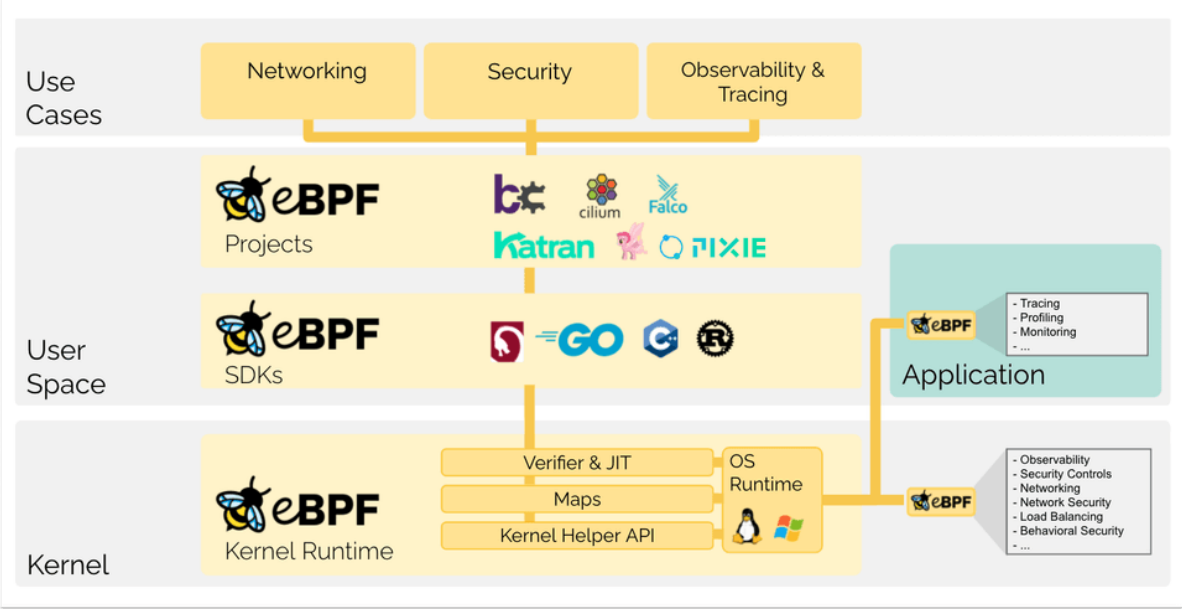
eBPF 介绍

官网：https://ebpf.io/

eBPF 简介：https://ebpf.io/what-is-ebpf/#introduction-to-ebpf

eBPF 是一项革命性技术，起源于 Linux 内核，可以在特权上下文（例如操作系统内核）中运行沙盒程序。它用于安全有效地扩展内核的功能，而无需更改内核源代码或加载内核模块。

eBPF 在 linux 系统中的位置如下：



术语介绍

eBPF 和 BPF 代表什么

BPF（最初的BPF有时被称为cBPF（classic Berkeley Packet Filter，经典BPF）以区别于eBPF） 最初代表伯克利数据包过滤器，专门为过滤网络数据包而创造。但现在 eBPF（扩展 BPF）可以做的不仅仅是数据包过滤，这个缩写词不再有意义。eBPF 现在被认为是一个独立的术语，不代表任何东西。在 Linux 源代码中，术语 BPF 仍然存在，并且在工具和文档中，术语 BPF 和 eBPF 通常可以互换使用。

为什么称为 bee

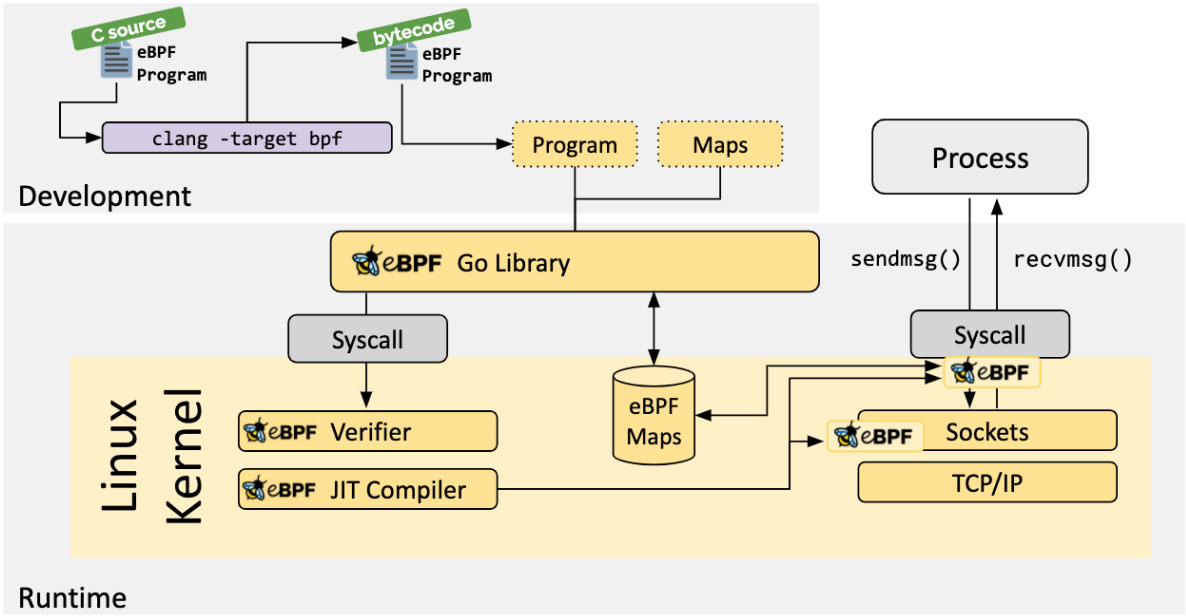
bee 是 eBPF 的官方标志，最初由 Vadim Shchekoldin 创建。在第一届 eBPF 峰会上进行了投票，这只蜜蜂被命名为 eBee。（有关徽标可接受用途的详细信息，请参阅 Linux 基金会品牌指南。）

