NETWORK AUTOMATION FORUM

WS-A3 - Network App Development: Design, Deploy, & Automate

Autocon3 - Deck 1

Mau - Deepinder - Emre

AUTOCON 3

THE NETWORK AUTOMATION CONFERENCE

Who we are?

- **Network automation** professionals from **Nokia**
- Diverse backgrounds, generations, and experiences
- United by a passion for learning, building, and improving

Introduction & Foundations

- **Workshop Goals & Key Principles**: Declarative automation, event-driven design (Mau)
- **Technology Overview**: Containerlab, Kind, iPerf, Reconcilers, gNMIc (Mau)
- **Managing Container Images**: Building, posting locally, and pushing to online registries (Emre)
- **Preparing an App for Containers**: Structuring code for easy deployment (Emre)

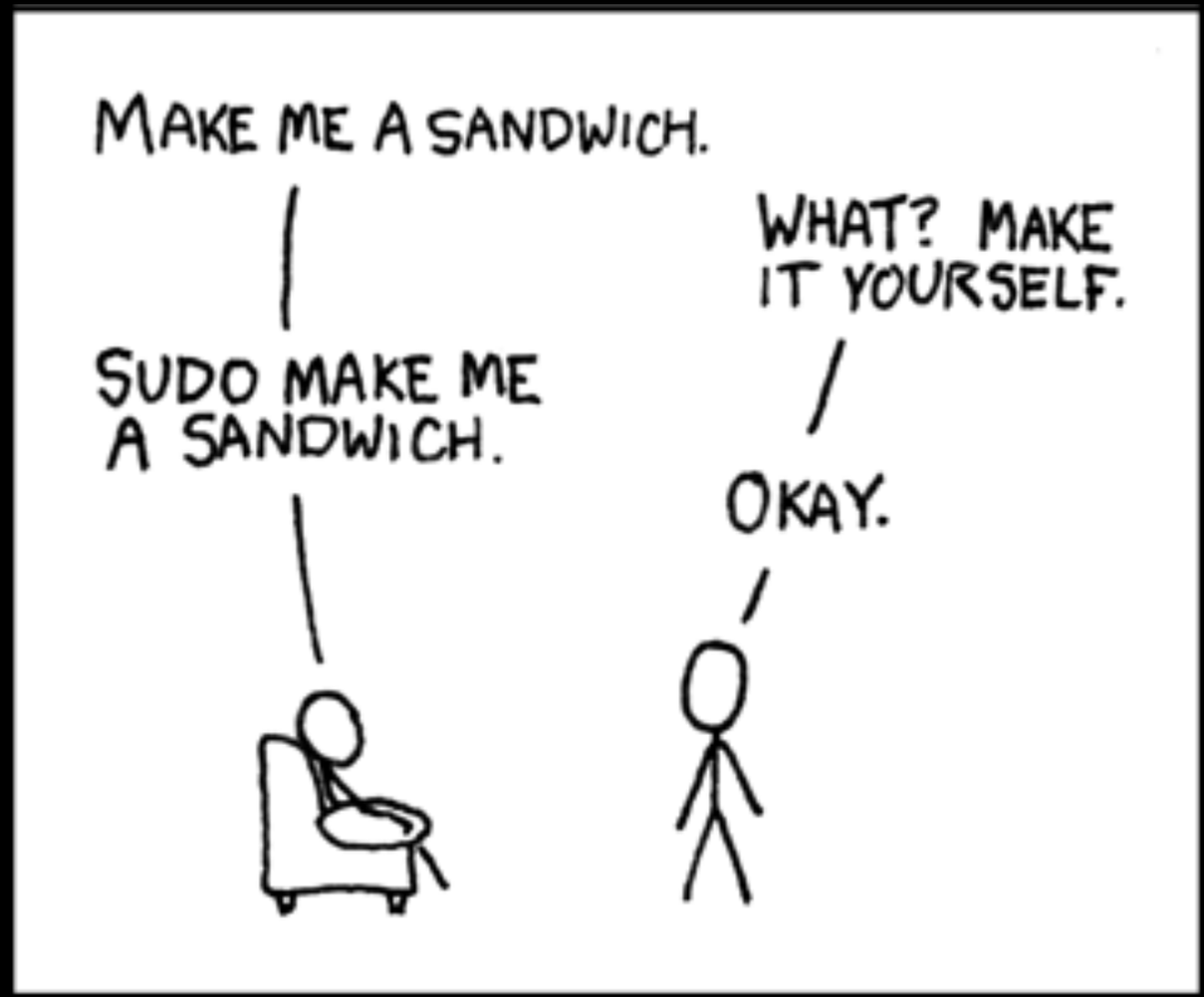
Goals


Be a hard working, efficient employee
Boss starts to take notice
You're given more responsibilities without a pay increase or promotion
You're given more responsibilities without a pay increase or promotion

Goals

- Is there a better way?
- Discover tools that simplify and scale
- Build more with less code
- Learn from use cases and lessons learned
- Apply practical DevOps and automation techniques
- Create resilient, shareable, and maintainable solutions

