MMIX MMNOG
Forum 2026

FLOW COLLECTOR VISUALIZATION FOR NETWORK OPERATIONS

MMIX MMNOG 8

YANGON

2026

SAW YAN PAING

CCIE #57007

ARK PREMIUM SOLUTIONS LIMITED

WHY NETWORK FLOW IS IMPORTANT?

- Traffic visibility
- Network Performance Monitoring
- Security Monitoring
- Capicity Planning

FLOW PROTOCOLS

- Netflow
- sFlow
- IPFIX
- Jflow and other proprietary like Huawei Netstream

NETFLOW

Developer: Cisco

Released: Mid 1990s

Version: v5 , v9

Method: "v5, v9 can use sampling method (1 out of N packets)", older versions track all packets

Scalability: Due to sampling rather than capturing every packet, suitable for high-speed networks (10Gbps–100Gbps+)

Flow Data: earlier version use 7-tuple, verion 5 and later 5-tuple (src.add, dst.add, src.port, dst.port, protocol), Flexible-netflow can add additional context

Port: udp (2055/9555/9025/9026)

SFLOW (SAMPLED FLOW)

Developer: InMon Corporation

Released: 2001

Version: v5

Method: sampling method (1 out of N packets)

Scalability: Due to sampling rather than capturing every packet, sFlow is highly efficient for high-speed networks (10Gbps–100Gbps+)

Flow Data: 5-tuple (src.add, dst.add, src.port, dst.port, protocol)+L2-4 Header+L2 information+Interface counter

Port: udp 6343

IPFIX (IP FLOW INFORMATION EXPORT)

Developer: Internet Engineering Task Force (IETF)

Released: 2008

Version: v10

Method: Deterministic Sampling (every n -th), Random Sampling (1 out of N packets)

Scalability: Due to sampling rather than capturing every packet, suitable for high-speed networks (10Gbps–100Gbps+)

Flow Data: 5-tuple (src.add, dst.add, src.port, dst.port, protocol) + Ingress Interface + Packet/Byte Counts + Timestamp

Port: default 4739 (UDP/TCP), 2055, 9995, 2056 (alternative)

JFLOW (IP FLOW INFORMATION EXPORT)

Developer: Juniper

Released: 2001

Version: v5, v8, v9

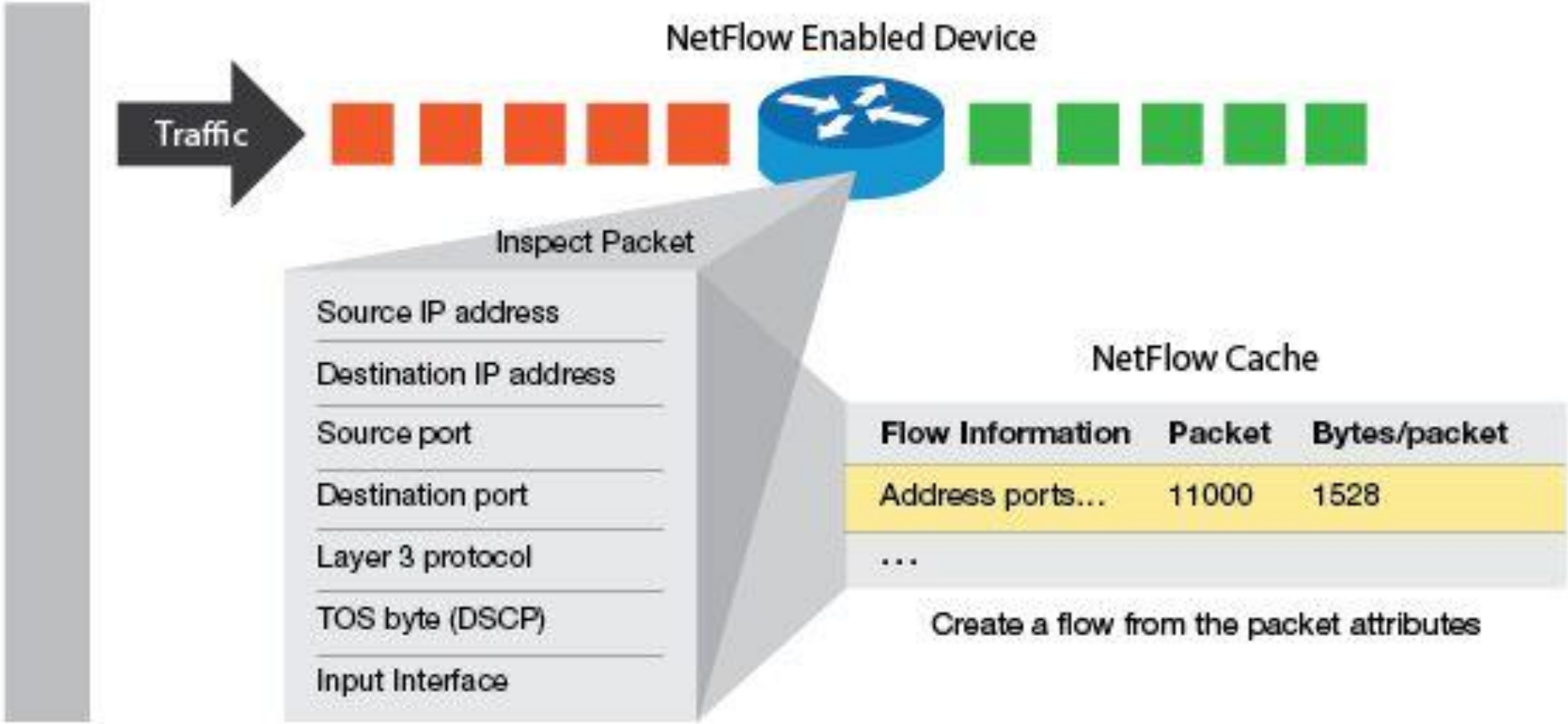
Method: Random Sampling (1 out of N packets)

Scalability: Due to sampling rather than capturing every packet, suitable for high-speed networks (10Gbps–100Gbps+)

Flow Data: 5-tuple (src.add, dst.add, src.port, dst.port, protocol)

Port: 2055, 9995, or 9996 (UDP)

UNDERSTANDING FLOW RECORD



FLOW SAMPLING

- Monitoring 100% of traffic ❌ (high CPU, memory, storage)
- Monitor **1 out of N packets** ✅ (efficient, scalable)

