Configuring Loki

Network Monitoring & Management

Loki is a scalable log storage engine which is simple to install and run. It proves an API for adding and querying logs, and can be queried via a Grafana web interface.

You are going to work with your classmates to configure the following logging pipeline:

```
(devices) ----> rsyslog ----> promtail ----> loki
```

All the commands in this sheet can be run as “root”. Use `sudo -s` to get a root shell if necessary.

Start loki

Loki and promtail have been pre-installed on your campus server instance (srv1.campusX.ws.nsrc.org).

ONE of the users in your group should log into your srv1 instance and start loki:

```
systemctl enable loki   # start on future boots
systemctl start loki    # start now
journalctl -eu loki     # check for no errors during startup
```

You can check it is running by making a query to its API (https://github.com/grafana/loki/blob/master/docs/api.md) :

```
curl localhost:3100/ready; echo
```

It should respond with “Ready”, but it may take a minute or two before it gets into that state.

Ingesting logs

The tool “promtail” can be used to pick up logs and forward them to loki. It supports reading logs from:

- syslog (framed TCP)
- local logfiles (e.g. Apache logs)
- journald

The configuration we have provided (in `/etc/loki/promtail.yaml`) listens for syslog messages on TCP port 5140.

Again, ONE user in your group should do the following:

```
systemctl enable promtail   # start on future boots
systemctl start promtail    # start now
journalctl -eu promtail     # check for no errors during startup
```

Now we need to configure rsyslog on srv1 to forward to promtail. Create a file `/etc/rsyslog.d/25-promtail.conf` with the following contents:

```
*.*     action(type="omfwd" protocol="tcp"
               target="127.0.0.1" port="5140"
               Template="RSYSLOG_SyslogProtocol23Format"
               TCP_Framing="octet-counted")
```

(Note that `*.*` matches messages of all facility and severity, so we will forward both router logs and srv1's own system logs to loki)

Restart rsyslog and check for errors:

```
systemctl restart rsyslog
journalctl -eu rsyslog
```

Now send a test message:

```
logger "testing 123"
```

You should see that this still appears in the files that rsyslog itself writes:

```
grep testing /var/log/syslog
```

However, hopefully it has also been sent via promtail to loki. To check this, we need to perform some queries.

Querying

This part can be done by everyone in the group.

The command-line tool for querying loki is called "logcli". Log into srv1 and type "logcli" by itself to get some basic help.

Every log message is categorised using one or more "labels". Assuming that at least one log message has been ingested, we should be able to query for labels now. Try this command:

```
logcli labels
```

Note that the first line of logcli output is the API URL that it is communicating with. The remainder are the labels which exist in the log index. Hopefully there will be labels "host", "facility", "severity".

If not, see “Troubleshooting” below.

To check what different *values* of this label loki has seen, run this:

```
logcli labels host
```

In other words, this is the list of all the hosts which loki has seen messages from. Hopefully you’ll see “srv1” here - if not, see “troubleshooting” below. (Again, the first line is just the URL that the query used)

Now you can create a log query using logQL. The simplest query is just to show all log messages with a given label value:

```
logcli query '{host="srv1"}'
```

You should now see some log messages. Notice how each line starts with a timestamp and a list of labels. If there are any labels which are the same on all the messages shown, they are extracted as “common labels” to reduce clutter.

logcli by default will show you the last 30 matching logs over the last hour. There are flags available to change this:

```
logcli query '{host="srv1"}' --limit=10 --since=2h
```

Troubleshooting

If you are not able to see any logs, here are some things to check:

- Check that services are running

  ```
  systemctl status loki
  systemctl status promtail
  systemctl status rsyslog
  ```

- Check for errors in service logs

  ```
  journalctl -eu loki
  journalctl -eu promtail
  journalctl -eu rsyslog
  ```

- Run tcpdump on “localhost” socket

  rsyslog->promtail and promtail->loki communicate on localhost (127.0.0.1) so you can check this using tcpdump:

  ```
  tcpdump -i lo -nnA -s0
  ```

Now in another window generate a log message

```
logger "testing 123"
```

Do you see any connection on tcp port 5140 (rsyslog to promtail), and/or port 3100 (promtail to loki)?

