# Red Hat Decision Manager 7.6

## Policy-based decision and control in ONAP using Ansible and Red Hat Decision Manager

Last Updated: 2020-05-22

# Red Hat Decision Manager 7.6 Policy-based decision and control in ONAP using Ansible and Red Hat Decision Manager

Red Hat Customer Content Services
brms-docs@redhat.com

## Legal Notice

Copyright © 2020 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## Abstract

This document describes the use of Ansible and Red Hat Decision Manager technology in Open Network Architecture Platform (ONAP) architecture.

# Table of Contents

# PREFACE

This document describes how to use Ansible and Red Hat Decision Manager technology in Open Network Architecture Platform (ONAP) architecture. This explains how automation and rules management systems can provide a comprehensive network policy engine. The proposed policy engine provides the necessary automations for proactive response to network and service conditions without human intervention. This is also known as closed loop automation. ONAP is a open source networking project that provides a platform for real-time, policy-driven orchestration and automation of physical and virtual network functions. ONAP enables service providers to rapidly automate new services and support complete lifecycle management.

# CHAPTER 1. TECHNOLOGY AND DEFINITIONS

## 1.1. OPEN NETWORK AUTOMATION PLATFORM

Open Network Automation Platform (ONAP) provides orchestration and automation capabilities for physical as well as virtual network functions. It reduces the cost and time needed to implement new service offerings by exploiting new paradigms of software defined networking as well as network function virtualization. ONAP provides the following capabilities:

- Builds service by modeling resources and their relationships.
- Automates the instantiation of services based on policies.
- Monitors the services using an analytics framework.
- Automates the life cycle management actions on services and underlying resources based on events and policies.

## 1.2. POLICY

You can define a policy as a condition, requirement, constraint, decision or a need that must be provided, evaluated, maintained, and enforced. A policy can also be defined at a lower or functional level, such as a machine-readable rule or software condition or assertion which enables actions to be taken based on a trigger or request, specific to particular selected conditions in effect at that time.

The following table lists the policies that affect networking.

**Table 1.1. Policies affecting networking**

| Policy | Description |
|---|---|
| Virtual Network Function (VNF) placement | Rules governing where to place VNFs, including affinity rules |
| Data and feed management | What data to collect and when, their retention periods, and when and where to send events about issues |
| Access control | Who all can have access to which data |
| Trigger conditions and actions | Determine which conditions are actionable, and define what to do under those conditions |
| Interactions | How to handle the interactions between change management and fault and performance management |

# CHAPTER 2. RULES ENGINES AND RED HAT DECISION MANAGER

A rules engine is a software element that makes decisions based on business logic built using declarative format. Rules engines can be very basic (providing support for simple rules) or very complicated (based on sophisticated algorithms).

Red Hat Decision Manager provides you with a flexible set of decision capabilities, including a rules engine based on the PHREAK algorithm (an evolution of the well known Rete algorithm), a DMN Compliance Level 3 runtime, support for PMML, and a Complex Event Processing (CEP) engine.

The PHREAK reasoning algorithm runtime can easily scale to hundreds of thousands of rules in a single rules execution environment, while providing low-latency and high-performance business rules execution.

Because it is a lightweight engine, you can use Red Hat Decision Manager as a decision service and runtime in various architectures and deployment topologies, including, embedded systems and microservices architectures. You can combine the decision engine with multiple application frameworks and runtimes.

Red Hat Decision Manager is built on open standards (DMN, XML, JSON, JAX-RS) and de-facto standards (Git, Maven, Eclipse, IntelliJ). It has the following capabilities:

- Automates decision making
- Incorporates sophisticated decision logic easily
- Separates decision logic from program code and defines it in simple, declarative, business-friendly terms, making it easier and efficient to implement, manage, audit, and change
- Provides a business expert friendly and developer friendly UI to collaborate effectively to build policies

These capabilities facilitate improved business agility, consistent and efficient decision execution, shorter development cycles, and faster time to market.

# CHAPTER 3. AUTOMATION AND ANSIBLE

Ansible is an automation engine that completely automates network functions, cloud provisioning, configuration management, application deployment, intra-service orchestration, and other automation needs.

