OpenShift Reference Architecture

A Guide for Implementing Kubernetes Admission Control and Application Communication Authorization with Styra DAS, Styra Load, and OPA

Reference Architecture Objectives

Whether companies are working with traditional, modernized, or cloud-native applications, Red Hat OpenShift is a leading platform to build and deploy applications at scale on Kubernetes. Meanwhile, Styra Declarative Authorization Service (DAS), Styra Load, and Open Policy Agent (OPA) provide the leading suite of commercial and open-source tools to implement policy-as-code authorization and admission control for cloud-native applications.

This Styra OpenShift Reference Architecture provides OpenShift architects and administrators with guidance to implement admission control and application communication authorization using Styra DAS, Styra Load, and OPA. The document details key definitions and the most important concepts for using Styra DAS as your Policy-as-Code management platform.

Below, we'll first discuss the key architectural components of any real-world authorization decision. These focus on the ways that policies and data come together to make authorization decisions, as well as how policies can be centrally managed and organized to satisfy even complex requirements in heterogenous environments at scale.

Then, we will delve into the heart of the piece: The use cases and deployment architectures for policy as code within Red Hat OpenShift.

Policy as Code Architecture

Policy Enforcement Point and Policy Decision Point
A Policy Enforcement Point (PEP) is a component that intercepts requests and enforces policies on them. A Policy Decision Point (PDP) is a component that evaluates policies and returns decisions to the PEP. OPA is a general-purpose policy engine that can act as a PDP for various use cases. One of them is Kubernetes admission control, where OPA can validate and mutate Kubernetes resources before they are created or updated. Another one is service mesh traffic authorization, where OPA can authorize requests between microservices based on attributes such as source, destination, method, path, etc. Styra Load is Styra's enterprise OPA distribution with additional features and performance improvements.

Policy Control Plane
Styra DAS is a platform that enables users to manage policies across different systems and applications. It acts as a central control plane for OPA and Styra Load. Styra DAS provides a unified interface to create, edit, and monitor policies for OPA and Styra Load instances running in different environments, such as Kubernetes, Terraform, or HTTP APIs. Styra DAS also offers features such as policy testing, debugging, and analysis to help users ensure their policies are correct and effective.

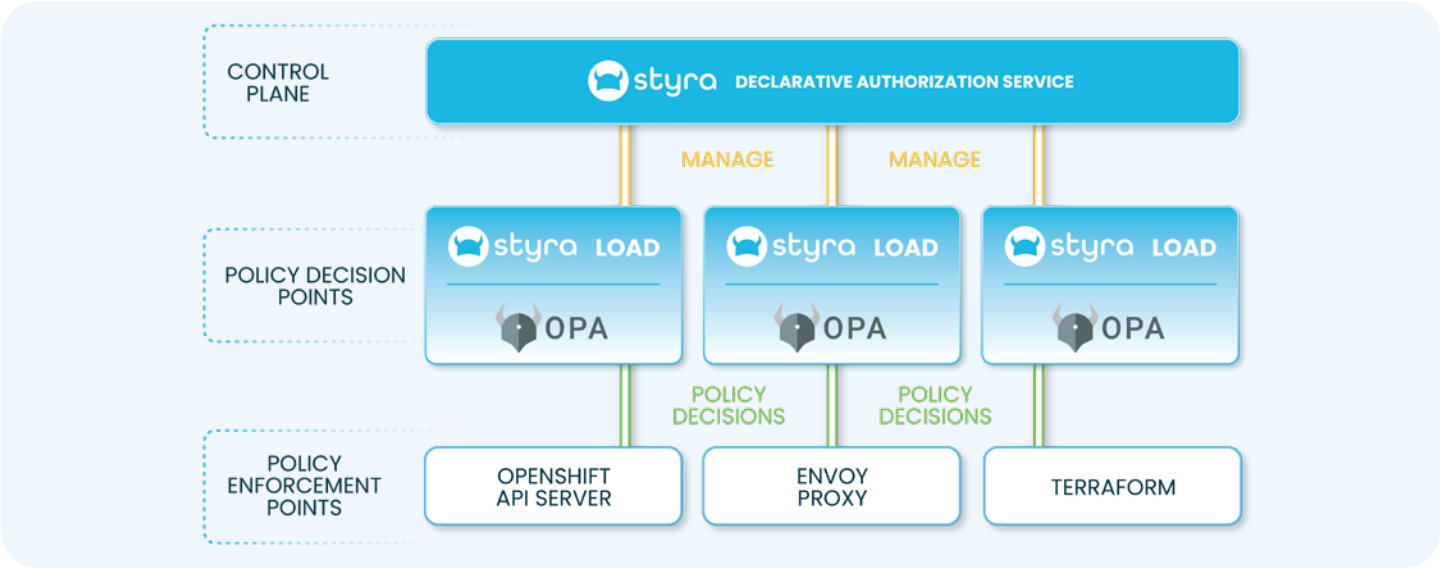


Diagram 1: The three layers of policy control.

