eBPF FOUNDATION

eBPF for the Infrastructure Platform

How Modern Applications Leverage Kernel-Level Programmability

by Andrew Green

November 2025

Contents

Executive Summary

Over the past ten years, the eBPF community, ecosystem, project, and product landscapes have grown considerably. Organizations now benefit from a wide range of tools that can help them protect and optimize their infrastructure and applications by instrumenting the kernel. In the next ten years we expect this trend only to accelerate. We are past the early adopters' phase, where eBPF was reserved to hyperscalers and niche use cases. **eBPF is becoming the strategic platform of choice for infrastructure teams.**

eBPF-based tools have become **indispensable in today's infrastructure platforms.** The observed performance improvements, enhanced security, flexibility in instrumentation, and overhead reduction represent major advancements compared to traditional security, observability, and networking applications.

This is because eBPF provides control over otherwise restricted parts of the operating system. Unlike traditional agents and kernel modules, these programs execute directly within the operating system kernel with access to system data structures, network packets, and kernel events.

This model eliminates the performance tax of frequent user–kernel transitions and provides a secure sandbox for custom logic. **Developers can reprogram kernel behavior dynamically,** enabling rapid iteration and innovation across many areas of the kernel, even some that have been unchanged for decades. This promotes a rapid innovation cycle and the opportunity to define granular and custom application logic safely.

In this paper, we will look at how the eBPF landscape evolved, how it changed existing and redefined product categories, and **how it is shaping the future of infrastructure platform development.**

Product Categories Enhanced Using eBPF

Since eBPF was merged into the Linux kernel, it has **reshaped the foundation of modern infrastructure.** We've seen a range of tools and products that are either replacing legacy counterparts or even defining new categories altogether. Some of these include:

- Container networking
- Runtime security
- Network security
- Cloud observability
- LLM security
- FinOps

Compared to scaling up and scaling out when hitting capacity constraints, eBPF addresses infrastructure scalability by reprogramming how existing infrastructure operates. This enables organizations to process significantly more requests on the same hardware, be it physical or virtual.

Each new generation of infrastructure products is effectively an eBPF application under the hood. In the sections below, we provide examples to better illustrate how developers are **converging on the single principle that the fastest, most programmable, and secure path is leveraging eBPF.**

Virtualized Networking

There are many flavors of virtualized networking, including software-defined networking for on-premises networking, cloud networking for orchestrating hyperscaler networks, and container networking for Kubernetes.

Virtualization introduced enormous flexibility into networking, but also an invisible overhead with each layer of abstraction. By letting packet processing, policy enforcement, and routing logic happen directly in the kernel, **eBPF can improve both performance and scalability of virtualized networks,** realizing performance gains at the software level that are otherwise unachievable with traditional cloud and container networking constructs.

SCALABILITY EXAMPLE #1

INCREASING VIRTUALIZED NETWORK THROUGHPUT TO MATCH PHYSICAL NETWORK THROUGHPUT

Both virtualization and containerization introduce some overhead for managing the software-defined functions. For example, the virtual ethernet device (veth) acts as a tunnel between network namespaces to create a bridge to a physical network device in another namespace, inducing latency and limiting throughput compared to host networking. Container Networking Interface (CNI) plugins attach a Kubernetes Pod to the node it's hosted on by a veth device.

To improve network throughput to match that of the host, organizations can replace the legacy veth device with **Netkit**, which **eliminates the software abstraction overhead.**

For example, the default Linux networking available for containers uses iptables. It configures the IP packet filter rules of the kernel firewall, but it was never designed for highly scaled operations of today. The sequential rule matching and stiff IP based rules struggle with frequently changing IP addresses. eBPF on the other hand uses efficient hash tables allowing for constant time lookups, maintaining consistent performance even as the number of services within a cluster increases substantially.

Another example is **Katran**, which uses eBPF with XDP to achieve ultra-low latency by running a packet handling routine immediately after a packet is received by the network interface card (NIC) and before the kernel intercepts it, bypassing the traditional Linux networking stack entirely.

