Pagina inicial

Programe dinamicamente o kernel para rede, observabilidade, rastreamento e segurança eficientes.

Os programas são verificados para serem executados com segurança Conecte-se em qualquer lugar do kernel para modificar a funcionalidade Compilador JIT para velocidade de execução quase nativa Adicione recursos do sistema operacional em tempo de execução Organizações de todos os setores usam eBPF na produção.

Google

O Google usa eBPF para auditoria de segurança, processamento de pacotes e monitoramento de desempenho.

Netflix

A Netflix usa eBPF em escala para obter insights de rede.

Android

O Android usa eBPF para monitorar o uso da rede, energia e perfil de memória.

S&P Global

S&P Global usa eBPF por meio do Cilium para rede em múltiplas nuvens e no local.

Shopify

Shopify usa eBPF por meio do Falco para deteção de intrusões.

Cloudflare

Cloudflare usa eBPF para segurança de rede, monitoramento de desempenho e observabilidade de rede.

Mais estudos de caso

Aqui estão algumas das organizações que estão usando o eBPF na produção.

Se você estiver usando eBPF e não estiver nesta lista, envie uma solicitação pull.

Por que ebpf?

O que é ebpf

desempenho

O eBPF melhora drasticamente o processamento ao ser compilado JIT e executado diretamente no kernel.

Segurança

programas eBPF são verificados para não travar o kernel e só podem ser modificados por usuários privilegiados.

Flexibilidade

Modifique ou adicione funcionalidades e casos de uso ao kernel sem precisar reiniciá-lo ou corrigi-lo.

O eBPF resultou em uma nova geração de ferramentas que permite aos desenvolvedores diagnosticar problemas facilmente, inovar rapidamente e ampliar a funcionalidade do sistema operacional.

eBPF é o futuro da rede para o kernel Linux e o Google tem o prazer de fazer parte do padrão em evolução que criou. Embora o eBPF já tenha entrado nas pilhas de produção de inúmeras empresas, ainda estamos no início da curva de inovação que o eBPF como tecnologia desbloqueia.

Por muitos anos, o eBPF desempenhou um papel crítico na aceleração do desenvolvimento do kernel. Estamos entusiasmados em apoiar o trabalho da comunidade eBPF, permitindo-lhes construir as ferramentas necessárias para impulsionar a próxima geração de desenvolvimento de sistemas Linux.

eBPF é um novo tipo de software que fornece recursos super poderosos, dando origem a uma indústria de tecnologias de rede, desempenho e segurança.

A Netflix foi pioneira no uso do eBPF para observabilidade, fornecendo informações sobre inúmeras áreas que antes eram difíceis ou proibitivamente caras de instrumentar.

O que é possível com eBPF?

Rede

Acelere o processamento de pacotes sem deixar espaço no kernel. Adicione analisadores de protocolo adicionais e programe facilmente qualquer lógica de encaminhamento para atender aos requisitos em constante mudança.

Observabilidade

Coleta e agregação no kernel de métricas personalizadas com geração de eventos de visibilidade e estruturas de dados de uma ampla variedade de fontes possíveis, sem a necessidade de exportar amostras.

Rastreamento e perfil

Anexe programas eBPF a pontos de rastreamento, bem como a pontos de investigação de aplicativos de usuário e de kernel, fornecendo poderosas habilidades de introspeção e insights exclusivos para solucionar problemas de desempenho do sistema.

Segurança

Combine a visualização e a compreensão de todas as chamadas do sistema com uma visão em nível de pacote e soquete de todas as redes para criar sistemas de segurança operando em mais contexto com um melhor nível de controle.

Palestras comunitárias ebpf.

O que é eBPF?

eBPF é uma tecnologia revolucionária com origem no kernel Linux que pode executar programas em sandbox num contexto privilegiado, como o kernel do sistema operacional.

Ele é usado para estender com segurança e eficiência os recursos do kernel sem a necessidade de alterar o código-fonte do kernel ou carregar módulos do kernel.

Historicamente, o sistema operacional sempre foi um local ideal para implementar observabilidade, segurança e funcionalidade de rede devido à capacidade privilegiada do kernel de supervisionar e controlar todo o sistema.