Ansible is designed for multi-tier deployments because it models your IT environment by describing how all of your systems interrelate, rather than just managing one system at a time.

Ansible is easy to deploy because it uses no agents or additional custom security infrastructure. Additionally, it uses the YAML language, in the form of Ansible Playbooks, which lets you describe the automation jobs in a language similar to English.

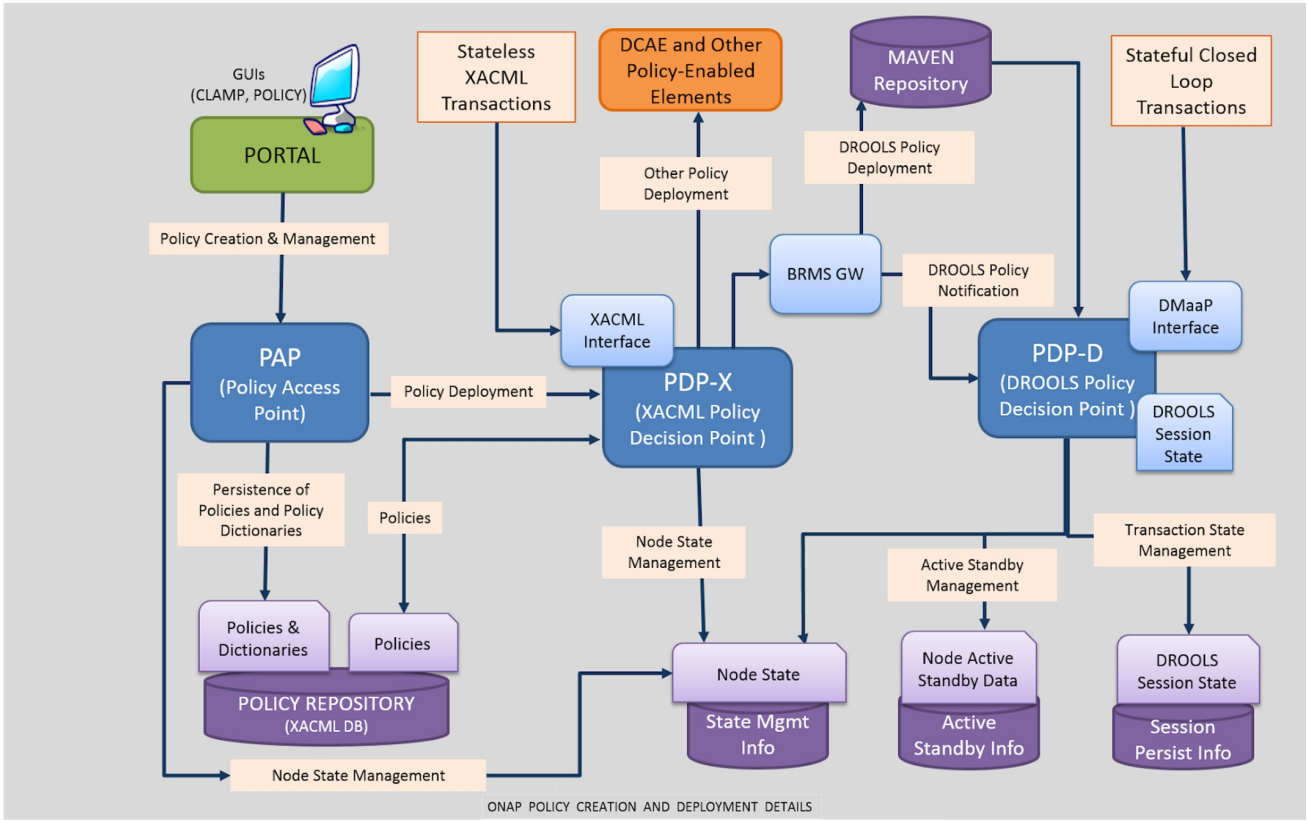
Ansible engine has two additional optional components:

- Ansible Tower: Red Hat Ansible Tower enables you to scale IT automation, manage complex deployments, and increase your organization's productivity. You can centralize and control your organization's IT infrastructure by using a visual dashboard, role-based access control, job scheduling, integrated notifications, and graphical inventory management. You can also embed Ansible Tower into existing tools and processes by using its REST API and CLI.