Declarative vs Imperative

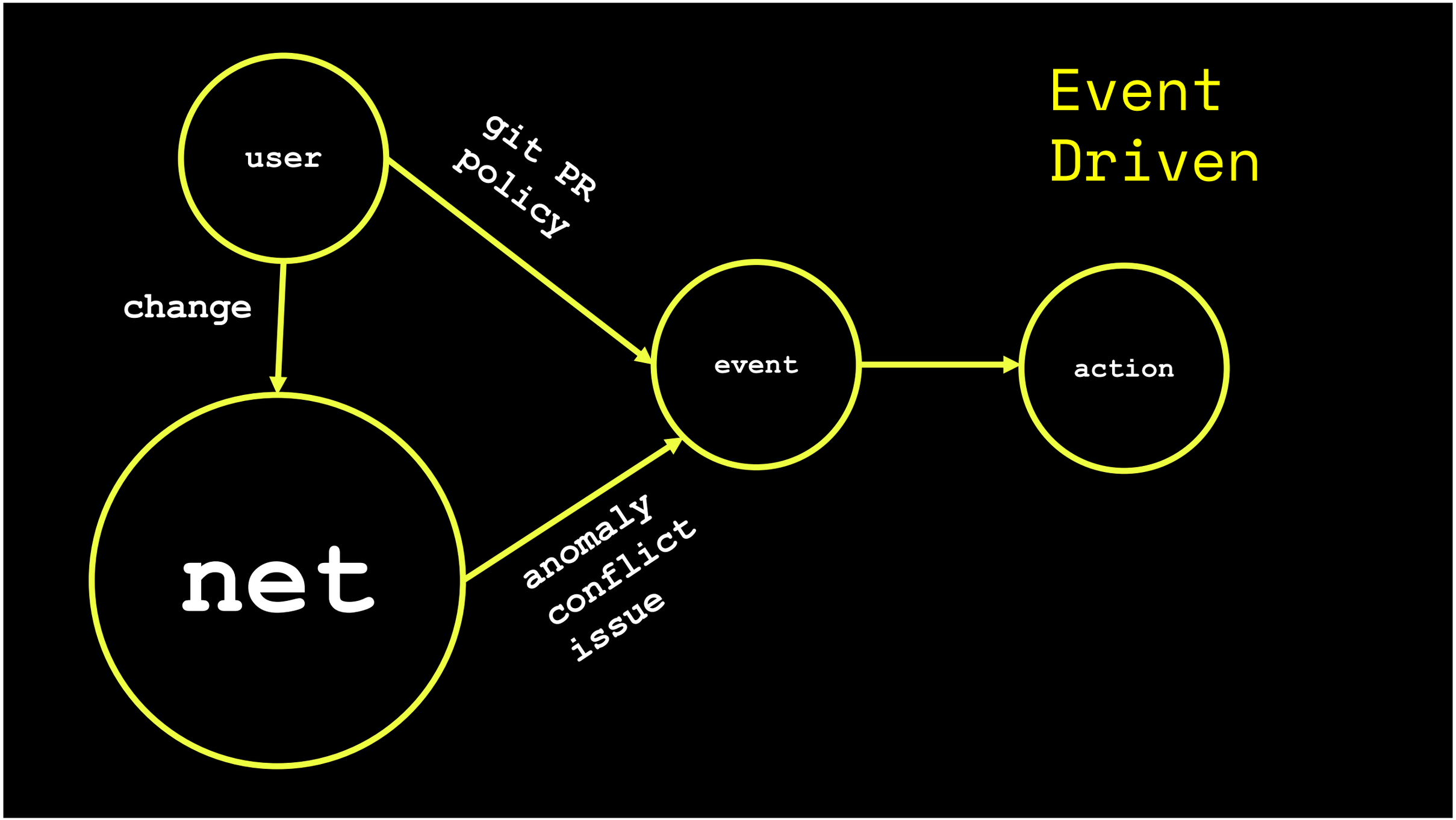


```
- name: Run iperf3 clients to multiple destination ports
  hosts: iperf_clients
  gather_facts: no
  vars:
    iperf_server_ip: "192.168.1.100"
    destination_ports: [5201, 5202, 5203]
    parallel_clients_per_port: 2

  tasks:
    - name: Run iperf3 client instances
      ansible.builtin.shell: |
        nohup iperf3 -c {{ iperf_server_ip }} -p {{ item.1 }} -t 60 >
/tmp/iperf3_{{ item.0 }}_port{{ item.1 }}.log 2>&1 &
      loop: "{{ query('subelements', [range(1,
parallel_clients_per_port+1), destination_ports]) }}"
      async: 0
      poll: 0
      ignore_errors: yes
```

Imperative

Declarative

```
apiVersion: example.com/v1
kind: IperfClient
metadata:
  name: iperf3-client
spec:
  targetIP: "172.254.102.101"  # Replace with server
cluster IP
  initPort: 30001
  endPort: 30005
  image: iperf3-client:0.1a
```


Event
Driven
user
git PR
policy
change
event
action
net
anomaly
conflict
issue


NAF
NETWORK AUTOMATION FORUM
Lab Prep

Lab Setup Options

 **Two Ways to Join the Labs:**

- **GitHub Codespaces (Recommended)**
  - No installation needed — runs in your browser
  - Requires a GitHub account and ~4 hours of Codespaces credits
  - Remember to switch to 4 vCPUs
- **Local Deployment**
  - Run everything using **VS Code** + **Docker Desktop**
  - Requires 16 GB RAM & 8 Cores minimum
  - Watch out for WSL

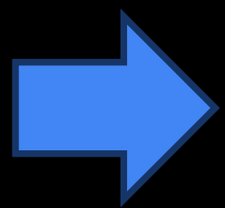
Required Tools & Extensions

🧰 **Local Setup Checklist (if not using Codespaces):**

- Install:
  - VS Code
  - Docker Desktop
  - Git
- VS Code Extensions:
  - Dev Containers
  - Remote - SSH

Port Mapping

codespaces

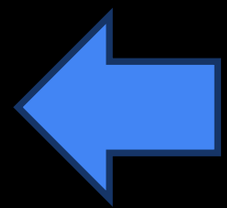


PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 5

Port	Forwarded Address	Runn
2181	https://laughing-eureka-j...	/usr/b
8001	https://laughing...	/usr/b
9092	https://laughing-eureka-j...	/usr/b
9093	https://laughing-eureka-j...	/usr/b
9095	https://laughing-eureka-j...	/usr/b

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 13

Port	Forwarded Addr...	Running P
2181	localhost:2181	/usr/bin/dc
2222	localhost:2222	sshd: /usr/
3000	localhost:3000	/usr/bin/dc
8001	localhost:8001	/usr/bin/dc
9092	localhost:9092	/usr/bin/dc
9093	localhost:9093	/usr/bin/dc
9095	localhost:9095	/usr/bin/dc

