SKOGUL FOR SYSADMINS

KRISTIAN LYNGSTØL

FORNEBU, SEPTEMBER 2022

AGENDA

1. Install, start, stop
2. Fundamentals
3. Documentation
4. Batch and burn
5. Telemetry
6. Enrichment
7. Misc

Installation

```
# wget https://github.com/telenornms/..../skogul...rpm
# yum localinstall skogul-...rpm
# vim /etc/skogul/conf.d/....
```

```
# systemctl restart skogul
# journalctl -u skogul
$ man skogul
$ cd /usr/share/doc/skogul-*/
```

CONFIGURATION

- Stored in `/etc/skogul/conf.d`
- JSON
- `skogul -show -f /etc/skogul/conf.d`

MODULES

- Configuration instantiates a list of modules
- Type: underlying implementation (sql, http, tcp, json, etc)
- Configured name: specific instance
- Multiple instances of the same module type can be set up. E.g.: Multiple influx senders.

```
{
  "receivers": {
    "myAPI": {
      "type": "http",
      "handlers": {
        "/": "someHander"
      }
    }
  }
}
```

```
{
  "receivers": {
    "myAPI": {
      "type": "http",
      "handlers": {
        "/": "someHander"
      }
    }
  }
}
```

```
{
  "receivers": {
    "myAPI": {
      "type": "http",
      "handlers": {
        "/": "someHander"
      }
    }
  }
}
```

Modules are independent, but can cross reference each other.

Modules are independent, but can cross reference each other.

Example: Receive data on HTTP and UDP, but send to the same influx instance.

Module categories:

- Receiver
- Handler (sort of)
- Parser
- Encoder
- Transformer
- Sender

BASIC CONFIG

```
{
  "receivers": { ... },
  "handlers": { ... },
  "parsers": { ... },
  "encoders": { ... },
  "transformers": { ... },
  "senders": { ... }
}
```

BASIC CONFIG

```
{
  "receivers": { ... },
  "handlers": { ... },
  "parsers": { ... },
  "encoders": { ... },
  "transformers": { ... },
  "senders": { ... }
}
```

PS: Handler is just a collection of parser, transformer(s) and sender.

```
{
  "receivers": {
    "myReceiver": {
      "type": "udp",
      "handler": "foohandler"
    }
  },
  "handlers": {
    "foohandler": {
      "parser": "juniper",
      "sender": "print"
    }
  }
}
```