SCALABILITY EXAMPLE #2

LOWERING PACKET FILTERING LATENCY FOR HIGH-VOLUME TRAFFIC

XDP, an eBPF-based high-performance network data path, hooks at the lowest level before the network stack for coarse packet filtering. This is suitable for high-volume traffic filtering such as denial-of-service protection. Some studies have shown that **XDP can yield four times the performance** in comparison to performing a similar task in the kernel using common packet filtering tools. **Cloudflare** uses eBPF to protect against Terrabit per second scale attacks.

Network Security

Firewalls have been synonymous with on-premises network security. In the cloud, security providers have virtualized their firewalls appliances and deployed them as cloud-based instances. These naturally have throughput limits and act as a chokepoint.

Instead of routing packets through external appliances, **packet inspection and filtering are done at the kernel level on each endpoint,** and are attached to network interfaces at both ingress and egress points using the Linux traffic control subsystem. This creates distributed firewalls and enforces microsegmentation. eBPF-based network security offers a wide range of technical features, including Layer 4/7 stateful filtering, reputation-based detection, microsegmentation, IP masquerading, and NAT.

This shift means network security no longer needs to live in a box, physical or virtual. With eBPF, it becomes **an integral, programmable layer of the operating system itself,** enabling zero-trust microsegmentation that scales as seamlessly as the workloads it protects.

Runtime Security

Several types of solutions secure processes or applications as they are executing, most notably being endpoint detection and response (EDR) and the proposed next-generation extended detection and response (XDR).

As EDR was only designed to protect endpoints, a more comprehensive solution, XDR, attempted to bring runtime security to clouds, networks, and virtualized appliances. However, the agents used to enforce security policies consume a high percentage of resources on the hosts they run on and require continuous management. The solutions that do not use kernel modules have their visibility and enforcement restricted by the kernel data made available via APIs.

SCALABILITY EXAMPLE #3

LOWERING OVERHEAD ASSOCIATED WITH RUNNING SENSORS AND AGENTS

Runtime security is often based on agents deployed on hosts to enforce OS-level controls and visibility. However, user-space agents have a high performance tax. **eBPF dramatically reduces observability overhead** by collecting metrics, logs, and traces directly in kernel space without expensive system calls. It can implement efficient memory management policies, custom garbage collection triggers, and event monitoring strategies that minimize operational overhead. Some eBPF-based agents and sensors can introduce as little as a **2% CPU load increase** relative to baseline levels, and near-zero memory increase.

SCALABILITY EXAMPLE #4

OBSERVABILITY AND OPTIMIZATION OF CPU CYCLES

eBPF enables continuous profiling and optimization by tracking CPU cycles, function call frequencies, and hot code paths in real-time, allowing developers to identify and optimize bottlenecks. eBPF programs can also implement custom scheduling policies and load balancing algorithms directly in the kernel, reducing context switches and improving CPU utilization for request processing. With this visibility, Meta **observed 20% performance gains** after changing a single character in their code.

Sched_ext is a Linux kernel framework that allows developers to write, test, and deploy custom CPU schedulers. Meta explored different scheduling strategies like work conservation, idle CPU selection, and soft-affinity and **achieved over 5% throughput improvement** on a latency-sensitive workload by optimizing CPU selection and confining low-priority tasks to dedicated CPU pools.

The **runtime security landscape has quickly adopted eBPF** and evolved into more robust products, where **enforcement and visibility are achieved by introducing dynamic, secure, low-overhead, and efficient programs that run directly in the kernel.** Some examples of categories that heavily use eBPF for runtime security are cloud detection and response (CDR) and application detection and response (ADR).