Types:

- Random Sampling
  1 out of N packets , eg. Sampling rate 1000 = 1 out of every 1000 packets
  #common use in sflow

- Systematic Sampling
  Every Nth packet, eg Sampling rate 1000 = every 1000th packet
  #common use in netflow, IPFIX

FLOW SAMPLE RATE

Sampling rate defines how much the traffic capture:

- 1:1 → No sampling (full visibility)
- 1:100 → 1% traffic
- 1:1000 → 0.1% traffic

1 Mbps sampled traffic at 1:1000, actual traffic ≈ **1 Gbps**

Recommended Rates:
Low Traffic (<100 Mbps): 1:1 or 1:10 (no sampling)
Medium Traffic (100-500 Mbps): 1:100
High Traffic (>1 Gbps): 1:1000 to 1:4096+

✅ **Use Sampling When:**

- Backbone / IGW / Core network monitoring
- High-speed interfaces (10G, 40G, 100G)
- DDoS trend detection
- Capacity planning

❌ **Avoid Sampling When:**

- Billing / charging systems
- Security forensics (need full data)
- SLA verification

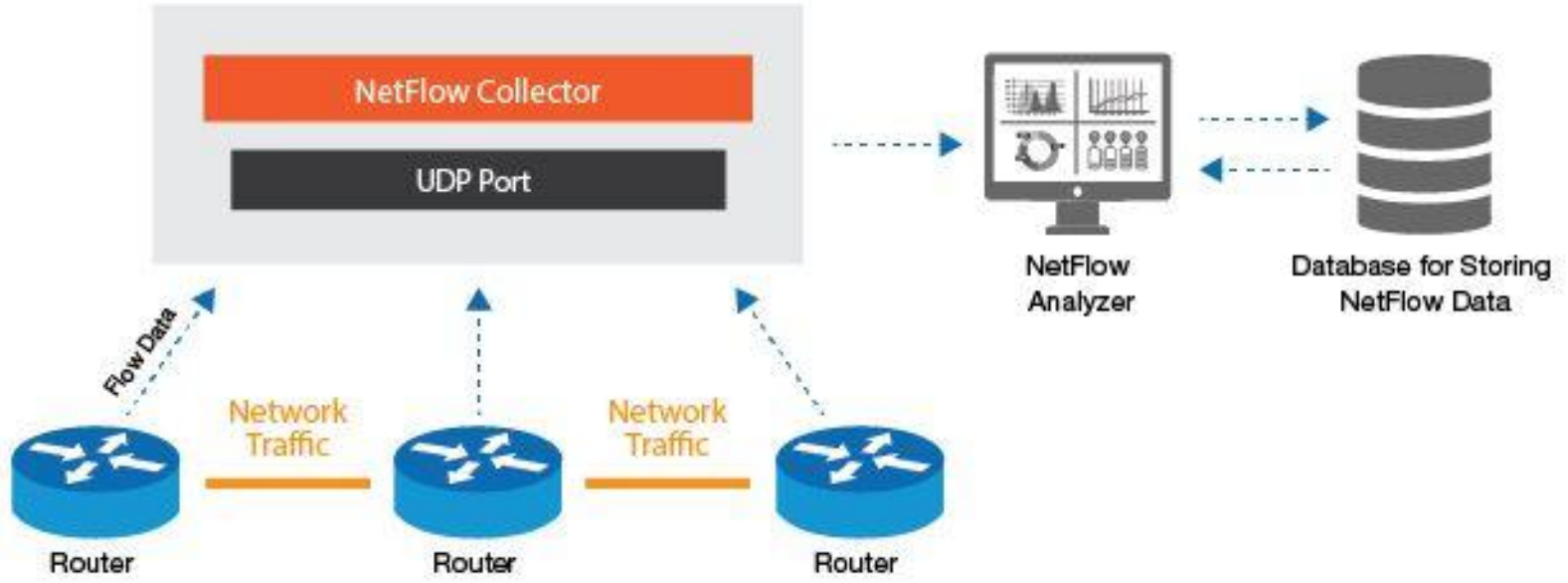
FLOW SAMPLE RATE

Link Speed	Large Flow	Sampling Rate	Polling Interval
10 Mbit/s	>= 1 Mbit/s	1-in-10	20 seconds
100 Mbit/s	>= 10 Mbit/s	1-in-100	20 seconds
1 Gbit/s	>= 100 Mbit/s	1-in-1,000	20 seconds
10 Gbit/s	>= 1 Gbit/s	1-in-10,000	20 seconds
40 Gbit/s	>= 4 Gbit/s	1-in-40,000	20 seconds
100 Gbit/s	>= 10 Gbit/s	1-in-100,000	20 seconds

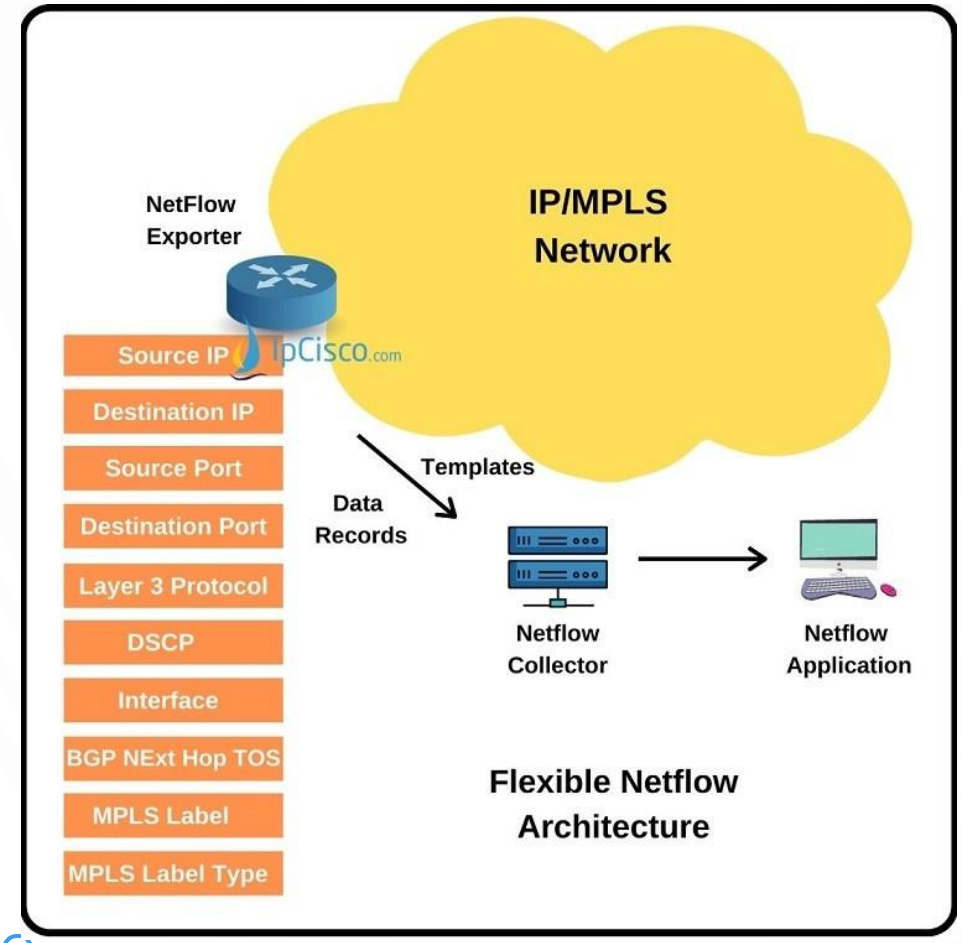
##Easy rule >> multiply by 1000 the link speed. Eg. 40Gbs = 40x1000 = 40000 (1:40000 sample rate)

