RFC - Galadriel Design Document

Background

SPIRE (SPIFFE Runtime Environment) is an open-source CNCF graduated project that provides unique, secure identities to software services, by validating their identities using properties of the cloud provider or on-premises equipment. SPIRE identities can take the form of x.509 certificates or JWT (JSON Web Token) tokens. In both cases, SPIRE uses its own public/private keypairs. The public keys are published in files called “trust bundles,” which are necessary for validating the x.509 certificates and JWTs to establish secure communication. Because trust bundles only contain public keys, they do not need to be secret (but they must not be tampered with).

SPIRE provides functionality to federate with other entities following SPIFFE (Secure Production Identity Framework for Everyone) Federation specifications. It has built-in functionality that allows the creation of multiple federation entries. The SPIRE Server exposes its trust bundle via an end-point API. A federated relationship must be manually configured by both parties in the relationship, so trust bundles can be exchanged.

Problem statement

The key limitation of existing SPIRE Federation options is that they require an administrator to set up a secure, public endpoint to serve the federation data (either through OIDC Federation or SPIFFE Federation). This is a substantial administrative hassle and, in some cases, may be impossible (for example for on-premises users).

A second limitation of the existing options is that federating many-to-many relationships requires manual creation of federation entries. For example, federating 5 trust domains to each other requires 20 administrator actions (each domain needs 4 new configuration changes to federate with the 4 others). There would be more actions when trust points change URLs or relationships change.

A third limitation relates to the lifecycle and audit of federated relationships. SPIRE does not provide a mechanism to manage and track the lifecycle of federated relationships. Current techniques rely on manual configuration changes, and do not allow for observability of trust bundle exchange.

Goal

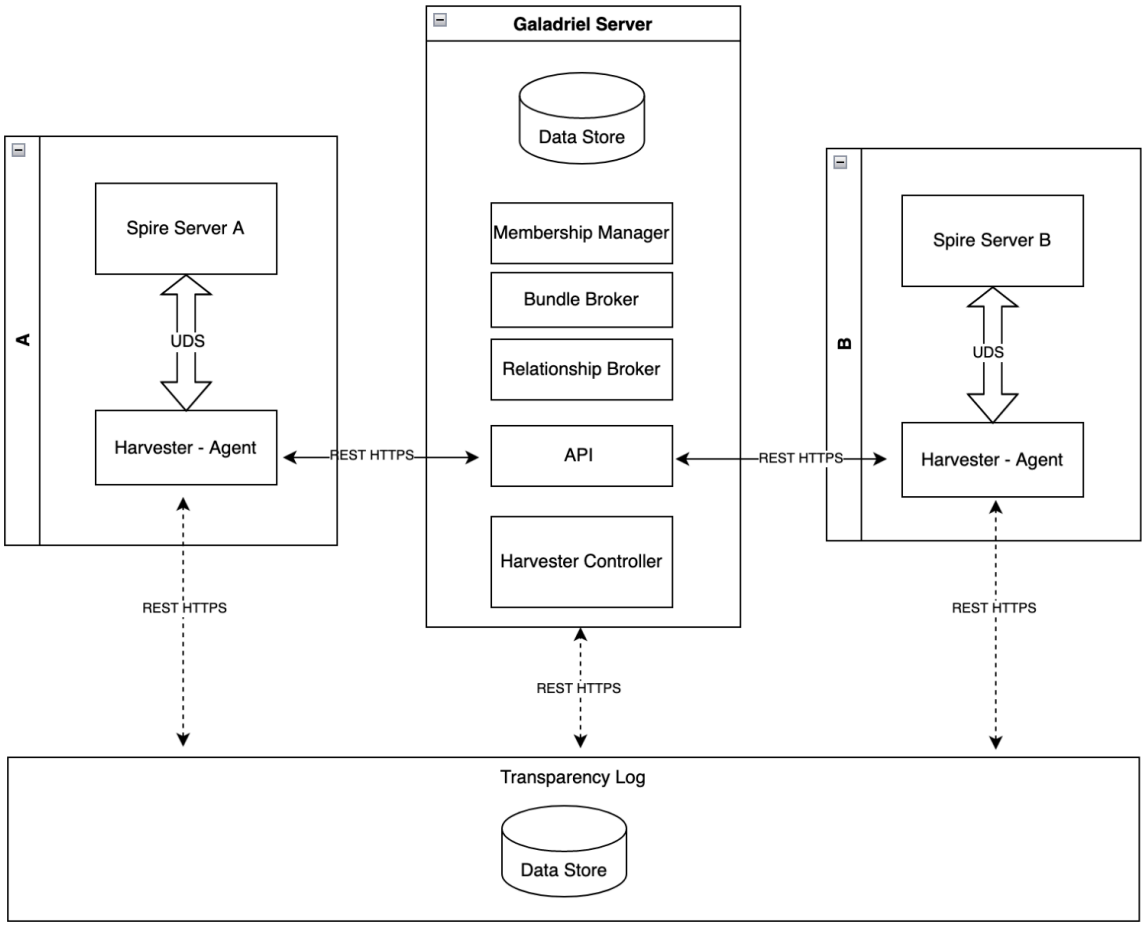
A system that allows scalable and easy configuration of federation among SPIRE servers, or among SPIRE servers and other external resources, and that serves as a central controller for managing and auditing federation relationships.

