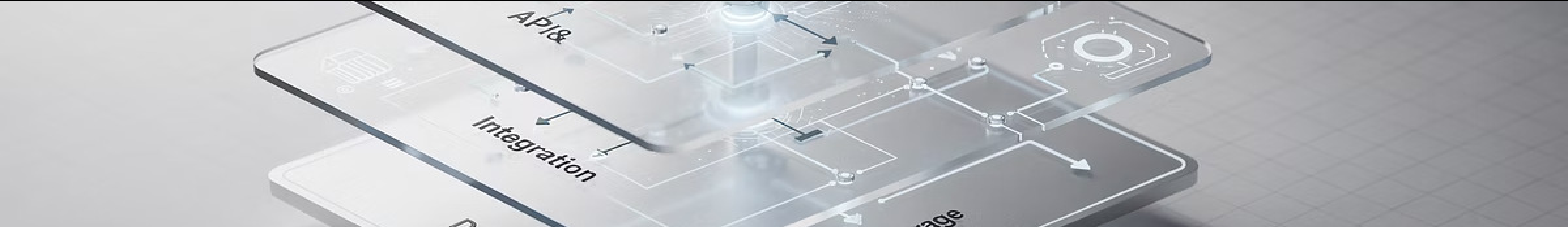
Understanding Ansible Automation Platform (AAP) Components

Enterprise-Level Architecture Explanation — Training Ready for IT Architects, Platform Engineers, and Enterprise DevOps Teams

RED HAT AAP

ENTERPRISE AUTOMATION

ARCHITECTURE DEEP DIVE



Ansible Automation Platform: Architecture Overview

Ansible Automation Platform (AAP) is Red Hat's enterprise-grade automation suite built on top of Ansible core. It transforms simple CLI-based playbook execution into a governed, scalable, and auditable enterprise automation system.

Centralized

Single control plane for all automation

Governed

RBAC, approvals, and audit trails

Scalable

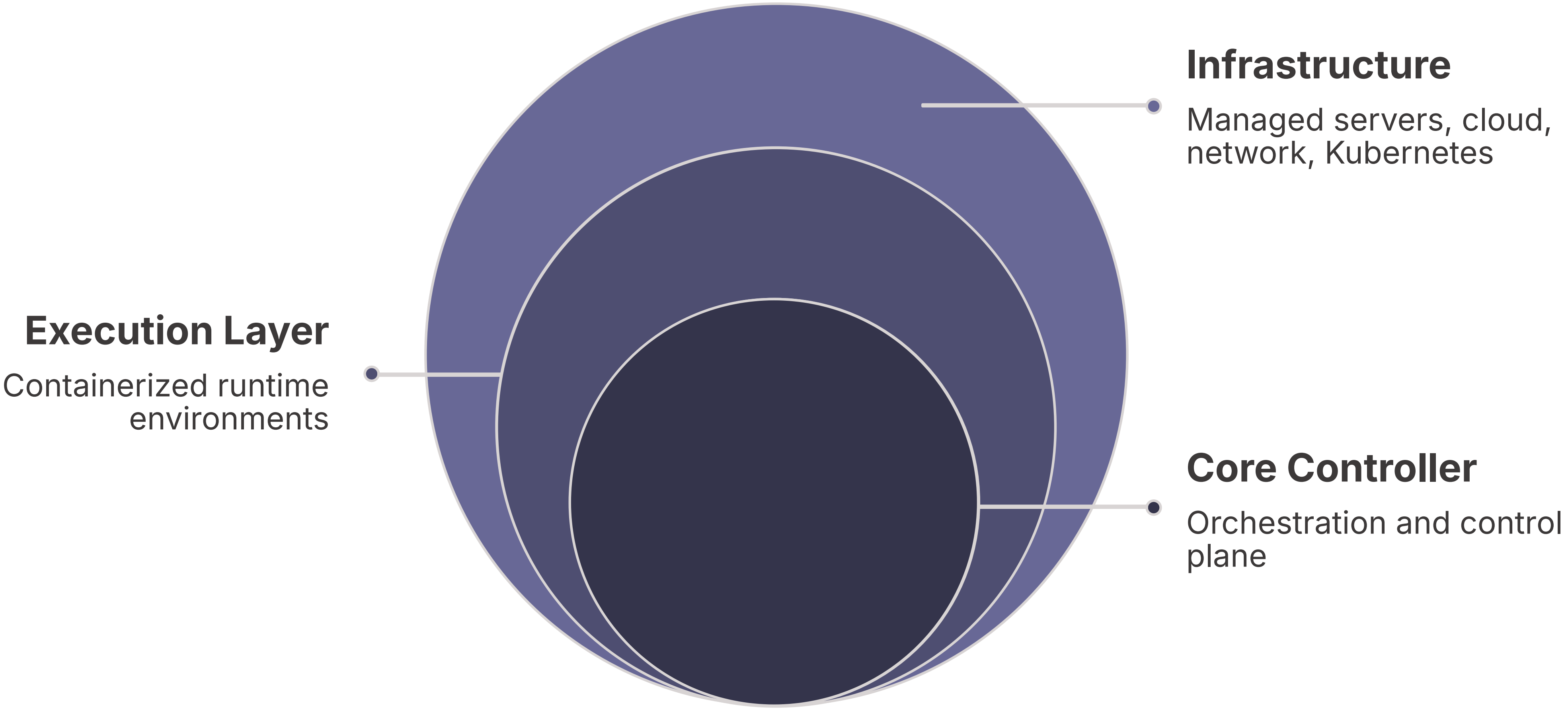
Distributed execution across environments