Systems, Stacks and Libraries in Styra DAS
A System in Styra DAS allows a team to manage OPA or Styra Load agents that have been integrated into a real-world software system: what policies should be used, what decisions have been made, are the agents healthy, and so on. The real world system can be an OpenShift or Kubernetes cluster, a service-mesh, or a custom application.

The same is true of an OpenShift Service Mesh instance. It maps one-to-one to a Styra DAS System of type Istio. As OpenShift Service Mesh is based on Istio it can be handled by the same type of policies and agent deployment. There are a few quirks that need to be taken into account when using Styra DAS's Istio system with Service Mesh; we will discuss these later.

Stacks are the next layer of policy management. A Stack also serves as a collection of rules. However, instead of controlling one cluster or mesh directly, a Stack oversees multiple Systems that share a specific type (example: Kubernetes), a common function (example: production) or contract (example: PCI compliance). When you define a stack rule, it applies to all the Systems the Stack manages. The rule is synchronized with all corresponding OPA or Styra Load instances, and it is monitored or enforced.

Finally Libraries are a mechanism that enables teams to share policies and policy fragments across the enterprise. The policies in the library can be written for any use case by any team. Styra DAS Systems and Styra DAS Stacks can use the policy library.

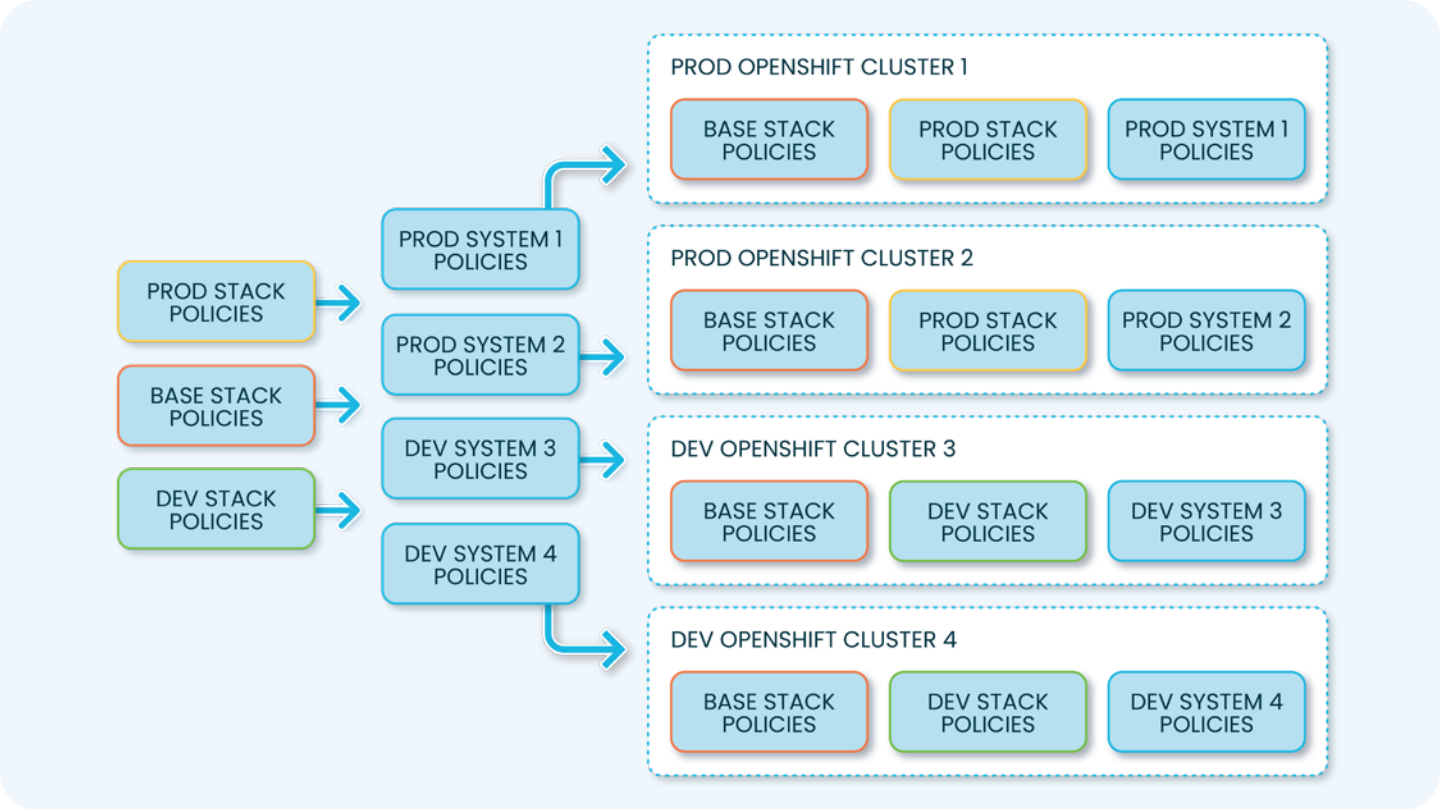


Diagram 2: Using Stacks and Systems for applying policies to OpenShift clusters

OpenShift Use-cases

Admission Control
Styra DAS installs agents (OPA or Styra Load) to OpenShift and configures the API server to use the agent as an Admission Control Webhook. An OpenShift cluster will be represented in Styra DAS by a System of type Kubernetes.

