KubeCon

CloudNativeCon

THE LINUX FOUNDATION

OPEN SOURCE SUMMIT

AI_dev

Open Source GenAI & ML Summit

China 2024

KubeCon | CloudNativeCon

THE LINUX FOUNDATION OPEN SOURCE SUMMIT

AI_dev Open Source GenAI & ML Summit

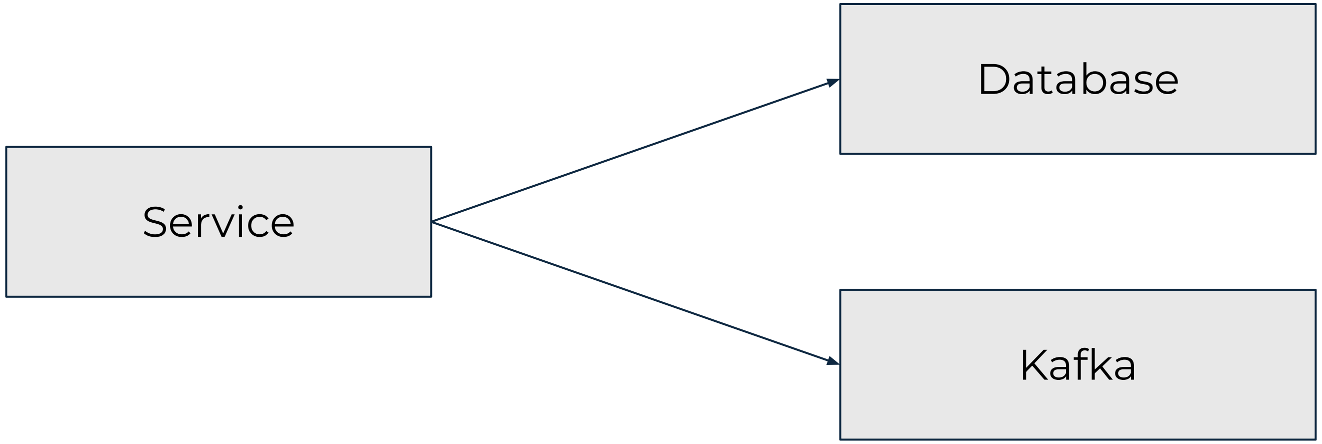
China 2024

Observability Supercharger

Build the Traffic Topology Map for Millions of Containers with Zero Code

Lim Teck Chuan
Chow Sheng Wei
Engineering Infra, Shopee

What is a topology map?



Why is it helpful?

Workloads	Container Cache, Database, Queue
Observability	Metrics, Logs, Traces
Platform	DataOps FinOps, AIOps, DevOps
Business Objectives	Datacenter Migration Dependency Graph, Resource Accountability

Dependency graph

- What?
  - Stateless/stateful service tagging
- Why?
  - Different workload types require different operational procedures

Dependency graph

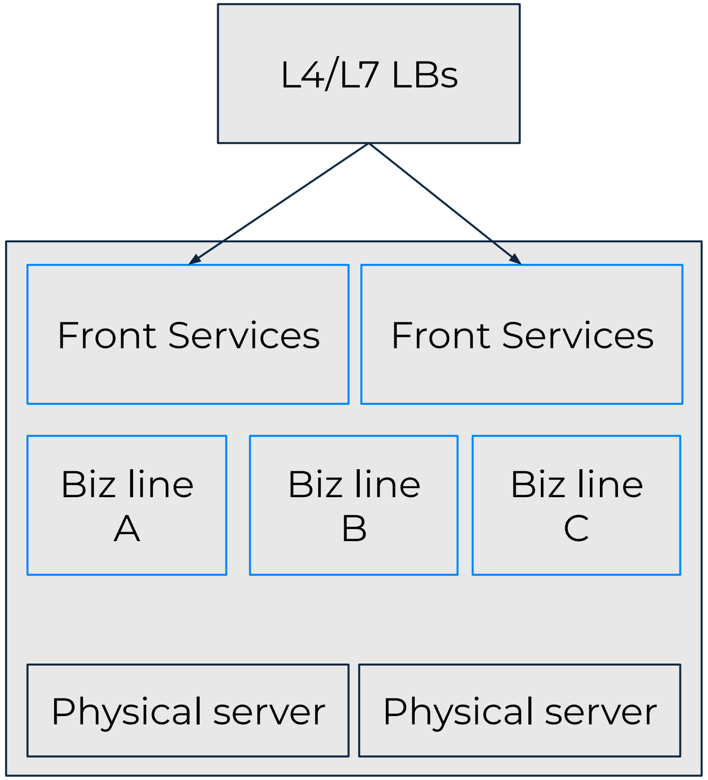
- What?
    - Service to middleware / storage relationships
    - Service to service call graphs
- Why?
    - Incident response

Resource accountability

- What?
  - Resource to service relationships
- Why?
  - Cost attribution and budgeting
  - Service migrations makes accounting difficult

