Cyral | WHITE PAPER

10 Reasons Why it is OK to Hate Database Proxies, but Love Sidecars!

Contents

10 Reasons Why it is OK to Hate Database Proxies, but Love Sidecars! 3

Designing Data Layer Sidecar for the Cloud Native World 5

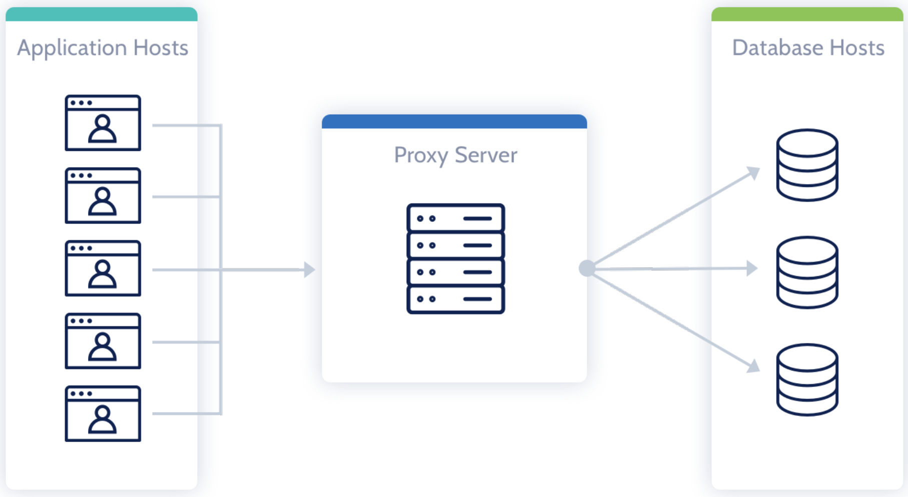
Comparison of Traditional Database Proxy and Data Layer Sidecar 7

10 Reasons Why it is OK to Hate Database Proxies, but Love Sidecars!

A brief history of the Database Proxy

A proxy is an interception service that sits between the client and the server. When the proxy is deployed close to the client, it is called a forward proxy. When the proxy is deployed closer to the server such that the clients do not know about the origin of the server, it is known as a reverse proxy.

A database proxy (ProxySQL, MaxScale, etc.) is basically a reverse proxy built to provide benefits like security, scalability, high availability, etc. for databases, key-value stores, message queues, etc.



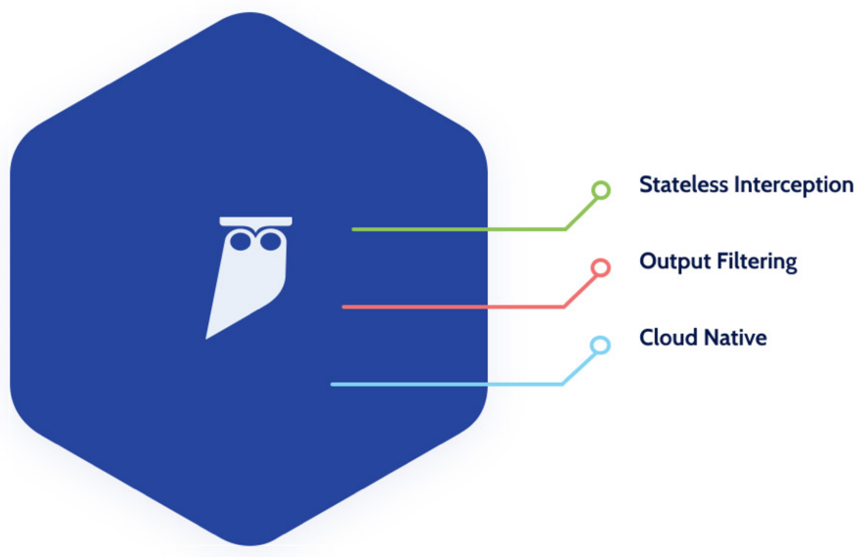
Before highly distributed data repositories (MongoDB, Cassandra, etc.) became popular, a database proxy enabled scaling and performance by providing connection pooling to the backend data repositories, high availability by routing requests to a healthy data backend (standby when the primary failed) reducing failover time. Such proxies are generally considered L4 or SQL-agnostic proxies and include HAProxy, Nginx, etc.