The System is responsible for building the Policy Bundle that contains all the admission control policies for the given cluster (this includes Rego policy from Stacks and Libraries additionally to the System-specific policy). The Policy Bundle will also contain any data needed for policy decisions (e.g. a list of on-call engineers allowed to make production changes).

A System is always associated with a single OpenShift cluster. There will be multiple OPA or Styra Load agents deployed for high availability, but each of those will be connected to the same System in Styra DAS. Admission control policies on an OpenShift cluster will be managed by a System of type “Kubernetes”. This is because the admission control and deployment mechanism is the same for Kubernetes as for OpenShift.

OpenShift Cluster Deployment
Styra DAS provides several methods to deploy OPA or Styra Load agents to an OpenShift cluster:

- A YAML file that can be used with `kubectl apply`
- A Helm 2 or Helm 3 chart
- A Kustomize file

A user can pick which of these methods is the most appropriate for their environment.

The deployment will install the following components all of which run as separate containers in the same Pod. The Pod itself is managed by a StatefulSet and scaled to 3 instances by default:

- OPA or Styra Load in an HA configuration with a Kubernetes Service providing load balancing.
- Styra Local Plane (SLP): Provides local caching of the policy bundle. It uses a Persistent Volume to store the bundle in case the Pod needs to restart and Styra DAS is not accessible. Also imports cluster configuration data into the agent to be used during policy evaluation.
- Datasources Agent: Uploads cluster configuration data to Styra DAS to be used for policy authoring.

