Network Log Aggregation with Loki

Dan Poltawski

UKNOF 51

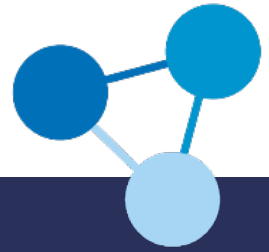
dan.poltawski@tnp.net.uk

dan@fedi.talktodan.com

Why Loki?

- Features
  - Open source (& self hosted)
  - Cost effective and easy to operate
    - Minimal infrastructure requirements
    - Highly available, horizontally scalable
  - Integrates with existing Prometheus infrastructure
  - Powerful Query Language (LogQL)
    - Metrics and Alerting
  - Supports Multi-tenancy

https://github.com/grafana/loki



Loki: contrast to other log solutions

`2023-04-04T10:01:02.123456789Z {ip="10.0.0.1", hostname="rtr1"}` `Login 'danp' from 10.0.0.2 (ssh)`

Indexed

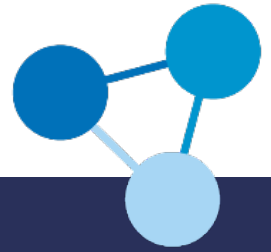
(Timestamp & Prometheus-style labels)

Unindexed

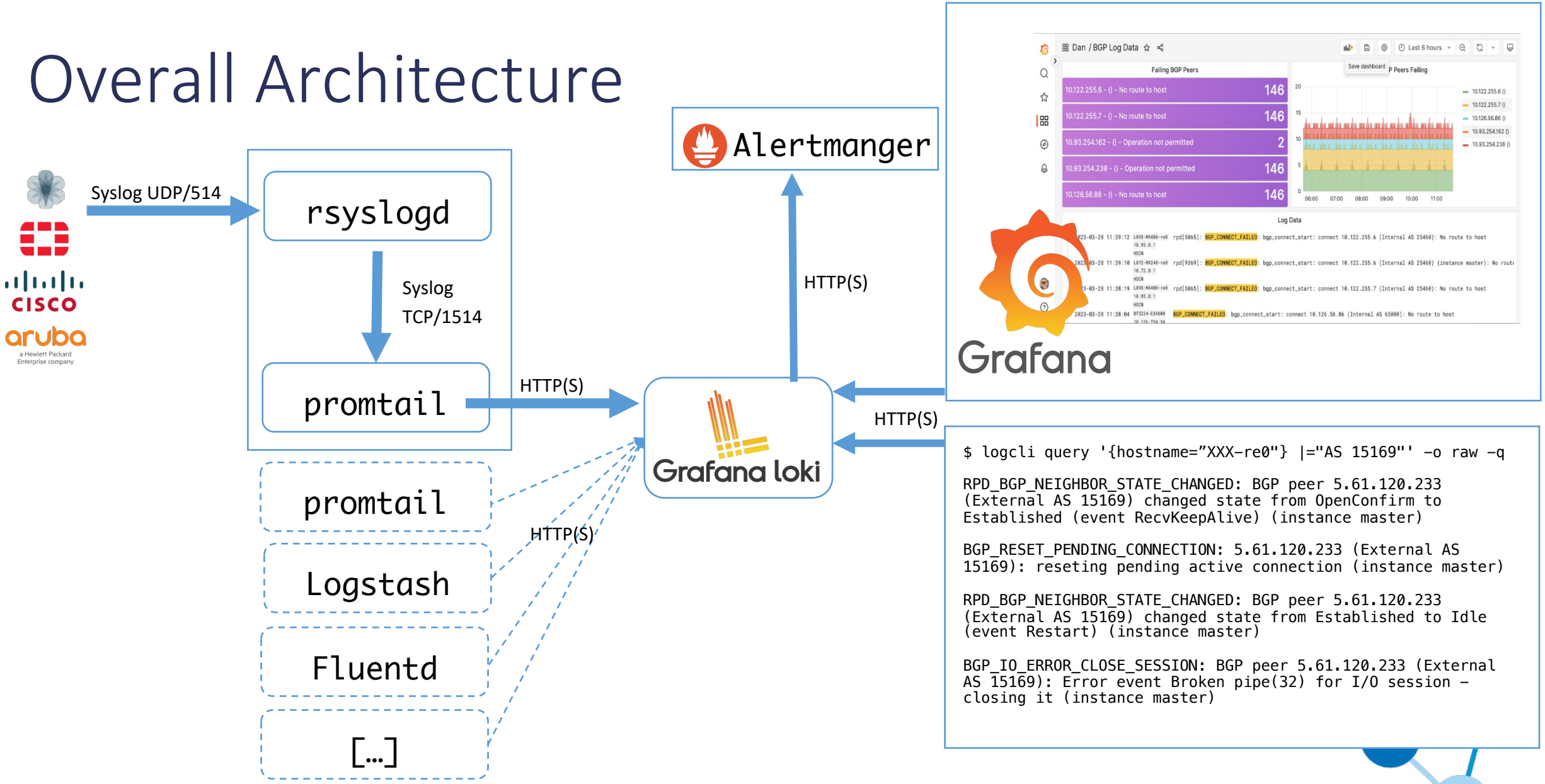
(Log content)