With applications moving to the cloud, and data volumes skyrocketing, modern data repositories started providing scalability and high availability functionality using data sharding and replication with a distributed coordinator-worker architecture. To shield the application logic from the underlying topology changes, SQL-aware database proxies such as ProxySQL, MaxScale, etc. started gaining traction. These proxies can perform tasks such as SQL read/write query routing by directing read queries to workers and write queries to the master in the coordinator. SQL-aware proxies are also used in such scenarios when there are needs to operate at SQL layer to cache SQL query responses to gain in performance, rewrite and block SQL queries for security reasons, etc.

With the maturity of container technology, especially docker, service oriented architecture using microservice as its composable unit started gaining widespread popularity. Cloud-native applications started to use microservices as their building blocks, lending themselves to the DevOps methodology of Continuous Integration and Continuous Deployment.

While the new architecture based on microservices have resulted in many benefits, it has exposed challenges specifically around security and traffic management. Communication between these disaggregated microservices resulted in an explosion in the east-west traffic, with no concrete perimeter to enforce security rules on and lack of a single ingress/egress point where traffic management could be performed. As a result, the traditional model of deploying a proxy between the application and the data repository (databases, data warehouses, etc.) no longer works in this new world.

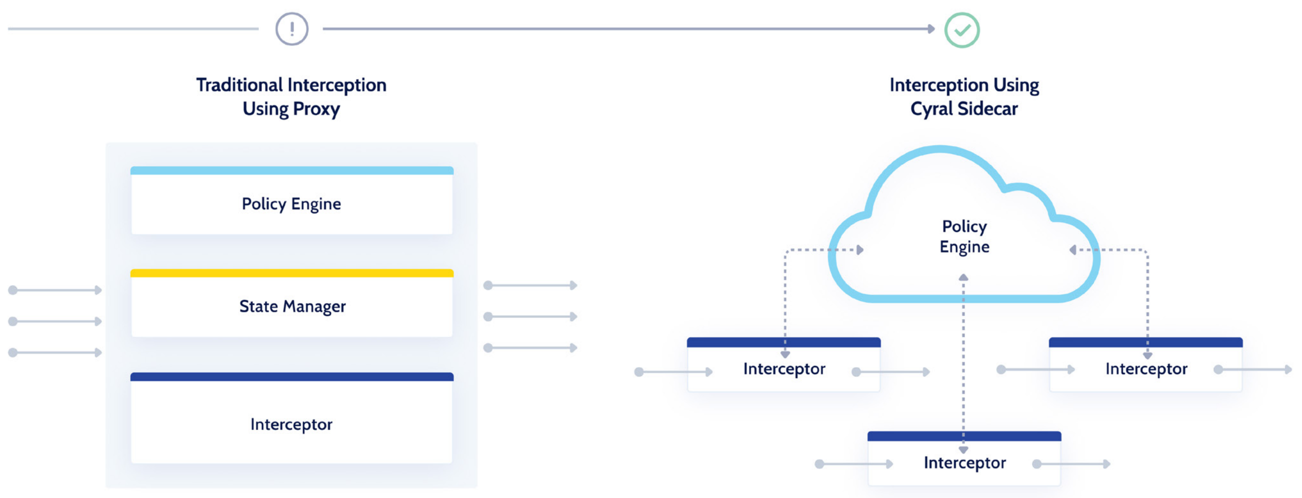
To solve this problem, Cyral invented the concept of a stateless interception service that can be deployed using a sidecar pattern.



