Hackathon Assignment: Enable Multi-Hostname Support for Harbor

Background

Harbor is an open-source cloud-native registry that stores, signs, and scans container images. Currently, Harbor only supports a **single hostname** per instance, which limits its flexibility in environments where internal and external access require different hostnames (e.g., `harbor.internal` for CI/CD pipelines and `harbor.example.com` for external users).

This hackathon challenge is to **design and implement a solution** that allows a single Harbor instance to be accessible via **multiple hostnames**, addressing the use case described in goharbor/harbor#8243.

Problem Statement

- Harbor currently only supports **one hostname** per instance.
- Users need to push images via an **internal hostname** (e.g., for CI/CD) while allowing pulls via an **external hostname** (e.g., for public or cross-network access).
- Without multi-hostname support, users must deploy multiple Harbor instances or use workarounds like reverse proxies, which add complexity.

Goal: Extend Harbor to support **configuring and managing multiple hostnames** for a single instance, ensuring seamless access and compatibility with existing features (e.g., authentication, replication, and vulnerability scanning).

Requirements

1. **Configuration**:
    - Allow administrators to specify **multiple hostnames** in Harbor’s configuration (e.g., via `harbor.yml` or UI).
    - Ensure all hostnames resolve to the same Harbor instance.
2. **Functionality**:
    - Support pushing/pulling images using **any configured hostname**.
    - Maintain consistency for features like:
        - Authentication (e.g., OAuth, LDAP).
        - Image signing and vulnerability scanning.
        - Replication and garbage collection.
3. **Security**:
    - Validate that all hostnames are **securely managed** (e.g., TLS/SSL support for each hostname).
    - Prevent conflicts or misuse (e.g., ensure only authorized hostnames are accepted).
4. **Compatibility**:
    - Ensure backward compatibility with existing single-hostname setups.

   - Test with common CI/CD tools (e.g., GitLab CI, Jenkins) and container runtimes (e.g., Docker, Kubernetes).
5. **Documentation**:
   - Provide clear instructions for configuring multi-hostname support.
   - Include examples for common scenarios (e.g., internal/external access).

Deliverables

1. **Code Implementation**:
   - Modify Harbor’s core components (e.g., `core`, `registry`, `portal`) to support multiple hostnames.
   - Update configuration files and APIs to accept a **list of hostnames**.
2. **Testing**:
   - Write unit and integration tests to verify multi-hostname functionality.
   - Test with real-world scenarios (e.g., pushing via `harbor.internal` and pulling via `harbor.example.com`).
3. **Documentation**:
   - Update Harbor’s official documentation to include multi-hostname setup instructions.
   - Provide a **quick-start guide** for users.
4. **Demo**:
   - Showcase the solution with a live demo, including:
     - Configuring multiple hostnames.
     - Pushing/pulling images using different hostnames.
     - Verifying security and compatibility.

Evaluation Criteria

Criteria	Weight
Functionality	30%
Code Quality	25%
Security	20%
Compatibility	15%
Documentation	10%

Resources

- Harbor GitHub Repository
- Harbor Documentation
- Issue #8243: Multi-Hostname Support
- Harbor Architecture Overview

Bonus Challenges

- **Dynamic Hostname Management**: Allow hostnames to be added/removed without restarting Harbor.
- **UI Integration**: Add a user interface for managing hostnames in the Harbor admin panel.
- **Load Balancer Support**: Ensure compatibility with load balancers (e.g., NGINX, Traefik).

Why This Matters

This feature will **simplify Harbor deployments** in hybrid or multi-network environments, reducing the need for workarounds and improving usability for DevOps teams.

Ready to hack? Fork the Harbor repo, implement your solution, and submit a pull request! 🚀

Hackathon Assignment: Implement an SNMP Source for Vector

Background

Vector is a high-performance, open-source observability data pipeline that collects, transforms, and routes logs, metrics, and events. While Vector supports a wide range of sources (e.g., files, Kafka, syslog), it currently **lacks native support for SNMP (Simple Network Management Protocol)**, a critical protocol for monitoring network devices like routers, switches, and servers.

This hackathon challenge is to **design and implement an SNMP source** for Vector, enabling users to ingest SNMP traps and polls directly into their observability pipelines. This feature is highly requested (see vectordotdev/vector#4567) and would fill a gap in Vector's capabilities, making it a more versatile tool for network monitoring.