API-Driven

Integrates with CI/CD and ITSM toolchains

High-Level Architecture Layers

AAP is organized as a layered stack — from user-facing interfaces down to managed infrastructure. Each layer has a distinct responsibility.



Infrastructure

Managed servers, cloud, network, Kubernetes

Execution Layer

Containerized runtime environments

Core Controller

Orchestration and control plane

Understanding this layered model is the foundation for designing scalable, enterprise-grade automation architectures on AAP.

Core Components of AAP

AAP is composed of five tightly integrated components, each addressing a distinct layer of enterprise automation.

Automation Controller

Orchestration and control plane — the operational hub for all job execution, scheduling, and governance.

Automation Hub

Central content repository for certified collections, roles, and execution environment images.

Execution Environments

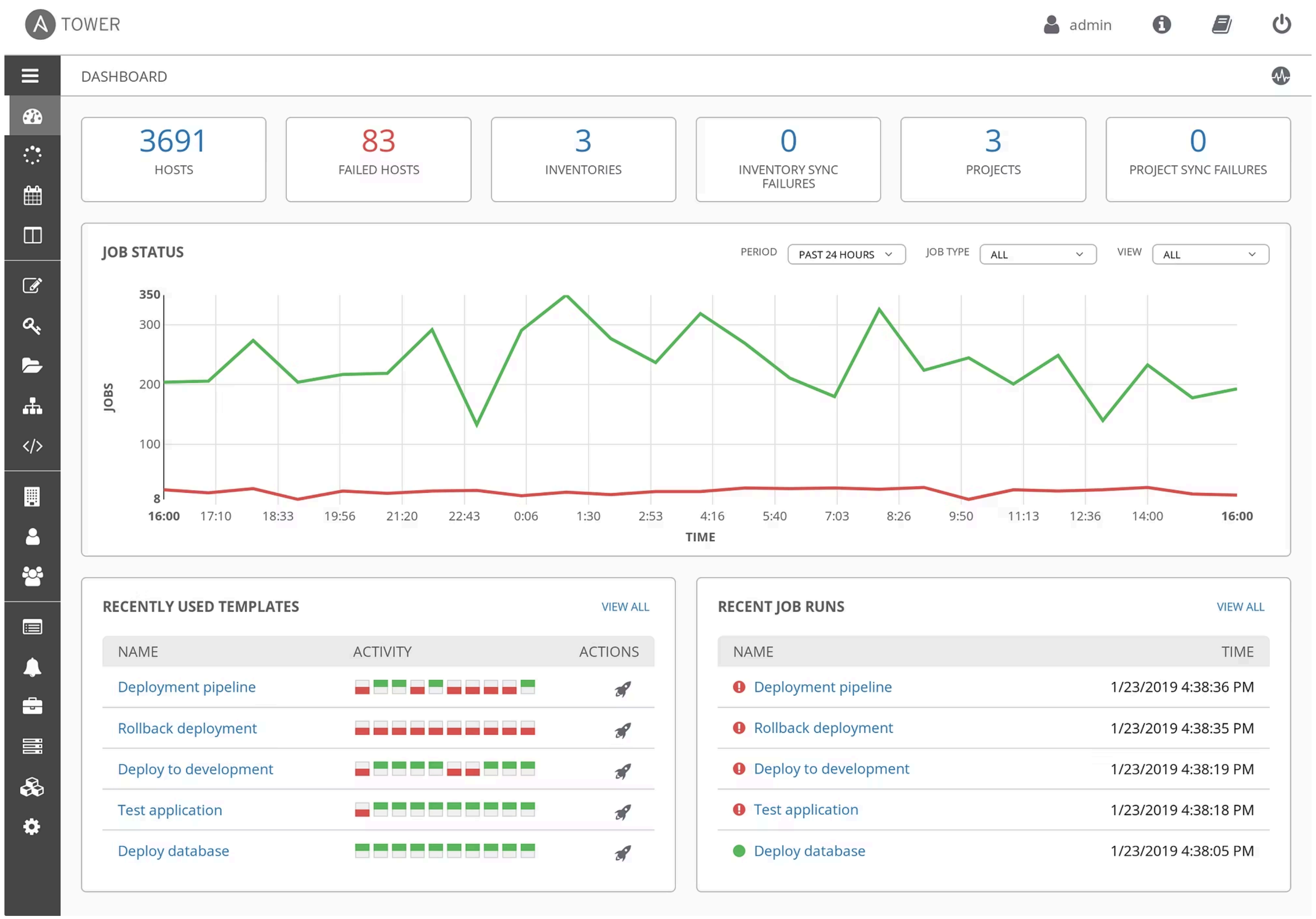
Containerized, portable Ansible runtimes that eliminate dependency drift and ensure reproducibility.