Using LogQL

Here are some examples you can try of LogQL (https://github.com/grafana/loki/blob/master/docs/logql.md) , loki's log query language.

You can select messages which match multiple labels simultaneously:

```
logcli query '{host="srv1",severity="notice"}'
```

You can filter for messages which contain a given string:

```
logcli query '{host="srv1"} |="test"'
```

Multiple filter conditions can be chained: you will get messages which match *all* the conditions simultaneously. Here you will see messages which contain the string "test" *and* contain the string "123".

```
logcli query '{host="srv1"} |="test" |="123"'
```

Or for messages which *do not* contain a given string:

```
logcli query '{host="srv1"} !="test"'
```

For more advanced uses, use regular expression operators with `~` instead of `=`. For example:

```
# severity label is "warning", "error" or "critical"
logcli query '{severity=~"warning|error|critical"}'

# message contains "test" or "done", but does not contain
"plugins" or "alerts"
logcli query '{host="srv1"} |~"test|done" !~"plugins|alerts"'
```

Live tailing

"Live tailing" lets you see logs as they arrive in real time. Run this command:

```
logcli query '{host="srv1"}' -t
```

Now login to another console and type:

```
logger boom
```

Go back to the original console where logcli is running, and the message should appear.

Aggregated logs

If you have already done the rsyslog exercise, then your network device logs will already be arriving at srv1 - and therefore should also be arriving in loki. Can you see them?

Hints:

- Check which label values you have for the “host” label
- Check which label values you have for the “facility” label (remembering that you configured devices to send messages with facility “local0”)
- Query logs for a specific device, or query all logs with facility “local0”

If you need any help, please ask your instructor. Remember that you’ll only see logs which arrived *after* you configured rsyslog to talk to promtail/loki, so if necessary you can login to one of your devices and make a config change, to force it to send some logs.

Grafana

Grafana can serve as a web interface to loki, including LogQL and live tailing.

Start Grafana

ONE of the users in your group should start the grafana server on srv1:

```
systemctl enable grafana-server   # start on future boots
systemctl start grafana-server    # start now
journalctl -eu grafana-server     # check for no errors during
startup
```

Configure Grafana

ONE of the users in your group should do the following configuration.

Login to the Grafana interface at http://oob.srv1.campusX.ws.nsrc.org/grafana (http://oob.srv1.campusX.ws.nsrc.org/grafana) . The initial login is “admin” and “admin”. It will prompt you to change the password - use the class password.

On the left-hand side, find the “Configuration” icon - it looks like a cog. Under here click “Data Sources”.

Click the blue button “Add data source”.

From the list of data sources shown, under “Logging & document databases”, click on “Loki”

Under HTTP, in the URL field, type `http://localhost:3100` (it may look like it’s already there, but you still need to type over it)

Click the blue “Save & Test”. If successful, you should get a green banner saying “Data source connected and labels found”.

Browse Grafana

ALL of the users in your group can now do the remaining exercise.

Login to the Grafana web interface as above.

From the left-hand set of icons, choose the “Explore” option (it looks like a compass)

At the very top of the page, next to “Explore”, ensure that “Loki” is selected from the drop-down menu.

You can now explore logs by selecting the “Log labels” menu, selecting any particular label (for example “host”) and then selecting one of the labels offered (e.g. “srv1”). Then click “Run query” at the top right.

This will build a query for you: `{host="srv1"}` and display the matching logs. You will get a bar graph showing the number of matching logs per second, and the logs themselves shown at the end.

Toggle the “Unique labels” switch to see the labels next to each message.

When you click on an individual message to expand it, there are filter buttons which let you drill down to logs with matching labels. For example, if you look at a cron line and it has label “app” value “CRON”, then clicking the plus-magnifying glass will adjust the query so that it shows only logs with `app="CRON"` .