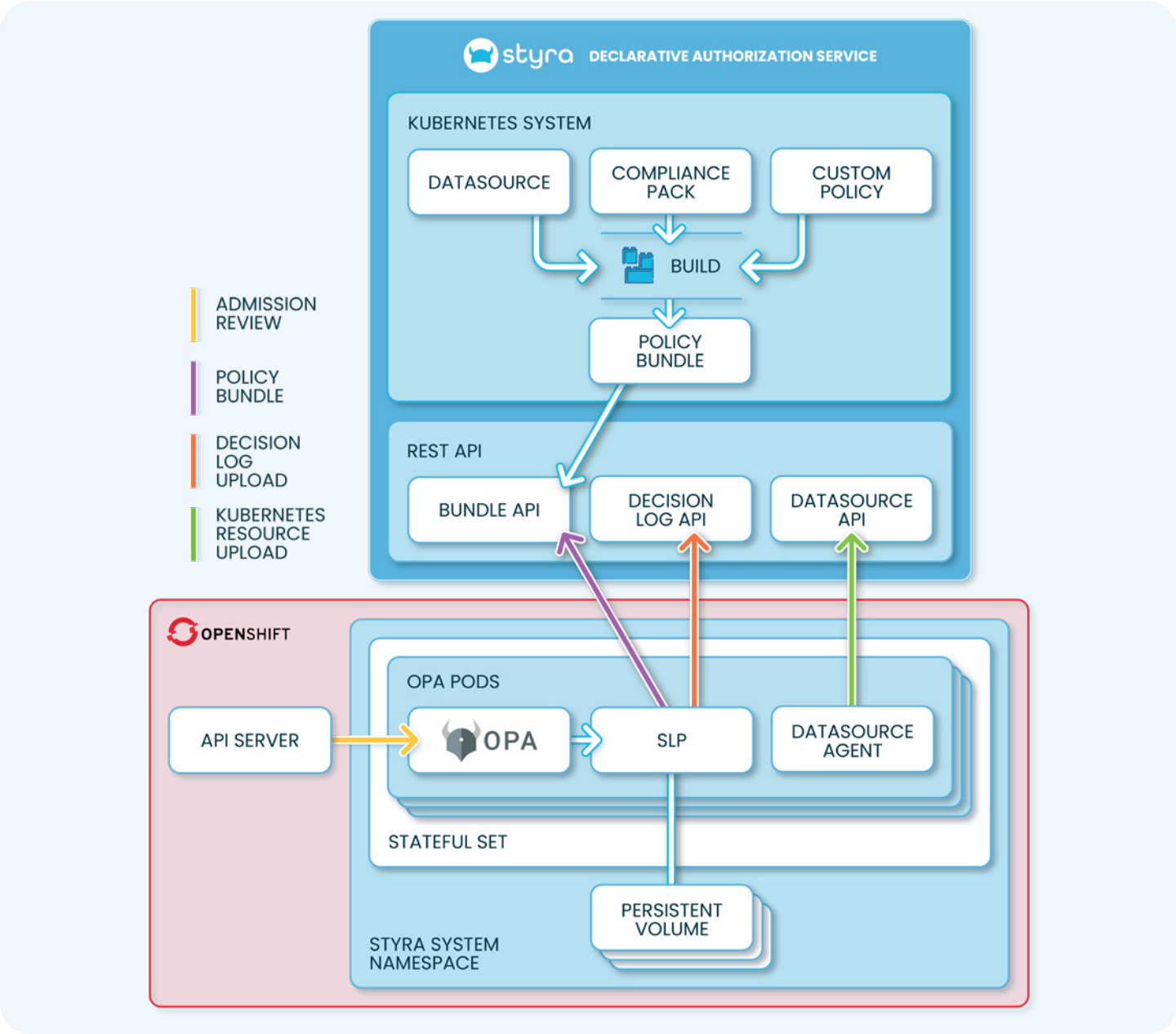


Diagram 3: Deployment of Styra components and Rego policy on an OpenShift cluster.

What's in a Bundle?
Styra DAS builds Rego policy bundles for OPA or Styra Load to download and execute. These bundles will contain all the runtime information necessary to perform policy decisions. When OPA or Styra Load request the latest bundle they always supply the ID of the System. This means an agent will always be connected to a single System.

Styra DAS will assemble a policy bundle from the following sources:

- Rego & JSON data files defined in the System level.
- Rego & JSON data files defined in any Stack that applies to the System.
- Rego & JSON data files defined in any Library that is referenced by the System.

Whenever any of the Rego policies or data gets updated, Styra DAS will build a new bundle. If the System is configured for automatic bundle deployment, each agent will receive this bundle the next time check the Bundle API for updates. If the System is configured for manual bundle deployment, a user action (either using the UI or the API) is needed before the bundle becomes available for download.

When the policy needs external data, this data needs to be imported into Styra DAS first (see the Policy Data Architectures section for more details).

Service Mesh Authorization

OpenShift Service Mesh is built on top of Istio, which supports connecting an external authorization service to the Envoy proxy that processes network traffic flowing through the mesh. In this architecture Styra DAS installs OPA or Styra Load as an external authorizer to the Envoy proxy attached as a sidecar to each application container. This allows Rego policy to control what traffic is allowed to flow in and out of each service on the mesh. Each Envoy proxy will forward each request to the agent, which can make a policy decision.