Ao mesmo tempo, o kernel de um sistema operacional é difícil de evoluir devido ao seu papel central e aos altos requisitos de estabilidade e segurança.

A taxa de inovação no nível do sistema operacional tem sido tradicionalmente menor em comparação com a funcionalidade implementada fora do sistema operacional.

Visão geral

eBPF muda esta fórmula fundamentalmente. Ao permitir a execução de programas em sandbox no sistema operacional, os desenvolvedores de aplicativos podem executar programas eBPF para adicionar recursos adicionais ao sistema operacional em tempo de execução.

O sistema operacional garante então segurança e eficiência de execução como se fosse compilado nativamente com o auxílio de um compilador Just-In-Time (JIT) e mecanismo de verificação.

Isto levou a uma onda de projetos baseados em eBPF cobrindo uma ampla gama de casos de uso, incluindo redes de próxima geração, observabilidade e funcionalidade de segurança.

Hoje, o eBPF é usado extensivamente para conduzir uma ampla variedade de casos de uso: fornecer redes de alto desempenho e balanceamento de carga em data centro modernos e ambientes nativos de nuvem, extrair dados de observabilidade de segurança refinados com baixa sobrecarga, ajudar os desenvolvedores de aplicativos a rastrear aplicativos, fornecendo insights para solução de problemas de desempenho, aplicação preventiva de segurança em tempo de execução de aplicativos e contêineres e muito mais.

As possibilidades são infinitas e a inovação que o eBPF está a desbloquear apenas começou.

O que é eBPF.io?

eBPF.io é um lugar para todos aprenderem e colaborarem no tema eBPF. eBPF é uma comunidade aberta e todos podem participar e compartilhar.

Se você deseja ler uma primeira introdução ao eBPF, encontrar mais material de leitura ou dar os primeiros passos para se tornar contribuidor de grandes projetos eBPF, o eBPF.io irá ajudá-lo ao longo do caminho.

O que significam eBPF e BPF?

BPF originalmente significava Berkeley Packet Filtre, mas agora que eBPF (BPF estendido) pode fazer muito mais do que filtragem de pacotes, a sigla não faz mais sentido. eBPF agora é considerado um termo independente que não significa nada. No código-fonte do Linux, o termo BPF persiste, e nas ferramentas e na documentação, os termos BPF e eBPF são geralmente usados ​​de forma intercambiável.

Qual é o nome da abelha?

A abelha é o logótipo oficial do eBPF e foi originalmente criada por Vadim Shchekoldin. No primeiro Summit da eBPF houve votação e a abelha foi chamada de eBee. ( Para obter detalhes sobre usos aceitáveis ​​do logótipo, consulte as Diretrizes da Marca da Linux Foundation )

Introdução ao eBPF

Os capítulos seguintes são uma rápida introdução ao eBPF. Se você quiser saber mais sobre o eBPF, consulte o Guia de referência do eBPF e XDP.

Quer você seja um desenvolvedor que deseja construir um programa eBPF ou esteja interessado em aproveitar uma solução que usa eBPF, é útil compreender os conceitos básicos e a arquitetura.

Visão geral do gancho

Os programas eBPF são orientados a eventos e executados quando o kernel ou um aplicativo passa por um determinado ponto de gancho.

Ganchos predefinidos incluem chamadas de sistema, entrada/saída de função, pontos de rastreamento de kernel, eventos de rede e vários outros.

Gancho Syscall

Se não existir um gancho predefinido para uma necessidade específica, é possível criar um teste de kernel (kprobe) ou um teste de usuário (uprobe) para anexar programas eBPF em quase qualquer lugar no kernel ou em aplicativos de usuário.

Visão geral do gancho

Como os programas eBPF são escritos? Em muitos cenários, o eBPF não é usado direta, mas indiretamente por meio de projetos como Cilium, bcc ou bpftrace, que fornecem uma abstração sobre o eBPF e não exigem a escrita direta de programas, mas oferecem a capacidade de especificar definições baseadas em intenções que são então implementado com eBPF.