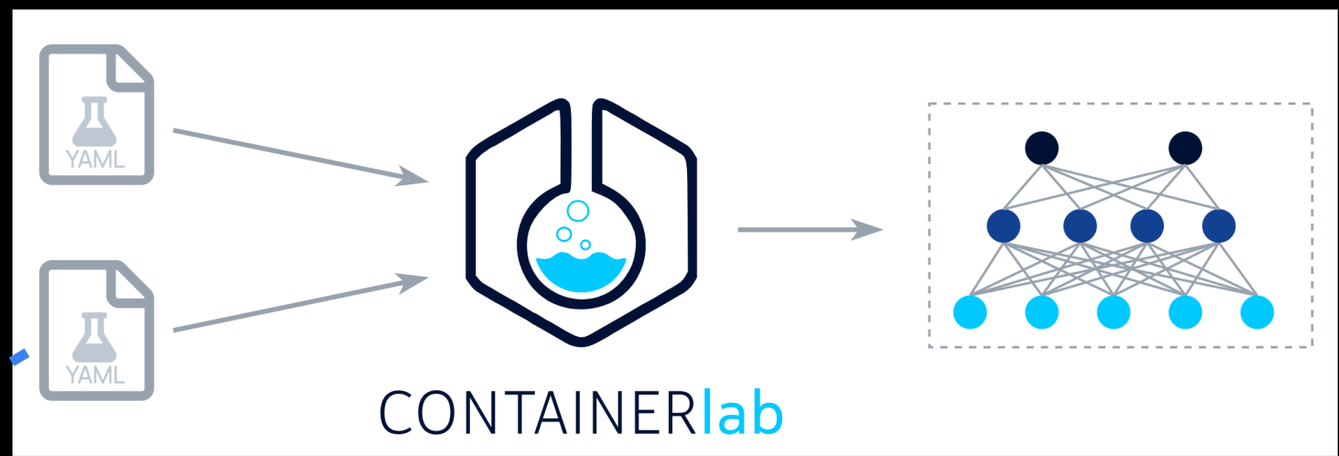


local

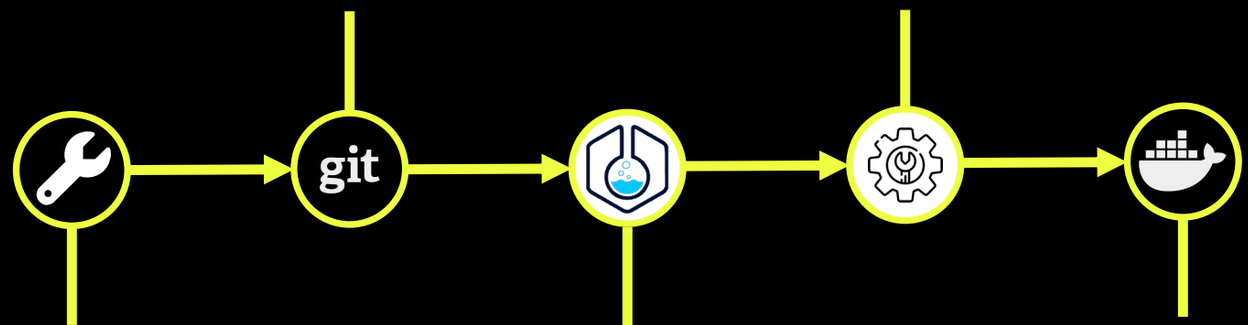

NA
NETWORK AUTOMATION FORUM
Technology Overview

Containerlab or clab

```
name: mylab

topology:

  nodes:
    [ ]: [ ]
    [ ]: [ ]

  links:
   - [ ]: [ ]
```



Clab: basic workflow



Install

docker

containerlab

Topology file

Create a lab

Clone existing lab

Deploy a lab

Interact w/ a lab

login to devices

modify configuration

Inspect running labs

Save a lab

Save running configs

Push lab files to a repo

Topology File (e.g. topo.yml)

```
topology:
  kinds:
    nokia_srlinux:
      type: ixrd2l
      image: ghcr.io/nokia/srlinux:24.7.2
  nodes:
    k8s01:
      kind: k8s-kind
    k8s02:
      kind: k8s-kind
    k8s01-control-plane:
      kind: ext-container
      exec:
        - "ip addr add 172.254.101.101/24 dev eth1"
        - "ip route add 172.0.0.0/8 via 172.254.101.1"
```

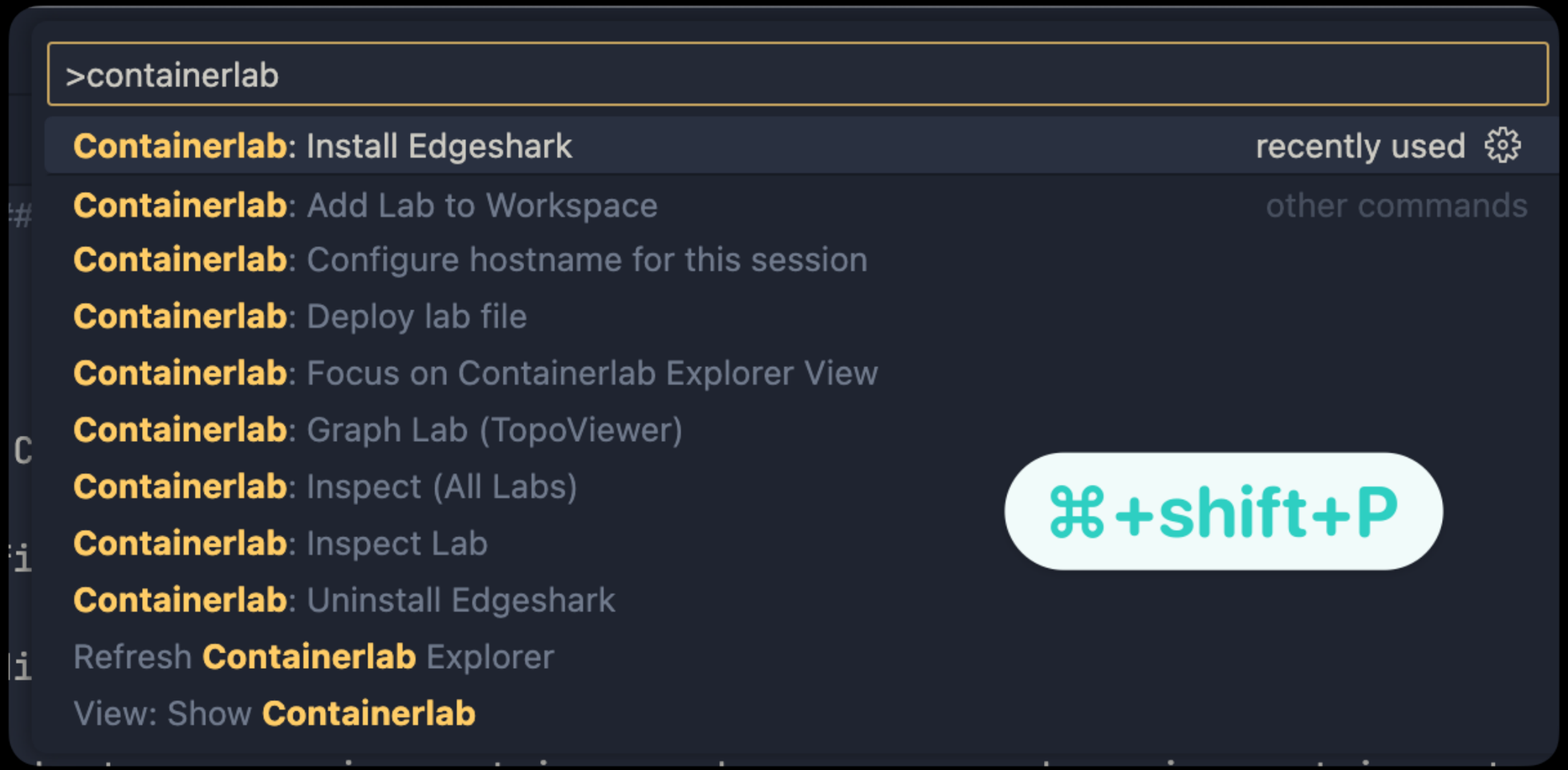
Clab: Deploy a lab

```
[srl-apps]# clab deploy -t topo.yml
INFO[0000] Containerlab v0.25.1 started
INFO[0000] Parsing & checking topology file: topo.yml
INFO[0000] Creating lab directory: /root/pygnmi-srl-apps/clab-dc-k8s
INFO[0000] Creating docker network: Name="kind",
IPv4Subnet="172.18.100.0/16", IPv6Subnet="", MTU="1500"
INFO[0000] Creating container: "grafana"
INFO[0000] Creating container: "SPINE-DC-2"
INFO[0000] Creating container: "prometheus"
INFO[0000] Creating container: "LEAF-DC-1"
INFO[0000] Creating container: "BORDER-DC"
INFO[0000] Creating container: "SPINE-DC-1"
INFO[0000] Creating container: "LEAF-DC-2"
```

