU Scaling in OpenShift Virtualization* :
developers.redhat.com/blog/2025/05/07/leveraging-ansible-event-driven-automation-automatic-cpu-scaling-openshift
- **Red Hat Developer Blog — What's new in Red Hat Ansible Automation Platform 2.7* :
developers.redhat.com/articles/2026/06/10/whats-new-red-hat-ansible-automation-platform-2-7
- **ansible-rulebook (GitHub, Apache-2.0)** : github.com/ansible/ansible-rulebook — documentation : docs.ansible.com/projects/rulebook
- **EDA Controller / eda-server (GitHub, Apache-2.0)** : github.com/ansible/eda-server
- **Red Hat — Documentation Ansible Automation Platform 2.7** :
docs.redhat.com/en/documentation/red_hat_automation_platform/2.7



Contacts & Communauté :

- Direction des Formations : learning@apertascientia.io
- Référent Handicap & Bien-être (Care) : care@apertascientia.io
- X (Twitter) : [@ApertaScientia](https://twitter.com/ApertaScientia)
- Communauté Discord : discord.gg/eDJzR5Vk85