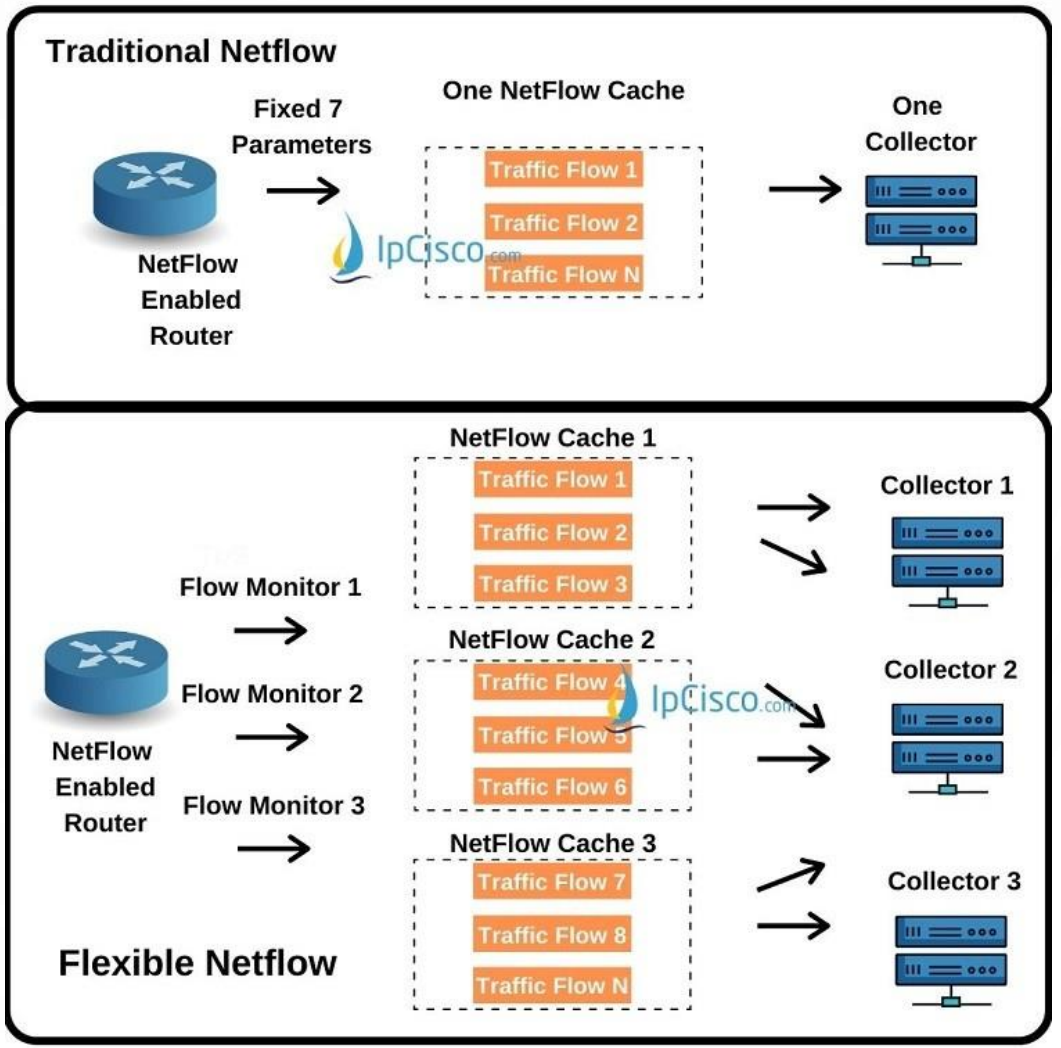
NETFLOW COMPONENTS

- **Flow exporter**
- **Flow collector**
- **Flow analyzer**



FLEXIBLE NETFLOW COMPONENTS





FLOW EXPORTERS

Devices

- Routers (most common)
- Switches
- Firewalls
- Network probes/Tap Devices

cisco router configuration example

1. Configure the Exporter

```
Router(config)# flow exporter my-exporter
Router(config-flow-exporter)# destination 1.1.1.1
```

2. Configure the Flow Record

```
Router(config)# flow record my-record
Router(config-flow-record)# match ipv4 destination address
Router(config-flow-record)# match ipv4 source address
Router(config-flow-record)# collect counter bytes
```

3. Configure the Flow Monitor

```
Router(config)# flow monitor my-monitor
Router(config-flow-monitor)# exporter my-exporter
Router(config-flow-monitor)# record my-record
```

4. Apply to an Interface

```
Router(config)# interface s3/0
Router(config-if)# ip flow monitor my-monitor input
```

"Exporters generates and sends flow data"

FLOW COLLECTORS AND ANALYZERS

Open Source

- ntopng
- ElastiFlow
- pmacct
- nfsen-ng

Commercial

- Solarwinds NTA
- Huawei NetStream Analyzer

  and NETSCOUT, Kentik, Cisco Stealthwatch

"Collectors receives flow records from multiple devices and Analyzers process the data, store in database"

Workshop Focus: NetFlow-Based Traffic Visualization & Enrichment !