- Indexing
  - Only indexes the metadata around source of log line
  - Small RAM requirements
- Log data
  - Stored **compressed** in block storage chunks
  - Example - 40GB log data from webserver
    - 100GB in Elasticsearch - **9GB** in Loki

- Filtering and Parsing
  - Happens at **query** time
  - Parallelised components for performance
  - Grafana Labs achieve query speeds of up of 80GB/s

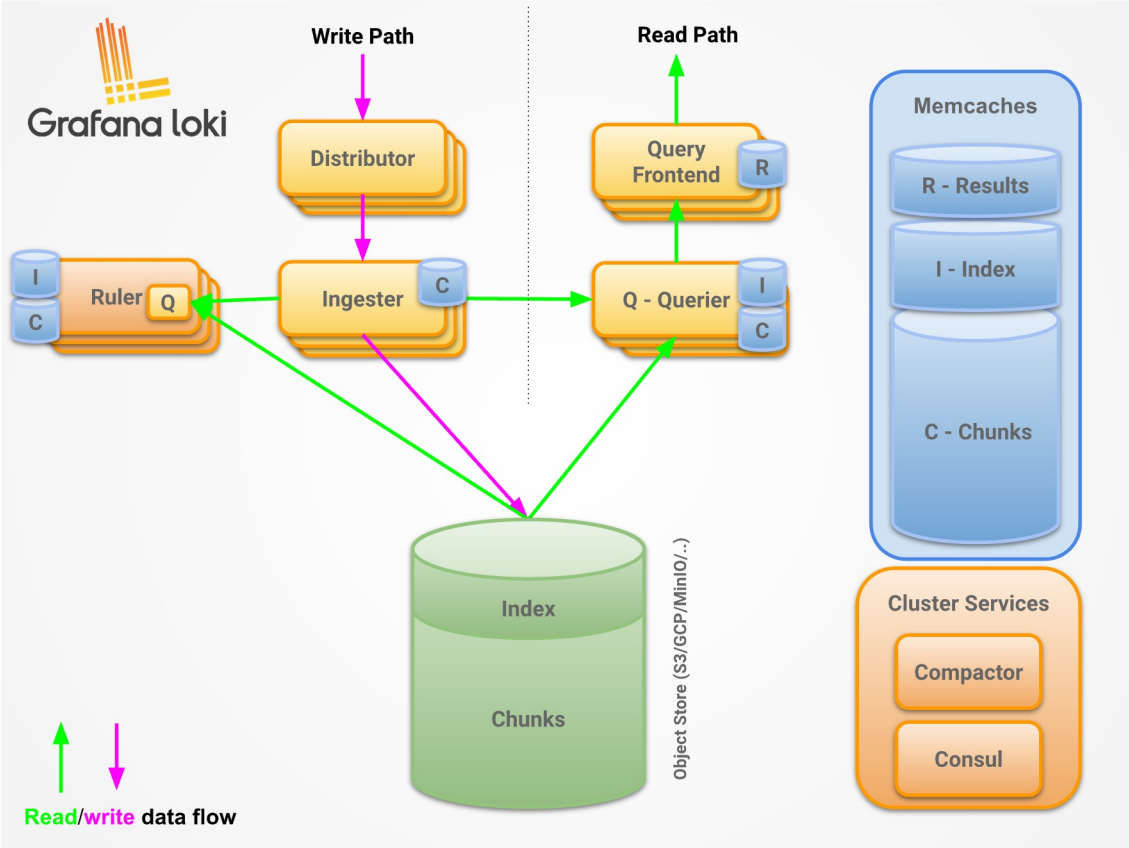


Overall Architecture



Loki architecture

- Single binary (golang)
- Deployment Options
  - Monolithic
  - Simple scalable deployment
  - Microservices
- Storage – only dependency
  - Local Filesystem
  - S3 (plus substitutes like MiniIO)
  - Azure blob storage
  - GCS
  - Openstack Swift
  - (etc)



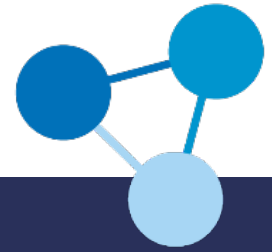
Full microservices architecture

Syslog Ingestion - Rsyslog & Promtail

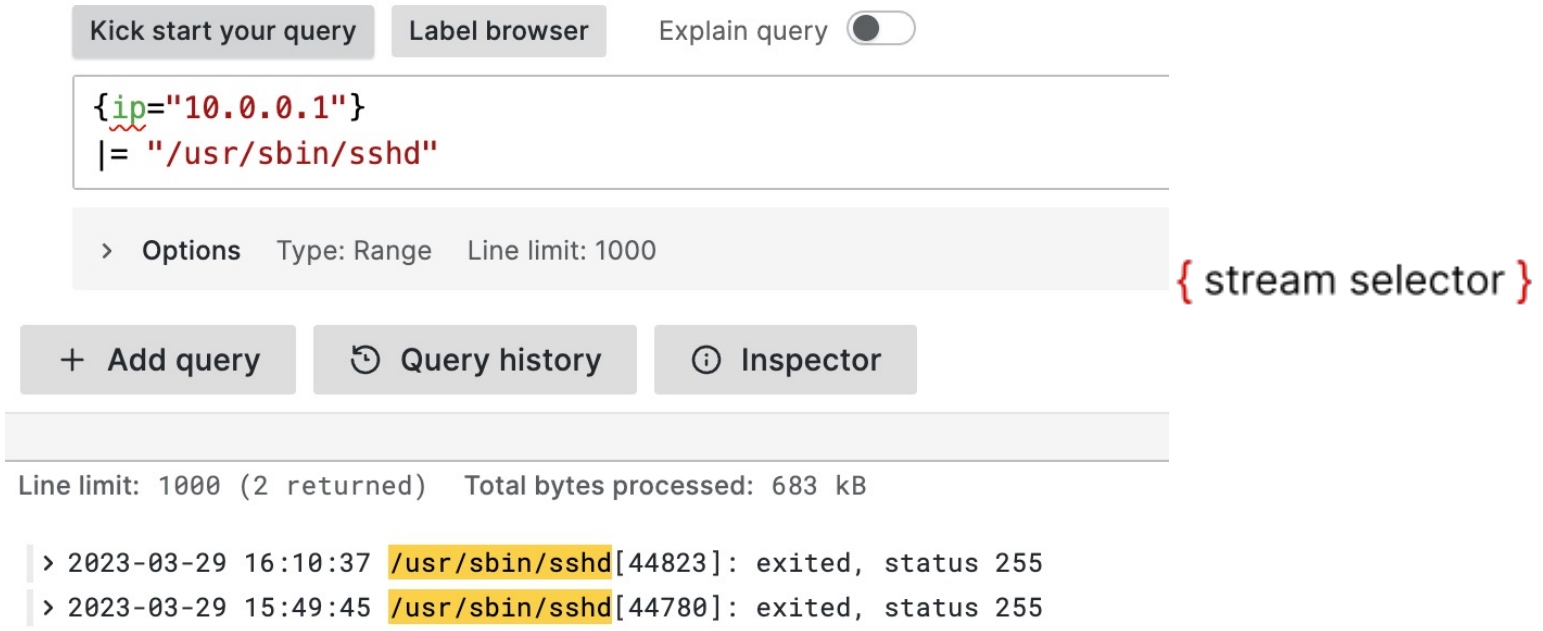
- Rsyslog
  - Forwarder in front of promtail to normalize messages via Syslog/UDP
  - Use `jsonmesg` to send all raw data to promtail
- Promtail
  - Ingests syslog message from rsyslog
  - Runs pipeline on messages
    - Parse data from rsyslog
    - Transform or filter lines
    - Generate Prometheus metrics
  - Send onward to loki

```
/etc/rsyslog.d/send-to-loki.conf
module(load="imudp")
input(type="imudp" port="514" ruleset="networking-loki")
template(name="loki" type="list") {
     constant(value="<")
     property(name="PRI")
     constant(value=">1 ")
     property(name="TIMESTAMP" dateFormat="rfc3339")
     constant(value=" ")
     property(name="HOSTNAME")
     constant(value=" ")
     property(name="APP-NAME")
     constant(value=" ")
     property(name="PROCID")
     constant(value=" ")
     property(name="MSGID")
     constant(value=" ")
     property(name="STRUCTURED-DATA")
     constant(value=" ")
     property(name="jsonmesg")
     constant(value="\n")
}
ruleset(name="networking-loki"){
    action(type="omfwd" protocol="tcp" target="127.0.0.1”
     port="1514” Template="loki" TCP_Framing="octet-counted”)
}
```