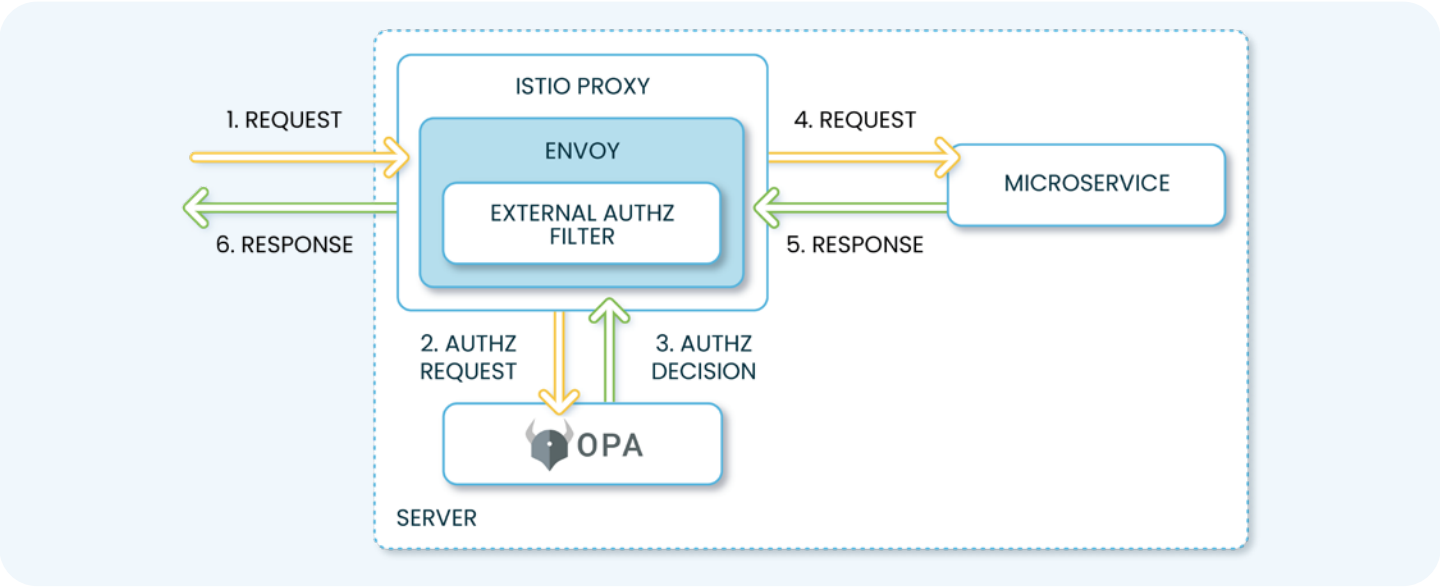


Diagram 4: Integrating OPA with Envoy for microservice traffic authorization.

Leveraging OPA's distributed architecture we can deploy an agent to run as a sidecar next to each Envoy proxy. This provides strong availability guarantees and low latency for policy evaluation at the cost of increased memory usage by the agent.

Styra DAS provides support for OpenShift Service Mesh with the Istio system type. As Service Mesh is built on top of Istio it doesn't require special handling in Styra DAS.

OpenShift Cluster Deployment
Deployment of the Istio System cluster local components is done on a per-namespace basis. This means the user has to run the installation script for each namespace they wish to be policy controlled by Styra DAS.

The installation steps will add the following to the target namespace:

- SLP deployment
- SLP configuration for connecting to the Styra DAS System
- OPA/Load configuration for connecting to SLP

Additionally the user has to add the OPA or Styra Load agent to the applications' manifest files to run as another sidecar container in the Pod.

Multi-cluster Deployment With RACM

Red Hat Advanced Cluster Management (RACM) allows efficient management of multi-cluster environments. Users of Styra DAS will need to create a System to represent each cluster and install the cluster-local components. RACM allows fully automating this process.

To achieve this automation we can utilize RACM's policy framework. We can define policies that mandate Styra DAS components to be installed on every cluster as well as a Styra DAS System to be created for each cluster.

Using a RACM policy we can run a Job on each managed cluster that will connect to Styra DAS and create a corresponding System (of type Kubernetes). Once the System is created the Job can download and run the installation script provided by the System to install the cluster-local components like the Styra Local Plane, OPA and the Datasources agent.

You can find a template policy in our Github repository that you can modify to suit your needs. The repository contains detailed usage instructions.

Our policy will create the following resources:

- The styra-system Namespace.
- A Secret with the credentials to the Styra DAS API.
- A Job which creates a System for the cluster in Styra DAS and runs the installation script for the System.
- An image build that creates the necessary Docker image for the Job's Pods.
- Verification for cluster-local component Pods to be up and running.