Clang

Se não existir nenhuma abstração de nível superior, os programas precisam ser escritos diretamente. O kernel Linux espera que os programas eBPF sejam carregados na forma de bytecode. Embora seja possível escrever bytecode diretamente, a prática de desenvolvimento mais comum é aproveitar um conjunto de compiladores como o LLVM para compilar código pseudo-C em bytecode eBPF.

Arquitetura de carregador e verificação

Quando o gancho desejado for identificado, o programa eBPF pode ser carregado no kernel Linux usando a chamada de sistema bpf. Isso normalmente é feito usando uma das bibliotecas eBPF disponíveis.

A próxima secção fornece uma introdução às cadeias de ferramentas de desenvolvimento disponíveis.

Go

À medida que o programa é carregado no kernel do Linux, ele passa por duas etapas antes de ser anexado ao gancho solicitado:

Verificação

A etapa de verificação garante que o programa eBPF seja seguro para execução.

Valida que o programa atende a diversas condições, por exemplo:

Carregador

O processo que carrega o programa eBPF contém os recursos necessários (privilégios). A menos que o eBPF sem privilégios esteja habilitado, apenas processos privilegiados podem carregar programas eBPF. O programa não trava nem prejudica o sistema. O programa sempre é executado até a conclusão (i.e. o programa não fica parado em loop para sempre, atrasando o processamento adicional )

Compilação JIT

A etapa de compilação Just-in-Time (JIT) traduz o bytecode genérico do programa no conjunto de instruções específico da máquina para otimizar a velocidade de execução do programa. Isso faz com que os programas eBPF sejam executados com a mesma eficiência que o código do kernel compilado nativamente ou como o código carregado como um módulo do kernel.

Mapas

Um aspecto vital dos programas eBPF é a capacidade de compartilhar informações coletadas e armazenar o estado. Para este propósito, os programas eBPF podem aproveitar o conceito de mapas eBPF para armazenar e recuperar dados num amplo conjunto de estruturas de dados. Os mapas eBPF podem ser acessados ​​a partir de programas eBPF, bem como de aplicativos no espaço do usuário por meio de uma chamada de sistema.

Arquitetura do mapa

A seguir está uma lista incompleta de tipos de mapas suportados para dar uma compreensão da diversidade nas estruturas de dados. Para vários tipos de mapas, estão disponíveis variações compartilhadas e por CPU.

Tabelas hash,

matrizes

LRU (menos usado recentemente)

Buffer de anel

Rastreamento de pilha

LPM (correspondência de prefixo mais longa)

Chamadas de ajuda

Os programas eBPF não podem chamar funções arbitrárias do kernel. Permitir isso vincularia os programas eBPF a versões específicas do kernel e complicaria a compatibilidade dos programas.

Em vez disso, os programas eBPF podem fazer chamadas de função para funções auxiliares, uma API bem conhecida e estável oferecida pelo kernel.

Ajudante:

O conjunto de chamadas de ajudantes disponíveis está em constante evolução.

Exemplos de chamadas de ajuda disponíveis:

Gerar números aleatórios

Obtenha a hora & a data atuais

acesso ao mapa eBPF

Obtenha o contexto do processo/cgroup

Manipule pacotes de rede e lógica de encaminhamento

Cauda e de função Chamadas

Os programas eBPF podem ser combinados com o conceito de cauda e função chamadas. As chamadas de função permitem definir e chamar funções dentro de um programa eBPF.

As chamadas Cauda podem chamar e executar outro programa eBPF e substituir o contexto de execução, semelhante à forma como a chamada do sistema execve() opera para processos regulares.

Cauda Chamadas

Segurança eBPF

Com grande poder também vem haver uma grande responsabilidade.

eBPF é uma tecnologia incrivelmente poderosa e agora funciona no centro de muitos componentes críticos da infraestrutura de software.

Privilégios necessários

A menos que o eBPF sem privilégios esteja habilitado, todos os processos que pretendem carregar programas eBPF no kernel do Linux devem estar rodando em modo privilegiado (root) ou exigir o recurso CAP\_BPF.