- Ansible Networking Add-on: When Ansible Engine is part of your automation strategy, the Networking Add-on provides support for select network modules that are developed, tested, and released in-house for popular networking platforms. Network modules ensure that operators and engineers have peace of mind, especially in mixed or hybrid network environments that can have differing needs and implementations.

# CHAPTER 4. POLICY WITH RED HAT DECISION MANAGER

**Figure 4.1. ONAP policy creation and deployment**




ONAP policy framework (also known as POLICY) consists of the following subcomponents:

- Policy Administration Point (PAP) provides interfaces for policy creation. The interfaces are integrated with a portal to provide a GUI.
- Policy Decision Point (PDP) is based on a specific rules technology. It has two components:
  - PDP-X is based on XACML technology. PDP-X is *stateless* and can be deployed as a resource pool of PDP-X servers. You can scale up the number of servers to increase both capacity (horizontal scalability) and availability.
  - PDP-D is based on Red Hat Decision Manager technology. PDP-D is *stateful* and uses Red Hat Decision Manager in its native, stateful way. The transactions persist as long as PDP-D is active. It provides the following capabilities:
    - Advanced control loops
    - Interfaces for various ONAP components to trigger actions and receive events
    - Maintains state throughout work-flows across the network while handling failures in the corrective actions
- Policy Enforcement Point (PEP) is where runtime policy enforcement is performed by ONAP subsystems that are policy-enabled or can respond to commands from a policy-enabled element such as a PDP. These subsystems include:
  - Master Services Orchestrator (MSO) provides orchestration at a very high level, with an end to end view of the infrastructure, network, and application scopes.

    - Active and Available Inventory (AAI) is the ONAP subsystem that provides real-time views of active, available, and allocated resources and services and their relationships.

    - Data Collection, Analytics, and Events (DCAE) subsystem, together with other ONAP components, gathers information regarding performance, usage, and configuration from the managed environment. Various analytic applications then use this data to detect any anomaly or significant events. Based on the analysis results, actions are triggered such as publishing to other ONAP components such as policy, MSO or controllers.

The main functions of the DCAE subsystem are as follows:

- Collect, analyze, transform, and store data for analysis.

- Provide a framework for analytics development.

These functions enable closed-loop responses by various ONAP components to events or other conditions in the network.

- Controllers : A controller is used to manage the state of a set of sub-domain specific resources (for example, an application or network). It executes the resource's configuration and instantiation, and is the primary agent in ongoing management such as control loop actions, migration, and scaling. ONAP uses the following controller types to manage resources in the execution environment, corresponding to their assigned controlled domain:

    - Network controller (Network configuration): A network controller (for example, a SDNC/software defined network controller), manages and controls network functions and services by carrying out its network configuration workflow and reporting the resulting status to MSO or AAI. Examples of Controlled Network Elements and services include those for Transport Network Functions, infrastructure networking (for instance, leaf, spine, and virtual switches), and Wide-Area-Networks (WANs).

    - Application controller (Application): Application controllers, such as APPC, are responsible for the lifecycle management of VFNs. The APPC HTTP API supports lifecycle management commands, enabling users to manage virtual applications and their components through other ONAP components.

Both of these controllers are based on an OpenDaylight Controller framework.

After a policy has been initially created or an existing policy has been modified, the Policy Distribution Framework sends the policy from the repository to its points of use, which include Policy Decision Points (PDPs) and Policy enforcement points before the policy is actually needed.

Examples of policy enforcement can include the following:

- Policy rules for data collection and retention by the DCAE data collection functionality.

- Analytic policy rules, identification of anomalous or abnormal conditions, and publication of events signaling detection of such conditions by DCAE analytic applications.

- Policy rules for associated remedial actions, or for further diagnostics, are enforced by the correct component in a control loop such as the MSO, a controller, or DCAE.

- Policy engines such as XACML and Red Hat Decision Manager also enforce policies and can trigger other components as a result (for example, causing a controller to take specific actions specified by the policy). Additionally, some policies (Guard Policies) may enforce checks against decided actions.