Cloud and On-prem Observability

Observability has long depended on agents, sidecars, and SDKs with each one sampling metrics, logs, and traces from different parts of the stack. But stitching these perspectives together has always meant gaps, latency, and overhead. **eBPF collapses that complexity by operating from a single vantage point in the kernel.** Some of this richer kernel-level data includes distributed traces, service mappings, protocol-specific usage such as HTTP, gRPC calls, SQL queries, Kafka, system calls, network interface events, and performance data **all without modifying a single line of code.**

For organizations with an on-premises footprint, eBPF also provides hardware-level observability, such as visibility over memory access patterns, disk I/O performance, and network interface behavior. This kernel-level visibility helps identify hardware contention, resource misallocation, and performance degradation across physical servers, storage devices, and network adapters. This is

particularly valuable for optimizing on-premises data centers where operators must understand the relationship between software behavior and underlying physical hardware.

For hybrid environments, this model is particularly powerful. Whether workloads run on-premises or across multiple clouds, **eBPF delivers consistent telemetry from the same kernel hooks, eliminating the blind spots** that arise from mixing vendor agents and hypervisor-based monitoring. The result is a unified, event-driven view of the entire system.

FinOps

To calculate cloud costs without relying only on data exposed by hyperscalers, eBPF probes are used to monitor system calls, payload level network traffic, and application behavior in real-time. eBPF can help determine how cloud services are used and tie this back to kernel events.

By observing CPU scheduling, network paths, and I/O patterns in real time, **eBPF makes invisible inefficiencies visible.** This lower-level data can also differentiate between tenants on shared services or multi tenant environments. It can also mean that costs can be attributed to each cloud service such as managed databases, object storage, or egress costs.

LLM Security and Observability

Protecting GenAI applications, specifically those that use LLMs, is a new challenge for security architects. Most tools designed to address LLM security issues have **directly opted for eBPF to trace the interactions** between applications and external large language models.

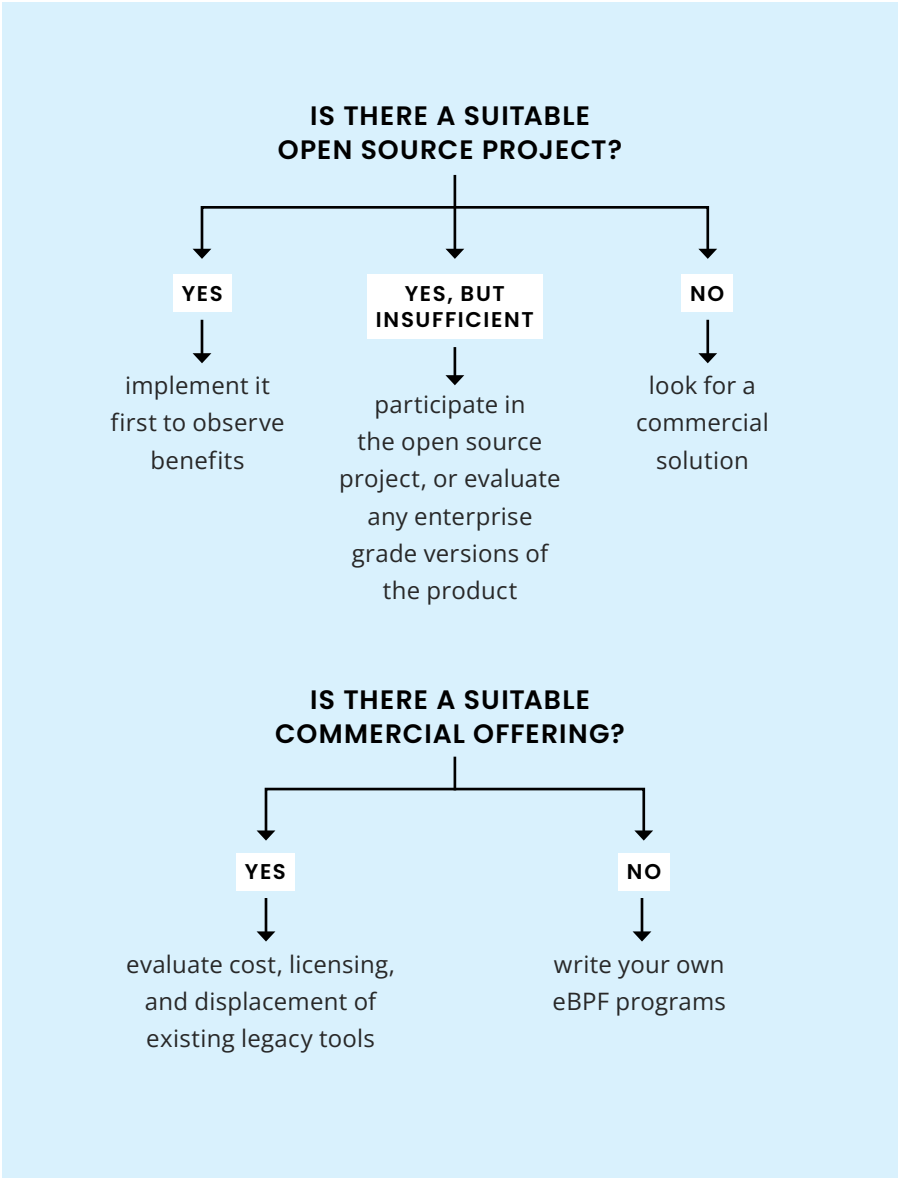
By instrumenting eBPF at the network layer, these solutions monitor and log the API calls made to these services, capturing information such as request and response data, latency, and error rates.