Event-Driven Ansible

Reactive automation engine that listens to real-time events and triggers playbooks automatically.

Private Automation Hub

Internal enterprise content governance — curate, approve, and distribute automation artifacts internally.



3691	83	3	0	3	0
HOSTS	FAILED HOSTS	INVENTORIES	INVENTORY SYNC FAILURES	PROJECTS	PROJECT SYNC FAILURES

RECENTLY USED TEMPLATES

NAME	ACTIVITY	ACTIONS
Deployment pipeline	🟥🟩🟩🟥🟩🟥🟥🟥🟥🟩	🚀
Rollback deployment	🟥🟥🟥🟥🟥🟥🟥🟥🟥🟥	🚀
Deploy to development	🟥🟩🟩🟩🟩🟥🟥🟩🟩🟩	🚀
Test application	🟥🟩🟩🟩🟩🟩🟩🟩🟩🟩	🚀
Deploy database	🟩🟩🟩🟩🟩🟩🟩🟩🟩🟩	🚀

RECENT JOB RUNS

NAME	TIME
❗ Deployment pipeline	1/23/2019 4:38:36 PM
❗ Rollback deployment	1/23/2019 4:38:35 PM
❗ Deploy to development	1/23/2019 4:38:19 PM
❗ Test application	1/23/2019 4:38:18 PM
● Deploy database	1/23/2019 4:38:05 PM

Automation Controller (Formerly Tower)

The **Automation Controller** — previously known as Ansible Tower — is the **control center** of the entire AAP platform. It exposes a Web UI, a full REST API, and a rich set of enterprise governance capabilities.

- **Web UI**
- **REST API**
- **RBAC**
- **Workflows**
- **Credential Vault**
- **Audit Logging**

What Automation Controller Solves

❌ Without Controller

- Anyone runs playbooks directly from the CLI
- No team-level governance or access control
- No centralized execution logging
- No approval workflow before production changes
- Credentials stored insecurely in scripts or files

✅ With Controller

- Centralized execution with full visibility
- Team-level RBAC permissions per environment
- Secure, encrypted credential vault
- Multi-step workflow approvals enforced
- Enterprise-grade audit trail for every job

Automation Controller: Key Features

Job Templates

Reusable definitions for automation jobs — specifying which playbook, inventory, credentials, and EE to use.

Projects

Git-backed playbook repositories that sync automatically on change or on schedule.

Inventories

Dynamic or static lists of managed hosts, grouped by environment, region, or role.

Workflows

Multi-step automation chains with conditional branching, approvals, and parallel execution.

Schedules & RBAC

Cron-like job scheduling and fine-grained team-based access control across all resources.

Enterprise Example: Controller in Action

The Scenario

- 500 Linux servers across Dev, QA, and Prod
- Security team must approve all Prod changes
- Multiple teams share the same platform

How Controller Handles It

- Only the DevOps team has Prod deployment rights via RBAC
- Workflow approval gate requires Security sign-off
- Every execution is logged with user, time, and outcome
- All SSH keys and API tokens are stored encrypted — never exposed

Automation Hub: Enterprise Content Repository

What It Provides

Automation Hub is the central repository for all certified and curated Ansible content — collections, roles, and execution environment images. Think of it as the **enterprise app store for Ansible automation artifacts**.