Proposal

Galadriel will extend the existing federation authorization techniques from SPIRE by centralizing the management and exchange of trust bundles via a hub. Galadriel will manage members and federation relationships via APIs. It will collect trust bundles from SPIRE servers, route them, and present them to other SPIRE servers and cloud resources. Galadriel can be used as the sole mechanism to manage and audit external relationships for SPIRE implementations.

Deep dive

This design proposes the creation of two main components: The Galadriel Server and Harvester. The components will facilitate the exchange of SPIRE-generated trust bundles among multiple SPIRE servers. The exchange will be enforced via relationship rules established at the Galadriel Server level, and the Harvesters will provide the routing of bundles to and from SPIRE servers via this central hub. A transparency log will be used to store and validate configurations of relationships in the server and trust bundles from the Harvesters.

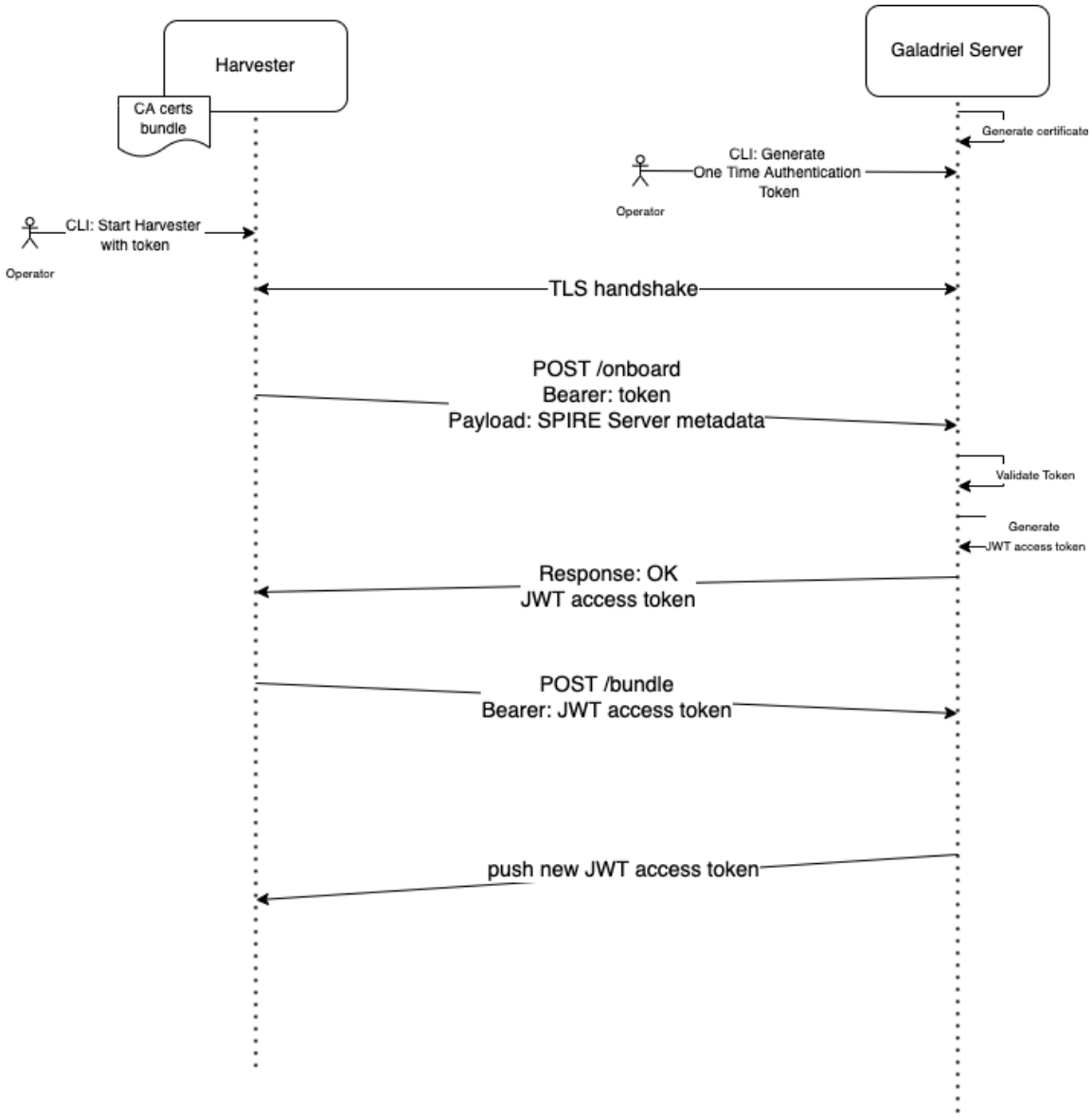


Communications Flow Among Components

Galadriel Harvester and Server

The Galadriel server will expose a REST API that will be consumed by the Harvesters via HTTPS.

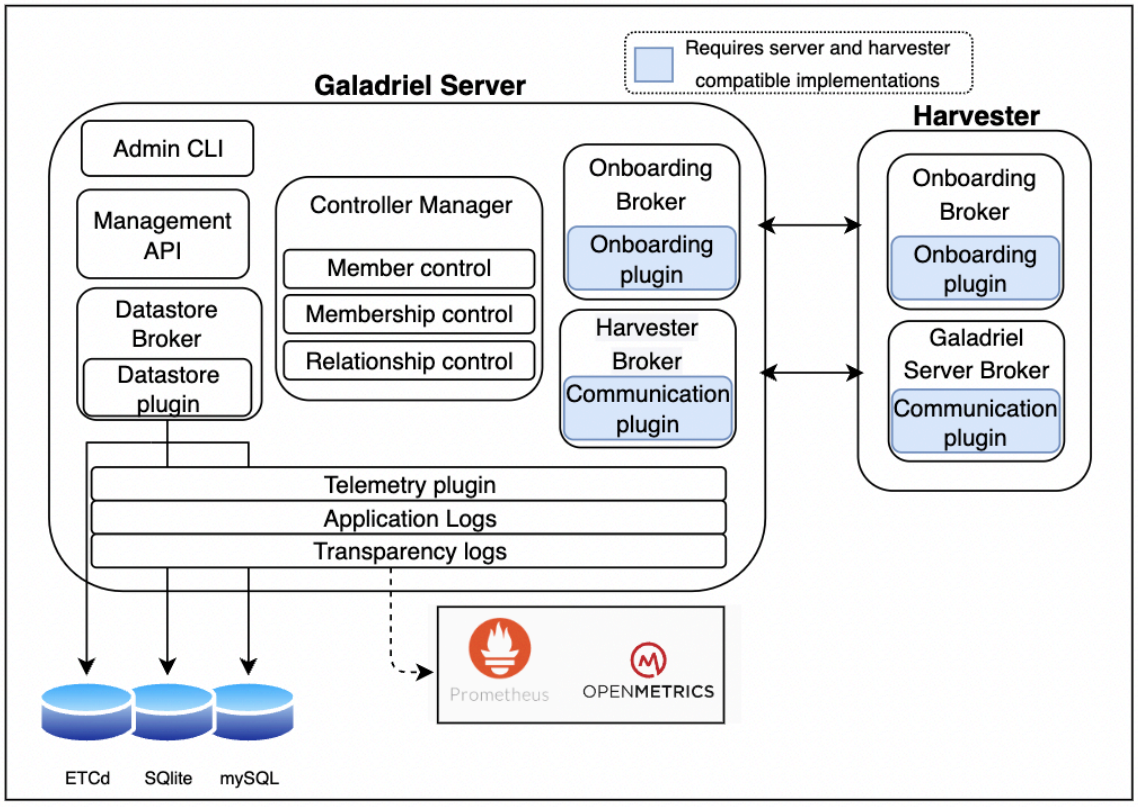
The Harvester will use the Join Token to authenticate to the Galadriel server APIs. Join Tokens will be generated by Galadriel server administrators during the SPIRE server onboarding process and will be shared manually to the SPIRE server administrators to be bootstrapped in the Harvester at startup time.