Problem Statement

- **SNMP is widely used** for monitoring network devices, but Vector does not natively support it.
- Users currently rely on workarounds (e.g., `snmp_exporter` + Prometheus or Fluentd plugins) to ingest SNMP data into Vector.
- A native SNMP source would simplify architectures, reduce latency, and improve integration with existing Vector pipelines.

Goal: Build an **SNMP source component** for Vector that can:

1. **Receive SNMP traps** (asynchronous notifications from devices).
2. **Poll SNMP devices** (active queries for metrics).
3. **Convert SNMP data** into Vector's internal event format for further processing (e.g., filtering, transforming, routing).

Requirements

1. **SNMP Trap Support**:
   - Listen for SNMP traps on a configurable UDP port (default: `162`).
   - Parse trap messages (v1, v2c, or v3) and convert them into Vector events.
   - Support customizable trap OIDs (Object Identifiers) and mappings to Vector fields.
2. **SNMP Polling Support**:
   - Actively poll SNMP-enabled devices at configurable intervals.
   - Support for common SNMP metrics (e.g., interface stats, CPU/memory usage).
   - Allow users to specify OIDs to poll and map them to Vector event fields.
3. **Configuration**:
   - Add a new `snmp` source to Vector's configuration (e.g., in `vector.toml`):

     Kopiëren
     ```
     [sources.my_snmp]

     type = "snmp"
     ```

```
mode = "trap" # or "poll"
address = "0.0.0.0:162" # for traps
community = "public" # for v1/v2c
version = "2c" # 1, 2c, or 3
poll_interval_secs = 60 # for polling mode
oids = ["1.3.6.1.2.1.1.5.0", "1.3.6.1.2.1.2.2.1.10"] # OIDs to poll
```

- Support SNMPv3 authentication/encryption (optional but ideal).

4. **Data Mapping**:
    - Convert SNMP data (e.g., OID values, varbinds) into structured Vector events.
    - Preserve metadata (e.g., device IP, timestamp, OID).
5. **Performance**:
    - Handle high volumes of traps or polls efficiently.
    - Minimize latency in processing and forwarding events.
6. **Compatibility**:
    - Ensure the SNMP source integrates seamlessly with Vector's existing transforms and sinks.
    - Test with real-world SNMP devices (e.g., Cisco routers, Linux servers with `net-snmp`).
7. **Documentation**:
    - Provide clear setup instructions and examples.
    - Document supported SNMP versions, configurations, and limitations.

Deliverables

1. **Code Implementation**:
    - Add a new `snmp` source to Vector's codebase (Rust).
    - Use existing SNMP libraries (e.g., `snmp-rs` or `netsnmp`) or implement a lightweight SNMP parser.
    - Update Vector's configuration schema to support the new source.
2. **Testing**:
    - Write unit and integration tests for trap/polling functionality.
    - Test with real or emulated SNMP devices (e.g., using `snmpsim` or a lab router).
3. **Documentation**:
    - Add usage examples to Vector's official docs.
    - Include a **quick-start guide** for common SNMP monitoring scenarios.
4. **Demo**:
    - Showcase the SNMP source in action, including:
        - Receiving traps from a device.
        - Polling a device for metrics.

        - Forwarding SNMP data to a sink (e.g., Loki, Elasticsearch).

Evaluation Criteria

Criteria	Weight
Functionality	30%
Code Quality	25%
Configuration	20%
Performance	15%
Documentation	10%

Resources

- Vector GitHub Repository
- Vector Source Development Guide
- Issue #4567: SNMP Source
- [Prior Art]:
    - fluent-plugin-snmp
    - Prometheus SNMP Exporter
    - InfluxData SNMP Integration
- SNMP Protocol RFCs

Bonus Challenges

- **SNMPv3 Support**: Implement authentication and encryption for secure SNMPv3 communication.
- **Dynamic OID Discovery**: Allow users to discover available OIDs from a device and auto-generate configurations.
- **Metrics Conversion**: Automatically convert SNMP metrics into Prometheus-compatible formats.

Why This Matters

An SNMP source would make Vector a **one-stop shop** for observability data, eliminating the need for external tools like `snmp_exporter`. This is especially valuable for network engineers and DevOps teams monitoring hybrid infrastructures.

Ready to build? Fork the Vector repo, implement your SNMP source, and submit a pull request!