Collections

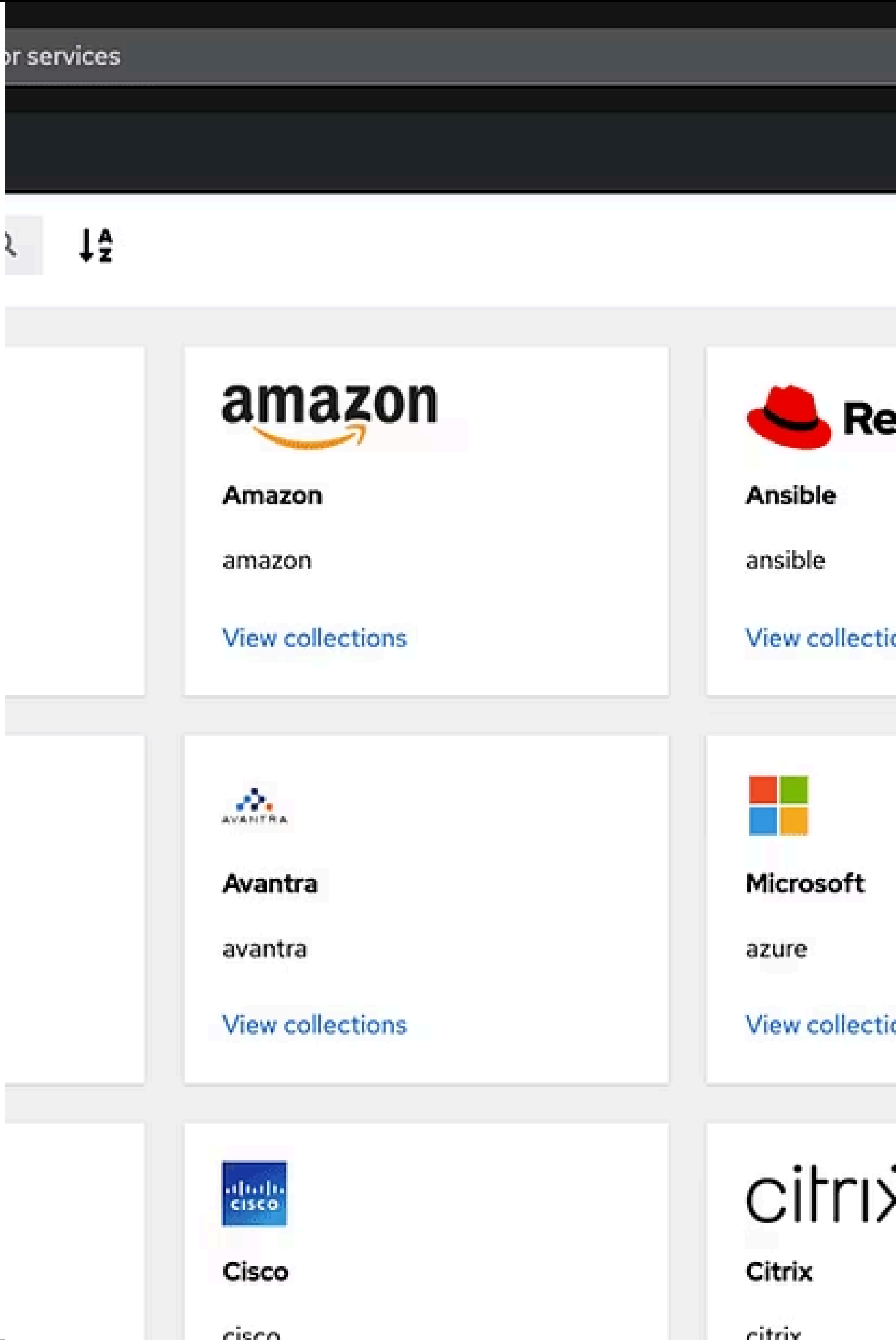
Packaged modules, plugins, and roles

Certified Content

Red Hat and partner-validated integrations

EE Images

Pre-built execution environment containers



Types of Automation Hubs

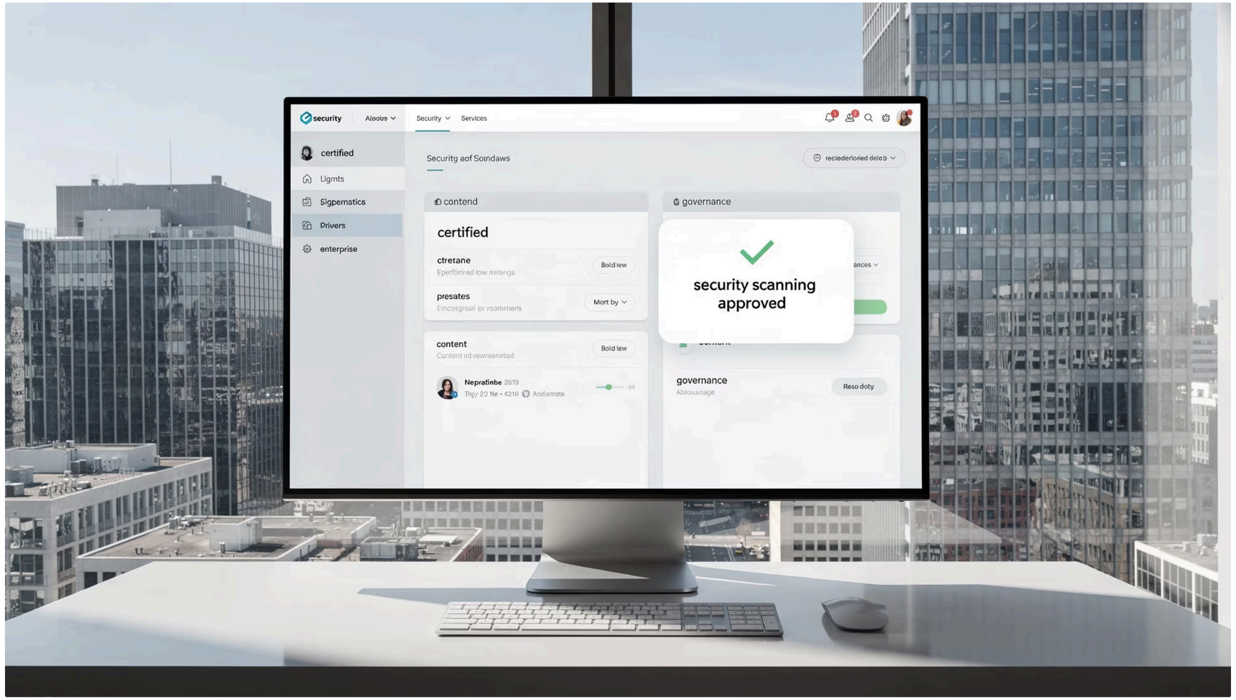
Public Automation Hub

Hosted and maintained by Red Hat at **console.redhat.com**. Provides access to Red Hat Certified Collections and community Galaxy content. Ideal for teams getting started or consuming vendor-certified integrations.

Private Automation Hub

Self-hosted within the enterprise network. Enables organizations to publish, curate, approve, and version-control internal automation content. Full air-gap support for restricted environments.

Why Automation Hub Matters in the Enterprise

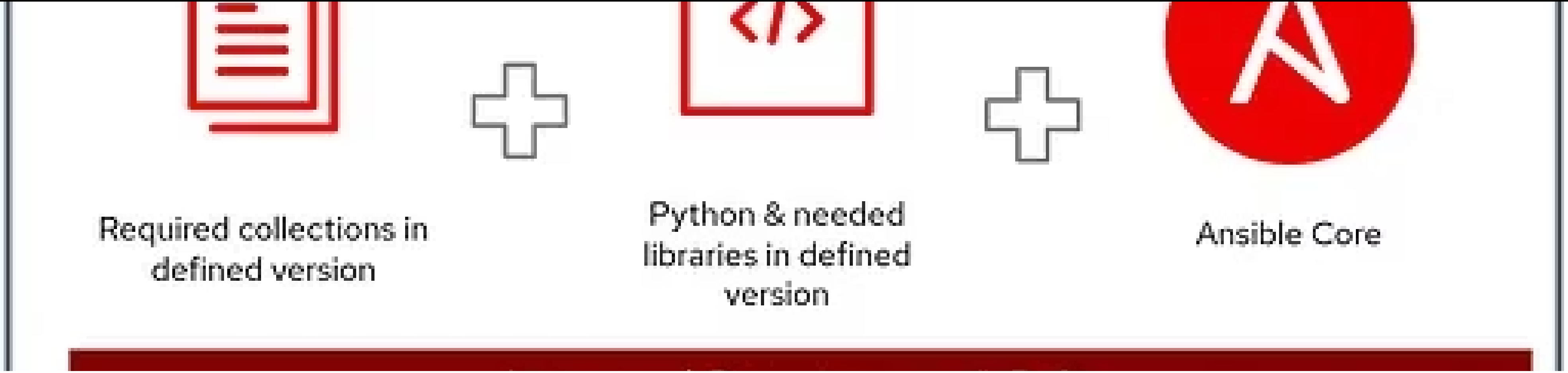


The Enterprise Challenge