Harvester and SPIRE Server

The Galadriel Harvester will be deployed in the same node as the SPIRE server and will use Unix Domain Sockets (UDS) exposed by SPIRE Server to establish communications. The UDS path should be configurable.

Galadriel Server



- Admin CLI: This component is a wrapper client around the set of REST APIs exposed by the Management API. It will be part of the same binary as the rest of other components.

- Management API: The Galadriel server will expose a series of APIs to the Harvesters and Galadriel administrators. The OpenAPI specification (v 3.0) will be used. The API definitions will provide operations for managing the following:
    - Organization
    - Federation Group: Represents a bridge for its Members. This allows Members of the same group to establish relationships. Members of distinct groups are not allowed to have relationships.
    - Member (SPIRE Servers in the same Trust Domain)
    - Relationship (Members that want to federate to each other)

    - Membership (Federation Group relationship)
    - Trust Bundle (exchange)

- Datastore Broker: Module that should provide higher-level, database agnostic abstractions (StoreNewRelationship() could be an example). The Interfaces defined by this component will be implemented by Datastore plugins. The plugins will then contain datastore specific routines.

- Controller Manager: This component is a control loop that will keep the server alive, allow for it to gracefully shut down, and manage the overall state of Galadriel Server. The Management API should only use objects exported by the Controller Manager in order to manage Members, Memberships, and Relationships, while the Controller Manager itself will know how to manage these functionalities, either by action (e.g. user input) or passively (e.g. verifying expiry dates). One argument for having these all part of the same module is to help avoid exporting methods related to these functionalities, as these should be used only by the Controller Manager. Another relevant observation is that this is the only component which will not have interfaces to be implemented, as other plugin-based components do.

- Onboarding Broker: Similar to the node-attestation for SPIRE, this module should be able to verify the authenticity of inbound connections from Harvesters and securely provide an authentication material of some form. A possible handshake could be: Join token and *Galadriel Server's* CA certificate are shared via an out-of-band mechanism > *Harvester* initiates the connection > *Galadriel Server* sends certificate signed by CA > *Harvester* validates *Galadriel Server's* certificate and sends join token > *Galadriel Server* validates token, sends encrypted signed token to be used by *Galadriel Server Broker.*

- Harvester Broker: Component responsible for exposing routes where Harvesters can reach out to, in order to push and pull updates from. Since this channel should be secure, the implementation for this module is dependent on the **Onboarding Broker**, although It would be ideal if we are able eliminate this dependency. It's worth noting that it is unclear at this point if a communication plugin will be needed, as this depends highly on user deployment topology, environment contingency restrictions, etc.