NetFlow Data Collection

- Capture traffic flows from network devices (routers, switches, firewalls)
- NetFlow/IPFIX and sFlow

Traffic Visualization

- Convert raw flow data into dashboards and graphs
- Identify top talkers, applications, and bandwidth usage trends
- Detect anomalies and unusual traffic patterns

Data Enrichment

- Interface names (SNMP)
- Routing information (BMP)
- Geo-location data (Ipinfo.io)
- Classifiers

AKVORADO: FLOW COLLECTOR, ENRICHER AND VISUALIZER

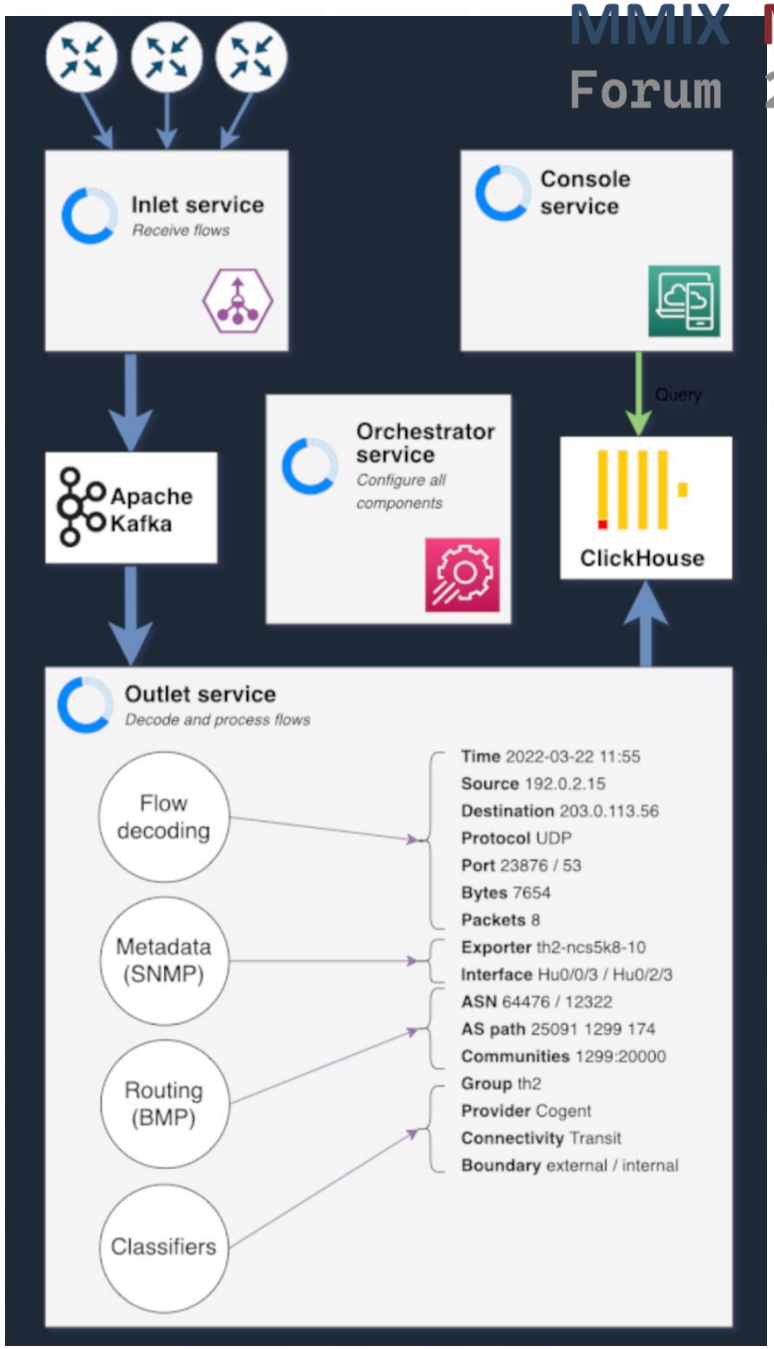
Open-source

- Developed by Vincent Bernat and Free (French ISP)
- https://www.akvorado.net , licensed under AGPLv3
- written in Go
- prerequisite golang(1.21+), nodejs(22+), npm, kafka(1.0+), Clickhouse (22.4+)

AKVORADO

Akvorado is split into four components:

- inlet service
- outlet service
- orchestrator service
- console service



REQUIREMENTS

Recommended Hardware:

- 8 vCPUs (AMD64 or ARM64)
- 100 GB Storage
- 16GB RAM

- Docker engine 23+ (if using docker image)

or

- Standalone Install (Dependencies)
  - Use pre-build binary/ compiling from source code

Dependencies

- Kakfa (1.0+)
- ClickHouse(22.4+)
- Go(1.21+)
- NodeJS(22+)
- PNPM

Standalone install guide

https://github.com/akvorado/akvorado/discussions/1513#discussion-7528574

QUICK START

1st Step

On Debian, install the **docker-compose** package.

#on Ubuntu, use the **docker-compose-v2** package

2nd Step

```
# mkdir akvorado
# cd akvorado
# curl -sL https://github.com/akvorado/akvorado/releases/latest/download/docker
-compose-quickstart.tar.gz | tar zxvf -
# docker compose up --wait
```

QUICK START

Next Step

To connect the own network devices:

1.Customize the configuration in **akvorado.yaml**:
    Set SNMP communities for your devices
    in **outlet** → **metadata** → **provider** → **communities**
    Configure interface classification rules
    in **outlet** → **core** → **interface-classifiers**

SUPER EASY!!!