Link to full configs snippet



LogQL – basics



{ stream selector }

optional log pipeline

filter operator

|=
!=
|~
!~

line filter expression

=
!=
=~
!~
>
>=
<
<=

label filter expression

parser expression
line format expression
label format expression

LogQL – Parsers: pattern

Log line:

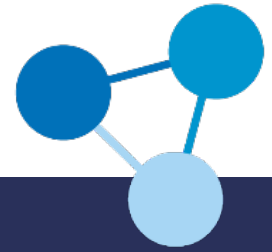
```
Failed password for root from 61.177.172.147 port 39028 ssh2
```

Query:

```
{hostname=”HOSTNAME-TEST"}
| pattern `Failed password for <username> from <srcip> port`
| srcip != ip(`10.0.1.0/27`)
| line_format `{{.hostname}} - {{.username}} ({{.srcip}})`
```

Output:

```
HOSTNAME-TEST - root (61.177.172.147)
```



LogQL – Parsers: logfmt

Log line:

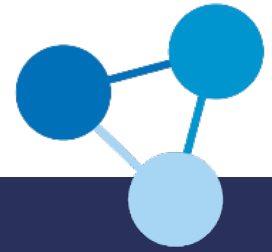
```
time=20:02:40 devname="Core-FG" eventtime=1680033760096179863 tz="+0100" logid="1059028704"
type="utm" subtype="app-ctrl" eventtype="signature" level="information" vd="Corporate" appid=15895
srcip=10.64.128.6 srccountry="Reserved" dstip=1.1.1.1 dstcountry="Australia" srcport=52026
dstport=443 srcintf="v1974_ContInt" srcintfrole="lan" dstintf="VCorp-SvrGlob" dstintfrole="wan"
proto=6 service="SSL" direction="outgoing" policyid=171 poluuid="575255da-1eac-51eb-ff8a-
466643d7739b" policytype="policy" sessionid=1267665482 action="pass" appcat="Network.Service"
app="SSL" hostname="cloudflare-dns.com" incidentserialno=252207910 url="/" msg="Network.Service: SSL"
apprisk="elevated" scertcname="cloudflare-dns.com”
```

Query:

```
{type="fortios"} | logfmt | subtype=`app-ctrl`, dstip=`1.1.1.1`
| line_format "src: {{.srcip}} dst: {{.dstip}}”
```