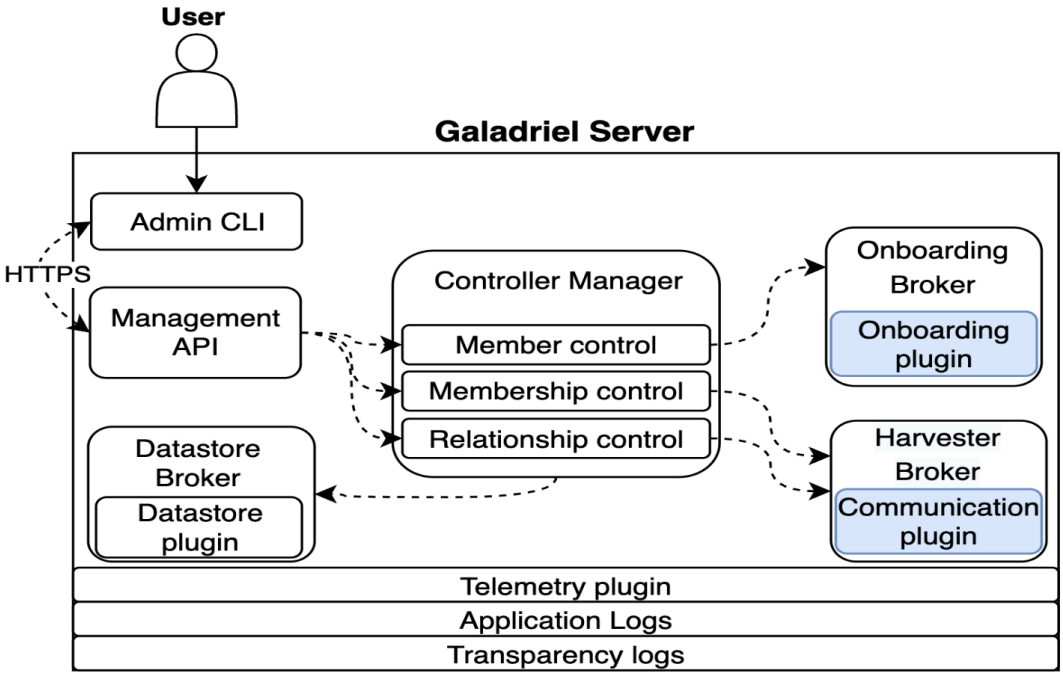
- Data Storage: The "Datastore" component will only be accessed via a data broker in the Galadriel server. The intent is that the Controller Manager, depicted on the "Communication Flow" topic will use internal calls to a datastore interface that is agnostic of the architecture. Thus, different types of databases, such as SQLite, Postgres, MongoDB or etcd can be used as a data repository. Galadriel must support multi-tenancy allowing multiple organizations and members to use the same deployment instance.

- Transparency Log: Transparency logs will be used to verify the authenticity of trust bundles and configurations for the multiple components of Galadriel. Trust bundles will be signed and sent to transparency logs by the Harvesters before being shared with their respective Galadriel Server.

- Metrics: Galadriel Server and Harvester will provide capabilities to export metrics for monitoring. The metrics broker will be agnostic of the architecture; thus, different types of metrics monitoring

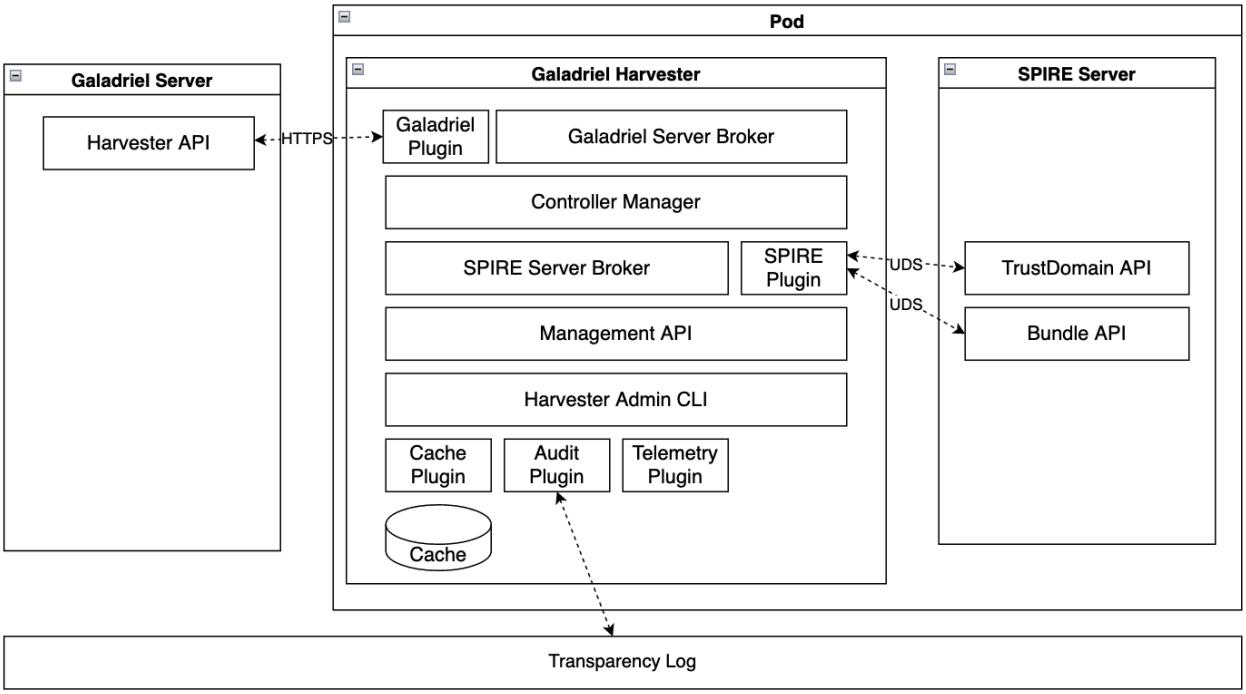
tools could be used. The metrics monitoring service to be used will be controlled via server configuration settings.

