Consolidating logs from multiple Linux nodes while filtering them at the source to reduce noise is a common and effective strategy. You'll generally use a combination of local filtering tools and a centralized log management system.

Here's a recommended approach:

1. Local Filtering and Forwarding (at the Node)

The best tool for filtering and forwarding logs natively on most modern Linux systems is **Rsyslog** or **Systemd-journald/Journalctl** (depending on your distribution).

Using Rsyslog for Filtering

Rsyslog is highly configurable and can filter messages based on a variety of criteria (facility, severity, content).

Filtering Configuration:

You can create a new configuration file, for example, `/etc/rsyslog.d/10-remote.conf`, on each node.

1. **Filter by Severity and Facility:** To send only messages of severity **warning** or higher (crit, alert, emerg) from the **authpriv** and **mail** facilities:

   ```
   # Send only warning or higher from authpriv and mail to the remote server
   authpriv,mail.warn     @@logserver.example.com:514
   ```

2. **Filter by Message Content (Property-Based Filtering):** To send all messages *except* those containing the word "CRON" (common noise):

   ```
   # Don't forward messages containing the word "CRON"
   :msg, contains, "CRON"    ~

   # Forward all other logs (except CRON) to the remote server
   *.* @@logserver.example.com:514
   ```

   The `~` discards the log line. The subsequent rule then forwards everything else.

Using Journald for Filtering (Optional)

If you're primarily using `systemd-journald`, you can use **`journalctl`** for temporary viewing/filtering, but for *persistent* forwarding, it's often configured to pipe its output into **Rsyslog** or a dedicated log forwarder.

2. Centralized Log Collection and Management

You need a central server to receive, store, index, and analyze the filtered logs. The most popular open-source stack is the **ELK/Kibana Stack** or **Loki/Grafana**.

System	Role	Description
Log Shipper (Fluentd/Fluent Bit/Logstash)	Sends logs from the node to the log receiver.	On the node, it reads the logs (often from a file or directly from the system journal) and sends them to the Log Receiver.
Log Receiver (Logstash)	Parses, filters, and transforms the incoming logs.	Receives data from all nodes, applies additional processing (if needed), and sends it to the Log Store.
Log Store (Elasticsearch/Loki)	Stores and indexes all the log data.	Provides fast, scalable storage and search capabilities.
Visualization (Kibana/Grafana)	Provides a UI for searching, analyzing, and graphing the logs.	The main interface for viewing your consolidated and filtered logs.

Recommended Log Shipper for Enhanced Filtering

For highly complex filtering and parsing *before* transmission, using a dedicated log shipper like **Fluent Bit** or **Logstash** on the node is often more powerful than basic Rsyslog filtering.

- **Fluent Bit** (Lightweight): Excellent for pre-processing, adding metadata, and filtering logs with minimal resource usage before sending them, typically to Elasticsearch or Loki.
  - **Filtering Example (Fluent Bit Config):** You can use a `grep` or `rewrite_tag` filter within the Fluent Bit configuration to drop noisy logs based on a regular expression before they are ever sent over the network.

Summary of Steps

1. **Choose your Local Filter:** Configure **Rsyslog** or a dedicated shipper like **Fluent Bit** on each Linux node.
2. **Implement Filtering Rules:** Define precise rules to **discard** or **ignore** the high-volume, low-value logs (like routine CRON jobs, expected minor service warnings, etc.) directly in the local configuration file.
3. **Configure Forwarding:** Set the local filter/shipper to forward the *remaining* (filtered) logs to your central log server (e.g., Logstash or Elasticsearch/Loki receiver).
4. **Set up Central Server:** Deploy a central log management system (e.g., ELK/Loki) to receive and store the data.

> **Ultimately these logs will be send from the consolidated syslog server to Loki, and the nodes will uses a syslog agent and Alloy forwarder.**

That's a very clear and modern logging pipeline! Focusing on **Rsyslog** (as the syslog agent) and the **Alloy/Loki** stack is an efficient approach.

Here is the plan to consolidate and filter your logs using **Rsyslog** on the nodes, forwarding them to a central **Syslog Server**, and then ingesting them into **Loki** via the **Alloy** forwarder.

---

1. Node-Level Filtering and Forwarding (Rsyslog)

On each Linux node, you'll configure **Rsyslog** to perform the noise reduction and forward the filtered logs to your central syslog server.

A. Implement Filtering Rules

Edit or create a configuration file on each node, typically in `/etc/rsyslog.d/`, for example, `/etc/rsyslog.d/10-remote.conf`.