Clab: Inspect a lab

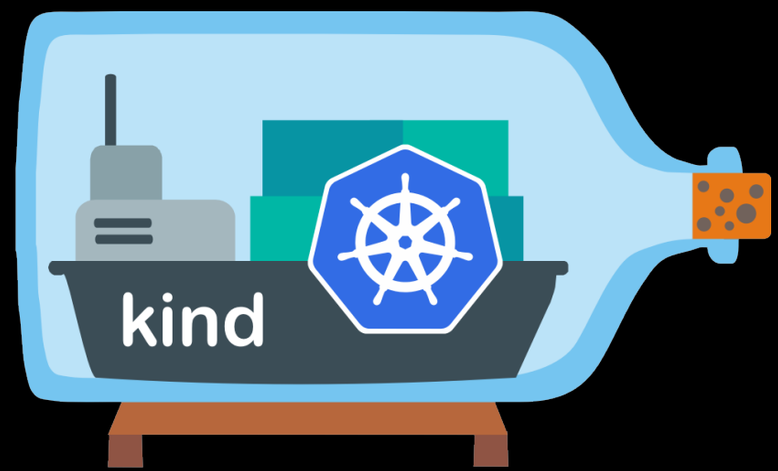
```
[srl-apps]# clab inspect -t topo.yml
INFO[0000] Parsing & checking topology file: topo.yml
+---+-------------------------+--------------+-------------------------------+---
| # |          Name           | Container ID |             Image             | Kin
+---+-------------------------+--------------+-------------------------------+---
| 1 | clab-dc-k8s-BORDER-DC   | 9876f09a5580 | ghcr.io/nokia/srlinux:21.6.4 | srl
| 2 | clab-dc-k8s-LEAF-DC-1   | 830369bb4d39 | ghcr.io/nokia/srlinux:21.6.4 | srl
| 3 | clab-dc-k8s-LEAF-DC-2   | 05d303e50816 | ghcr.io/nokia/srlinux:21.6.4 | srl
| 4 | clab-dc-k8s-SPINE-DC-1  | 574ff19416fb | ghcr.io/nokia/srlinux:21.6.4 | srl
| 5 | clab-dc-k8s-SPINE-DC-2  | e44d29973290 | ghcr.io/nokia/srlinux:21.6.4 | srl
| 6 | clab-dc-k8s-grafana     | e6d5221fa472 | grafana/grafana:latest        | lin
| 7 | clab-dc-k8s-prometheus  | 533473420ff1 | prom/prometheus:latest        | lin
+---+-------------------------+--------------+-------------------------------+---
```

Clab: VSCode


>containerlab
Containerlab: Install Edgeshark
recently used
Containerlab: Add Lab to Workspace
other commands
Containerlab: Configure hostname for this session
Containerlab: Deploy lab file
Containerlab: Focus on Containerlab Explorer View
Containerlab: Graph Lab (TopoViewer)
Containerlab: Inspect (All Labs)
Containerlab: Inspect Lab
Containerlab: Uninstall Edgeshark
Refresh Containerlab Explorer
View: Show Containerlab
⌘+shift+P

Kubernetes kind

- Local Kubernetes clusters using Docker container “nodes”.
- Designed for testing Kubernetes.
- Requirements:
  - go (1.17+)
  - Uses docker for node instances



Kind: Install

- You can install kind with:

```
go install sigs.k8s.io/kind@v0.18.0
```

- This will put kind in $(go env GOPATH)/bin.
  - You may need to add that directory to your $PATH as shown here if you encounter the error kind: command not found after installation.
- Kind uses docker for node instances
- Once you have docker running you can create a cluster with:

```
kind create cluster
```

Kind: Config

- For this demo we'll use ONE controller
- Use "kind load" to upload images for apps like iperf after cluster is created
  - You canto setup a private registry

```
[~/kind]# cat cluster_datacenter.yaml
---
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
  - role: control-plane
  - role: worker
  - role: worker
```

🚀 Why Use Kind + Containerlab?