What does our container ecosystem look like

- Traffic enters our L4/L7 load balancers
- Front services translate HTTP to RPC
- Pods run a variety of workloads
  - API services
  - Queue consumers
  - Cronjobs
- Physical servers
- Across 10 AZs



Various ways we tried

- Platform workflow
- Client side instrumentation
- Domain sniffing

Platform workflow

- How?
  - For all new clusters, relationship binding to a service is mandatory
  - For all existing clusters, do a one time data collection from service owners
- What?
  - Lots of legacy clusters had no bindings
  - Existing bindings had no guarantee of being correct
- Lessons learnt
  - Data collection takes way too long and costs a lot in terms of human labor

Client side instrumentation

- How?
  - Add instrumentation in client libraries (kafka, redis, etc..)
- What?
  - Long rollout
  - Required code changes
  - Not all services use the internal client libraries
- Lessons Learnt
  - Code changes takes a long time for rollout

Domain Sniffing

- How?
  - eBPF agent on all machines that intercepts connect() syscalls
  - Extract the domain from intercepted call
  - Extract service name via pid envvars
- What?
  - Only proved usage, we needed ownership
  - It would have attributed resources to wrong services (shared infrastructure teams)
- Lessons learnt
  - eBPF was likely the way forward
  - Needed to start intercepting on L7 to be more accurate

Objectives

- Non-intrusive
- Reliable
- Scalable on capabilities
- Sounds like …. https://github.com/pixie-io/pixie

Limitations

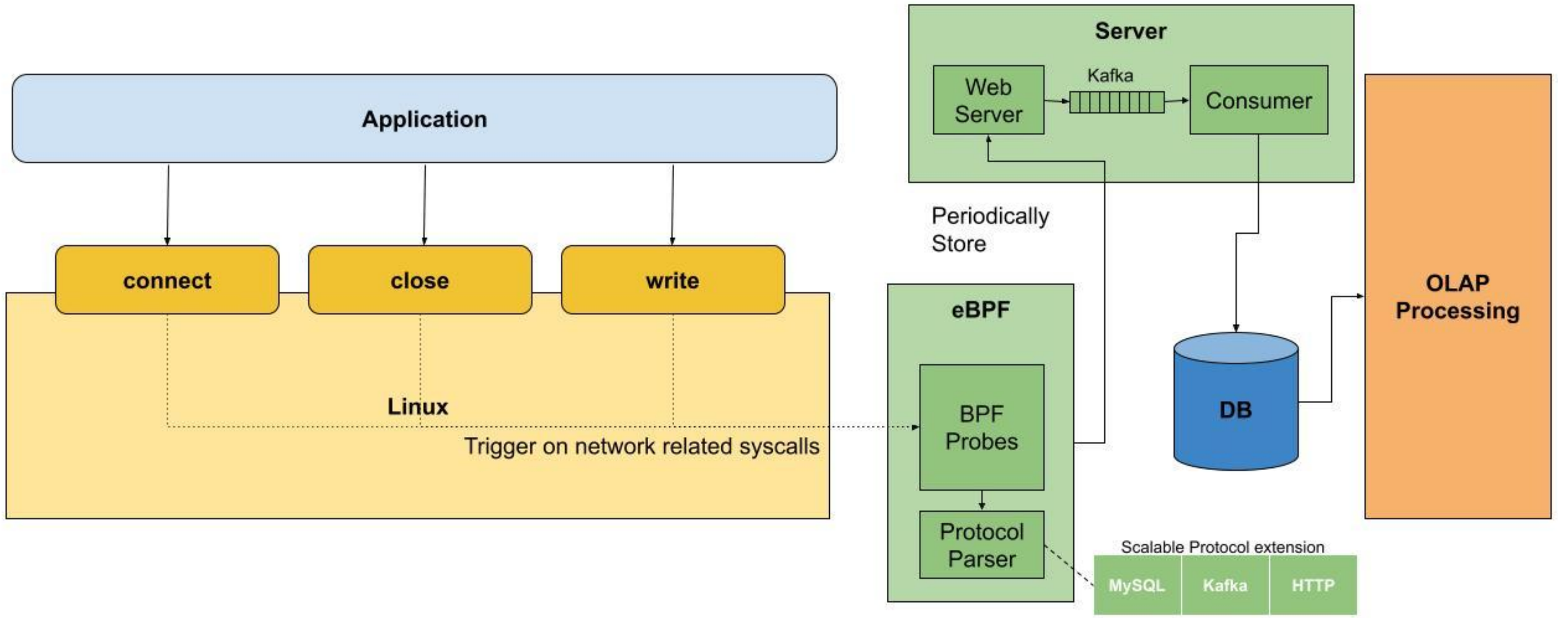
- Not all workloads are Kubernetes Pods
- Requires min specific version: Kubernetes v1.21+
- Studying Pixie indepthly reveals **PEM (Pixie Edge Module)**

Inspirations

- PEM as a protocol parser (known as Stirling in the original code base)
- Supports **10 protocols**
- **Fully scalable and extensible**
- Rewritten as Python BCC script