2.Configure your routers/switches to send flows to *Akvorado*:
    NetFlow: port 2055
    IPFIX: port 4739
    sFlow: port 6343

3.Restart all containers:
    docker compose down
    docker compose up --wait

CONFIGURATION IN AKVORADO.YAML

cd akvorado/
vim config/akvorado.yaml
#customize the asn and network

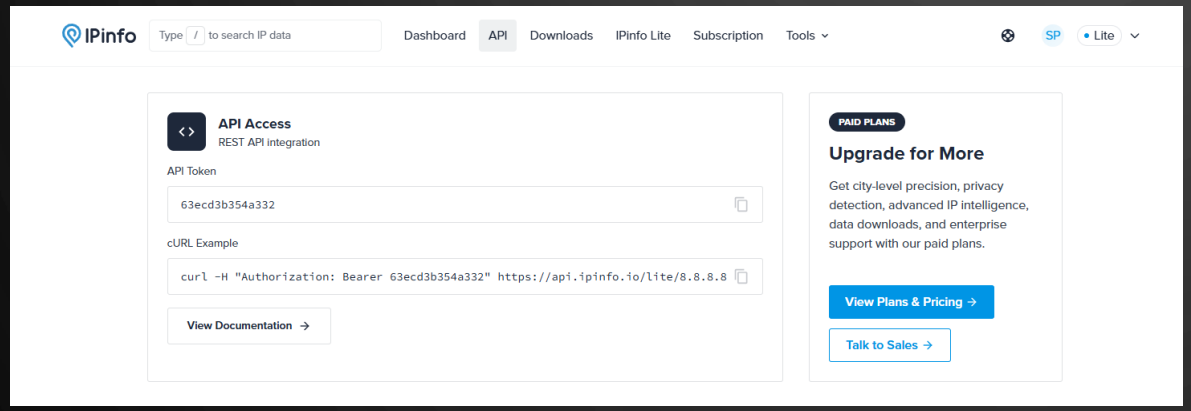
```
  asns:
    9654: mmix
  networks:
    # You should customize this section with your networks. This
    # populates the Src/DstNetName/Role/Site/Region/Tenant fields.
    103.116.194.0/24:
      name: ipv4-customers
      role: customers
```

To populate the asn and country from flow,
can choose ipinfo (default) and maxmind DB

```
geoip:
  optional: true
  # When running on Docker, these paths are inside the container. By default,
  # IPinfo databases are used. (https://ipinfo.io/)
  asn-database:
    - /usr/share/GeoIP/asn.mmdb
  geo-database:
    - /usr/share/GeoIP/country.mmdb
  # If you want to use MaxmindDB, check `.env`, `docker-compose-maxmind.yml` and
  # update these paths:
  #asn-database:
  # - /usr/share/GeoIP/GeoLite2-ASN.mmdb
  geo-database:
  # - /usr/share/GeoIP/GeoLite2-Country.mmdb
```

CONFIGURATION IN AKVORADO.YAML

Singup and open account at the https://ipinfo.io
copy the API token



vim docker/docker-compose-ipinfo.yml and paste the token key to fetch the information

```
---
services:
  geoip:
    extends:
      file: versions.yml
      service: ipinfo-geoipupdate
    restart: unless-stopped
    environment:
      IPINFO_TOKEN: 63ecd3b354a332
      IPINFO_DATABASES: country asn
      UPDATE_FREQUENCY: 48h
    volumes:
      - akvorado-geoip:/data
~
```

METADATA

set snmp communites to collect the metadata from the router. If do not want to use snmp, we can set the intaface information with the provider type static.

snmp v2

```
metadata:
  providers:
    - type: snmp
      credentials:
        ::/0:
          communities:
            - [redacted]
            - [redacted]
```

vi config/outlet.yaml

snmp v3

```
metadata:
  workers: 10
  providers:
    - type: snmp
      credentials:
        ::/0:
          user-name: monitoring
          authentication-protocol: SHA
          authentication-passphrase: "d$rkSec"
          privacy-protocol: AES192
          privacy-passphrase: "C10se"
```

static

```
metadata:
  providers:
    - type: static
      exporters:
        2001:db8:1::1:
          name: exporter1
          skip-missing-interfaces: true
          ifindexes:
            10:
              name: Gi0/0/10
              description: PNI Netflix
              speed: 1000
            11:
              name: Gi0/0/15
              description: PNI Google
              speed: 1000
```

INTERFACE-CLASSIFIERS

Classify the interface which are the transit/pni/ix/customer connectivity and the upstream provider to visualize on the dashboard.

vi config/outlet.yaml

```
interface-classifiers:
  # This is an example. This must be customized depending on the
  # descriptions of your interfaces. In the following, we assume
  # external interfaces are named "Transit: Cogent" Or "IX:
  # FranceIX".
  - |
    ClassifyConnectivityRegex(Interface.Description, "^(?i)(transit|pni|ppni|ix):? ", "$1") &&
    ClassifyProviderRegex(Interface.Description, "^[^ ]+? ([^ ]+)", "$1") &&
    ClassifyExternal()
  - ClassifyConnectivityRegex(Interface.Description, "^(?i)(cust|core|member):? ", "$1") &&
    ClassifyInternal()
  - ClassifyInternal()
```

Classifier rules are setting with regex.

Eg. The interface description is

interface g0/0/1

description transit: singtel <blahblahblah>

Akvorado set this interface as the external connectivity and transit interface, the upstream provider will singtel.

SEND FLOWS TO *AKVORADO*

vim config/inlet.yaml
configure the required configuration, there have the some options for tuning like “rate limit”, “decapsulation-protocol” to look inside the tunneling protocols, vxlan,gre,srv6,ipip.

```
---
flow:
  inputs:
    # NetFlow port
    - type: udp
      decoder: netflow
      listen: :2055
      workers: 4
      # Before increasing this value, look for it in the troubleshooting section
      # of the documentation.
      receive-buffer: 212992
    # IPFIX port
    - type: udp
      decoder: netflow
      listen: :4739
      workers: 4
      receive-buffer: 212992
    # sFlow port
    - type: udp
      decoder: sflow
      listen: :6343
      workers: 4
      receive-buffer: 212992
```

SEND FLOWS TO *AKVORADO (ROUTER SIDE)*

1.configure flow exporter

```
flow exporter akvorado
  destination [redacted] use-vrf management
  transport udp 2055
  source mgmt0
  version 9
    option exporter-stats timeout 60
    option interface-table timeout 60
```

2.configure flow record

```
flow record akvorado_flow
  match ipv4 source address
  match ipv4 destination address
  match ip protocol
  match ip tos
  match transport source-port
  match transport destination-port
  collect transport tcp flags
  collect counter bytes long
  collect counter packets long
  collect timestamp sys-uptime first
  collect timestamp sys-uptime last
```

Cisco 9K switches are not need sampled Packet can be process with line rate. We need to set the sampling rate to 1 at akvorado config.

```
core:
  default-sampling-rate: 1
```

3.configure flow monitor

```
flow monitor akvorado_mon
  record akvorado_flow
  exporter akvorado
```

4.apply to the interface

```
interface port-channel22
  ip flow monitor akvorado_mon input
```

This example for Nexus 9K switch.