🧪 **Test real networks + Kubernetes workloads** in a unified lab

🔗 **Integration** enables direct links between K8s pods and underlay nodes

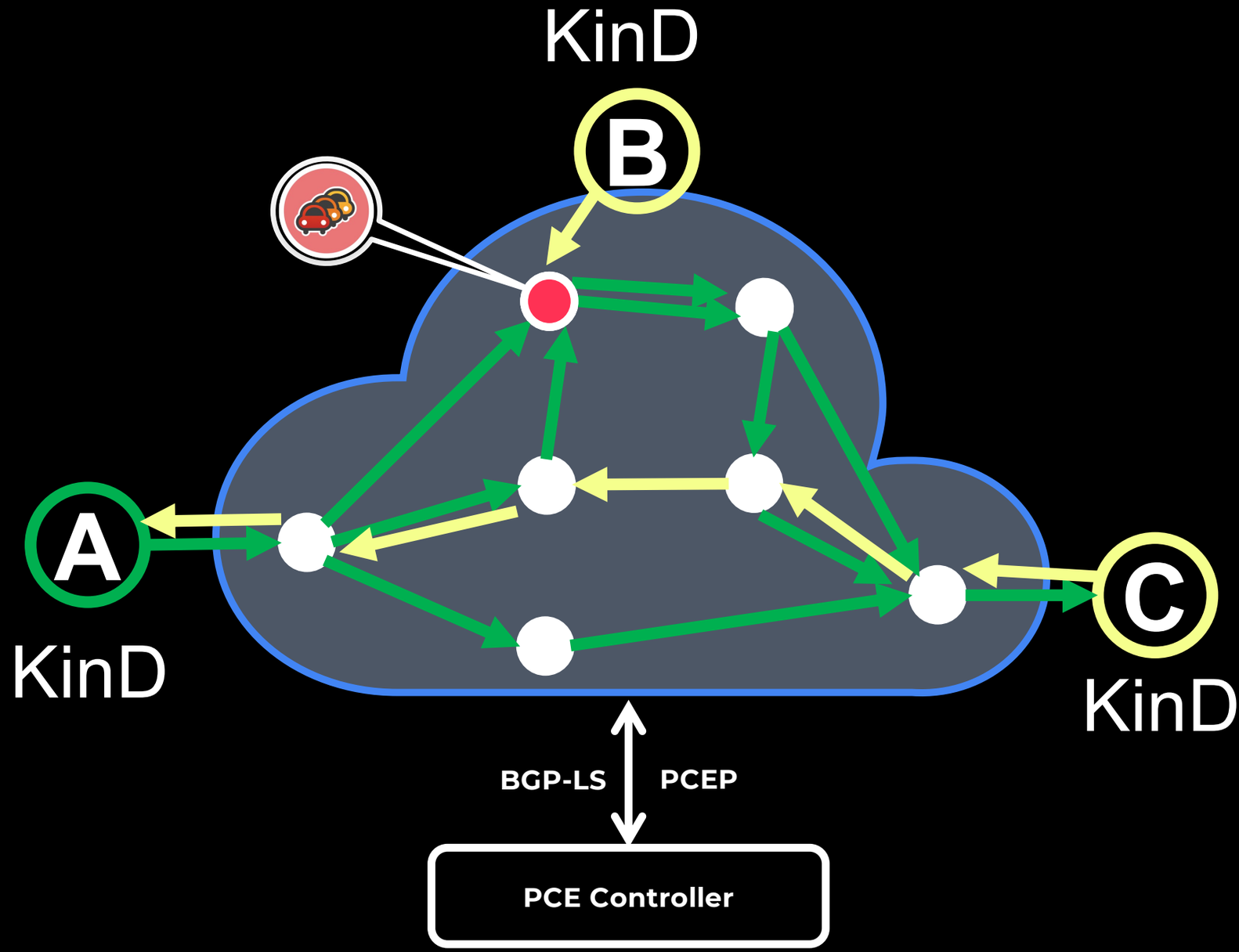
🧾 Defined in a single **YAML** file for automated deployment *

🔄 Useful for network function testing, CNFs, and automation labs

Clab+KinD: Components & Topology

- 🔷 k8s-kind nodes: create and manage Kind clusters
- 🔷 ext-container nodes: expose control-plane & worker nodes for linking
- 🧵 Use exec to configure interfaces like eth1 on K8s nodes
- 🔗 Define links between K8s nodes and network devices (e.g., SR Linux)
- 📦 startup-config: custom Kind YAML config for multi-node clusters
- 📡 Each Kind node joins the Docker network and underlay topology