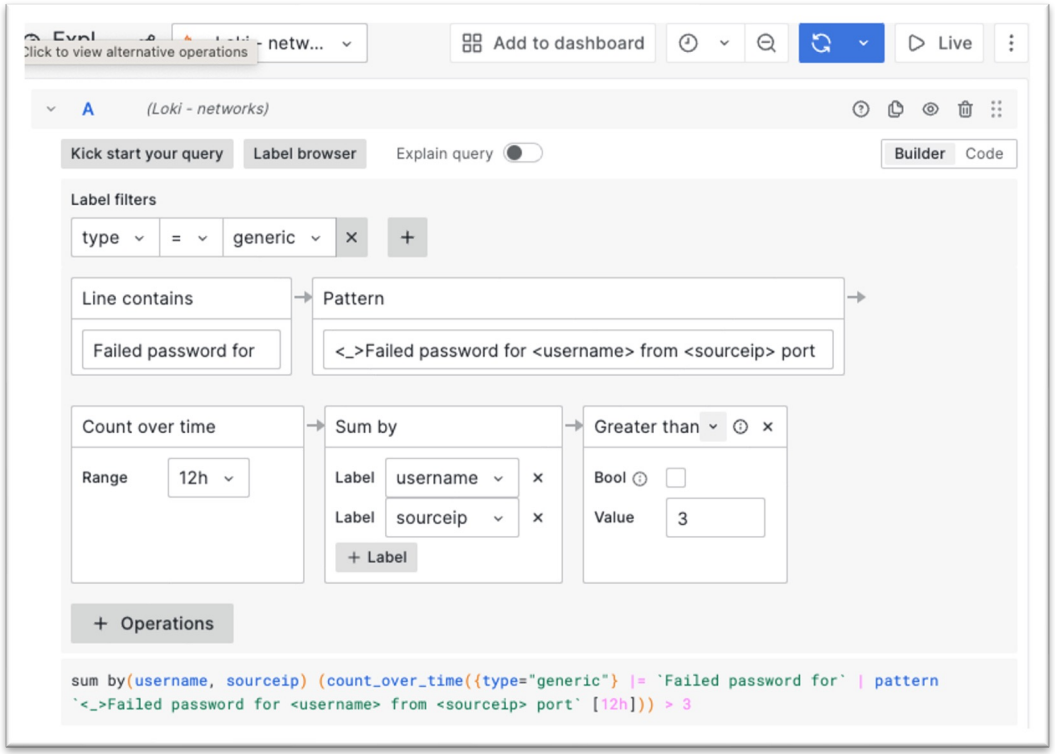
Output:

```
src: 10.64.128.6 dst: 1.1.1.1
```



LogQL Continued

- ..yet more parsers
  - `json` – parses json much like logfmt
  - `regexp` – golang RE2 syntax to extract labels
- Matching IP addresses with `ip("<pattern>")`
  - `ip("192.0.2.0"), ip("::1")`
  - `ip("192.0.2.0/24"), ip("2001:db8::/32")`
  - `ip("192.0.2.1-192.0.2.7"), ip("2001:db8::1-2001:db8::8")`
- Graphical query builder in Grafana!



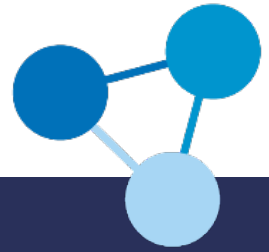
- Experiment with the LogQL Analyzer
  - https://grafana.com/docs/loki/latest/logql/analyzer/

Metric Queries / Alerting

```
sum by (ip, username, sourceip) (
    count_over_time(
        {type="generic"}
        |= "Failed password for"
        | pattern `<_>Failed password for <username> from <sourceip> port`
        [12h]
    )
) > 3
```

Table

ip	sourceip	username	Value #A
10.125.195.2	10.64.131.51	tnp-configbackups	24
10.126.56.166	10.64.131.51	tnp-configbackups	20
10.64.131.33	10.64.131.56	tnp-configbackups	20



the networking people

connectivity | consultancy | engineering

Questions?