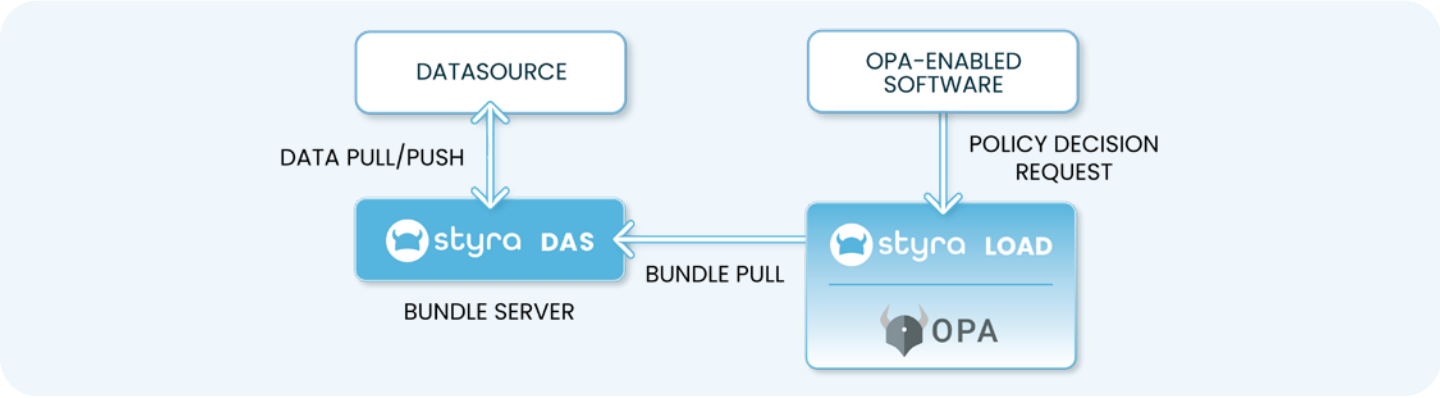
The policy will have to be configured on the Hub cluster which manages any clusters created with or imported into RACM. Once the policy is configured RACM takes care to run all the necessary automation on each managed cluster. The same is true when the policy gets updated.

You can find the policy file and the related Dockerfile in this repository.

Policy Data Architectures

Data is a crucial part of many authorization scenarios. Some might have little to no data and only rely on policy rules while others like RBAC are heavily data-based, assigning each user a set of permissions that have to be evaluated. The OPA or Styra Load agent has to have access to the necessary data during policy evaluation. There are several distinct ways we can achieve data distribution to our agents.

Bundle-based Data Distribution
OPA policy bundles can include data in the form of JSON files. This feature can be leveraged to manage data sources centrally in Styra DAS and push data to OPA or Styra Load using the bundle update mechanism.



Example: Certain namespaces on OpenShift can only be modified outside of office hours. The list of these protected namespaces is stored in a JSON file in an S3 bucket.

Styra DAS can poll this bucket for updates to the data file and pull the changes if there are any. Whenever there is a change in the data Styra DAS will build a new bundle and distribute it to the agents.

Pros:

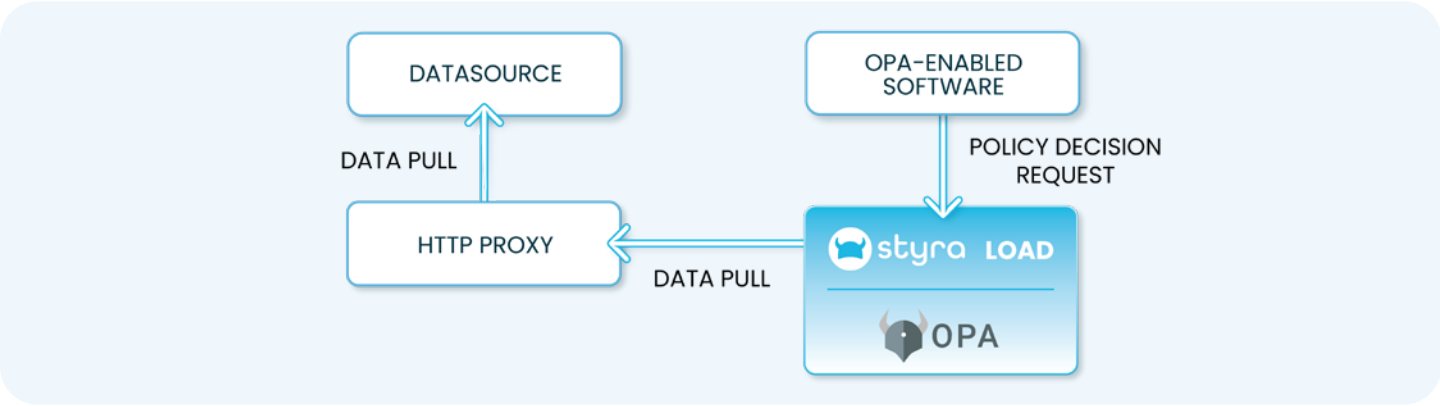
- Data is managed centrally in Styra DAS, which makes it visible and auditable.
- Data is available to support policy authoring in Styra DAS.