eBPF 架构模式

eBPF 分为用户空间程序和内核程序两部分：

- 用户空间程序负责加载 BPF 字节码至内核，如需要也会负责读取内核回传的统计信息或者事件详情
- 内核中的 BPF 字节码负责在内核中执行特定事件，如需要也会将执行的结果通过 maps 或者 perf-event 事件发送至用户空间
- 其中用户空间程序与内核 BPF 字节码程序可以使用 map 结构实现双向通信，这为内核中运行的 BPF 字节码程序提供了更加灵活的控制

eBPF 整体架构如下：



用户空间程序与内核中的 BPF 字节码交互的流程主要如下：

1. 使用 LLVM 或者 GCC 工具将编写的 BPF 代码程序编译成 BPF 字节码；
2. 使用加载程序 Loader 将字节码加载至内核；
3. 内核使用验证器（Verfier） 组件保证执行字节码的安全性，以避免对内核造成灾难，在确认字节码安全后将其加载对应的内核模块执行；
4. 内核中运行的 BPF 字节码程序可以使用两种方式将数据回传至用户空间：
    - maps 方式可用于将内核中实现的统计摘要信息（比如测量延迟、堆栈信息）等回传至用户空间；
    - perf-event 用于将内核采集的事件实时发送至用户空间，用户空间程序实时读取分析。

eBPF 实践

当前 eBPF 开发有三种方式：

- 基于 bcc 开发：bcc 提供了对 eBPF 开发，前段提供 Python API，后端 eBPF 程序通过 C 实现。特点是简单易用，但是性能较差；
- 基于 libebpf-bootstrap 开发：libebpf-bootstrap 提供了一个方便的脚手架；
- 基于内核源码开发：内核源码开发门槛较高，但是也更加切合 eBPF 底层原理。

这里以基于 bcc 开发的方式描述：

1. 安装 bcc 工具

BCC 工具可以让你使用 Python 和 C 语言组合来编写 eBPF 程序。

在 Ubuntu 低版本系统中安装 BCC 工具是比较简单的，可以使用以下命令：

```
install-bcc.h

#!/usr/bin/bash

# 安装
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 4052245BD4284CDD
echo "deb https://repo.iovisor.org/apt/$(lsb_release -cs) $(lsb_release -cs)
main" | sudo tee /etc/apt/sources.list.d/iovisor.list
sudo apt-get update
sudo apt-get install bcc-tools libbcc-examples linux-headers-$(uname -r)

# 检测
bcc -v
```

二进制安装：

```
sudo apt-get install bpfcc-tools linux-headers-$(uname -r)

bcc -v
```

使用源码安装：因为`repo.iovisor.org`上的版本老旧，且存在bug。

```
apt purge bpfcc-tools libbpfcc python3-bpfcc
wget https://github.com/iovisor/bcc/releases/download/v0.25.0/bcc-src-with-
submodule.tar.gz
tar xf bcc-src-with-submodule.tar.gz
cd bcc/
apt install -y python-is-python3
apt install -y bison build-essential cmake flex git libedit-dev   libllvm11
llvm-11-dev libclang-11-dev zlib1g-dev libelf-dev libfl-dev python3-distutils
apt install -y checkinstall
mkdir build
cd build/
cmake -DCMAKE_INSTALL_PREFIX=/usr -DPYTHON_CMD=python3 ..
make
checkinstall
```

踩坑：

1. tcptop 命令无法运行：

   > from bcc import BPF
   > ImportError: No module named bcc
   > `python3 ./tcptop`

2. AttributeError: /lib/x86_64-linux-gnu/libbcc.so.0: undefined symbol: bpf_module_create_b

   > eBPF 源码构建目录下 `cp -r  /eBPF/bcc/build/src/python/bcc-python3/bcc/\* /usr/lib/python3/dist-packages/bcc/`

3. LLVM ERROR:

   > 无法解决。😳😳

如果安装失败，可以参考官网安装文档，如下：https://github.com/iovisor/bcc/

2. 编写 eBPF 版的 hello world

步骤如下：

1. 使用 C 语言编写 eBPF 程序的内核态功能（也就是运行在内核态的 eBPF 程序）。
2. 使用 Python 编写加载代码和用户态功能。

> - 为什么不能全部使用 Python 编写呢？这是因为 LLVM/Clang 只支持将 C 语言编译成 eBPF 字节码，而不支持将 Python 代码编译成 eBPF 字节码。
> - 所以，eBPF 内核态程序只能使用 C 语言编写。而 eBPF 的用户态程序可以使用 Python 进行编写，这样就能简化编写难度。

新建 `ebpf.c` 输入以下内容：

```
int ebpf(void *ctx)
{
    bpf_trace_printk("Hi, ebpf.");
    return 0;
}
```

新建 `ebpf.py` 输入以下内容：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# 加载 BCC 库
from bcc import BPF

# 加载 eBPF 内核态程序
b = BPF(src_file="ebpf.c")

# 将 eBPF 程序挂载到 kprobe
b.attach_kprobe(event="do_sys_openat2", fn_name="ebpf")

# 读取并且打印 eBPF 内核态程序输出的数据
b.trace_print()
```

3. 运行

```
sudo python3 ebpf.py
```