Hackathon Assignment: Implement WebDAV (OwnCloud) Checksum Extension for SFTPGo

Background

SFTPGo is a fully featured, open-source SFTP/HTTP/WebDAV server that simplifies file transfer and sharing. While SFTPGo already supports the `X-OC-Mtime` extension for WebDAV (used by OwnCloud/Nextcloud), it **lacks support for checksum-based integrity checks**. This feature is crucial for environments where firewalls restrict traffic to HTTP(S), forcing users to rely on WebDAV for file transfers. Checksums ensure that files are transferred without corruption, providing an extra layer of reliability.

This hackathon challenge is to **add support for the OwnCloud checksum extension** in SFTPGo's WebDAV implementation, enabling clients like Rclone to verify file integrity during transfers.

Problem Statement

- Some firewalls **only allow HTTP(S) traffic**, making WebDAV the only viable option for file transfers to SFTPGo.
- SFTPGo currently **does not support checksums** for WebDAV, leaving users without a way to verify file integrity during transfers.
- Rclone (a popular cloud sync tool) already supports the OwnCloud checksum mechanism for both its client and server. Adding this to SFTPGo would enable seamless, integrity-checked transfers.

Goal: Implement the **OwnCloud checksum extension** for SFTPGo's WebDAV server, allowing clients to verify file integrity using on-the-fly checksums (e.g., SHA-1, MD5).

Requirements

1. **Checksum Extension Support**:

    - Implement the OwnCloud checksum extension in SFTPGo's WebDAV handler.
    - Support standard checksum algorithms (e.g., SHA-1, MD5) as defined in the OwnCloud WebDAV API.
    - Allow clients to request and validate checksums during file uploads/downloads.

2. **Compatibility with Rclone**:

    - Ensure the implementation is compatible with Rclone's checksum mechanism (see Rclone's WebDAV server implementation).
    - Test with Rclone as a client to verify end-to-end integrity checks.

3. **Configuration**:

    - Add a configurable option to enable/disable the checksum extension in SFTPGo's WebDAV settings.
    - Allow administrators to specify supported checksum algorithms (e.g., SHA-1, MD5).

4. **Performance**:
    - Calculate checksums efficiently, without significantly impacting transfer speeds.
    - Support streaming checksums for large files to avoid memory issues.
5. **Error Handling**:
    - Return appropriate HTTP status codes and error messages if checksum validation fails.
    - Log checksum mismatches for debugging.
6. **Documentation**:
    - Update SFTPGo’s documentation to explain how to enable and use the checksum extension.
    - Provide examples for configuring Rclone (or other clients) to use checksums with SFTPGo.

Deliverables

1. **Code Implementation**:
    - Extend SFTPGo’s WebDAV handler to support the OwnCloud checksum extension.
    - Add checksum calculation and validation logic for file uploads/downloads.
    - Update the configuration file (`sftpgo.json` or `sftpgo.yaml`) to include checksum settings.
2. **Testing**:
    - Write unit and integration tests for checksum calculation and validation.
    - Test with Rclone and other WebDAV clients (e.g., Nextcloud, Cyberduck).
    - Verify that checksums are correctly calculated and validated for files of varying sizes.
3. **Documentation**:
    - Document the new feature in SFTPGo’s official documentation.
    - Provide a **step-by-step guide** for enabling and using checksums.
4. **Demo**:
    - Showcase the feature by:
        - Uploading/download a file using Rclone with checksum validation.
        - Demonstrating how SFTPGo handles checksum mismatches.

Evaluation Criteria

Criteria	Weight
Functionality	30%
Compatibility	25%

Code Quality
20%
Performance
15%
Documentation
10%

Resources

- SFTPGo GitHub Repository
- OwnCloud WebDAV Checksum API
- Rclone WebDAV Server Implementation
- Issue #1922: WebDAV Checksum Extension
- WebDAV RFC 4918

Bonus Challenges

- **Multi-Algorithm Support**: Allow users to configure multiple checksum algorithms (e.g., SHA-256, BLAKE2).
- **Checksum Caching**: Cache checksums for frequently accessed files to improve performance.
- **UI Integration**: Add a Web UI option to enable/disable checksums and view validation results.

Why This Matters

Adding checksum support to SFTPGo's WebDAV server will **enhance data integrity** for users in restricted network environments. It will also improve compatibility with tools like Rclone, making SFTPGo a more versatile and reliable file transfer solution.

Ready to code? Fork the SFTPGo repo, implement the checksum extension, and submit a pull request! 🚀