How to Implement eBPF in Your Infrastructure

Organizations can currently choose from a wide range of eBPF based solutions to optimize and secure their applications and infrastructure. The **three main options are to leverage open source projects, adopt enterprise-grade distributions, or develop custom eBPF programs** tailored to their unique needs.

First they can choose from a dynamic open source ecosystem, which offers organizations a **low-risk, low-cost opportunity** to explore and build their eBPF footprint. Some open source projects also have commercial versions, often provided by the creators and maintainers of the OSS tools. Finally, organizations can also write and run their own eBPF programs if their **requirements are beyond the scope of the current tools,** or if they have specific requirements around regulations and data processing.

Working with the assumption that eBPF is the suitable solution, organizations should evaluate their implementations of eBPF from low-effort, low-cost first as follows:



Open Source Projects

The eBPF ecosystem has been consistently growing over the past ten years. Organizations can now leverage a wide range of open source tools, a survey of which you can see on **ebpf.io**, and typically include use cases such as:

- **Networking solutions -** Load balancers, container networking
- **Security tools -** Runtime security, intrusion detection
- **Observability platforms -** Monitoring, tracing, and profiling applications

Some of the most widely adopted projects that benefit from community development include Cilium for networking, its sub-project Tetragon for runtime security, and OpenTelemetry for observability.

Cilium is one of the first open source eBPF-based projects, and provides networking, security and observability for containerized environments. Cilium follows the Container Networking Interface (CNI) specification and has been specifically designed from the ground up to bring the advantages of eBPF to Kubernetes and to address the new scalability, security, and visibility requirements of container workloads.

Tetragon is a sub-project of Cilium that uses eBPF for transparent security observability combined with real-time runtime enforcement. The embedded runtime enforcement layer is capable of performing access control on kernel functions, system calls, and at other enforcement mechanisms.

OpenTelemetry is a collection of APIs, SDKs, and tools used to instrument, generate, collect, and export telemetry data. The Open Telemetry eBPF-based continuous profiler offers comprehensive, low-overhead whole-system profiling for Linux systems. It supports a wide range of programming languages, including native code without debug symbols, and provides deep insights into application behavior.