- You cannot consume random, unvetted open-source roles in production
- Content must be approved, tested, and version-pinned
- Certified integrations for AWS, Azure, SAP, and VMware require a trusted source

What Hub Provides

- Content governance and approval workflows
- Immutable version control for all artifacts
- Security scanning before publishing


Required collections in defined version
Python & needed libraries in defined version
Ansible Core

Execution Environments (EE)

Execution Environments are containerized, self-contained runtime environments for running Ansible automation — replacing fragile, machine-specific Python dependency setups entirely.

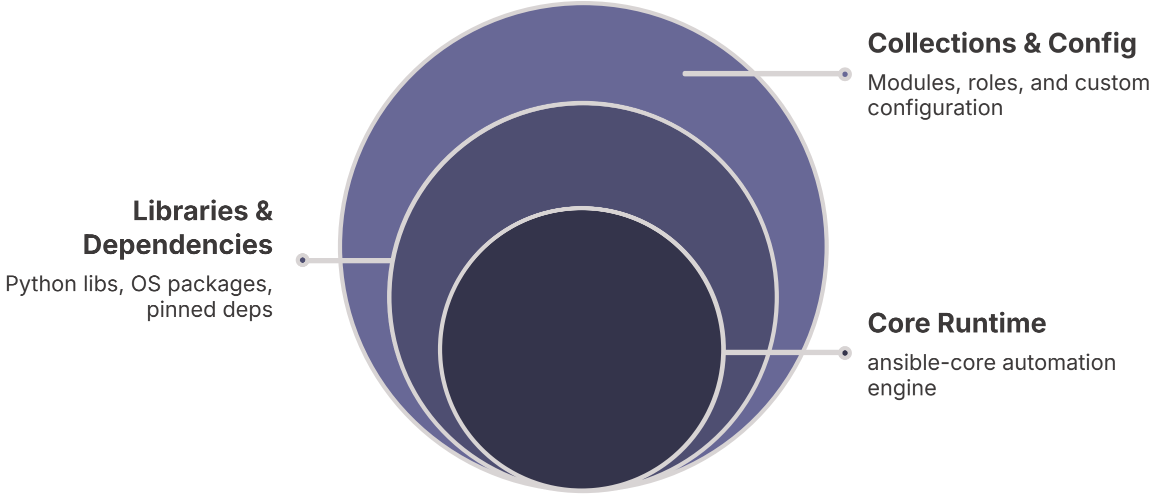
❌ Before EEs

Python dependency chaos across teams. "Works on my machine" failures in CI/CD. Unpredictable automation behavior across environments.