Communication Flow



Harvester

The Harvester will have a modular architecture with lightly coupled components that provide flexibility and maintainability of its components. Configuration settings will allow SPIRE administrators to define the Harvester's behavior and the mechanism to interface with the SPIRE server.



- Galadriel Plugin / Galadriel Server Broker: Responsible for all Galadriel Server communication-related tasks, it securely connects to the upstream Galadriel Server through the Harvester API to push Bundle updates, memberships and relationships consent approvals or denials, as well as retrieving the desired Federation Relationships that need to be applied on the adjacent SPIRE Server.

- SPIRE Plugin / SPIRE Server Broker: It's the only SPIRE-aware component and its main responsibility is to bridge all SPIRE-Harvester communication. It exposes an interface to fetch the adjacent SPIRE Trust Bundle and manage Federation Relationships as needed using SPIRE's Trust Domain and Bundle API. There will be a built-in plugin that communicates with a SPIRE Server through Unix Domain Socket.

- Controller Manager: It's the core component, which runs the control loop that reconciles the desired state coming from the Galadriel Server and the actual state of the SPIRE Server. It defines the public interface for all administrative tasks that are exposed to the administrator through the Management API, like membership and relationship consent responses.

- Management API: It exposes a REST API that gives access to the Controller Manager APIs over the network. It gives access to the Harvester on-boarding operations, and also to view, set, update, and/or delete a subset of configurations like Harvester status, trust bundle, membership, federation, audits, telemetry, etc. The Harvester will be able to issue client certificates signed by its own CA for administrators' use, allowing them to securely connect to perform administrative tasks.

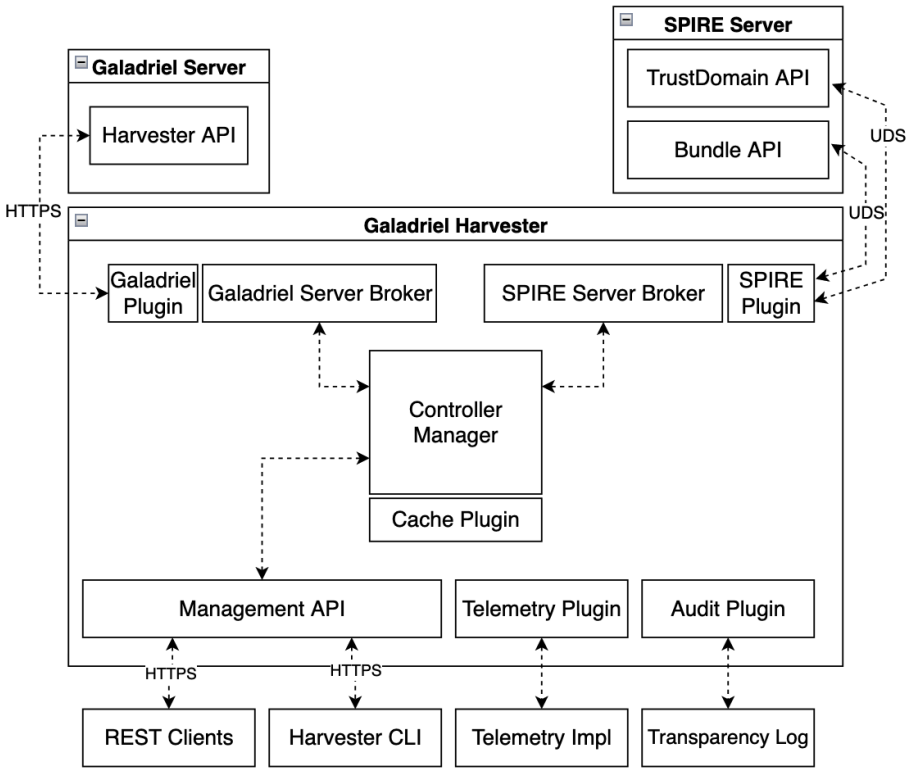
- Harvester CLI: This component is a wrapper client around the set of REST API exposed by the Management API. It will be part of the same binary as the rest of other components, instead of being shipped separately.