Isto significa que programas não confiáveis ​​não podem carregar programas eBPF. Se o eBPF sem privilégios estiver habilitado, processos sem privilégios podem carregar determinados programas eBPF sujeitos a um conjunto de funcionalidades reduzido e com acesso limitado ao kernel.

Verificador

Se um processo tiver permissão para carregar um programa eBPF, todos os programas ainda passarão pelo verificador eBPF. O verificador eBPF garante a segurança do próprio programa.

Isto significa, por exemplo:

Os programas são validados para garantir que sempre sejam executados até a conclusão, por ex. um programa eBPF nunca pode bloquear ou ficar em loop para sempre.

Os programas eBPF podem conter os chamados loops limitados, mas o programa só será aceite se o verificador puder garantir que o loop contém uma condição de saída que certamente se tornará verdadeira.

Os programas não podem usar variáveis ​​não inicializadas ou acessar a memória fora dos limites. Os programas devem atender aos requisitos de tamanho do sistema. Não é possível carregar programas eBPF arbitrariamente grandes.

O programa deve ter uma complexidade finita. O verificador avaliará todos os caminhos de execução possíveis e deverá ser capaz de concluir a análise nos limites do limite superior de complexidade configurado.

O verificador é uma ferramenta de segurança, verificando se os programas são seguros para execução. Não é uma ferramenta de segurança que inspeciona o que os programas estão fazendo.

Endurecimento

Após a conclusão bem-sucedida da verificação, o programa eBPF passa por um processo de endurecimento, dependendo se o programa é carregado a partir de um processo privilegiado ou não. Esta etapa inclui:

Proteção de execução do programa: A memória do kernel que contém um programa eBPF é protegida e tornada somente leitura. Se por qualquer motivo, seja um bug do kernel ou manipulação maliciosa, o programa eBPF for tentado ser modificado, o kernel irá travar em vez de permitir que ele continue executando o programa corrompido/manipulado.

Mitigação contra Spectre: Sob especulação, as CPUs podem prever mal as ramificações e deixar efeitos colaterais observáveis ​​que podem ser extraídos através de um canal lateral.

Para citar alguns exemplos: os programas eBPF mascaram o acesso à memória para redirecionar o acesso sob instruções transitórias para áreas controladas, o verificador também segue caminhos de programa acessíveis apenas sob execução especulativa e o compilador JIT emite Retpolines caso as chamadas finais não possam ser convertidas em chamadas diretas.

Cegueira constante: Todas as constantes no código são ocultadas para evitar ataques de pulverização JIT. Isso evita que invasores injetem código executável como constantes que, na presença de outro bug do kernel, poderiam permitir que um invasor saltasse para a secção de memória do programa eBPF para executar código.

Contexto de tempo de execução abstrato

Os programas eBPF não podem acessar diretamente a memória arbitrária do kernel.

O acesso a dados e estruturas de dados que estão fora do contexto do programa deve ser acessado por meio de auxiliares eBPF.

Isto garante um acesso consistente aos dados e torna qualquer acesso sujeito aos privilégios do programa eBPF, por ex.

um programa eBPF em execução pode modificar os dados de certas estruturas de dados se a modificação puder ser garantida como segura. Um programa eBPF não pode modificar aleatoriamente estruturas de dados no kernel.

Por que eBPF?

O poder da programabilidade

Vamos começar com uma analogia. Você lembra-se do GeoCidades? Há 20 anos, as páginas da web eram quase exclusivamente escritas em linguagem de marcação estática (HTML).

Uma página web era basicamente um documento com um aplicativo (navegador) capaz de exibi-lo. Olhando para as páginas da web hoje, as páginas da web tornaram-se aplicativos completos e a tecnologia baseada na web substituiu a grande maioria dos aplicativos escritos em linguagens que exigem compilação.

O que permitiu esta evolução?

Geocidades

A resposta curta é programabilidade com a introdução do JavaScript. Ele desbloqueou uma revolução massiva que resultou na evolução dos navegadores para sistemas operacionais quase independentes.

Por que a evolução aconteceu?