1. **Discard Noisy Logs:** Use the `~` action to discard (stop processing) messages that are high volume and low value.

   ```
   # Stop processing logs containing "CRON" or "rsyslogd-2007"
   :msg, contains, "CRON"     ~
   :msg, contains, "rsyslogd-2007"    ~
   ```

2. **Filter by Severity/Facility:** Filter out messages based on their standard syslog severity (e.g., only send logs with a priority of **`warn`** or higher, discarding low-level **`info`** and **`debug`** logs).

   ```
   # Discard all info, notice, and debug messages from all facilities
   *.* stop
   *.warn    action(type="omfwd" Target="SYSLOG_SERVER_IP" Port="514"
   Protocol="tcp")
   ```

   - **Note:** The `stop` action is useful if you want to explicitly stop processing the low-priority messages that haven't been forwarded, but the filtering below is more explicit.

3. **Forward Remaining Logs:** After filtering out the noise, forward all remaining logs to your central syslog server.

   ```
   # Forward everything else (that wasn't discarded above) to the central
   server
   *.* action(type="omfwd" Target="SYSLOG_SERVER_IP" Port="514"
   Protocol="tcp")
   ```

B. Apply and Restart

Save the configuration file and restart the rsyslog service on each node.

```
sudo systemctl restart rsyslog
```

---

2. Central Syslog Server Configuration (Rsyslog + Alloy)

The central server receives the filtered logs from all nodes and uses the **Alloy** forwarder (a specialized agent for the Grafana ecosystem) to send them to Loki.

A. Syslog Server (Rsyslog) Configuration

Configure Rsyslog on the central server to listen for incoming TCP/UDP connections.

Edit `/etc/rsyslog.conf` and uncomment or add these lines:

```
# provides UDP syslog reception
module(load="imudp")
input(type="imudp" port="514")

# provides TCP syslog reception
module(load="imtcp")
input(type="imtcp" port="514")

# Write all incoming remote logs to a single file for Alloy to scrape
$template RemoteLogFormat,"%timegenerated% %HOSTNAME% %syslogtag%%msg:::drop-last-lf%\n"
if $fromhost != "127.0.0.1" then action(type="omfile" file="/var/log/remote.log" template="RemoteLogFormat")
```

This configuration ensures all incoming logs are written to a single, structured file: `/var/log/remote.log`. This is the file **Alloy** will monitor.

B. Alloy Configuration

The **Alloy** forwarder (based on Grafana Agent Flow) will use a **`loki.source.file`** component to tail the `/var/log/remote.log` file and ship the log lines to Loki.

Create an Alloy configuration file (e.g., `alloy.river`).

```
// Component to define the logging target (your Loki instance)
loki.process "syslog_logs" {
  forward_to = [loki.write.default.receiver]

  // Parse the log line to extract fields like timestamp and hostname
  stage {
    match {
      // Regex to parse the syslog format: TIME HOST TAG: MSG
      regex = "^(?P<time>[^ ]+ [^ ]+ [^ ]+) (?P<host>[^ ]+) (?P<tag>[^:]+): (?P<msg>.*)$"
    }
  }

  // Set the extracted time as the log timestamp
  stage {
    timestamp {
      source = "time"
      format = "Jan _2 15:04:05" // Adjust based on your Rsyslog format
    }
  }

  // Set Loki labels for indexing/searching
  stage {
    label {
      host = "host"
      tag  = "tag"
    }
  }

  // Remove the temporary fields used for parsing
  stage {
    output {
      drop_fields = ["time", "host", "tag"]
    }
  }
}

// Component to read the file and feed it to the processor
loki.source.file "remote_syslog" {
  targets    = [{"__path__" = "/var/log/remote.log", "job" = "remote_syslog"}]
  forward_to = [loki.process.syslog_logs.receiver]
}

// Component to write the processed logs to Loki
loki.write "default" {
  endpoint {
    url = "http://LOKI_SERVER_IP:3100/loki/api/v1/push"
  }
}
```

- **Action:** Install Alloy, replace `SYSLOG_SERVER_IP` and `LOKI_SERVER_IP` with your actual addresses, and start the Alloy service.

Key Benefits of This Architecture

Component	Benefit
Rsyslog Filtering (Nodes)	**Reduced Network Load:** Noisy, low-value logs (like routine CRONs) are dropped *before* transmission, saving bandwidth and processing time.
Central Syslog Server	**Single Point of Ingestion:** Provides a reliable, standardized ingestion point for all nodes, simplifying firewall rules and centralizing initial processing.