SEND FLOWS TO *AKVORADO (ROUTER SIDE)*

```
flow record Akvorado
    match ipv4 tos
    match ipv4 protocol
    match ipv4 source address
    match ipv4 destination address
    match transport source-port
    match transport destination-port
    collect routing source as 4-octet
    collect routing destination as 4-octet
    collect routing next-hop address ipv4
    collect transport tcp flags
    collect interface output
    collect interface input
    collect counter bytes
    collect counter packets
    collect timestamp sys-uptime first
    collect timestamp sys-uptime last
!
flow record Akvorado-IPV6
    match ipv6 protocol
    match ipv6 source address
    match ipv6 destination address
    match transport source-port
    match transport destination-port
    collect routing source as 4-octet
    collect routing destination as 4-octet
    collect routing next-hop address ipv4
    collect transport tcp flags
    collect interface output
    collect interface input
    collect counter bytes
    collect counter packets
    collect timestamp sys-uptime first
    collect timestamp sys-uptime last
!
```

```
sampler random1in100
    mode random 1 out-of 100
!
flow exporter AkvoradoExport
    destination <akvorado-ip> vrf monitoring
    source Loopback20
    transport udp 2055
    version 9
    option sampler-table timeout 10
!
flow monitor AkvoradoMonitor
    exporter AkvoradoExport
    cache timeout inactive 10
    cache timeout active 10
    record Akvorado
!
flow monitor AkvoradoMonitor-IPV6
    exporter AkvoradoExport
    cache timeout inactive 10
    cache timeout active 10
    record Akvorado-IPV6
!
```

```
interface GigabitEthernet0/0/3
    ip flow monitor AkvoradoMonitor sampler random1in100 input
    ip flow monitor AkvoradoMonitor sampler random1in100 output
    ipv6 flow monitor AkvoradoMonitor-IPV6 sampler random1in100 input
    ipv6 flow monitor AkvoradoMonitor-IPV6 sampler random1in100 output
!
```

This example for IOS XE.

Cisco netflow cannot send the sampler rate information, We need to set same sampler rate at the akvorado config, otherwise the flow data cannot be process.

vim config/outlet.yaml

```
core:
  default-sampling-rate: 100
```

troubleshoot with following command

```
mmix@vm-2:~/akvorado$  curl -s http://127.0.0.1:8080/api/v0/outlet/metrics | grep 'akvorado_outlet_core.*errors'
# HELP akvorado_outlet_core_flows_errors_total Number of flows with errors.
# TYPE akvorado_outlet_core_flows_errors_total counter
akvorado_outlet_core_flows_errors_total{error="input and output interfaces missing",exporter="10.0.2.245"} 34
akvorado_outlet_core_flows_errors_total{error="sampling rate missing",exporter="10.0.2.245"} 399301
```

SEND FLOWS TO *AKVORADO (ROUTER SIDE)*

```
sampler-map sampler1
 random 1 out-of 32768
!
flow exporter-map akvorado
 version v9
  options sampler-table timeout 10
  template options timeout 10
 !
 transport udp 2055
 source Loopback20
 destination <akvorado-ip> vrf private
!
flow monitor-map monitor1
 record ipv4
 exporter akvorado
 cache entries 100000
 cache timeout active 10
 cache timeout inactive 10
 cache timeout rate-limit 2000
!
flow monitor-map monitor2
 record ipv6
 exporter akvorado
 cache entries 100000
 cache timeout active 10
 cache timeout inactive 10
 cache timeout rate-limit 2000
!
```

```
interface Bundle-Ether4000
 flow ipv4 monitor monitor1 sampler sampler1 ingress
 flow ipv6 monitor monitor2 sampler sampler1 ingress
!
```

```
router bgp <asn>
 address-family ipv4 unicast
  bgp attribute-download
!
 address-family ipv6 unicast
  bgp attribute-download
```

Optionally, push the AS path to the forwarding database, and the source and destination AS will be present in NetFlow packets:

This example for Cisco NCS 5500 and ASR 9000.

BMP PROVIDER

Using to collect the AS PATH, ASN, Communities.

vi config/outlet.yaml

```
routing:
  provider:
    type: bmp
    listen: 0.0.0.0:10179
    collect-asns: true
    collect-aspaths: true
    collect-communities: false
```

Router#

```
bmp server 1
 host <akvorado-ip> port 10179
 flapping-delay 60
bmp server all
 route-monitoring policy post inbound
router bgp 65400
 vrf public
  neighbor 192.0.2.100
   bmp-activate server 1
```

ISSUE# With many routes, BMP can have performance issues when a peer disconnects. If you do not need full accuracy, limit the number of BMP peers and export the LocRIB.

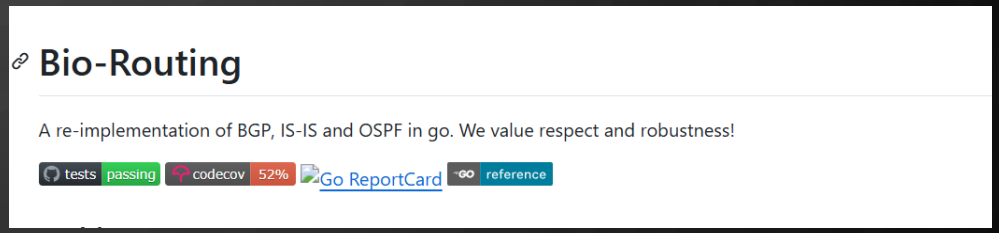
BIORIS PROVIDER

an alternative to the internal BMP.