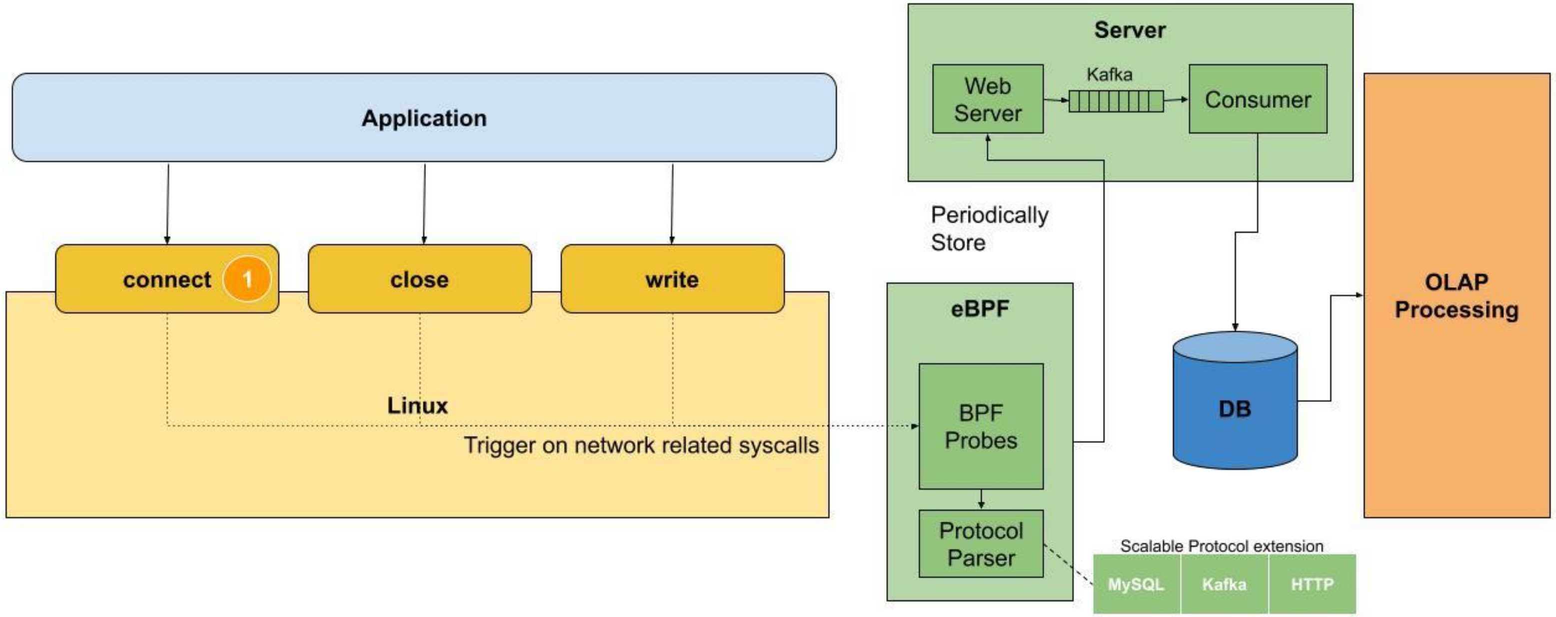
Architecture

- Probed syscalls: connect / close / write / tcp_v4_connect / tcp_v6_connect / udp_recvmsg