Component	Benefit
Alloy Forwarder	**Advanced Parsing:** Allows for sophisticated parsing, labeling, and timestamp extraction using processing stages *before* data reaches Loki, ensuring efficient indexing and query performance.
Loki	**Index Reduction:** Loki indexes only labels (metadata), not the full log text, making it highly cost-effective and scalable for your consolidated, filtered logs.

> **Can this be done with the rsyslog instance run on each node in a Podman container?**

Yes, running an **Rsyslog instance in a Podman container on each node is absolutely viable** and is a powerful way to manage the log filtering and forwarding.

This approach effectively turns the Rsyslog container into a dedicated, isolated "log collection sidecar" for the host node.

Here's the plan for how you would implement this architecture:

---

1. Node Logs vs. Container Logs

You must distinguish between two types of logs on the host and choose an appropriate input method for the Rsyslog container.

Log Source	Description	Rsyslog Container Input Method
Host System Logs (e.g., Kernel, Systemd, SSH)	Logs written by the host OS's native processes, typically sent to the local system journal/syslog socket (`/dev/log`).	**Mount the Syslog Socket:** Bind-mount the host's syslog socket (`/dev/log` or equivalent) into the container.
Application File Logs (e.g., NGINX access, custom app logs)	Logs written directly to files by applications running natively on the host.	**Bind-Mount Log Directories:** Use Podman mounts to make specific log files/directories (e.g., `/var/log/nginx/`) visible inside the container.

---

2. Podman Run Configuration (on each Node)

You need to run the Rsyslog container with the necessary mounts and network configuration.

A. Mount Host Logs

The most crucial step is making the host's logs accessible to the Rsyslog process inside the container.

1. **Syslog Socket Input (`imuxsock`):** The standard way Linux processes log to the system is via the Unix domain socket, often located at `/dev/log`.

```
# Podman command fragment for host logs
--volume /dev/log:/dev/log:ro
```

This allows the Rsyslog running inside the container to receive log events in real-time from the host's logging system, just as if it were running natively.

2. **Log File Input (`imfile`):** To read specific application log files (for filtering/forwarding), bind-mount those directories.

   ```
   # Podman command fragment for application logs
   --volume /var/log/app:/var/log/host_app:ro
   ```

   Inside the container, you would then configure Rsyslog's `imfile` module to read from `/var/log/host_app/application.log`.

B. Network and Configuration

The Rsyslog container needs network access to send the logs to the central Syslog Server (and ultimately, Loki/Alloy).

1. **Network:** By default, containers use their own network stack, but they can reach external IPs. You don't need to use `--net=host` unless you have complex network needs.
2. **Configuration:** It's best practice to separate your custom filtering rules from the container image, allowing you to update the configuration without rebuilding the image.

   ```
   # Podman command fragment for configuration
   --volume /etc/rsyslog.d/custom_rules.conf:/etc/rsyslog.d/99-custom.conf:ro
   ```

   The `custom_rules.conf` file on the host contains your **filtering logic** and the **forwarding rule** pointing to the central Syslog Server.

Full Example Podman Run Command

```
podman run -d \
    --name rsyslog-forwarder \
    --restart=always \
    --volume /dev/log:/dev/log:ro \
    --volume /var/log/nginx:/var/log/host_nginx:ro \
    --volume /etc/rsyslog.d/custom_forwarder.conf:/etc/rsyslog.d/99-forwarder.conf:ro \
    rsyslog/rsyslog-minimal:latest
```

---

3. Rsyslog Filtering Configuration (in `99-forwarder.conf`)

This configuration, mounted into the container, performs the crucial filtering before forwarding the traffic to your central Syslog Server.

```
# 1. Input Module for reading application files (using the bind mount)
module(load="imfile")

# 2. Input to read a specific file log from the host (mounted at /var/log/host_nginx)
input(type="imfile"
      File="/var/log/host_nginx/access.log"
      Tag="nginx-access"
      Facility="local6")

# 3. Filtering and Discarding Noise
# Discard all 'info' and 'debug' messages from the 'daemon' facility
daemon.info,debug     ~

# Discard high-volume, low-value cron logs based on content
:msg, contains, "CRON"    ~

# 4. Forwarding to Central Server (The remaining, filtered logs)
# Ensure you use the correct syntax for reliable forwarding (TCP/RELP recommended)
# Replace CENTRAL_SYSLOG_SERVER with the IP or hostname
*.* @@CENTRAL_SYSLOG_SERVER:514
```

This setup ensures that only the logs that survive the filtering rules are sent over the network, drastically reducing noise before the logs even reach the central Syslog Server/Alloy/Loki pipeline.