- Cache Plugin: It provides trust domain bundles caching to facilitate bundle exchange. There will be a single built-in cache plugin that will use in-memory cache.

- Audit Plugin: Implements standard public logging for audit tasks. There will be a single built-in audit plugin that will use Rekor as the underlying system.

- Telemetry Plugin: Implements a standard interface for metrics collection. There will be a single built-in plugin that leverages OpenTelemetry.

Communication flow break-down



Initialization flow

What happens when all components start?

Step by step description

1. Galadriel Server starts up
2. The server will read the configuration
    a. Connect to the defined data store
    b. Retrieve certificate material to start API
    c. Retrieve membership data from the data store
3. Server will retrieve validate membership and member data against transparency logs
4. Server will start up API
5. The Harvester starts in the same node as SPIRE server A
6. The Harvester uses configuration settings to
    a. Establish communications with SPIRE server
    b. Retrieve trust bundle from SPIRE server and cache it
    c. Establish communications with Galadriel server
7. The Harvester will push the trust bundle to the Galadriel server
8. The Harvester will push the trust bundle to the transparency log
9. The Galadriel server will:
    a. Validate trust bundle against transparency log
    b. Validate origin request against data store
    c. Cache the trust bundle
    d. If validation does not pass, the SPIRE server status in Galadriel will be set to “quarantine” and Galadriel admins will be notified
10. The Harvester will request from the Galadriel Server all the active relationships for the Federation Group(s) it belongs to
11. The Harvester will validate that the SPIRE server has the correct federation configuration based on relationships received from the Galadriel server
    a. If the SPIRE server does not have the dynamic federated relationships in it, the Harvester will create them.
12. The Harvester will request all trust bundles from other SPIRE servers
13. The Harvester will validate trust bundles against transparency logs
14. The Harvester will make trust bundles available to SPIRE server
15. The Harvester will start the approval relationships module
    a. It request all pending relationship approvals and based on configuration will auto-approve or queue pending request for manual approval

Definitions

- Member: A Member is the entity that’s created in the Galadriel Server database when a Galadriel Harvester connects to it, and it represents a successfully onboarded set of SPIRE Servers. This also means that the mechanism for securely communicating between a Galadriel Server <> Harvester is defined in this step.A Member holds together all the base, long-lasting information about the SPIRE server, like it's SPIFFE ID, contact information and others. Unlike the other two entities we will see below (Membership and Relationship), a Member does not have a TTL.

- Membership: The Membership, on the other hand, is the living aspect of a Member, temporarily associating its existence with one of the Federation Groups managed by that Server. They expire and new Relationships require an active, valid Membership to previously exist before being established.

- Relationship: A Relationship essentially defines which trust bundles gets routed to which Harvester. It defines an established mutual consent of trust between two Members with an active Membership. An important aspect about Relationships is that, similarly to Memberships, they also expire.

The current state

Galadriel is in active development and is publicly available at https://github.com/HewlettPackard/galadriel.
There is already a Proof of Concept that offers a limited sub-set of the features outlined in this document, but the goal is to have incremental releases until completion.