vi config/outlet.yaml

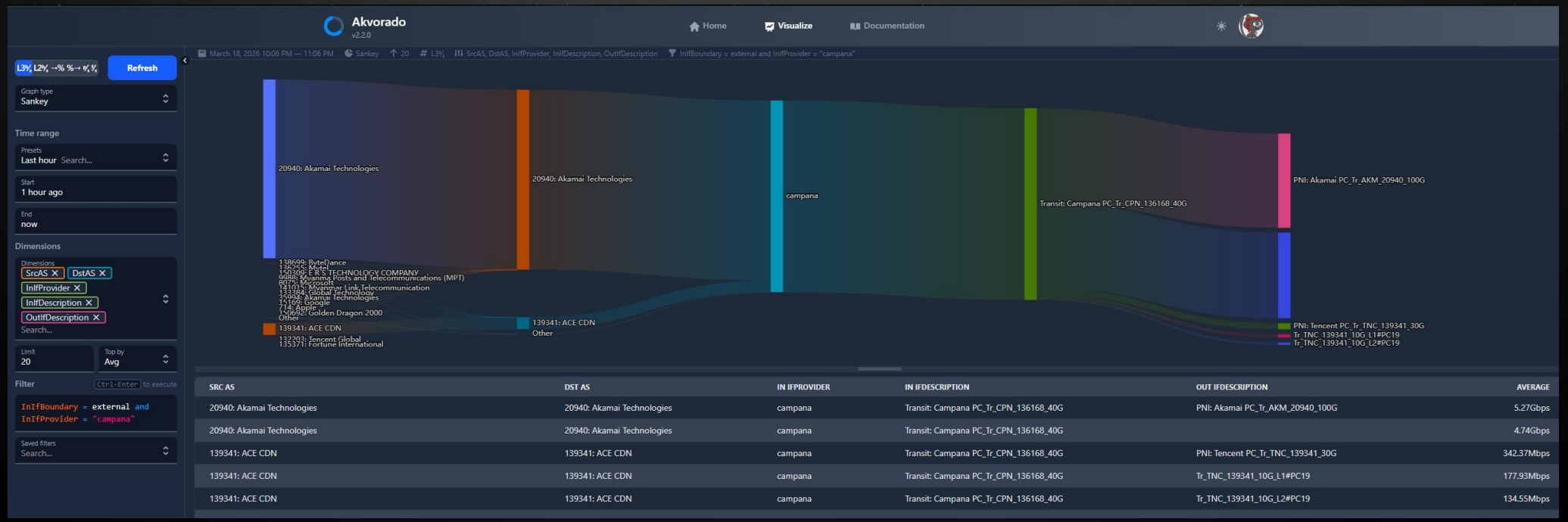
```
routing:
  provider:
    type: bioris
    risinstances:
      - grpcaddr: 192.0.2.15:4321
        grpcsecure: true
        vrf: 0:0
```

Third-paty BMP server



MMIX MMNOG Forum 2026

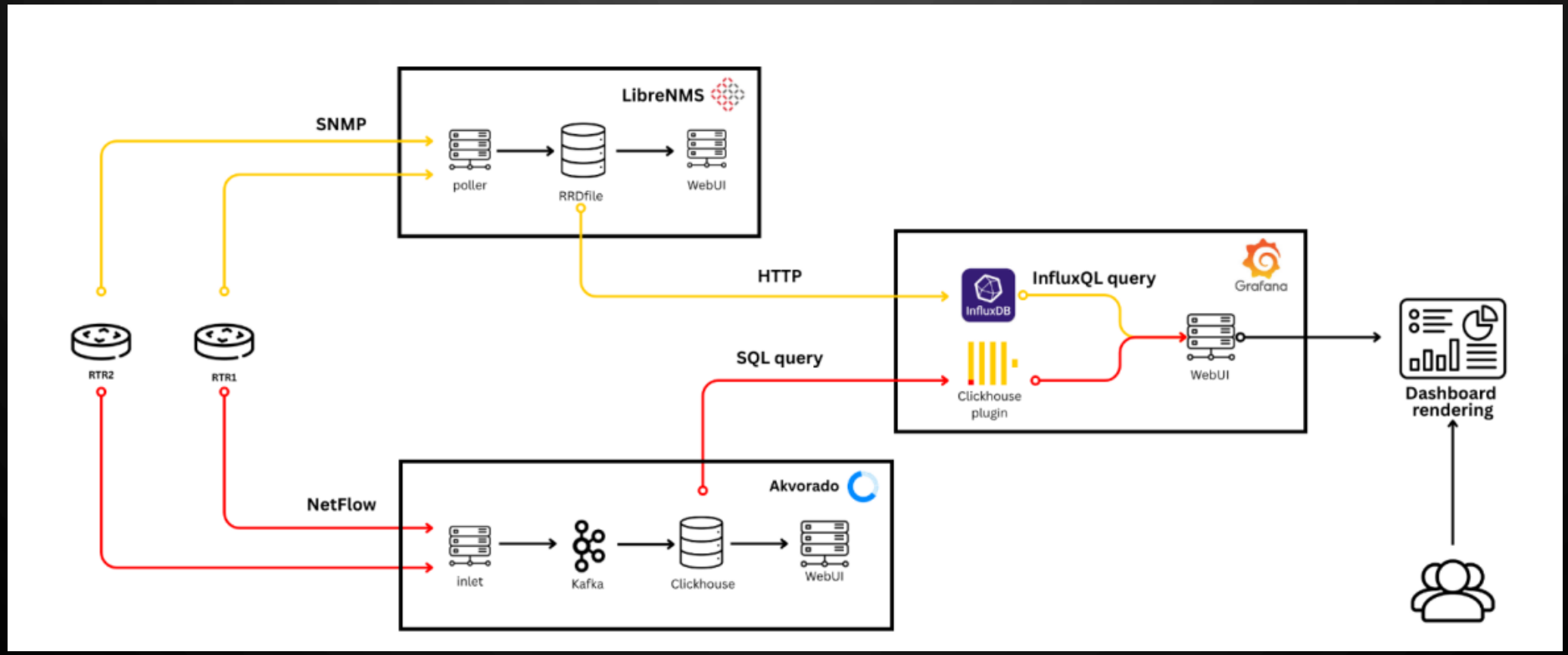
EVERY THINGS OK!