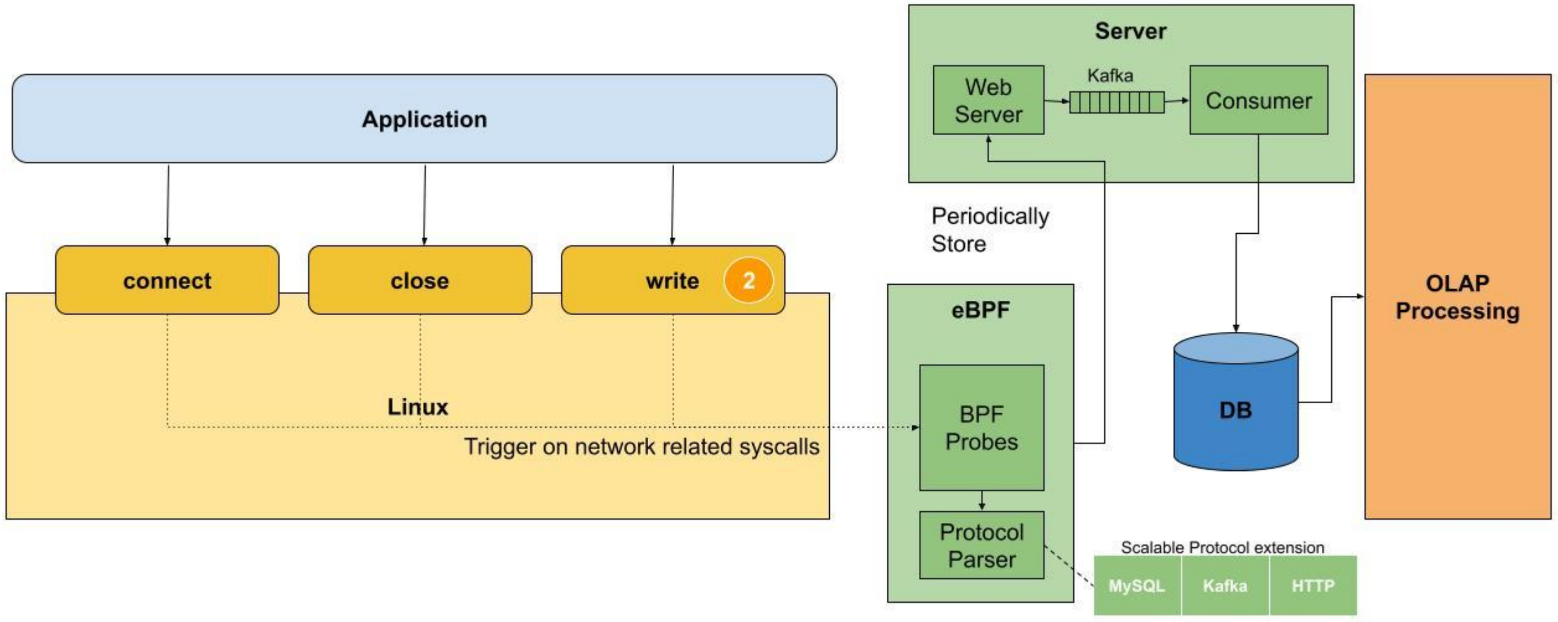
Walkthrough

- Application sends MySQL network request
- Connect syscall is probed and focus only on AF_INET & AF_INET6 connection
- Stores fd in BPF hash



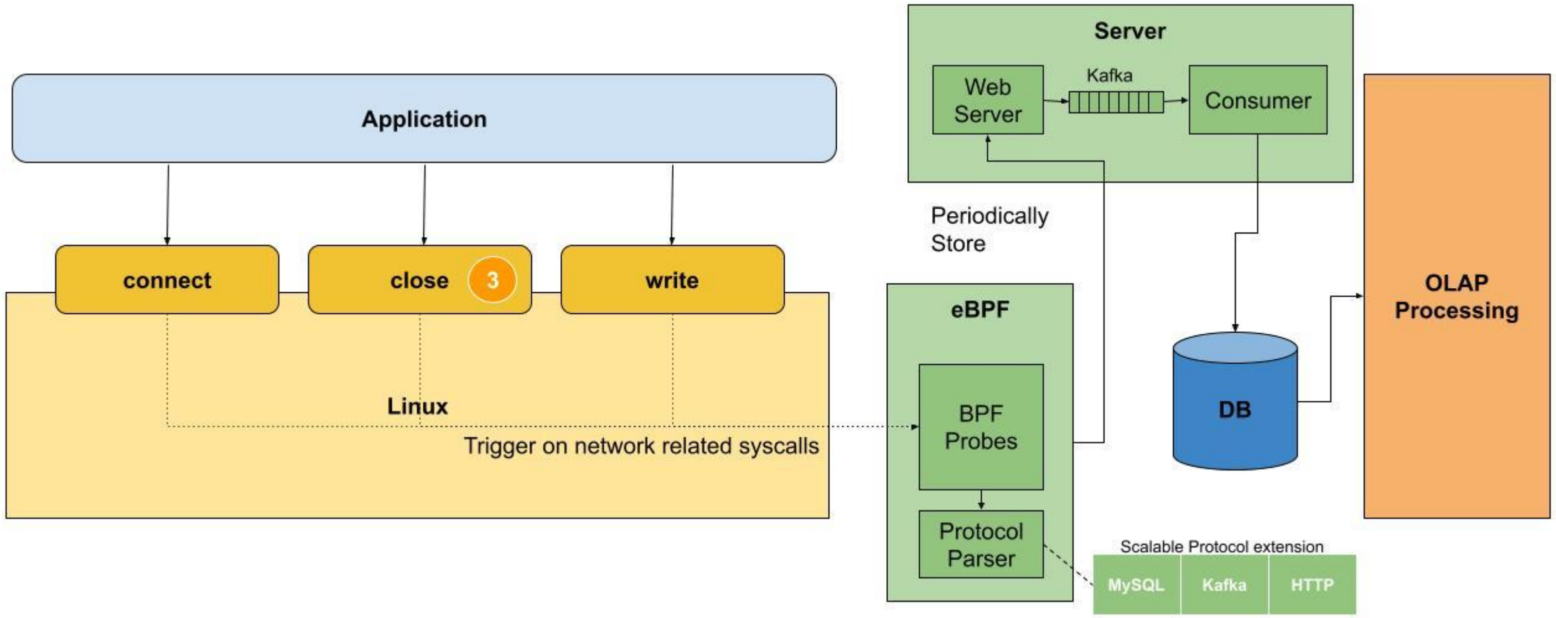
Walkthrough

- Write syscall is probed and reference fd
- If fd is within BPF hash, copy write buffer and send to user space using perf output