AUTO MODULES

Implicitly defined by their type name when a zero-configuration instance is valid.

```
debug
  Prints received metrics to stdout.

  Aliases: print

  This  module  does  not have to be explicitly defined in
  your configuration file. If you refer to it by one of it's
  implementation names, a default-variant of it will be used.

  Settings:

  encoder - EncoderRef
         Which encoder to use. Defaults to prettyjson.
  prefix - string
         Prefix to print before any metric
```

```
debug
  Prints received metrics to stdout.

  Aliases: print

  This  module  does  not have to be explicitly defined in
  your configuration file. If you refer to it by one of it's
  implementation names, a default-variant of it will be used.

  Settings:

  encoder - EncoderRef
        Which encoder to use. Defaults to prettyjson.
  prefix - string
        Prefix to print before any metric
```

```
debug
  Prints received metrics to stdout.

  Aliases: print

  This  module  does  not have to be explicitly defined in
  your configuration file. If you refer to it by one of it's
  implementation names, a default-variant of it will be used.

  Settings:

  encoder - EncoderRef
        Which encoder to use. Defaults to prettyjson.
  prefix - string
        Prefix to print before any metric
```

```
{
  "receivers": { "type": "tester", "handler": "foo" },
  "handlers": {
    "foo": {
      "parser": "myP",
      "transformers": [ "n" ],
      "sender": "p"
    }
  },
  "parsers": { "myP": { "type": "json" } },
  "transformers": { "n": { "type": "now" } },
  "senders": { "p": { "type": "print" } }
}
```

```
{
  "receivers": { "type": "tester", "handler": "foo" },
  "handlers": {
    "foo": {
      "parser": "myP",
      "transformers": [ "n" ],
      "sender": "p"
    }
  },
  "parsers": { "myP": { "type": "json" } },
  "transformers": { "n": { "type": "now" } },
  "senders": { "p": { "type": "print" } }
}
```

```
{
  "receivers": { "type": "tester", "handler": "foo" },
  "handlers": {
    "foo": {
      "parser": "json",
      "transformers": [ "n" ],
      "sender": "p"
    }
  },
  "parsers": { "json": { "type": "json" } },
  "transformers": { "n": { "type": "now" } },
  "senders": { "p": { "type": "print" } }
}
```

```
{
  "receivers": { "type": "tester", "handler": "foo" },
  "handlers": {
    "foo": {
      "parser": "json",
      "transformers": [ "n" ],
      "sender": "p"
    }
  },
  "parsers": { "json": { "type": "json" } },
  "transformers": { "n": { "type": "now" } },
  "senders": { "p": { "type": "print" } }
}
```

```
{
  "receivers": { "type": "tester", "handler": "foo" },
  "handlers": {
    "foo": {
      "parser": "json",
      "transformers": [ "now" ],
      "sender": "p"
    }
  },
  "parsers": { "json": { "type": "json" } },
  "transformers": { "now": { "type": "now" } },
  "senders": { "p": { "type": "print" } }
}
```

```
{
  "receivers": { "type": "tester", "handler": "foo" },
  "handlers": {
    "foo": {
      "parser": "json",
      "transformers": [ "now" ],
      "sender": "p"
    }
  },
  "parsers": { "json": { "type": "json" } },
  "transformers": { "now": { "type": "now" } },
  "senders": { "p": { "type": "print" } }
}
```

```
{
  "receivers": { "type": "tester", "handler": "foo" },
  "handlers": {
    "foo": {
      "parser": "json",
      "transformers": [ "now" ],
      "sender": "print"
    }
  },
  "parsers": { "json": { "type": "json" } },
  "transformers": { "now": { "type": "now" } },
  "senders": { "print": { "type": "print" } }
}
```

```
{
  "receivers": { "type": "tester", "handler": "foo" },
  "handlers": {
    "foo": {
      "parser": "json",
      "transformers": [ "now" ],
      "sender": "print"
    }
  },
  "parsers": { "json": { "type": "json" } },
  "transformers": { "now": { "type": "now" } },
  "senders": { "print": { "type": "print" } }
}
```

```
{
  "receivers": { "type": "tester", "handler": "foo" },
  "handlers": {
    "foo": {
      "parser": "json",
      "transformers": [ "now" ],
      "sender": "print"
    }
  }
}
```

CONF.D

By default, loads all .json config files in `/etc/skogul/conf.d/` and merges it.

```
$ ls /etc/skogul/conf.d
telegraf.json
telemetry-forwarding.json
common.json
```

STARTUP

Every receiver is started in parallel. Everything follows from that.

Unless modules reference each other, they are completely isolated.

Use unique names for modules for future-safety.

Use unique names for modules for future-safety.

"forwarding" - high chance of naming collision.

Use unique names for modules for future-safety.

"forwarding" - high chance of naming collision.

"telemetry-forwarding", *"fwa-forwarding"* - low chance of naming collision.

DOCUMENTATION

Manual page is a mix of auto-generated per module, and hand written.

If it's not in the manual page, it's not configurable.

```
tcp
    Listen for data on a tcp socket, reading one collection
    per line.

    Settings:

    address - string
           Address and port to listen to.

           Example(s): [::1]:3306

    handler - HandlerRef
           Handler used to parse, transform and send data.
```

```
udp
    Accept  UDP  messages, one UDP message is one
    container. Combine with  protobuf  parser  to
    receive Juniper telemetry.

    Settings:

    address - string
           Address and port to listen to.

           Example(s): [::1]:3306

    backlog - int
           Number of queued messages that are not
           delivered before the  receiver  starts
           blocking.  Defaults  to 100. Even when
```

```
udp
    Accept  UDP  messages, one UDP message is one
    container. Combine with  protobuf  parser  to
    receive Juniper telemetry.

    Settings:

    address - string
           Address and port to listen to.

           Example(s): [::1]:3306

    backlog - int
           Number of queued messages that are not
           delivered before the  receiver  starts
           blocking.  Defaults  to 100. Even when
```

```
12
13      backlog - int
14              Number of queued messages that are not
15              delivered before the  receiver  starts
16              blocking.  Defaults  to 100. Even when
17              this is full, the  kernel  will  still
18              buffer  data  on  top  of  this value.
19              Higher values  give  smoother  perfor-
20              mance, but slightly more memory usage.
21              Actual memory usage is a factor of av-
                erage message size * backlog-fill.
22
23
24      buffer - int
25              Set  kernel  read  buffer.  Default is
26              kernel-specific.  Bumping  this   will
37              make  it  easier to handler bursty UDP
```

```
40      handler - HandlerRef
41              Handler used to parse, transform and
42              send data.
43
44      packetsize - int
45              UDP Packet size note: max. UDP read
46              size is 65535
47
48      threads - int

49              Number of worker go routines to use,
50              which loosely translates to parallel
51              execution. Defaults to number of CPU
52              threads, with a minimum of 20. There
53              is no correct number, but the value
54              depends on how fast your senders are.
```