✅ With EEs

Consistent, portable, OCI-compliant containers. Every engineer runs the exact same runtime. Reproducible behavior from laptop to production pipeline.

What's Inside an Execution Environment?


Collections & Config
Modules, roles, and custom configuration
Libraries & Dependencies
Python libs, OS packages, pinned deps
Core Runtime
ansible-core automation engine

Runtime Components

- **ansible-core** — the automation engine itself
- **Python libraries** — required for modules and plugins
- **Collections** — pre-packaged roles and modules
- **Dependencies** — OS and Python packages pinned to specific versions

Built With

- Podman (default for RHEL-based builds)
- Docker and any OCI-compatible runtime
- **ansible-builder** CLI tool

Why Execution Environments Matter

Consistent Runtime

Every team member and every CI/CD pipeline runs the exact same container — no version drift, no surprises.

Portable Automation

EEs move seamlessly between laptops, CI runners, OpenShift clusters, and air-gapped environments.

Cloud-Native Scale

Deploy and scale EEs as Kubernetes pods, enabling horizontal scaling of automation execution capacity on demand.

Enterprise Impact: Specialized Execution Environments

Organizations build **purpose-specific EEs** tailored to each automation domain — keeping runtimes lean, secure, and auditable.

Network Automation EE

Includes NAPALM, netcommon, and vendor-specific collections for Cisco, Juniper, and Arista.

Cloud Automation EE

Bundles AWS, Azure, GCP, and VMware collections with cloud SDK dependencies pre-installed.

Security Automation EE

Contains SIEM integrations, CIS compliance modules, and security hardening playbooks.

Kubernetes Execution

EEs run natively as pods on OpenShift or upstream Kubernetes — enabling horizontal scale-out of automation workers.

Event-Driven Ansible (EDA)

Event-Driven Ansible shifts automation from *scheduled execution* to *real-time reactive response*. Instead of polling on a cron schedule, EDA listens to live event streams and triggers the right automation instantly.