Enterprise-Grade Versions of Open Source Projects

Some technology vendors, usually the creators and maintainers of OSS projects, also offer an enterprise-grade version of the open source tools. **These typically entail additional features, enterprise-grade support, and graphical user interfaces.**

Isovalent, now part of Cisco, are the creators of Cilium and Tetragon. Organizations can opt for the Isovalent Enterprise Platform as an enterprise-grade, **hardened distribution of the open source projects** Cilium, Hubble, and Tetragon. Isovalent Enterprise Networking for Kubernetes provides advanced network policy capabilities, including DNS-aware policy, L7 policy, and deny policy, enabling fine-grained control over network traffic for micro-segmentation and improved security.

For those who want to use OpenTelemetry at scale and with support, Splunk, a major contributor to the OpenTelemetry project, offers a **preconfigured and fully supported distribution** of the OpenTelemetry Collector. It supports trace instrumentation with no code modification and comes with default configuration and out-of-the-box support for Splunk Application Performance Monitoring and Splunk Infrastructure Monitoring.

Commercial eBPF-Based Products

Many technology vendors across security and observability are **now using eBPF to develop more sophisticated features.** For example, Datadog's Universal Service Monitoring uses eBPF to automatically discover, map, and monitor services and dependencies without requiring customers to instrument any code. Another example is Cloud Network Monitoring, which

uses eBPF to provide detailed but lightweight visibility into the network traffic. Similarly, CrowdStrike's newly announced Runtime Cloud Data Protection uses eBPF to detect and block unauthorized data movements in real time.

Internal eBPF-Based Tools

Some organizations, especially hyperscalers, are developing in-house eBPF-based tools to address use cases specific to their infrastructure. **Meta, for example, developed an eBPF-based profiling orchestrator** that collects observability data out-of-process, including CPU time spent in function calls and execution paths, call stacks for native and non-native languages, off-CPU time and service request latency analysis, and AI/GPU profiling and memory tracking.

The tool provides low-overhead data collection, avoiding additional instrumentation inside binaries and maintaining efficient performance. Following the deployment, Meta was able to achieve a **20% reduction in CPU cycles, equating to a 10-20% reduction in the number of required servers for Meta's top services.**

Netflix has also developed a command-line tool designed to streamline the performance optimization and monitoring of eBPF programs. Netflix's eBPF footprint continuously increases, so they have created **bpftop** to apply the same rigor to these applications as with the rest of the tech stack. bpftop provides a dynamic real-time view of running eBPF programs. It displays the average execution runtime, events per second, and estimated total CPU % for each program.

Who Can Write eBPF-based Programs?

Anyone can develop their proprietary or internal eBPF programs. Even though the eBPF open source community and the comprehensive landscape of commercial products cover a wide range use cases today, organizations may choose to write their own programs for custom logic or infrastructure-specific use cases.

Writing eBPF-based programs has some prerequisites, including:

- **Developers and Skills:** developers need knowledge in C programming, Linux kernel internals, and system-level programming. They need understanding of kernel data structures, and system call interfaces. There is also a learning curve associated with eBPF itself, such as learning about helper functions, memory access patterns that satisfy the verifier's safety checks, and debugging techniques for kernel-space code.
- **Tech stack prerequisites:** writing eBPF needs modern Linux environments with newer kernel version for optimal eBPF support, along with **LLVM**/Clang compiler toolchains, and development libraries. For compatibility across kernel versions the kernel must support BPF Type Format for compile once-run everywhere (CO-RE). A range of infrastructure projects on **ebpf.io** can help organizations in their efforts to write their own eBPF programs, including CO-RE eBPF libraries and frameworks, development tools, compilers, debuggers, and development environments for eBPF

Given the right use cases and ability to develop and manage the tools in-house, eBPF can be a very strong advantage for managing infrastructure at scale.

eBPF as the Platform of Choice for Infrastructure Teams

As eBPF adoption increases and today's use-case specific tools mature into more comprehensive platforms, eBPF will increasingly become the infrastructure building block of choice across hyperscalers, cloud native organizations, service providers, and even SMBs.