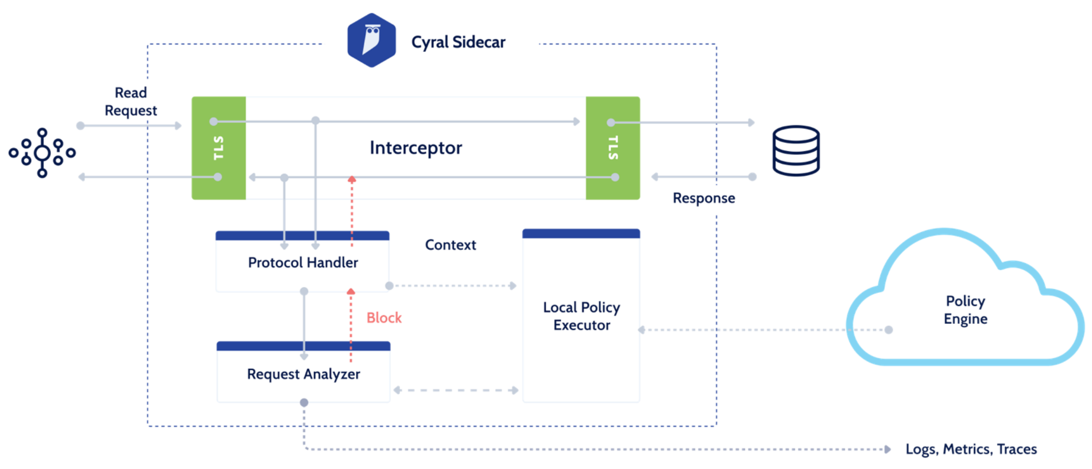
Designing Data Layer Sidecar for the Cloud Native World

With high availability and scalability often baked into the architecture and deployment model of cloud-native applications, Cyral's data layer sidecar essentially acts as a circuit breaker for these cloud native applications, natively integrating with various service orchestration tools (e.g. Kubernetes). The key design aspects of Cyral's sidecar are:

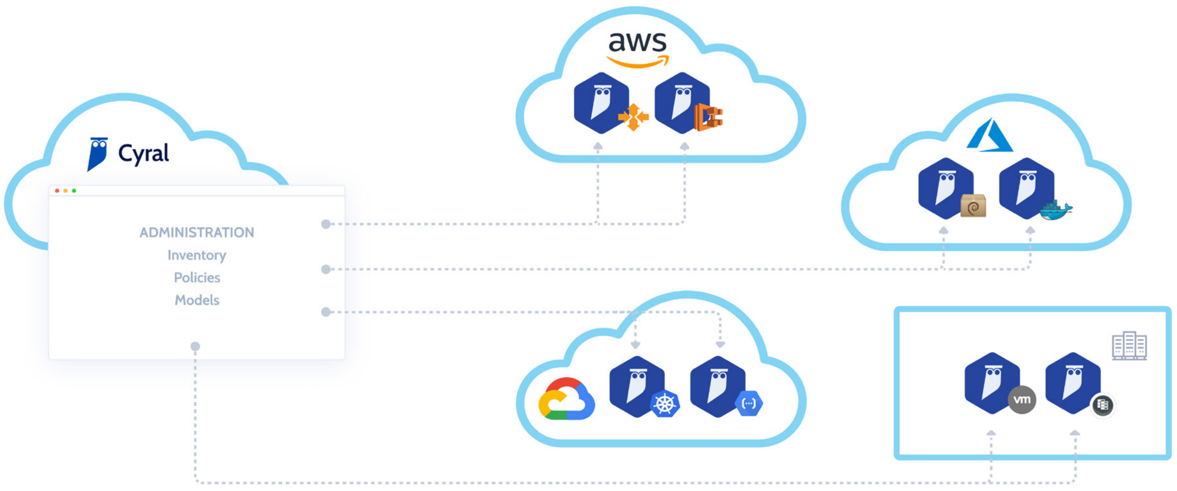
1. Stateless – Unlike traditional application proxies, our sidecar defers all session state management to the data layer connections themselves. This elegant design allows multiple sidecars to be deployed in a high-availability configuration and enables a true fail-open design.



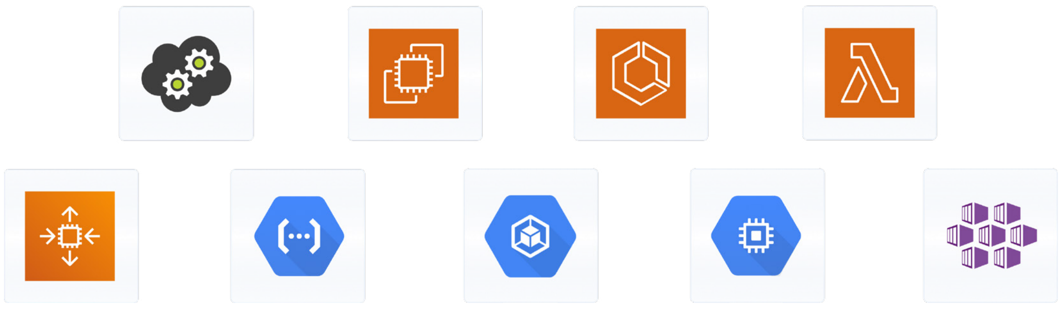
2. Output Filtering – One key insight behind our sidecar is that it can pass read requests to the data layer without any delay, while blocking their corresponding results if the request is determined malicious or disallowed. This analysis of the request happens asynchronously, while the data layer is processing it in parallel, allowing the original read operation to happen without any extra delay.