Cons:

- The bundle update frequency is a bottleneck for data distribution latency.
- The datasource might be close to the agent, thus the roundtrip to Styra DAS adds to latency and network costs.

Pull Data During Policy Evaluation

This method uses the agent's built-in Rego method http.send to pull data during policy evaluation. OPA or Styra Load will directly access the datasource or an HTTP proxy in front of it and retrieve the data. The data will be local to the single evaluation during which it is retrieved.



Example: A policy authorizing traffic over Service Mesh to a microservice needs the list of permissions the user has. These are stored in Active Directory. A custom service connects to AD over LDAP and exposes user permission data over HTTP

When the policy is evaluated the agent will get the current user from the JWT token stored in the HTTP request it is authorizing. The policy will then call http.send to connect to the HTTP Proxy service and retrieve the permission data. Once this is done the policy will use the data to make the authorization decision.

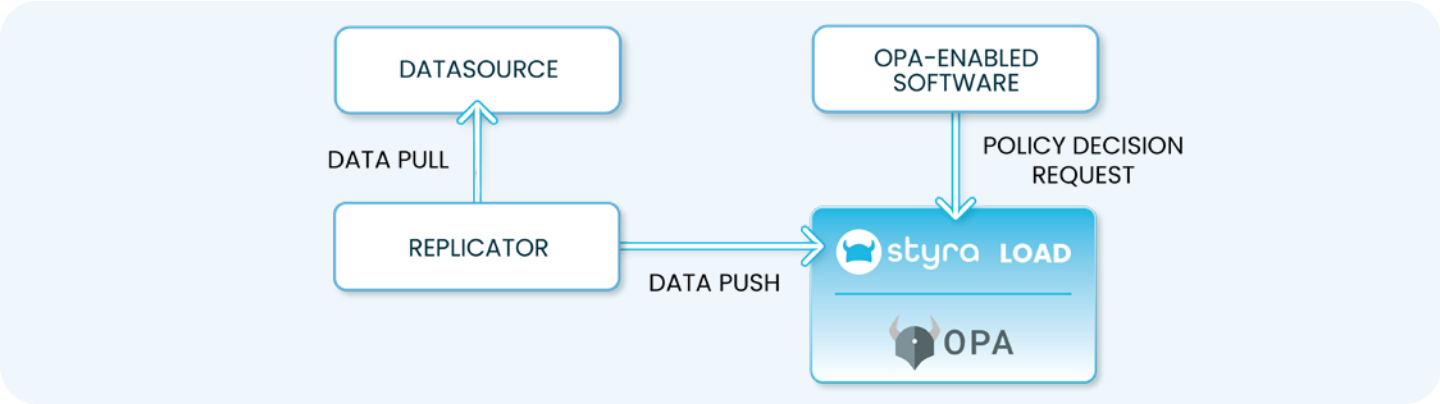
Pros:

- Data is freshly retrieved at the moment of policy evaluation.
- Can scope data requests to what's needed for a single policy decision. E.g.: Loading only a single user's data instead of keeping a local copy of all users' data in the agent.

Cons:

- Introduces network latency to policy evaluation.
- Policy evaluation can fail if service dependency is not available on the network.
- Agent needs to manage credentials to external services.
- Unless the datasource itself provides an HTTP interface a proxy needs to be created.

Push to Agent
This method uses a replicator service, which polls the data source and pushes data to OPA or Styra Load using the Data API. The replicator is usually a simple custom written service.



Example: A policy authorizing traffic over Service Mesh to a microservice needs the list of permissions the user has. These are stored in Active Directory. A replicator service is created that checks for data changes in AD every minute and pushes updated data into all agents. The agents assume they have up to date data on all users in memory.

The replicator can keep the data in the agent fresh, even near-real-time. How well it can be implemented depends on the datasource. A datasource that allows listening to data change events will allow a replicator that can immediately update data in the agents.