Developer Tools Will Lower the Barrier to Entry

Historically, eBPF development required deep knowledge of C, kernel internals, and system-level programming. Today, with a growing ecosystem of tools and frameworks and more specialized tools that abstract some of the lower-level complexities associated with kernel programming is making kernel-level programmability accessible to a broader audience. These include:

- New developer-friendly SDKs with frameworks in additional languages like Python, Go, and Rust, make it easier to interact with kernel data structures and events without deep C expertise.
- IDE integrations and debugging tools, such as code completion extensions that offer VS Code autocomplete functionality for common eBPF patterns.
- Standardized APIs such as memory monitoring libraries, event handling, and process monitoring REST services.
- Testing frameworks, such as memory profiling test suites for reproducible out-of-memory scenarios with automated test binaries for validation.

Providing High Fidelity Data for AI Agents

To help with inference, large language models require semantically meaningful data. By collecting kernel-level telemetry covering system calls, network activity, resource utilization, and application behavior, eBPF enables organizations to create a **unified, interpretable dataset for AI agents.** This approach ensures that LLMs and other AI systems can access reliable, real-time metrics, enabling better inference, monitoring, decision-making, and adaptation to changing workload requirements.

AI Workload Optimization

Large-scale AI workloads expose every inefficiency in compute, networking, and storage. With considerations such as ten thousand node clusters, GPU synchronicity, and elephant flows, every component has compounding effects, and any **marginal gains can translate into considerable savings across vast datasets.** eBPF's ability to optimize CPU process scheduling, lower network latencies, and provide deep kernel visibility becomes essential for minimizing training times and maximizing resource utilization.

The Building Block for Infrastructure Platforms

The eBPF ecosystem began with single purpose tools and programs designed to solve specific problems at hyperscale companies. The lessons learned from their early experimentation on kernel programmability have gone on to shape the foundations that the ecosystem rests upon. **These building blocks have become the core of the projects and products that leverage eBPF and represent the next generation of infrastructure platforms.** They can be combined, extended, and reused to create modular, extensible platforms capable of adapting in real time to the demands of cloud native, hybrid, and multi-cloud environments.

We can look at the evolution of Cilium as an example of how an eBPF project can become a platform. Cilium was initially created as an IPv6-only container networking project. Today, the project seamlessly connects workloads and infrastructure across Kubernetes, cloud, data centers, and on-premises deployments. In addition, with the insight that eBPF offers, it also offers network observability and runtime security.

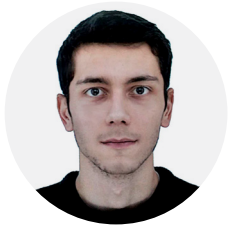
This trajectory is expected from a wider range of projects and products. Networking, security, observability, and even how the kernel itself functions are all interrelated thus the building blocks used for one, can also carry over into the next category. **Consumers of eBPF based products should think about buying into an eBPF-based platform rather than looking at individual tools.**

eBPF Resources for All

For those who want to know more about eBPF, there are plenty of resources for all learning styles, such as for those who:

- Learn by doing - a wide range of well-documented open source projects **are available here**
- Learn through formal pathways - **here are a wide range of interactive labs**.
- Learn by reading - Comprehensive O'Reilly books such as **Learning eBPF** and **Security Observability with eBPF**
- Learn from others' experience - **eBPF case studies** published by the eBPF Foundation
- Learn by watching - **the eBPF documentary, Unlocking the Kernel**

The **eBPF Foundation** brings together a cross-platform community of eBPF-related projects from across the open source ecosystem in an independent forum. The foundation provides a forum to collaborate on a common technical vision, vocabulary, security best practices, and general roadmap, to be applied within separate workstreams, for example Kubernetes, operating system kernels, and enterprise communities.



About the Author

Andrew Green is an enterprise IT research analyst and writer covering security, networking, and infrastructure. Lately, he's been deciphering the AI market and its impact on enterprise products. Andrew has been a GigaOm analyst since 2020. He also works with technology vendors to produce technical content, competitive analysis, and market landscape assessments, and shares independent and non-sponsored content on Substack and LinkedIn.