KinD: Simulate Traffic thru the core



gNMIc

YANG

Protobuf

gRPC

HTTP2

TLS/TCP

Encodes YANG content
Encodes the gNMI operations

4 Operations:
- Capabilities
- Get
- Set
- Subscribe

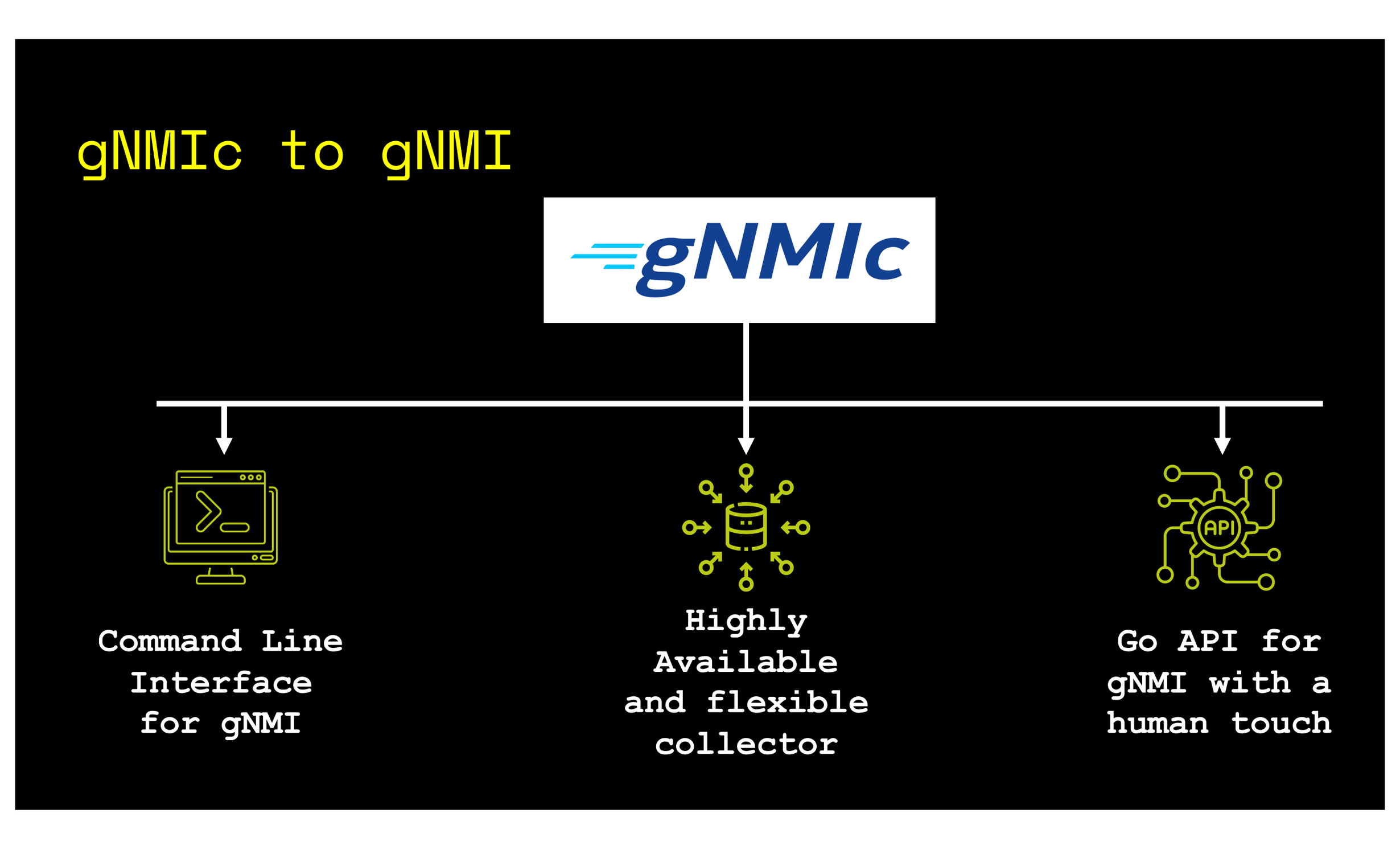
Client-Server RPC framework

gRPC builds on HTTP2 features to multiplex multiple streams over a single TCP connection

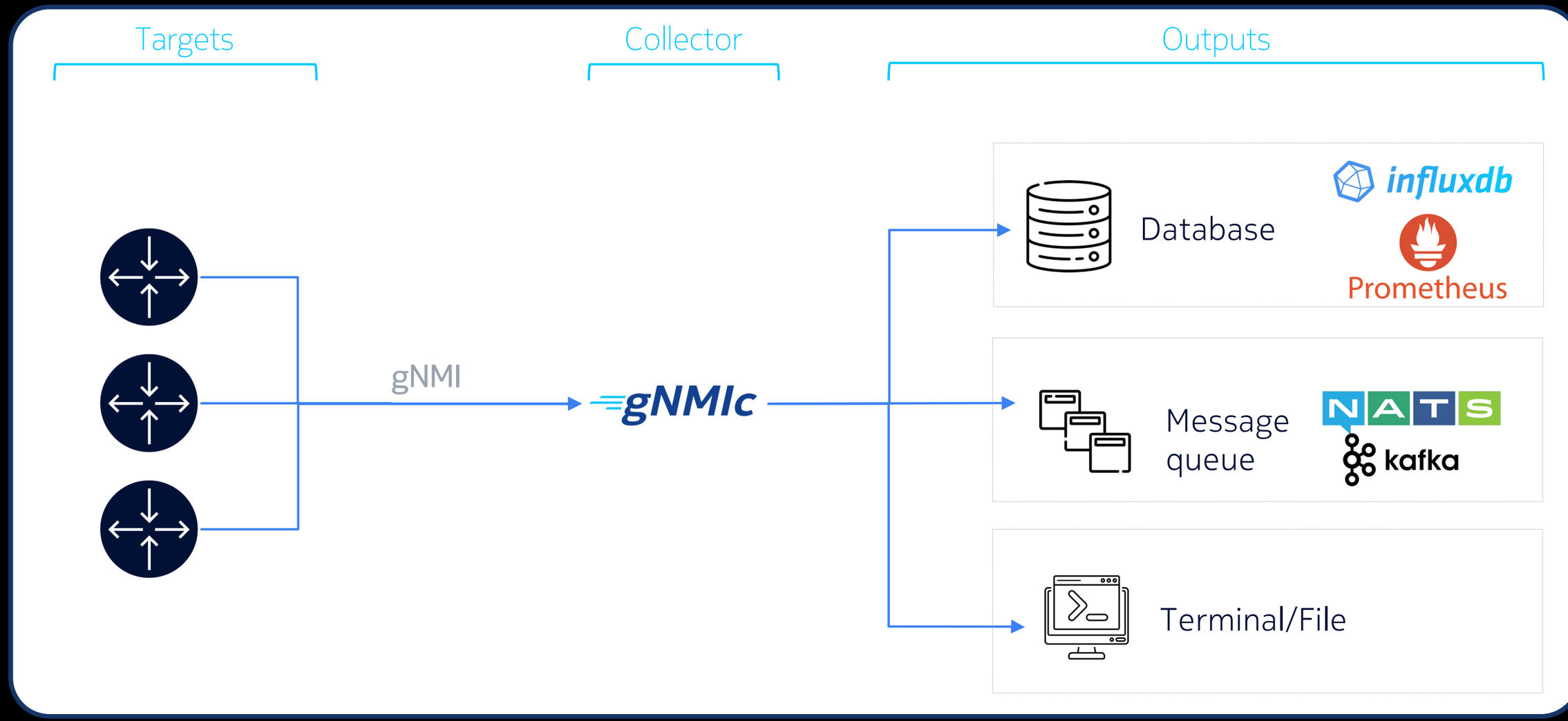
gNMIc: GNMI Service Definition

- **Capabilities**: The client requests the target's capabilities (supported YANG models and their revision date)
- **Get**: The client retrieves a snapshot of the data identified by a path from the target
- **Set**: The client modifies the configuration of the target
- **Subscribe**: The client can subscribe to data identifies a set of paths, the target will stream back the data to the client periodically or when it changes.