Os programadores não estavam mais tão vinculados aos usuários que executavam versões específicas de navegadores. Em vez de convencer os organismos de normalização de que era necessária uma nova etiqueta HTML, a disponibilidade dos blocos de construção necessários dissociou o ritmo de inovação do navegador subjacente da aplicação executada no topo.

É claro que isso é um pouco simplificado, pois o HTML evoluiu ao longo do tempo e contribuiu para o sucesso, mas a evolução do HTML em si não teria sido suficiente.

Antes de pegar este exemplo e aplicá-lo ao eBPF, vejamos alguns aspetos-chave que foram vitais na introdução do JavaScript:

Segurança: código não confiável é executado no navegador do usuário. Isso foi resolvido colocando programas JavaScript em sandbox e abstraindo o acesso aos dados do navegador.

Entrega Contínua: A evolução da lógica do programa deve ser possível sem a necessidade de enviar constantemente novas versões de navegadores. Isso foi resolvido fornecendo os blocos de construção corretos de baixo nível, suficientes para construir lógica arbitrária.

Desempenho: A programabilidade deve ser fornecida com sobrecarga mínima. Isso foi resolvido com a introdução de um compilador Just-in-Time (JIT). Para todos os itens acima, contrapartidas exatas podem ser encontradas no eBPF pelo mesmo motivo.

O impacto do eBPF no kernel Linux

Agora vamos voltar ao eBPF. Para compreender o impacto da programabilidade do eBPF no kernel Linux, é útil ter um entendimento de alto nível da arquitetura do kernel Linux e como ele interage com os aplicativos e o hardware.

Arquitetura do kernel

O principal objetivo do kernel Linux é abstrair o hardware ou hardware virtual e fornecer uma API consistente (chamadas de sistema) permitindo que aplicativos sejam executados e compartilhem os recursos.

Para conseguir isso, um amplo conjunto de subsistemas e camadas é mantido para distribuir essas responsabilidades. Cada subsistema normalmente permite algum nível de configuração para atender às diferentes necessidades dos usuários.

Se um comportamento desejado não puder ser configurado, uma mudança no kernel será necessária, historicamente, deixando duas opções:

Suporte nativo

Altere o código-fonte do kernel e convença a comunidade do kernel Linux de que a alteração é necessária. Espere vários anos para que a nova versão do kernel se torne uma mercadoria.

Módulo do Kernel

Escreva um módulo do kernel

Corrija-o regularmente, pois cada versão do kernel pode quebrá-lo

Risco de corromper o seu kernel Linux devido à falta de limites de segurança

Com o eBPF, está disponível uma nova opção que permite reprogramar o comportamento do kernel Linux sem exigir alterações no código-fonte do kernel ou carregar um módulo do kernel.

Em muitos aspetos, isso é muito semelhante ao modo como o JavaScript e outras linguagens de script desbloquearam a evolução de sistemas que se tornaram difíceis ou caros de mudar.

Cadeias de ferramentas de desenvolvimento

Existem diversas cadeias de ferramentas de desenvolvimento para auxiliar no desenvolvimento e gerenciamento de programas eBPF. Todos eles atendem a diferentes necessidades dos usuários:

BCC

BCC é uma estrutura que permite aos usuários escrever programas python com programas eBPF incorporados neles.

A estrutura é direcionada principalmente para casos de uso que envolvem criação de perfil/rastreamento de aplicativos e sistemas, onde um programa eBPF é usado para coletar estatísticas ou gerar eventos e uma contraparte no espaço do usuário coleta os dados e os exibe num formato legível por humanos.

A execução do programa python irá gerar o bytecode eBPF e carregá-lo no kernel.

Bcc Bpftrace

bpftrace é uma linguagem de rastreamento de alto nível para Linux eBPF e está disponível em kernels Linux semi-recentes (4.x).

bpftrace usa LLVM como back-end para compilar scripts para bytecode eBPF e faz uso de BCC para interagir com o subsistema Linux eBPF, bem como recursos de rastreamento Linux existentes: rastreamento dinâmico de kernel (kprobes), rastreamento dinâmico de nível de usuário (uprobes) e pontos de rastreamento.