HandlerRef means "the name you configured for a handler". Similar for SenderRef etc.

`/usr/share/doc/skogul-.../` contains several examples, and richer documentation

```
$ skogul -help
Usage of ./skogul:
  -d string
      Path to skogul configuration files. Deprecated, use -f.
  -f string
      Path to skogul config to read. Either a file or a direc
  -help
      Print more help
  -loglevel string
      Minimum loglevel to display ([e]rror, [w]arn, [i]nfo, [
  -make-man
      Output RST documentation suited for rst2man
  -pprof string
      Enable profiling over HTTP, value is http endpoint, e.g
  -show
      Print the parsed JSON config instead of starting
```

```
          net | Sends json data to a network endpoint.
enrichmentupdater | Updates the enrichment database of an enr
         http | Sends Skogul-formatted JSON-data to a HTTP endp
              | other Skogul instance?). Highly useful in scena
              | data collection methods spread over several ser
          mnr | Sends M&R line format to a TCP endpoint.
          sql | Execute a SQL query for each received metric, u
              | Any query can be run, and if multiple metrics a
              | same container, they are all executed in a sing
              | which means the batch-sender will greatly incre
              | Supported engines are MySQL/MariaDB and Postgre
         null | Discards all data. Mainly useful for testing th
              | decoupled from storage.
        debug | Prints received metrics to stdout.
         dupe | Sends the same metrics to all senders listed in
       fanout | Fan out (load balance) to a fixed number of thr
```

```
37          | decoupled from storage.
38    debug | Prints received metrics to stdout.
39     dupe | Sends the same metrics to all senders listed in
40   fanout | Fan out (load balance) to a fixed number of thr
41          | data on. This is rarely needed, as receivers sh
42     file | Writes metrics to a file.
43   influx | Send to a InfluxDB HTTP endpoint. The sender ca
44          | send the data to a single measurement, send it
45          | extracted from the metadata of a metric, or a c
46          | the "measurement" serves as a default measureme
47          | metric doesn't have the key presented in
48          | "measurementfrommetadata".

49      log | Logs a message, mainly useful for enriching deb
50          | conjunction with, for example, dupe and debug.
51  backoff | Forwards data to the next sender, retrying afte
```

```
 95          | and completeness also depends on the modules. E
 96          | gathered are: parse errors, send errors, number
 97          | messages.
 98       stdin | Reads from standard input, one collection per l
 99          | to pipe collections to Skogul on a command line
100        test | Generate dummy-data. Useful for testing, includ
101          | with the http sender to send dummy-data to an o
102          | instance.

103         tcp | Listen for data on a tcp socket, reading one co
104         sql | Periodically poll a database for information. S
105        http | Listen for metrics on HTTP or HTTPS. Optionally
106          | authentication. Each request received is passed
107          | a single HTTP receiver can listen for multiple
108          | on URL used.
109        file | Reads from a file, then steps. Assumes one coll
```

```
127             | the original.
128      switch | Conditionally apply transformers.
129         now | Forcibly set a timestamp (set to now/current ti

130             | to ensure the container is valid even if the so
131             | provide a timestamp.
132      enrich | PROTOTYPE/ALPHA: Static enrichment. Use a json-
133             | document to enrich incoming metrics with additi
134             | docs/examples/ directory for an actual example.
135             | production ready, and should only be used if yo
136             | file bug reports and update your setup as the t
137    metadata | Enforces custom-rules on metadata of metrics.
138   timestamp | Extract a timestamp from the container data.
139   templater | Executes metric templating. See separate docume
140             | skogul templating works.
141        data | Enforces custom rules for data fields of metric
```

BATCH AND BURN

Performance and isolation with little effort!

Performance and isolation with little effort!