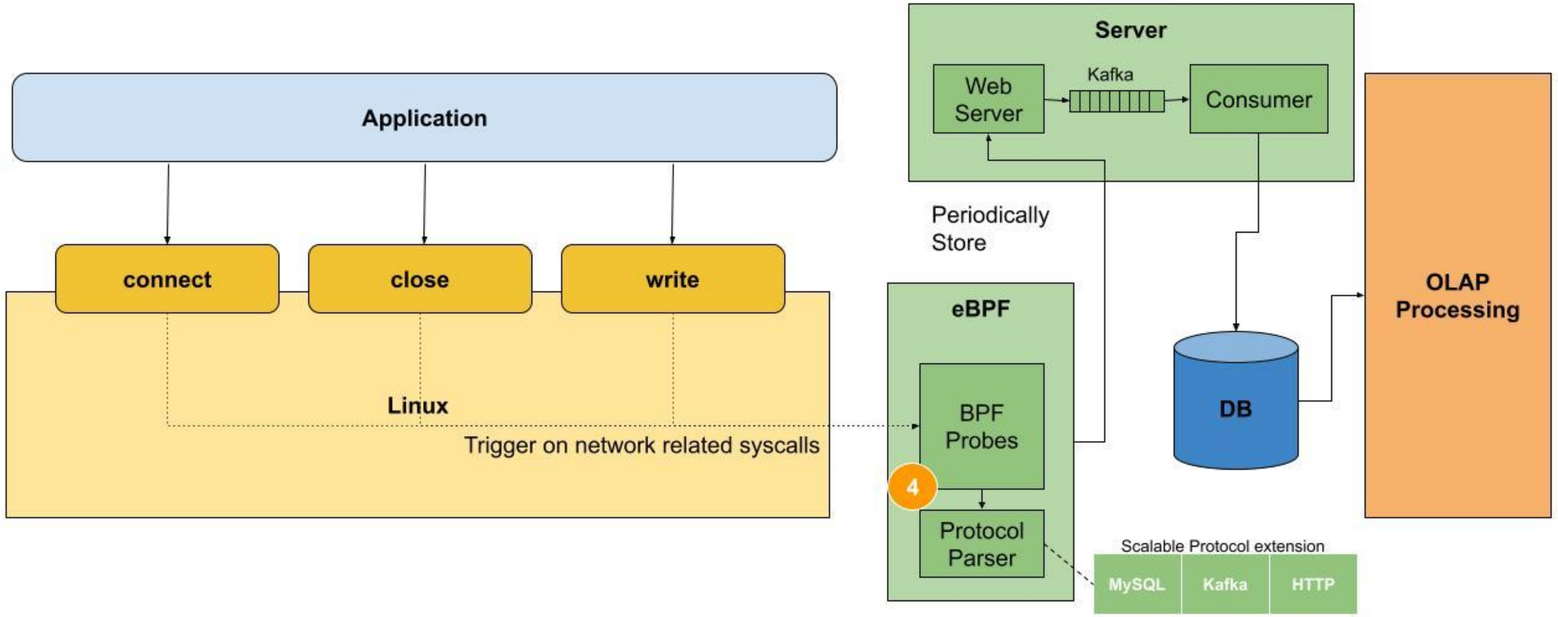
Walkthrough

- Release the fd from BPF hash



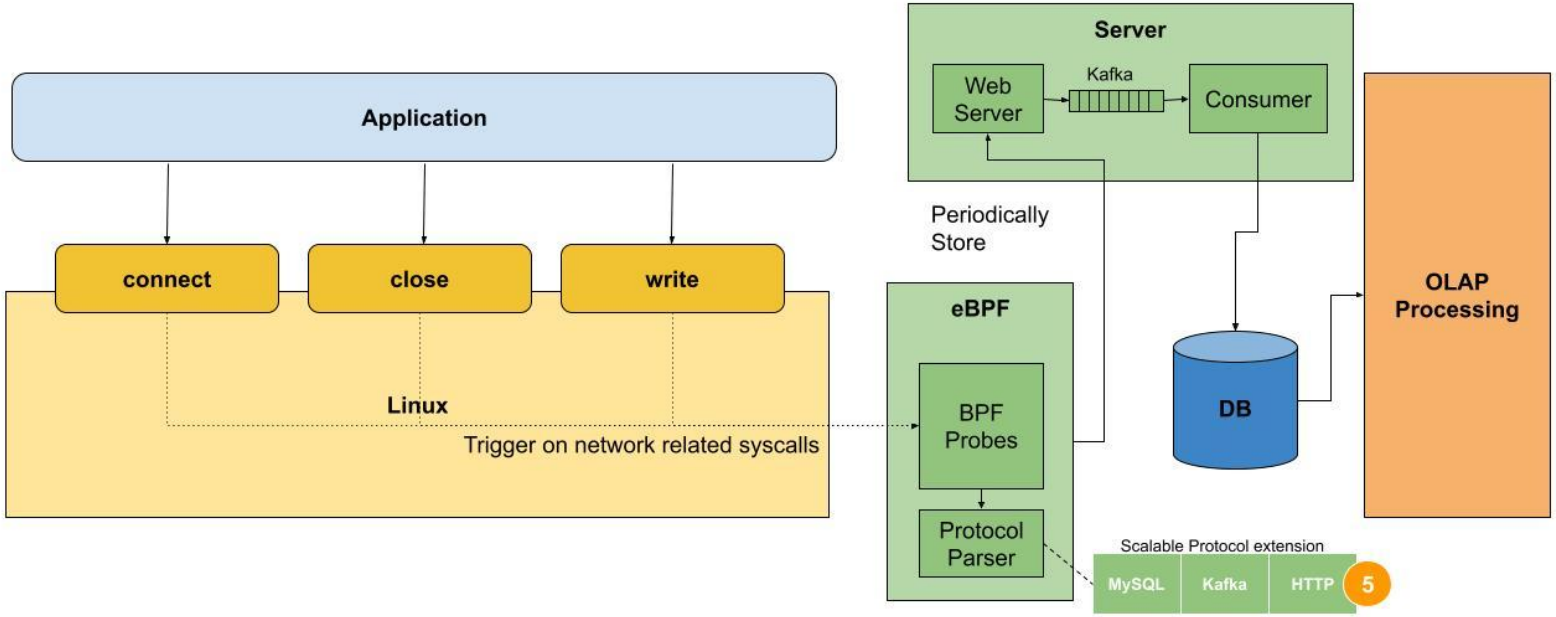
Walkthrough

- eBPF receives write buffer from perf output in user space