A linguagem bpftrace é inspirada em awk, C e rastreadores predecessores, como DTrace e SystemTap.

Bpftrace

eBPF Go Biblioteca

A biblioteca eBPF Go fornece uma biblioteca eBPF genérica que separa o processo de obtenção do bytecode eBPF e o carregamento e gerenciamento de programas eBPF.

Os programas eBPF são normalmente criados escrevendo uma linguagem de nível superior e, em seguida, usam o compilador clang/LLVM para compilar para o bytecode eBPF.

libbpf C/C++ Biblioteca

A biblioteca libbpf é uma biblioteca eBPF genérica baseada em C/C++ que ajuda a descolar o carregamento de arquivos de objeto eBPF gerados a partir do compilador clang/LLVM no kernel e geralmente abstrai a interação com a chamada do sistema BPF, fornecendo APIs de biblioteca fáceis de usar para aplicativos.

Libbpf

Leitura adicional

Se você quiser saber mais sobre o eBPF, continue lendo usando os seguintes materiais adicionais:

**Pegar Iniciar**

Pegar Iniciar com eBPF

Experimente o laboratório

Tomando emprestado o exemplo do opensnoop do relatório de Liz Rice, este laboratório ensina você a lidar com uma ferramenta eBPF, observá-la carregando os seus componentes e até mesmo adicionar o seu próprio rastreamento ao código-fonte do eBPF.

Leia os livros

Leia o artigo “O que é eBPF?” e “aprendizado eBPF” O’Reilly Books de Liz Rice ou BPF Performance ferramenta de Brendan Gregg para começar. Ao longo dos livros, você aprenderá o que é eBPF e por que ele é tão poderoso, os recursos que ele oferece.

Assista o vídeo

Mergulhe na história do eBPF com esta palestra de John Fastabend. Começando nos “primeiros dias” de 2014, abrange os principais projetos, empresas e players que influenciaram o cenário das redes Linux da época e como eles possibilitaram a criação do eBPF.

Leia a documentação para saber mais sobre

o eBPF Wikipédia

Artigo da Wikipédia sobre eBPF

Stackoverflow eBPF

Faça perguntas e leia as respostas

Comunidade eBPF no Reddit

Discuta o subsistema do kernel

eBPF Guia de referência BPF e XDP

Aprenda os detalhes internos e a programação do BPF

Documentação BPF

Documentação BPF no Kernel Linux

Q&A design BPF

FAQ para kernel relacionadas perguntas

Subscrever o eCHO Notícias quinzenal.

Fique por dentro das últimas notícias e informações do eBPF e Cilium

Se inscrever

Explore livros eBPF de especialistas do setor

Aprofunde-se na tecnologia eBPF com tutoriais de especialistas do setor

Tutoriais de rastreamento eBPF

Aprenda como usar o eBPF para rastreamento,

desde a simples execução de ferramentas Bcc até o desenvolvimento com bpftrace e Bcc.

Tutoriais XDP

Aprenda as etapas básicas necessárias para escrever programas para o XDP de maneira eficaz, desde o que significa anexar vários programas a uma interface.

Explorador do compilador

Suporte BPF para executar o compilador interativamente no seu navegador da web e explorar o assembly

Vídeos introdutórios do eBPF

Eventos

Conferências e encontros eBPF

Cabeçalho

Aprender

O que é eBPF

Iniciar

Estudos de caso

Paisagem do Projeto

Aplicativo infraestrutura

Eventos

Todos os eventos

Comunidade

Boletim de Notícias

Contribuir

Blogue

Fundação

Rodapé

Sobre

Loja de brindes

Marca

nota de rodapé

Sobre

Loja de brindes

Marca

Mantido pela comunidade eBPF.

Viu um bug? Registrar um problema

© 2023 autores eBPF.io

O conteúdo do site ebpf.io está licenciado sob uma Licença Creative Commons Atribuição 4.0 Internacional.

SEO

Título: 'eBPF - Introdução, tutoriais e recursos comunitários',

descrição:

‘eBPF é uma tecnologia revolucionária que pode executar programas em sandbox no kernel Linux sem alterar o código-fonte do kernel ou carregar um módulo do kernel.’