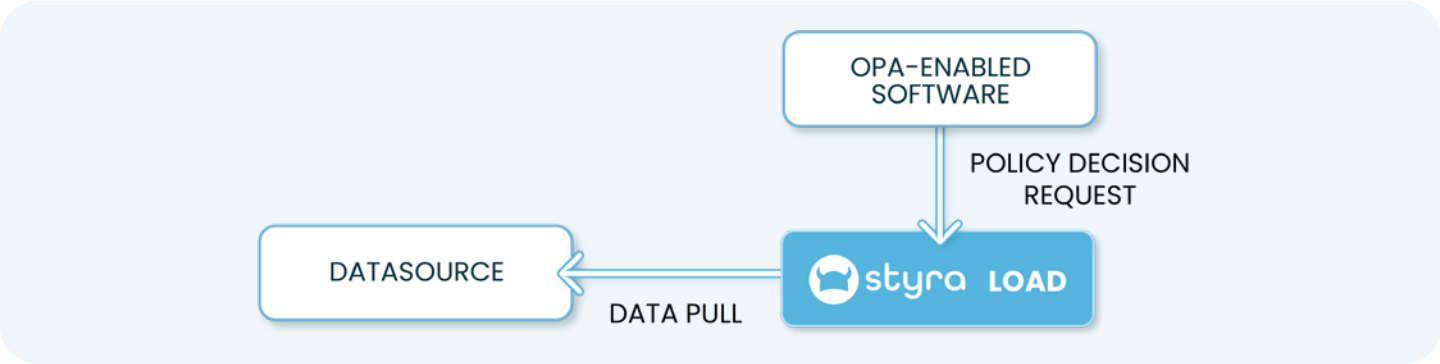
Pros:

- Data in agents can be kept up to date without resorting to network calls during evaluation.
- Policy evaluation is not blocked by data retrieval

Cons:

- A replicator service has to be created for each datasource
- Not all data sources will support efficient retrieval of data updates
- The replicator needs to know all agent API endpoints and relay the data to each of them.

Pull by Agent (Styra Load)
Styra Load contains functionality to pull data directly from different data sources. This is done in the background, not during policy evaluation. This leads to a more efficient version of the "Push to agent" method because it doesn't require a replicator. The replicator is built into the Styra Load agent itself.



Example: A policy authorizing traffic over Service Mesh to a microservice needs the list of permissions the user has. These are stored in Active Directory, but each data update is also put on a Kafka topic for other services to be notified. Styra Load is configured to connect to the Kafka topic and consume data updates in real time.

Pros:

- No replicator service needs to be created.
- Each agent can pull data independently.
- Real time data updates are possible if the data source allows it.

Cons:

- The datasource will have to serve each agent instance. If the number of instances grows too large they can overwhelm the datasource.

Self-hosting Styra DAS

Styra DAS can be used both as a SaaS product and a self-hosted application. When run in self-hosted mode Styra DAS is a Kubernetes-native product that can be deployed using Helm charts or custom installation methods. The customer is responsible for managing the Kubernetes cluster, the data storage services (Elasticsearch and PostgreSQL), and the image registries. Styra is responsible for providing the Styra DAS images, documentation, and support.

Installation is provided in the form of a Styra Helm chart and a set of Docker images. The Helm chart is hosted here in the Styra Helm repository. The Docker images are hosted here in the Styra image registry.

Customers can choose to configure their clusters to pull images from the Styra registry or transfer the images to their own registry. In both cases authentication with registry.styra.com will be necessary with the registry username and password provided by the managed Styra DAS tenant.

Styra DAS requires an OpenShift cluster with at least 4 nodes and 16 GB of memory each. The cluster should have access to the internet or a proxy for license validation and updates. The cluster should also have persistent volumes enabled for data storage.

Styra DAS relies on Elasticsearch and PostgreSQL for data storage. The customer can choose to run these services in-cluster or externally. Managed offerings for both are supported.

For more details consult the Styra DAS Self-hosted guide.

Conclusion

Styra DAS together with OPA and Styra Load provide a perfect policy-as-code solution for both OpenShift admission control and Service Mesh traffic authorization. It combines the versatility of OPA's distributed deployment model with centralized policy management. This way DevOps, security and compliance teams can fully secure their OpenShift deployed applications using a single technology stack.

For more information on how to get started with Styra DAS for OpenShift, set up a demo with our stellar solutions architects.

About Styra

Styra enables enterprises to define, enforce and monitor policy across their cloud-native environments. With a combination of open source (Open Policy Agent) and commercial products (Styra DAS and Styra Load), Styra provides policy-as-code authorization, enabling security, operations, and compliance to protect applications and infrastructure. **Learn more at** www.styra.com

styra