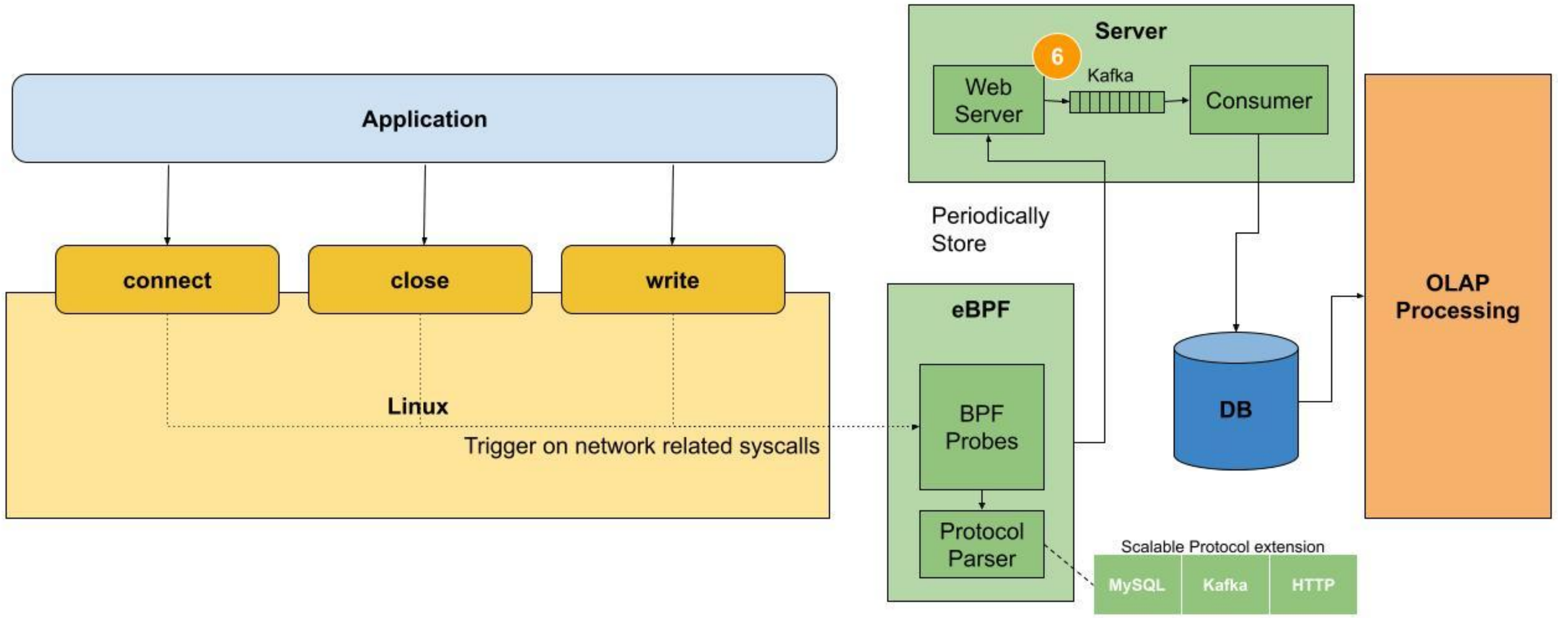
Walkthrough

- Protocol parser attempts to decode the write buffer via various protocols (MySQL, Kafka, HTTP)