The following steps illustrate a sample policy flow:

1. A new data flow is intercepted by the Policy Enforcement Point (PEP).
2. The PEP forwards the request to the Policy Decision Point (PDP).
3. The PDP evaluates the request against the policies it is configured with.
4. The PDP reaches a decision (Permit / Deny / Not Applicable / Indeterminate) and returns it to the PEP.

# CHAPTER 5. AUTOMATION WITH ANSIBLE

Ansible provides an easy to implement automation mechanism for various physical as well as virtual network functions using protocols such as **netconf** or **cli**. As currently used in ONAP under APPC architecture, Ansible provides the VNF management framework that enables an almost CLI like set of tools in a structured form. It is agentless, which means that the target VNF does not require any additional software. This construct enables management of any VNF in a consistent manner whether it supports a standard interface or protocol such as **netconf** or not. Any action (for example configure, restart, and health check) can be executed on the VNF by constructing a *playbook* (or *set of playbooks*) that is executed by an Ansible on the VNF through SSH.

The Ansible Extension for APP-C allows management of VNFs through the following architecture:

- Ansible Directed Graph (DG) - The Ansible Directed graph is a generic directed graph that you can use to invoke any playbook through Ansible (and therefore any APP-C action, since in Ansible, VNF actions map to playbooks) corresponding to an LCM action.

- APP-C Ansible Adapter - The ansible adapter is an OSGI bundle in the APP-C Karaf container that interacts with the Ansible server. It is a set of REST calls that performs two actions. It first submits a request for a playbook to be executed, and second, if required it gets the results of the playbook after execution (if in synchronous mode).

- APP-C/Ansible server interface - Ansible libraries are written in Python and therefore cannot be executed natively from within the APP-C Karaf container. Instead, the design calls for an Ansible Server that can execute the Ansible playbooks and exposes a **REST** interface that is compliant with requirements of APP-C. These requirements are documented as the Server API Interface that any compliant Ansible Server must support. Exact implementation of the Ansible Server is left open and does not affect APP-C operations as long as the server follows the interface. For purposes of evaluation, a reference web server that implements this APP-C/Ansible Server interface has been developed and the code is available from the App-C ONAP repository under the **appc-adapters/appc-ansible-adapter/appc-ansible-example-server** path.

For an illustration of the workflow when an application controller receives an event, see the APPC Ansible Adapter page.

# CHAPTER 6. CLOSED LOOP WITH OPNFV COMPONENTS

Barometer is an OPNFV project that develops features in **collectd** to collect metrics and events suitable for NFV deployments.

Virtual function Event Stream (VES) is a ONAP project that provides a common model and format for use by NFV Service Providers (SPs) for managing VNF health and lifecycle. The project goal is to reduce the effort to develop and integrate telemetry data from various sources (primarily NFVI, VNFs and PNFs) into automated policy based management systems, by moving towards a common event stream format and collection system.

The Barometer VES Plug-in consumes the **collectd** events and forwards them in VES format to a pre-configured VES/DCAE collector.

The DCAE collector, which receives events and metrics in the VES format, publishes them on the ONAP message bus, and saves them in a database.

# CHAPTER 7. REFERENCES

- *ONAP Case Solution Architecture* (requires a Linux Foundation account for login)
- *AT&T ECOMP Whitepaper*
- *ONAP Policy architecture*
- *ONAP Policy framework Project*
- *Controllers*
- *ONS F2F Casablanca planning (see policy)*
- *OPNFV Barometer project*
- *Barometer: Taking the pressure off of assurance and resource contention scenarios for NFVI*
- *Drools Community Documentation 5.3.0*
- *APPC Ansible Adapter*
- *APPC User Guide*
- *ONAP Project Specific Breakouts*
- *VES Home*
- *VNF Event Stream*
- Network functions
- Cloud provisioning
- Configuration management
- Application deployment
- Intra-service orchestration
- PHREAK algorithm
- DMN v1.2 FEEL Compliance Level 3
- PMML

# APPENDIX A. VERSIONING INFORMATION

Documentation last updated on Friday, May 22, 2020.