gNMIc to gNMI



gNMI collector

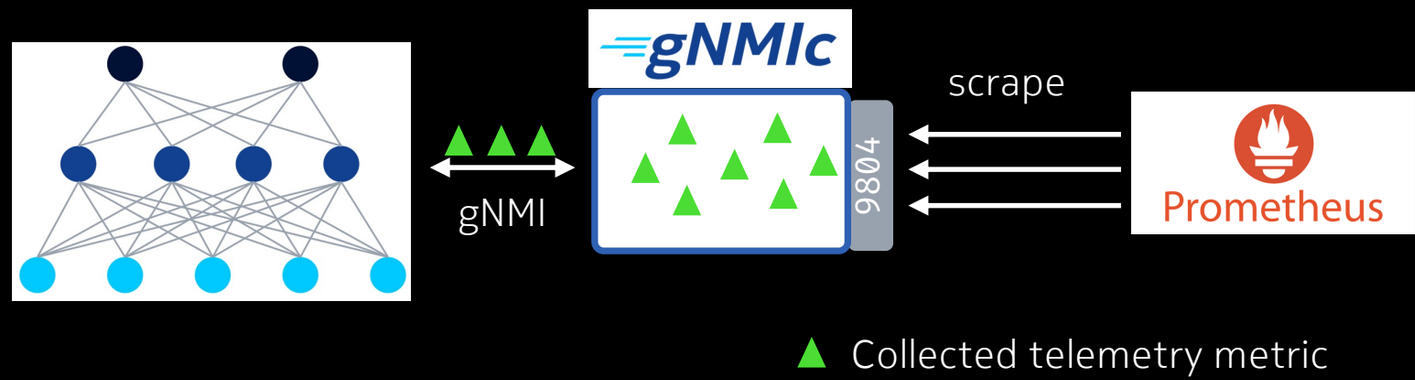


gNMI collector

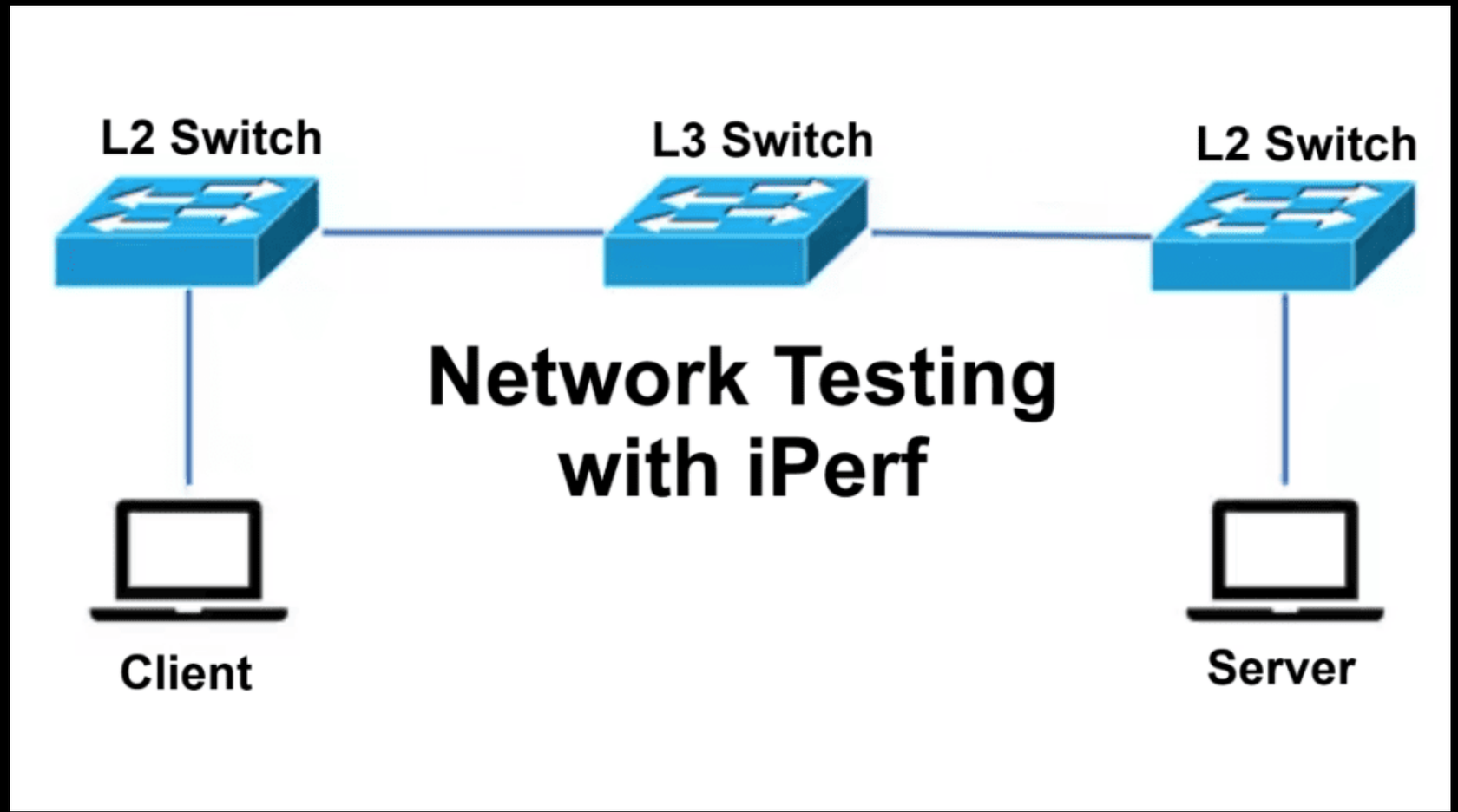
More details of Prometheus on the second part of the Workshop!

```
outputs:
  output1:
    type: prometheus
    listen: ":9804"
    export-timestamps: false
    strings-as-labels: false
```

```
targets:
  leaf1:
    outputs:
      - output1
```



iperf


L2 Switch
L3 Switch
L2 Switch
Network Testing
with iPerf
Client
Server

iperf: what?

- A network testing tool to measure bandwidth, loss, and latency.
- Focuses on TCP, UDP, and SCTP traffic performance.
- Designed for active testing between a client and a server.
- Supports modern features like JSON output and multi-threading.
- Common in performance tuning, validation, and benchmarking.

iperf: Why?

- Simple CLI interface, fast setup.
- Supports custom test durations, window sizes, and parallel streams.
- Useful for verifying network speed or identifying bottlenecks.
- Provides real-time throughput metrics.
- Ideal for both LAN and WAN environments.

iperf: Key features

- Client-server architecture (one side runs -s, the other -c).
- Customizable port, protocol, and test duration.
- Compatible with IPv4 and IPv6.
- Reports throughput, jitter, packet loss (UDP).
- Output can be human-readable or machine-parsed (JSON).

iperf: example

-s: Start as server
-c: Run as client
-p: Specify TCP/UDP port

Ensure firewall allows chosen port

Useful for parallel test sessions or avoiding conflicts

```
# Server (listening on port 5202)
iperf3 -s -p 5202

# Client (connect to server on port 5202)
iperf3 -c 192.168.1.10 -p 5202
```

iperf in the lab

- Wrapping up iperf in a container
- Managing multiple instances to different ports
- Using k8s for orchestration
- Declarative


NETWORK AUTOMATION FORUM
Preparing an App for
Container Deployment

Why Dockerfile Matters (vs Manual Setup)

Why Dockerfile Matters (vs Manual Setup)

- ✅ **Automates builds** – no need to set up environment manually
- ✅ **Ensures consistency** – same setup across all machines and teams
- ✅ **Version-controlled** – stored in Git, tracks changes over time
- ✅ **Portable & repeatable** – build once, run anywhere
- ❌ **Without Dockerfile** – manual setup = error-prone and hard to reproduce

Dockerfile: Key Components

- FROM – Base image to build your container from (e.g., python:3.11-slim)
- WORKDIR – Sets the working directory inside the container
- COPY – Copies files from your local machine to the container
- RUN – Executes shell commands during the image build process
- CMD / ENTRYPOINT – Defines default command(s) when container starts
- EXPOSE – Documents the port the container listens on (optional)

🧪 Why Prepare Apps for Containers?