Try a query which matches on the content of the message. Under “Line contains”, enter “pam” (without the quotes). This should build the following query, which shows any message which contains the string “pam”:

```
{host="srv1"} |="pam"
```

At the very top-right, next to Run Query, is a button “Live”. Click on this to get live log tailing. Login to srv1 from another machine, and the web browser should show additional matching logs as they occur.

Additional exercises

These are optional improvements to the logging configuration - you can try them out if you have time, or just keep them as a reference.

Logging source IP addresses

The “hostname” field within a syslog message is easily forged, and it would be more secure if we could also record the originating IP address of each message.

We can do this by getting rsyslog to send `hostname/ipaddress` in the hostname field - this is a convention started by syslog-ng (https://www.syslog-ng.com/technical-documents/doc/syslog-ng-open-source-edition/3.23/administration-guide/59#global-options-chain-hostnames) - and getting promtail to split this into two labels.

Edit `/etc/loki/promtail.yaml` and replace the `relabel_configs` section of the syslog job, so it looks like this:

```
- job_name: syslog
  syslog:
    listen_address: 127.0.0.1:5140
  relabel_configs:
    - source_labels: [__syslog_message_severity]
      target_label: severity
    - source_labels: [__syslog_message_facility]
      target_label: facility
    - source_labels: [__syslog_message_hostname]
      regex: '([^/]*)'
      target_label: host
    - source_labels: [__syslog_message_hostname]
      regex: '(.*)/(.*)'
      replacement: '$1'
      target_label: host
    - source_labels: [__syslog_message_hostname]
      regex: '(.*)/(.*)'
      replacement: '$2'
      target_label: ip
    - source_labels: [__syslog_message_app_name]
      target_label: app
```

Hostnames which don’t contain a slash are copied to the “host” label as before. But for hostnames which do contain a slash, the part before goes to “host” and the part afterwards to “ip”.

Restart promtail and check for errors:

```
systemctl restart promtail
journalctl -eu promtail
```

Now replace `/etc/rsyslog.d/25-promtail.conf` with the following:

```
template(name="Custom_SyslogProtocol23Format" type="string"
     string="<%PRI%>1 %TIMESTAMP:::date-rfc3339%
%HOSTNAME%/%FROMHOST-IP% %APP-NAME% %PROCID% %MSGID%
%STRUCTURED-DATA% %msg%\n")

*.*     action(type="omfwd" protocol="tcp"
               target="127.0.0.1" port="5140"
               Template="Custom_SyslogProtocol23Format"
               TCP_Framing="octet-counted")
```

This replaces the built-in RSYSLOG_SyslogProtocol23Format (https://www.rsyslog.com/doc/v8-stable/configuration/templates.html#reserved-template-names) template with a custom version with `%HOSTNAME%/%FROMHOST-IP%` in the hostname field.

Restart rsyslog and check for errors:

```
systemctl restart rsyslog
journalctl -eu rsyslog
```

Make some logs (e.g. login to router and do `conf t`, `exit`)

Check to see if a new “ip” label has been seen, and what values it has:

```
logcli labels
logcli labels ip
```

Query your logs using the source IP label as a filter:

```
logcli query '{ip="100.68.X.Y"}'
```

(replacing X and Y as appropriate)

You can match multiple values with a regular expression. In a regular expression, `.` matches any character, so you need `\.` to match a literal dot. `\d` matches any single digit (0-9), and `\d+` matches any sequence of one or more digits.