Webhooks

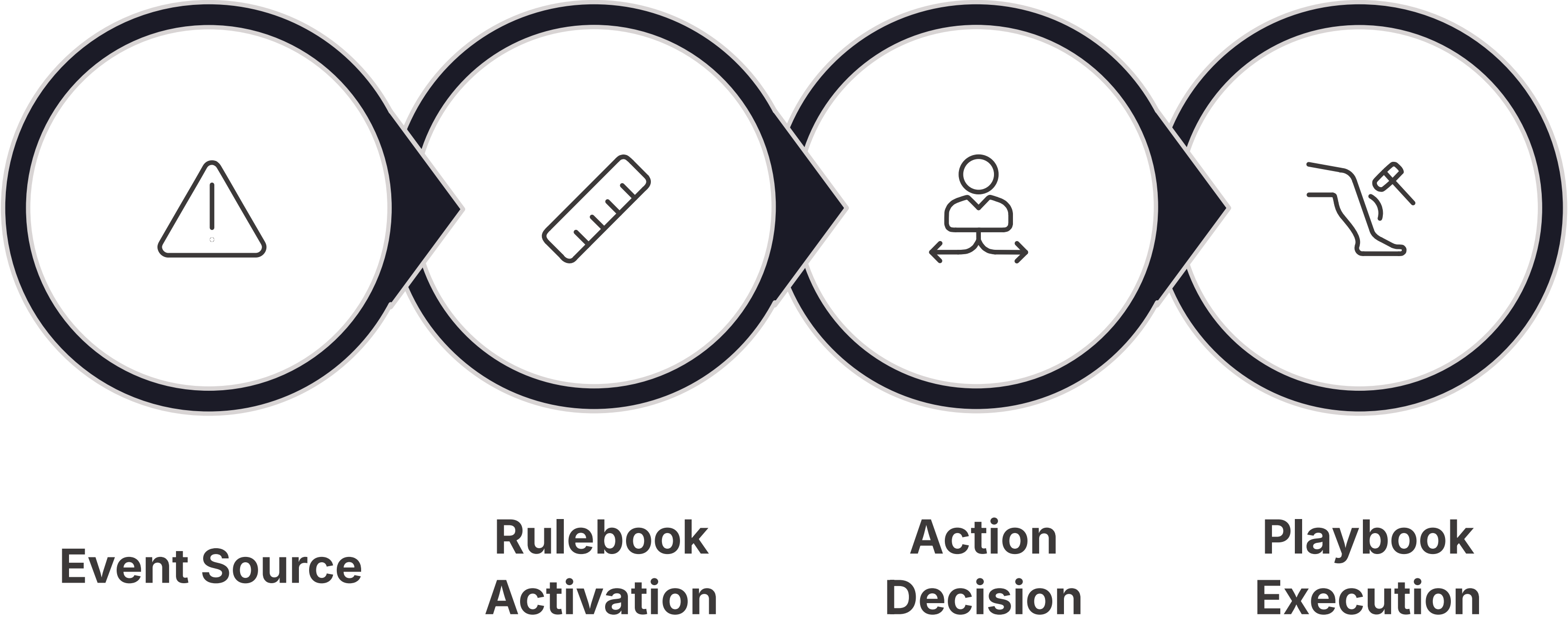
Security Alerts

Monitoring Events

Kafka / Message Queues

Cloud Events

How Event-Driven Ansible Works



Event Source

Rulebook Activation

Action Decision

Playbook Execution

Real-World Example: A server's CPU exceeds 90% → monitoring tool fires an alert → EDA rulebook matches the condition → Controller is triggered to restart the service, scale the VM, post to Slack, or open a ServiceNow incident — all without human intervention.

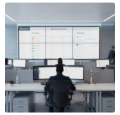
Event-Driven Ansible: Enterprise Use Cases

Security Response

Automatically isolate a compromised host, revoke credentials, and trigger a SIEM incident the moment a threat is detected.

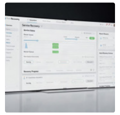
Cloud Auto-Scaling

Respond to load spikes by provisioning additional VMs or pods in real time — before users experience degradation.