Batches up to 10 metrics for up to 1 second by default. Can easily and safely be adjusted to 1000 - rarely sensible to batch for longer than 1sec

Performance and isolation with little effort!

Batches up to 10 metrics for up to 1 second by default. Can easily and safely be adjusted to 1000 - rarely sensible to batch for longer than 1sec

Burner sends overflow to alternate sender instead of blocking.

Performance and isolation with little effort!

Batches up to 10 metrics for up to 1 second by default. Can easily and safely be adjusted to 1000 - rarely sensible to batch for longer than 1sec

Burner sends overflow to alternate sender instead of blocking.

Come to think of it, I should up the default from 10 to 100...

```
{
  "type": "batch",
  "burner": "null",
  "next": "to-influx"
}
```

Uses null-sender to discard(burn) data if influx is too slow, instead of blocking.

VERY useful for lab/testing without impacting production.

Uses null-sender to discard(burn) data if influx is too slow, instead of blocking.

VERY useful for lab/testing without impacting production.

Could alternatively use any other sender as "burner". E.g.: log to disk, send to a queue, etc.

TELEMETRY

TELEMETRY

Receiver: UDP

TELEMETRY

Receiver: UDP

Parser: protobuf/junos

TELEMETRY

Receiver: UDP

Parser: protobuf/junos

Transformers: TONS OF THEM

TELEMETRY

Receiver: UDP

Parser: protobuf/junos

Transformers: TONS OF THEM

Sender: influx

brut-dmz-udp: Receive, parse and forward over HTTP

mor-influx: receive on HTTP, transform, store

Transformers can add/modify timestamps, rename fields, extract data to metadata, etc

... split a hash or an array into individual metrics.

Or enrich data!

ENRICHMENT

Enrichment transformer applies enrichment

Enrichment transformer applies enrichment

O(1) performance

Enrichment updater (sender) updates enrichment database

Enrichment updater (sender) updates enrichment database

Partial updates by default

Enrichment updater (sender) updates enrichment database

Partial updates by default

O(n) performance during update, blocking - room for improvement

Uses regular Skogul receivers to update the database

```
{
  "transformers": {
    "enrichID": {
      "type": "enrich",
      "keys": ["sysName", "ifName" ]
    }
  },
  "senders": {
    "enrichUP": {
      "type": "eupdater",
      "enricher": "enrichID"
    }
  },
  "receivers": {
    "api": {
      "type": "http",
```

```
{
  "transformers": {
    "enrichID": {
      "type": "enrich",
      "keys": ["sysName", "ifName" ]
    }
  },
  "senders": {
    "enrichUP": {
      "type": "eupdater",
      "enricher": "enrichID"
    }
  },
  "receivers": {
    "api": {
      "type": "http"
```

```
      "type": "http",
      "handlers": {
        "/": "ahandler"
      }
    },
    "eup": {
      "type": "sql",
      "connstr": "....",
      "query": "select NOW() as time, sysname, ifname, custome
      "driver": "mysql",
      "handler": "eup"
    }
  },
  "handlers": {
    "eup": {
      "parser": "json",
```

```
        "driver": "mysql",
        "handler": "eup"
      }
    },
    "handlers": {
      "eup": {
        "parser": "json",
        "sender": "enrichUP"
      },
      "ahandler": {
        "parser": "json",
        "transformers": [ "enrichID" ],
        "sender": "print"
      }
    }
}
```

```
        "driver": "mysql",
        "handler": "eup"
      }
    },
    "handlers": {
      "eup": {
        "parser": "json",
        "sender": "enrichUP"
      },
      "ahandler": {
        "parser": "json",
        "transformers": [ "enrichID" ],
        "sender": "print"
      }
    }
}
```

MISCELLANEOUS

Use "tester" receiver to generate random data to test sender-pipeline

Use "tester" receiver to generate random data to test sender-pipeline

"Does sending to Kafka work at all?"

Use "print"/"debug" sender to output result on stdout

Use "print"/"debug" sender to output result on stdout

"Does my script sending to the Skogul HTTP api come through ok?"

Use "dupe" sender to duplicate data

Use "dupe" sender to duplicate data

Send to production, stage, file, stdout, etc at the same time

Write config file as regular user without touching the regular one - completely isolated.

When it works: Move it to system-wide instance (through ansible 😇)

LOGGING

Never been happy with it, but it's getting gradually better. Feedback wanted!