However in LogQL the backslashes need to be doubled-up to make a valid query:

```
logcli query '{ip=~"100\\.68\\.\\d+\\.\\d+"}'
```

To avoid this issue, you can surround the entire value with backticks instead of double-quotes (new feature in loki 1.5):

```
logcli query '{ip=~`100\.68\.\d+\.\d+`}'
```

!!! Note

```
In all cases, the single-quotes outside the expression are very
important.
These stop the shell from doing its own expansions on
backslashes and
backticks.
```

Improving rsyslog parsing

Unfortunately, there is wide variation of how different vendor devices format logs. The “standard” defines a tag field:

```
timestamp hostname tag message
```

The tag can include the process name and pid, like `sshd[12345]:` . But some devices don’t include a tag: this means you may get a device which sends a message like this:

```
Invalid sequence number
```

but it gets interpreted by rsyslog (and forwarded to loki) as:

```
app="Invalid"
msg=" sequence number"
```

This displays wrongly in loki and is harder to search.

You can fix this by tweaking some settings in rsyslog, so that it only splits out the tag field if it ends in a colon. To do this, edit `/etc/rsyslog.conf` , find this section:

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

and add the following after it (replacing any RulesetParser section you already have):

```
# Some devices miss the "tag" field from syslog messages
# https://www.rsyslog.com/doc/v8-
stable/configuration/modules/pmrfc3164.html
```

```
parser(name="custom.rfc3164"
      type="pmrfc3164"
      force.tagEndingByColon="on"
      permit.squareBracketsInHostname="on"
      detect.YearAfterTimestamp="on")
# Replace the default parser chain
# https://www.rsyslog.com/doc/v8-
stable/configuration/ruleset/rsconf1_rulesetparser.html
module(load="pmciscoios")
$RulesetParser rsyslog.ciscoios
$RulesetParser rsyslog.rfc5424
$RulesetParser custom.rfc3164
```

This still allows devices to send using the newer and more strictly defined rfc5424 format, but if not, falls back to our customised rfc3164 parser which requires tags to end with a colon.

(Why isn't this the default behaviour? It's because the original "standard" RFC 3164 doesn't require a colon at the end of the tag, so rsyslog follows the standard. But in practice, most devices which send a tag *do* end it with a colon)

Application log aggregation

On srv1, there is a log file `/var/log/apache2/access.log` . Can you get this into loki too? Yes - by using promtail to read it.

Edit `/etc/loki/promtail.yaml` and add the following to the end under the `scrape_jobs` section. Alignment is important: the dash before `job_name` must align exactly with the dash of the previous `job_name` .

```
  - job_name: apache2_access
    static_configs:
      - targets: [localhost]
        labels:
          job: apache2
          __path__: /var/log/apache2/access.log
```

Restart promtail and check for errors:

```
systemctl restart promtail
journalctl -eu promtail
```

Using your web browser, refresh a page on srv1, or generate a web request from the command line:

```
curl localhost
```

Now check your logs:

```
logcli query '{job="apache2"}'
```

The results should look something like this:

```
Common labels: {filename="/var/log/apache2/access.log",
job="apache2"}
2020-03-06T14:39:01Z {} 127.0.0.1 - - [06/Mar/2020:14:39:01
+0000] "GET / HTTP/1.1" 200 1856 "-" "curl/7.47.0"
```

You are storing the raw log lines (https://httpd.apache.org/docs/2.4/logs.html#accesslog) .

It is possible to perform additional processing in a pipeline (https://github.com/grafana/loki/blob/master/docs/clients/promtail/configuration.md#pipeline_stages) , to extract fields from this into labels. However, you need to be careful to avoid “high cardinality” labels - that means labels which can have many different values. Loki assembles logs with the same labels into the same chunks, but if you have many different labels then loki can be forced to track many small chunks which makes it very inefficient and may crash.