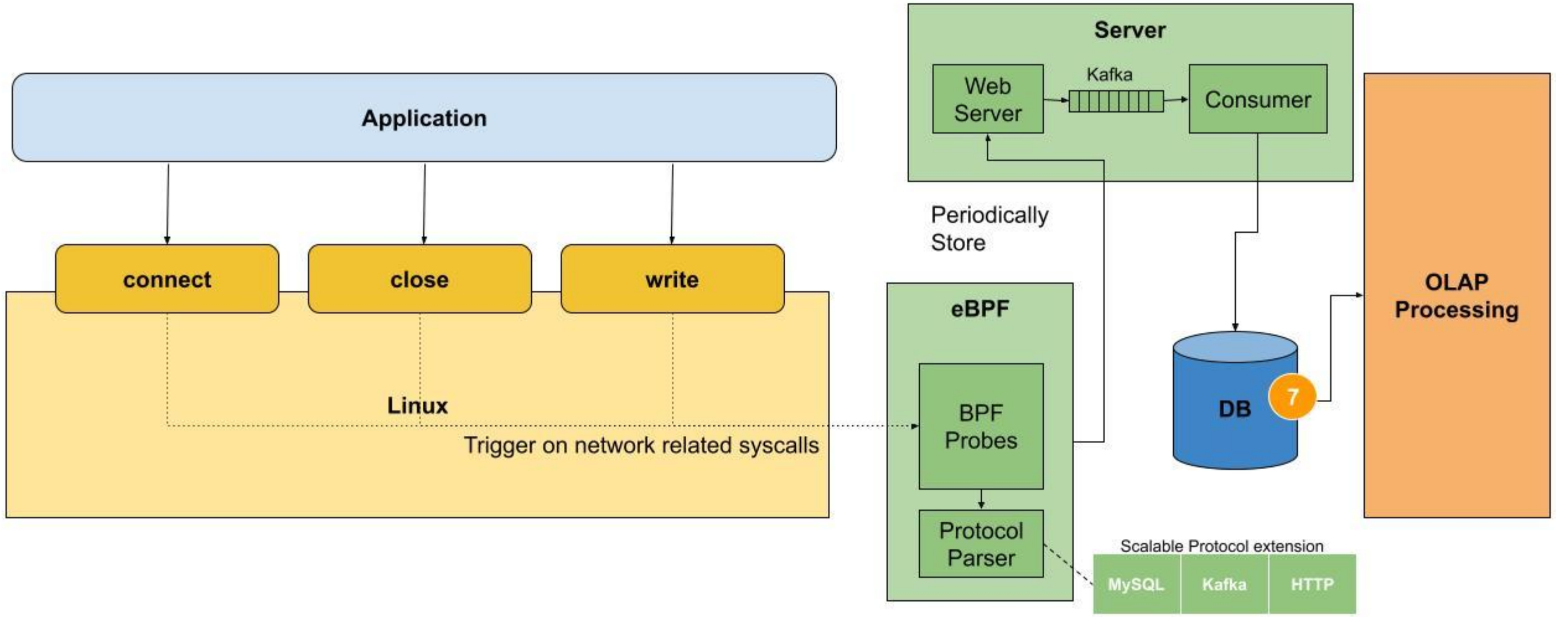
Walkthrough

- After successful decoding, decoded data is sent to a web server for storage
- Producer - subscriber design to prevent DDoS