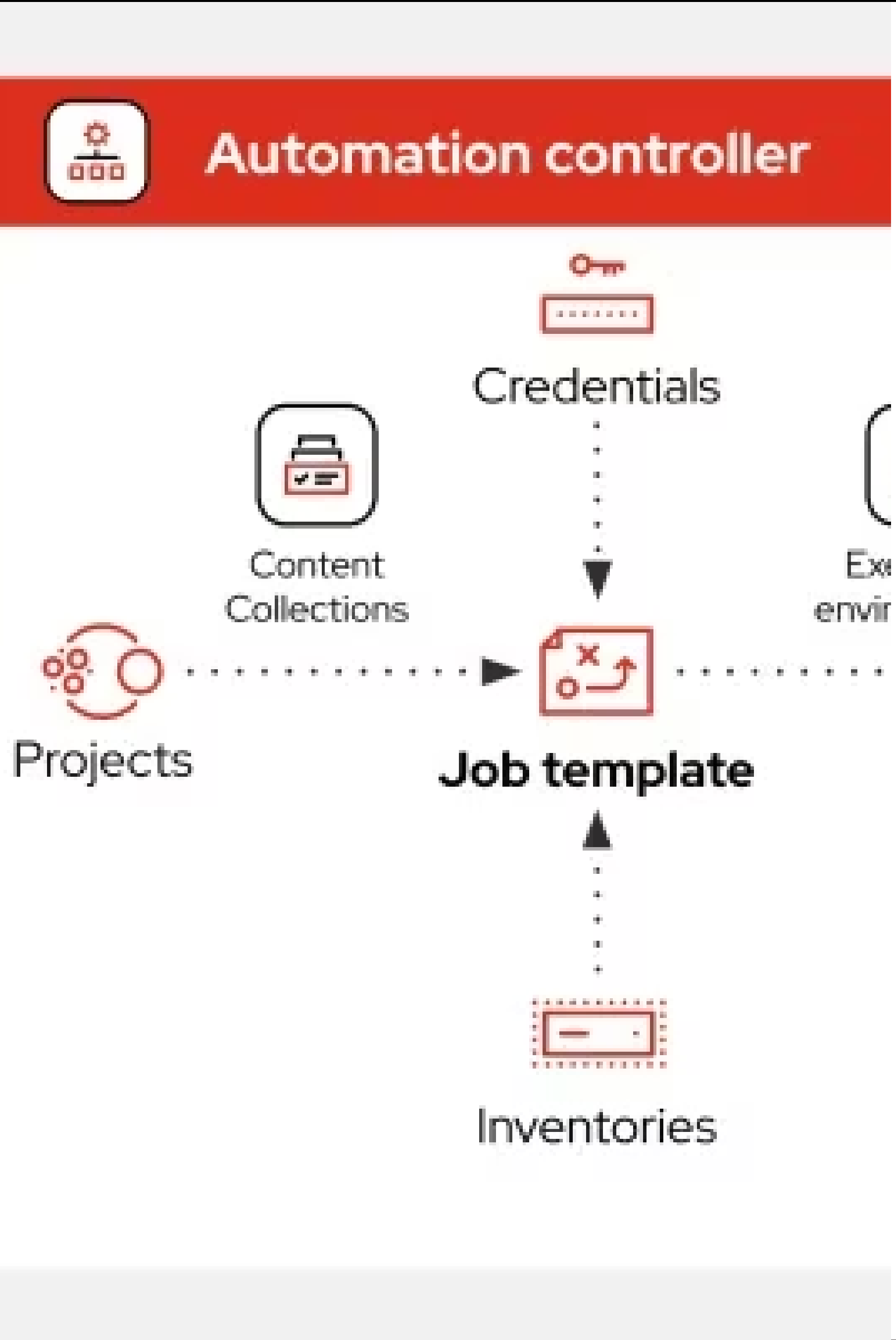
ITSM Integration

Automatically create and enrich ServiceNow incidents when infrastructure anomalies are detected — reducing MTTR dramatically.



Self-Healing Infrastructure

Restart crashed services, re-apply configuration drift, or re-provision failed nodes without manual operator involvement.



How All Components Work Together

AAP's five components aren't independent tools — they form a **tightly integrated automation platform**. Each component plays a precise role in delivering secure, scalable, and governed automation at enterprise scale.

Controller

Orchestrates and governs

Hub

Supplies certified content

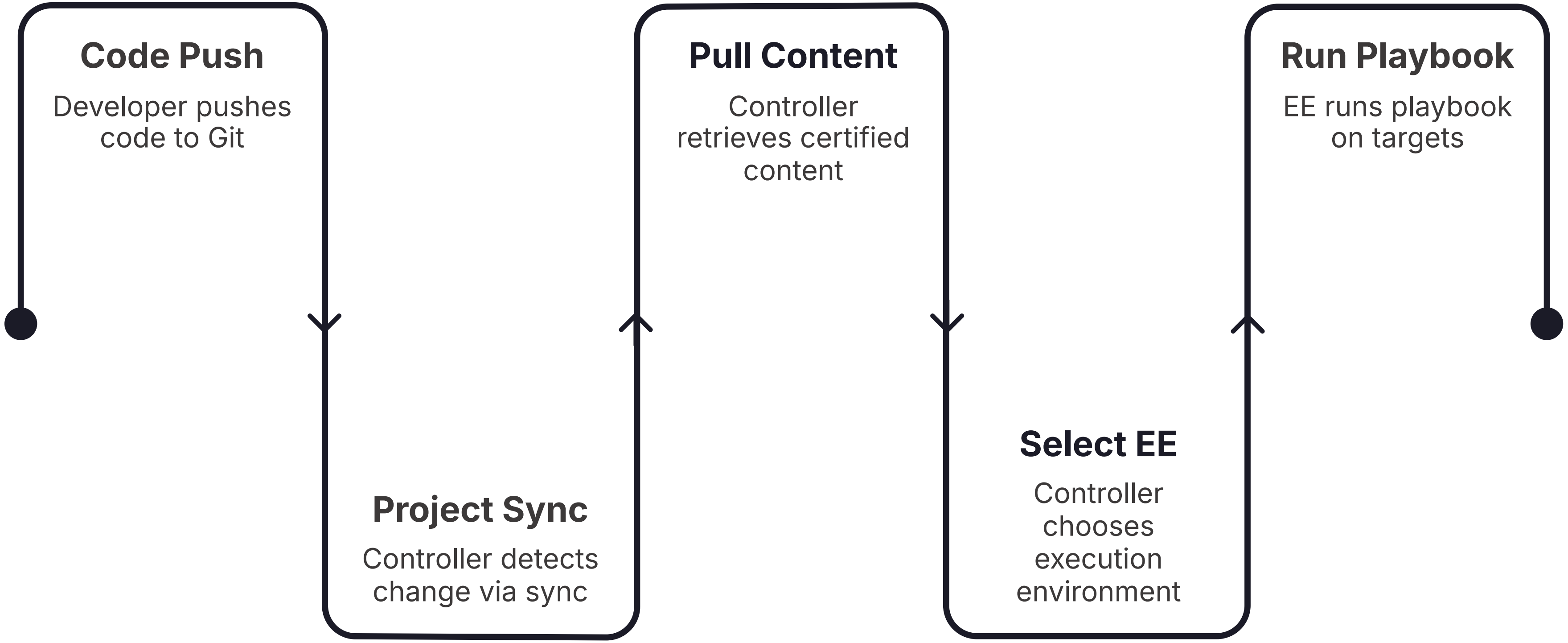
EE

Provides consistent runtime

EDA

Triggers reactive automation

End-to-End Workflow: Putting It All Together



Code Push

Developer pushes code to Git

Project Sync

Controller detects change via sync

Pull Content

Controller retrieves certified content

Select EE

Controller chooses execution environment

Run Playbook

EE runs playbook on targets

Key Takeaway: Every AAP component contributes a distinct function — governance, content, runtime, reactivity — yet they operate as a single, unified automation platform. This integrated architecture is what separates AAP from simple CLI-based Ansible usage.