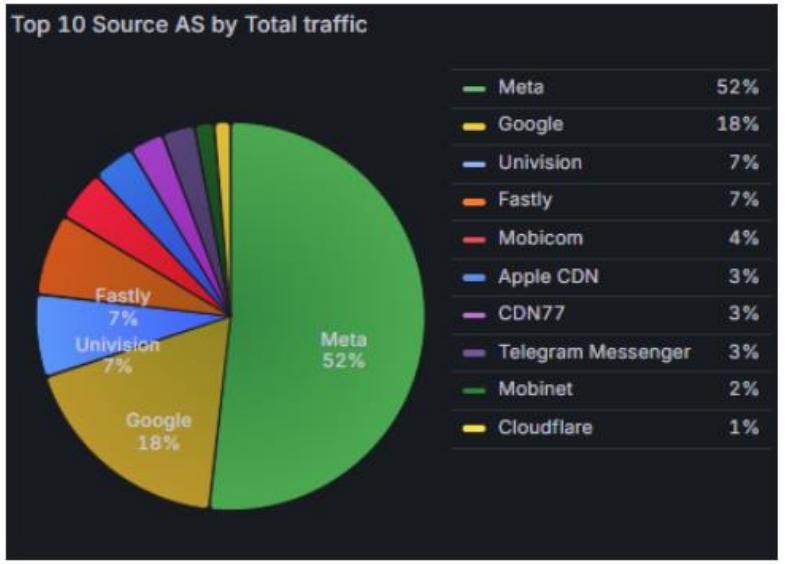
SRC AS	DST AS	IN IFPROVIDER	IN IFDESCRIPTION	OUT IFDESCRIPTION	AVERAGE
20940: Akamai Technologies	20940: Akamai Technologies	campana	Transit: Campana PC_Tr_CPN_136168_40G	PNI: Akamai PC_Tr_AKM_20940_100G	5.27Gbps
20940: Akamai Technologies	20940: Akamai Technologies	campana	Transit: Campana PC_Tr_CPN_136168_40G		4.74Gbps
139341: ACE CDN	139341: ACE CDN	campana	Transit: Campana PC_Tr_CPN_136168_40G	PNI: Tencent PC_Tr_TNC_139341_30G	342.37Mbps
139341: ACE CDN	139341: ACE CDN	campana	Transit: Campana PC_Tr_CPN_136168_40G	Tr_TNC_139341_10G_L1#PC19	177.93Mbps
139341: ACE CDN	139341: ACE CDN	campana	Transit: Campana PC_Tr_CPN_136168_40G	Tr_TNC_139341_10G_L2#PC19	134.55Mbps

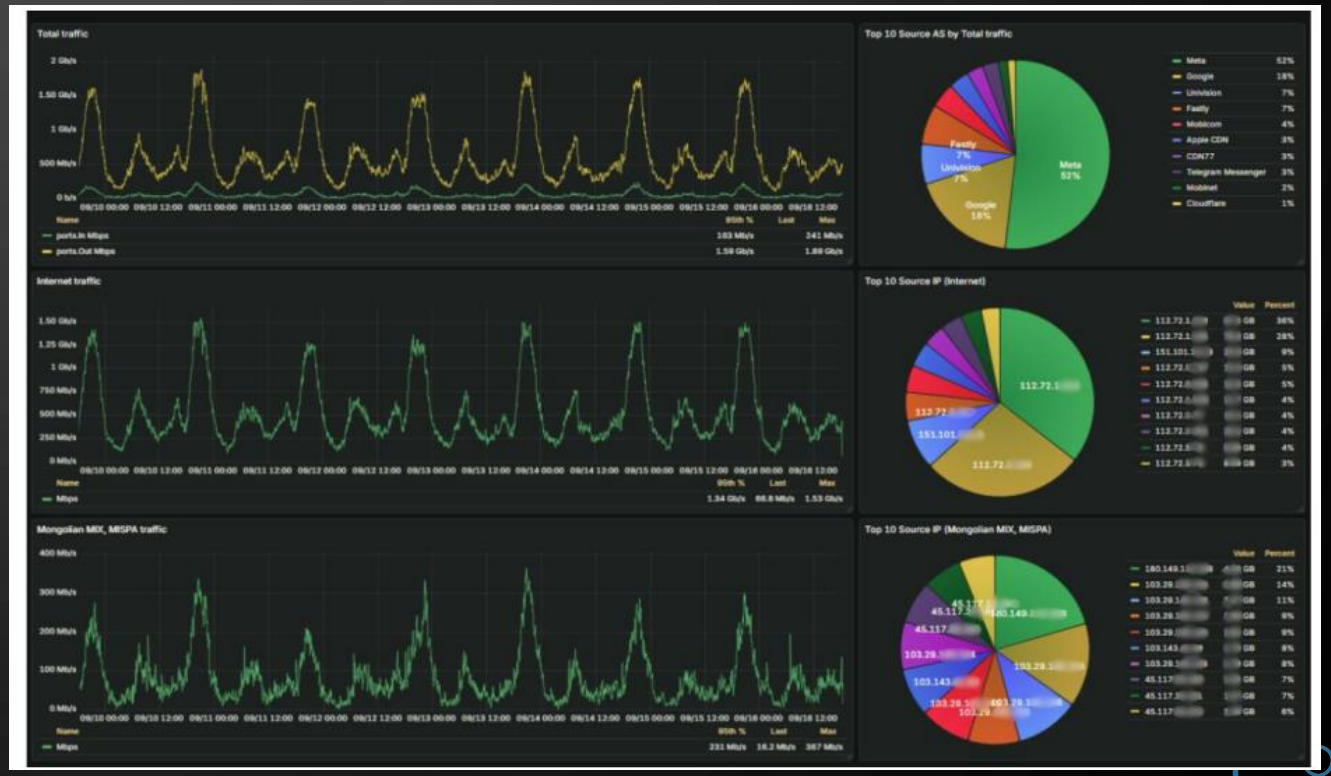
BONUS !

MONITORING SYSTEM DESIGN



CUSTOMIZED DASHBOARD AT GRAFANA





GRAFANA PLUGIN

Akvorado is developed by Free, a French ISP, and is licensed under the AGPLv3 license.

A demo site using fake data and running the latest stable version is available on demo.akvorado.net. It is the direct result of running `docker compose up` on a fresh checkout but flow ports are not accessible (you cannot send your own flows). Please, be gentle with this resource. The demo site also enables you to browse the documentation for the current version (the one in `docs/` is for the next version).

By default, *Akvorado* is using IPinfo databases for geolocation data.

A Grafana plugin is available.

THANK YOU ☺