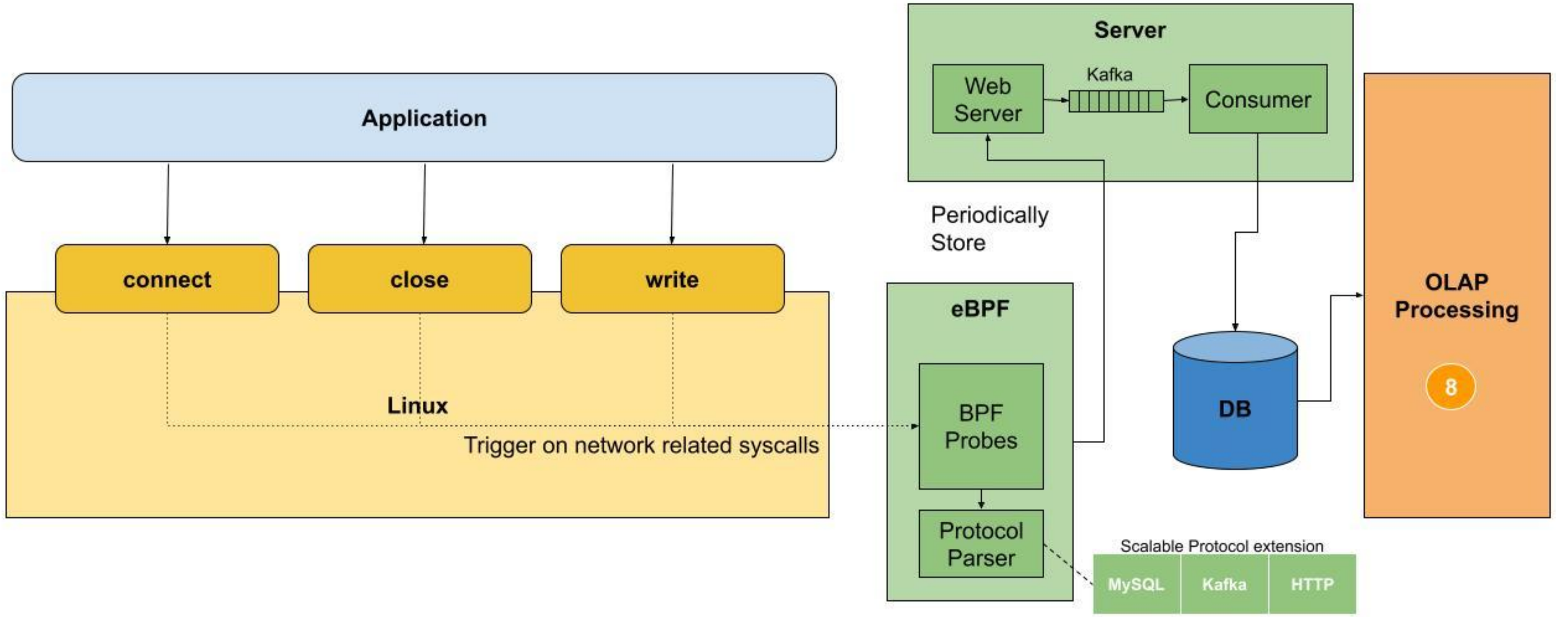
Walkthrough

- Database syncs to data warehouse using RTI (real time ingestion)



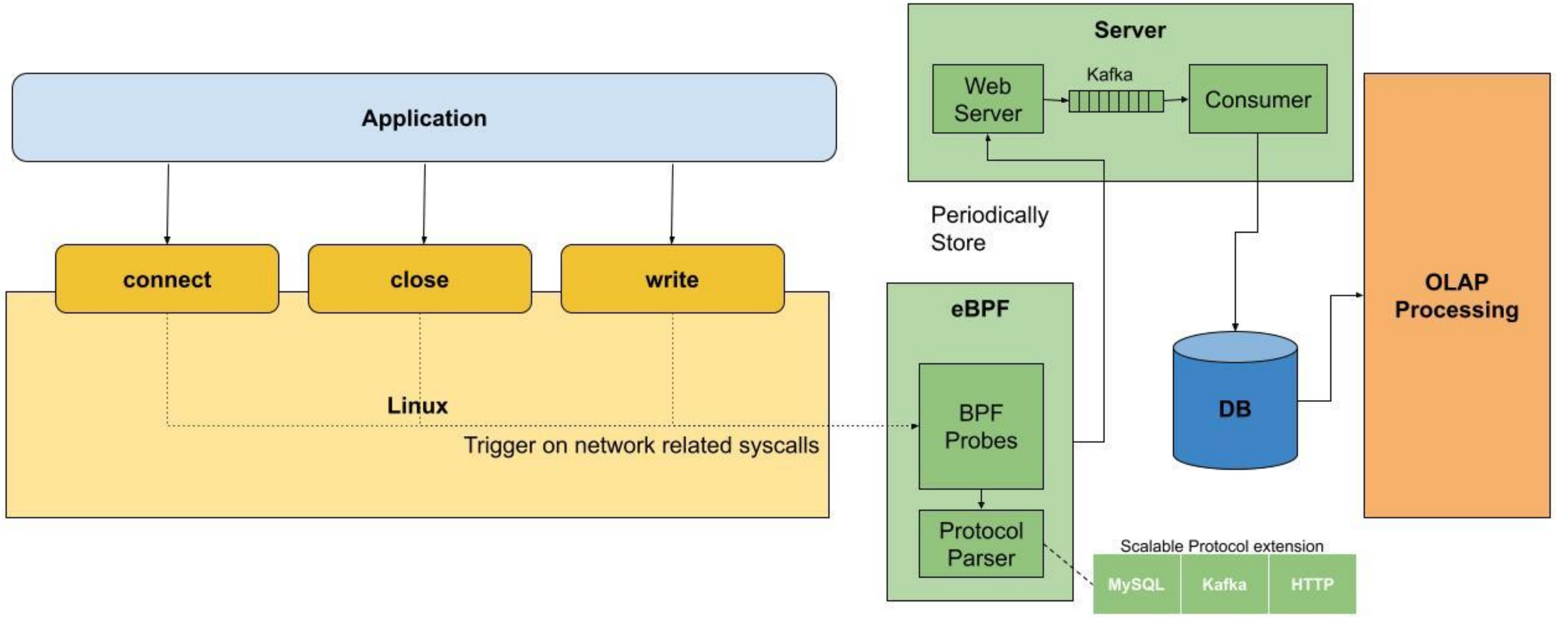
Walkthrough

- OLAP processing in data warehouse for empowerment & association



Requirements

- Linux OS
- HTTP web server
- Relational database
- Data warehouse (optional)



What did we achieve?

- Over 1 million traceable middleware / database connections
- Over 10 millions traceable DNS connections
- Over 5 millions verified middleware / database usages

Key Takeaways

- Lightweight (A simple eBPF python script)
- Scalable (Able to work with both legacy systems / Kubernetes ecosystem)
- Empower data with DataOps