Contribuir: {

pt: {

titulo: ‘Como contribuir com o eBPF’

descrição: ‘Quer saber como contribuir com o eBPF? Aprenda como começar a contribuir para projetos eBPF’

palavras-chave: 'ebpf, bpf, contribuir, primeiros passos, comunidade

estudos de caso: {

pt: {

título: { 'eBPF estudos de caso’

descrição: ‘Saber como o eBPF é utilizado em diferentes projetos’

palavras-chave: 'ebpf, bpf, contribuir, primeiros passos, comunidade'

aplicação: {

título: ‘Panorama de aplicações eBPF’

descrição: ‘Um diretório de infraestrutura central baseada em eBPF'

palavras-chave: 'ebpf, bpf, paisagem, diretório, código-fonte',

a infraestrutura: {

titulo: 'Panorama de infraestrutura central eBPF',

descrição: 'Um diretório de infraestrutura central baseada em eBPF',

palavras-chave: 'ebpf, bpf, paisagem, diretório, código-fonte',

Boletim de Notícias: {

título: 'Subscrever o eCHO Notícias quinzenal - eBPF',

descrição: 'Checar as últimas notícias e informações das comunidades eBPF e Cilium',

blog: {

título: 'eBPF - Blog',

description: 'As últimas notícias, atualizações e artigos sobre eBPF e tópicos relacionados.'

Estudos de caso

Estudos de Caso eBPF

Aqui estão algumas das organizações que estão usando o eBPF na produção. Se você estiver usando eBPF e não estiver nesta lista, envie uma solicitação pull.

Laboratórios

Explore o eBPF com laboratórios interativos

Aplicação

Principais aplicações

Emergindo

A infraestrutura

Infraestrutura principais

Emergindo

Bibliotecas eBPF

Bibliotecas Auxiliares eBPF

perguntas frequentes

Esses projetos são da Fundação eBPF?

Esta página lista vários projetos de código aberto que usam eBPF como tecnologia central subjacente. Esses projetos não estão todos sob a Fundação eBPF, mas estão listados aqui como uma pesquisa do panorama dos projetos eBPF hoje.

Adicione seu projeto

Certifique-se de que o projeto atenda aos requisitos a serem listados. Veja abaixo. Abra uma solicitação pull e forneça as informações necessárias. Use um dos projetos já listados como modelo. A ordenação dos aplicativos é baseada no número de estrelas do Github (maior para menor), atualizado trimestralmente.

A solicitação pull será revisada pela comunidade e mesclada por um dos mantenedores. Se você tiver alguma dúvida, fique à vontade para perguntar no Slack.

Você está mantendo um projeto listado?

Se você mantém um dos projetos listados e gostaria de ajustar o conteúdo. Entre em contacto no Slack ou abra uma solicitação pull diretamente. Requisitos para um projeto ser listado Os projetos podem ser listados nesta página como "Principais" ou "Emergentes". Os requisitos para ser listado como "Emergente" são:

O projeto deve ser de código aberto. Todo o código-fonte deve ser licenciado sob uma licença de código aberto. Qualquer documentação deve ser licenciada sob uma licença aberta. O projeto deve usar o eBPF como tecnologia central subjacente (em outras palavras, um projeto perderia o seu propósito se as partes do eBPF fossem removidas) ou ajudar a acelerar a adoção do eBPF na produção.

O projeto deve ser mantido ativamente.

O projeto deve estar aberto à colaboração e ter um modelo de governança seguindo as melhores práticas de código aberto.

Para ser listado como um projeto “Grande”, um projeto deve atender a todos os requisitos acima, mais:

O projeto deve ter mais de 50 colaboradores.

O projeto deve ser utilizado em ambientes de produção com uma quantidade significativa de usuários. Como essas informações podem não ser facilmente descobertas a partir de um link para o projeto, tais informações devem ser incluídas na descrição da solicitação pull.

Iniciar

Participe de eventos eBPF para aprender com especialistas do setor

Mais eventos

Saiba mais sobre o eBPF em artigos da comunidade Mais artigos