3. SaaS-based Control Plane – Our customers can deploy sidecars in several different ways, and easily administer them using a SaaS based control plane. All integrations and provisioning can be managed centrally from here. It offers intuitive workflows to implement security policies and react to threats.



Cyral sidecars can be deployed in customer's cloud or on-prem environment as a Kubernetes service, autoscaling group, cloud function or host-based install. All the data flows and sensitive information stays inside the customer's environment where the sidecar is deployed, creating no risk of spillage.



Comparison of Traditional Database Proxy and Data Layer Sidecar

	Database Proxy	Cyral's Data Layer Sidecar
Cloud Native	Bolted-on using scripts, IP address hardcoding, etc., with often no good integration with cloud orchestration platforms such as Kubernetes Not suitable for the highly distributed microservices world. Spoke and hub model -traffic from microservices forced to come to the proxy which then sends to the destination	Deployed using latest cloud orchestration platforms like Kubernetes Designed to be used with tools like Terraform, CloudFormation, etc. Supports a modern mesh architecture rather than spoke and hub
Application Affinity	A proxy is deployed in the same network where services reside but in a different compute rack Services are proxy aware, configured to talk to the proxy rather than the destination	Sidecars can reside in the same host as that where the service runs on Services are not sidecar aware, and traffic is routed to/from the services using IPTABLE rules or BPF
Function	Stateful traffic inspection Load balancing	Stateless traffic inspection Circuit breaker capability (Prevent data breaches, unauthorized data access, etc.)
Deployment	High availability through overprovisioning Used for North-South traffic	High availability through autoscaling Designed to be omnidirectional
Control Plane	Configured using non-standard formatting Minimal support for REST APIs (e.g. update pool of servers to be load balanced) Generally, no centralized control plane for managing a distributed set of proxies	Configured using JSON or YAML Support for REST APIs or APIs over gRPC Centralized control plane to manage the various sidecars
Scalability	No horizontal scaling No elasticity Introduces extra hop between the service and the data repository, thereby introducing network latency	Horizontal scaling and elasticity, integrates with container orchestration platforms such as Kubernetes Stateless circuit-breaker design results in negligible latency for traffic to the data repository
Security	TLS encryption/decryption No identity protection No native support for authenticating the connecting services No support for authorization policies	TLS encryption/decryption Inbuilt identity protection using mTLS Provides native authentication and authorization capability
APIs	Minimal API support Lacks integration with modern tools for logging, monitoring, visualization, CI/CD, etc.	API-first design Supports integration with DevOps tools such as Prometheus, Grafana, ElasticSearch, FluentD, and Kubernetes, for logging, monitoring, visualization, CI/CD, etc.
Telemetry	Basic log data Basic transaction success or failure log data	Rich telemetry data with event logs, metrics and traces Identifying individual connections and transactions with focus on the four golden signals of latency, traffic, errors and saturation (e.g. transaction latency, query per second for traffic, errors for TCP connection resets, etc.) Advanced integration with logging and observability stack such as ELK, Prometheus and Grafana
Continuous Deployment	No support for CI/CD	Supports continuous deployment because it can integrate with CI/CD tools, such as Jenkins X, Spinnaker, etc.

About Cyral

Cyral delivers enterprise data security and governance across all data services such as S3, Snowflake, Kafka, MongoDB, Oracle and more. The cloud-native service is built on a stateless interception technology that monitors all data endpoint activity in real-time and enables unified visibility, identity federation and granular access controls. Cyral automates workflows and enables collaboration between DevOps and Security teams to automate assurance and prevent data leakage. Cyral is venture-backed by Redpoint, A.Capital, Costanoa and SVCI. Follow the company on Twitter at @CyralInc

cyral.com/demo