- ✅ **Environment variables** make the app configurable (no hardcoded values)
- ✅ **Volumes** allow persistent data and external config files
- 🔄 **Separation of config from code** — easier updates and reuse
- 📦 **Portability** — run the same container anywhere with different settings
- 🔐 **Secrets & credentials** managed securely via env or mounted files
- 🛠️ **Dev → Test → Prod** flexibility without changing the app logic

Prep your App

- Python:

```
parser = argparse.ArgumentParser(description='Kafka Alert Exporter to Prometheus')
  parser.add_argument('--bootstrap-servers', required=True, type=str, help='Kafka Bootstrap Server')
  parser.add_argument('--cert', type=str, help='CA certificate path for Kafka')
  parser.add_argument('--port', type=str, default="8001", help='HTTP server port (default: 8001)')
  parser.add_argument('--config', required=True, type=str, help='YAML Alert topics and metrics')
  parser.add_argument('--debug', action='store_true', help='Activate debug mode')
```

- Dockerfile:

```
ENV KAFKA_BOOTSTRAP_SERVERS=localhost:9092
ENV METRICS_PORT=8001
ENV ALARMS_FILE=/app/alarms.yml

CMD ["python3", "./kafka-alert-exporter.py", "--bootstrap-servers", "${KAFKA_BOOTSTRAP_SERVERS}", "--
port", "${METRICS_PORT}", "--alarms", "${ALARMS_FILE}"]
```

Key Concepts for Preparation

- ⚙️ Environment Variables
  - Who does not love hardcoded passwords in a Github Repo?

```
grafana:
  image: grafana/grafana:latest
  container_name: kafka-alert-grafana
  volumes:
    - ./grafana/datasources → Grafana data sources
    - ./grafana/dashboards → Grafana dashboards
  environment:
    - Admin password = secret
  depends_on:
    - prometheus
  ports:
    - 3000:3000
  restart: unless-stopped
```

Key Concepts for Preparation

- 🏷️ Tagging the Image
  - Pick a donut, any donut. Surprise bugs included!

🧩 Docker Compose vs Manual Docker

- ✅ **Compose:** defines multi-container apps in one YAML file
- ✅ **Single command deploy** – docker-compose up starts everything
- ✅ **Easier networking** – containers auto-connect via service names
- ✅ **Environment config** – use .env files for flexible settings
- ❌ **Manual docker run** – repetitive, error-prone, hard to scale

Docker Compose Code Modularity

```
FROM ubuntu:22.04

# Install dependencies
RUN apt-get update && \
    apt-get install -y wget gnupg2 curl software-properties-common

# Install Prometheus
RUN useradd --no-create-home --shell /bin/false prometheus && \
    mkdir /etc/prometheus /var/lib/prometheus && \
    cd /tmp && \
    wget https://github.com/prometheus/prometheus/releases/download/v2.52.0/prometheus-2.52.0.linux-amd64.tar.gz && \
    tar xvf prometheus-2.52.0.linux-amd64.tar.gz && \
    cd prometheus-2.52.0.linux-amd64 && \
    cp prometheus promtool /usr/local/bin/ && \
    cp -r consoles console_libraries prometheus.yml /etc/prometheus

# Install Grafana
RUN wget -q -O - https://packages.grafana.com/gpg.key | apt-key add - && \
    add-apt-repository "deb https://packages.grafana.com/oss/deb stable main" && \
    apt-get update && \
    apt-get install -y grafana

# Expose Ports
EXPOSE 9090 3000

# Start both services
CMD service grafana-server start && \
    prometheus --config.file=/etc/prometheus/prometheus.yml
```

Docker Compose Code Modularity

```yaml
version: '3.8'

services:
  prometheus:
    image: prom/prometheus:latest
    container_name: prometheus
    ports:
      - "9090:9090"
    volumes:
      - ./prometheus/prometheus.yml:/etc/prometheus/prometheus.yml

  grafana:
    image: grafana/grafana:latest
    container_name: grafana
    ports:
      - "3000:3000"
    environment:
      - GF_SECURITY_ADMIN_USER=admin
      - GF_SECURITY_ADMIN_PASSWORD=admin
    volumes:
      - grafana-storage:/var/lib/grafana

volumes:
  grafana-storage:
```

Docker Compose: YAML Components

- services – Main section to define each container
- build / image – Either build from Dockerfile or pull an image
- ports – Maps container ports to host ports ("8080:80")
- volumes – Mounts host paths or named volumes
- environment – Sets environment variables inside the container

Health Checks & Dependencies in Docker

- HEALTHCHECK (in Dockerfile) – tests if the container is still working
  - Example: HEALTHCHECK CMD curl --fail http://localhost:80 || exit 1
- Helps monitor service status for orchestrators (e.g., Compose, Swarm)
- Dependencies handled in Docker Compose with depends_on
  - Example: web depends_on: [db, redis]
- depends_on controls start order, but not readiness
  - Use health checks + restart policies for full dependency control

📦 Why We Need Container Registries

- ✅ **Central storage** – push and pull images from a shared location
- ✅ **Versioning** – tag and manage image versions easily
- ✅ **Team collaboration** – share images across dev, test, and prod
- ✅ **CI/CD integration** – automate image builds and deployments
- ✅ **Security & auditing** – scan images, enforce access control
- ❌ Without a registry – sharing = manual, error-prone, and insecure

How to manage Container Images

- Image built and tested — now what?
- ☁️ Public Repositories
  - Push it to share and deploy anywhere
  - E.g. Docker Hub
- 🔐 Private Registries
  - Push it to your teams private registry for secure access

🧪 Push Image to Local Docker Registry

- 1. **Tag the image** with the local registry address:
  - docker tag my-app:latest localhost:5000/my-app:latest
- 2. **Push it to the local registry**:
  - docker push localhost:5000/my-app:latest

✅ This assumes your **local registry** is already running on port `5000`
(e.g., started with `docker run -d -p 5000:5000 registry:2`)


NAF
NETWORK AUTOMATION FORUM
Questions?