For example, the source IP address may be high cardinality if your web site could be accessed from anywhere on the Internet - this means it is not a good candidate for a label. However the status code (e.g. “200”) only has a small number of possible values so is safe in a label, and also useful (e.g. you can quickly search for requests which generated a “500” status)

To test this, add the following to the end of the scrape config. The new section “pipeline_stages” should line up with “static_configs” above it.

```
    pipeline_stages:
      - regex:
          expression: '^\S+ \S+ \S+ \[[^\]]*\] "[^"]*" (?
P<code>\d+) \d+'
      - labels:
          code:
```

Restart promtail again. Generate some requests:

```
curl localhost
curl localhost/foobar
```

Finally, check the logs again:

```
logcli query '{job="apache2"}'
```

The results should look something like this:

```
Common labels: {filename="/var/log/apache2/access.log",
job="apache2"}
2020-03-06T15:02:28Z {code="404"} 127.0.0.1 - -
[06/Mar/2020:15:02:28 +0000] "GET /foobar HTTP/1.1" 404 432 "-"
"curl/7.47.0"
2020-03-06T15:02:27Z {code="200"} 127.0.0.1 - -
[06/Mar/2020:15:02:27 +0000] "GET / HTTP/1.1" 200 1856 "-"
"curl/7.47.0"
```

Now you can quickly search for logs which have an error code starting 4 or 5:

```
logcli query '{job="apache2",code=~"[45].+"}'
```

Linux syslog aggregation

If you still have spare time, remember that you also have other Linux hosts (host1-6) in your campus. Can you configure them to log centrally as well?

For syslog there are two main options: you could get the host's rsyslog server to send to rsyslog on srv1 (TCP or UDP port 514), or you could get it to write syslog messages directly to loki (TCP port 5140).

You could also run promtail on the remote hosts, which would be useful if you also needed to pick up application logs from those hosts.

How to configure any of these options is left as an exercise.

Further reading

- Grafana loki datasource (https://grafana.com/docs/grafana/latest/features/datasources/loki/)
- Loki docs (https://github.com/grafana/loki/tree/master/docs)
- LogQL (https://github.com/grafana/loki/blob/master/docs/logql.md)
- GrafanaCONline video on Loki (https://www.youtube.com/watch?v=TcmvmqbrDKU)

To control how long Loki keeps logs for, see Loki Storage Retention (https://github.com/grafana/loki/blob/master/docs/operations/storage/retention.md) . Old logs can be automatically deleted by setting `retention_deletes_enabled` , `retention_period` and `max_look_back_period` .

For storing large volumes of logs, Loki can be configured to use remote storage (https://github.com/grafana/loki/blob/master/docs/operations/storage/README.md) , including the ability to store log chunks into S3-compatible storage such as minio (https://min.io/) .

Reference: installation

This section is for reference only - it documents how loki and promtail are installed.

All the loki components are available as pre-built binaries.

Fetch and unpack the latest release from the releases page (https://github.com/grafana/loki/releases/) :

```
wget
https://github.com/grafana/loki/releases/download/vX.Y.Z/loki-
linux-amd64.zip
wget
https://github.com/grafana/loki/releases/download/vX.Y.Z/logcli-
linux-amd64.zip
wget
https://github.com/grafana/loki/releases/download/vX.Y.Z/promtai
l-linux-amd64.zip

mkdir /opt/loki
unzip loki-linux-amd64.zip -d /opt/loki
unzip logcli-linux-amd64.zip -d /opt/loki
unzip promtail-linux-amd64.zip -d /opt/loki

ln -s /opt/loki/logcli-linux-amd64 /usr/local/bin/logcli
```

Create a data directory for loki to use:

```
mkdir /etc/loki
mkdir /var/lib/loki
chown syslog:syslog /var/lib/loki
```

Use a text editor to create a systemd unit file `/etc/systemd/system/loki.service` with the following contents:

```
[Unit]
Description=Loki logging server
Documentation=https://grafana.com/oss/loki/
After=network-online.target
# Fix shutdown delays: if promtail is running on the same host,
# loki should start first, and shutdown after.
Before=promtail.service

[Service]
User=syslog
Group=syslog
Restart=on-failure
RestartSec=5
```

```
WorkingDirectory=/var/lib/loki
EnvironmentFile=/etc/default/loki
ExecStart=/opt/loki/loki-linux-amd64 $OPTIONS

[Install]
WantedBy=multi-user.target
```

Also create an options file `/etc/default/loki` with the following contents:

```
OPTIONS='-config.file=/etc/loki/loki.yaml -log.level=warn'
```

(`-log.level=warn` is to suppress excessive log messages (https://groups.google.com/d/msg/lokiproject/ZmgpAQ5vCjc/i9xCFeVEAQAJ) from loki's storage backend)

Create the initial `/etc/loki/loki.yaml` with the following contents (https://raw.githubusercontent.com/grafana/loki/master/cmd/loki/loki-local-config.yaml) :

```
auth_enabled: false

server:
  http_listen_port: 3100
  grpc_listen_port: 9096

common:
  path_prefix: /var/lib/loki
  storage:
    filesystem:
      chunks_directory: /var/lib/loki/chunks
      rules_directory: /var/lib/loki/rules
  replication_factor: 1
  ring:
    instance_addr: 127.0.0.1
    kvstore:
      store: inmemory

schema_config:
  configs:
    - from: 2020-10-24
      store: boltdb-shipper
      object_store: filesystem
      schema: v11
      index:
        prefix: index_
```

```
        period: 24h

ruler:
  alertmanager_url: http://localhost:9093/alertmanager

compactor:
  retention_enabled: true

limits_config:
  retention_period: 91d
  enforce_metric_name: false
  reject_old_samples: true
  reject_old_samples_max_age: 168h

# By default, Loki will send anonymous, but uniquely-
identifiable usage and configuration
# analytics to Grafana Labs. These statistics are sent to
https://stats.grafana.org/
#
# Statistics help us better understand how Loki is used, and
they show us performance
# levels for most users. This helps us prioritize features and
documentation.
# For more information on what's sent, look at
#
https://github.com/grafana/loki/blob/main/pkg/usagestats/stats.g
o
# Refer to the buildReport method to see what goes into a
report.
#
# If you would like to disable reporting, uncomment the
following lines:
analytics:
  reporting_enabled: false
```

Note that this configuration is insecure. In practice, you should front loki with an authentication proxy (https://github.com/grafana/loki/blob/master/docs/operations/authentication.md) , or use iptables to limit access.

Create `/etc/systemd/system/promtail.service` :

```
[Unit]
Description=Promtail
```

```
Documentation=https://github.com/grafana/loki/tree/master/docs/clients/promtail
After=network-online.target

[Service]
User=syslog
Group=syslog
Restart=on-failure
RestartSec=5
WorkingDirectory=/var/log
EnvironmentFile=/etc/default/promtail
ExecStart=/opt/loki/promtail-linux-amd64 $OPTIONS

[Install]
WantedBy=multi-user.target
```

Create `/etc/default/promtail`:

```
OPTIONS='-config.file=/etc/loki/promtail.yaml'
```

Create `/etc/loki/promtail.yaml`:

```
# Promtail can expose prometheus metrics
server:
  http_listen_address: 127.0.0.1
  http_listen_port: 9080
  grpc_listen_port: 0

# How to find the loki server(s)
clients:
  - url: http://127.0.0.1:3100/loki/api/v1/push

# Logs to read and forward
scrape_configs:
  - job_name: syslog
    syslog:
      listen_address: 127.0.0.1:5140
    relabel_configs:
      - source_labels: [__syslog_message_severity]
        target_label: severity
      - source_labels: [__syslog_message_facility]
        target_label: facility
```

```
      - source_labels: [__syslog_message_hostname]
        target_label: host
      - source_labels: [__syslog_message_app_name]
        target_label: app
```

Finally, tell systemd to read our new service files:

```
systemctl daemon-reload
```

Grafana is available (https://grafana.com/docs/grafana/latest/installation/) pre-packaged in many formats, including deb packages for Ubuntu:

```
curl -fsSL https://apt.grafana.com/gpg.key
>/etc/apt/trusted.gpg.d/grafana.asc
echo "deb https://apt.grafana.com stable main"
>/etc/apt/sources.list.d/grafana.list
apt-get update
apt-get install grafana
```