

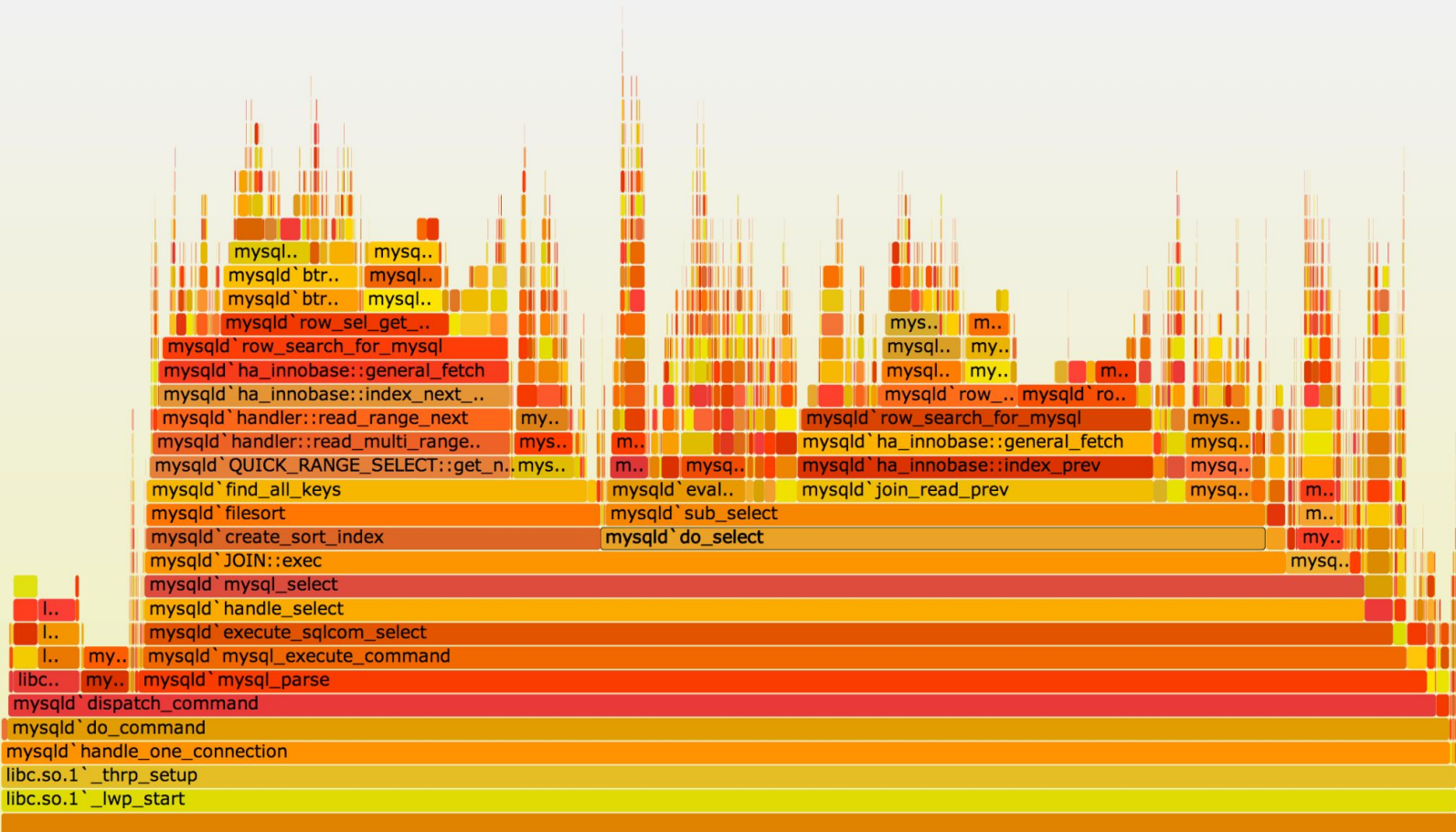


# Visualizing Execution with FlameGraphs

## What you learn in the this session

- What FlameGraphs are and how to read them
- How to generate them
- What the performance impact is on supporting them
- How to generate them from inside a container

## Flame Graph

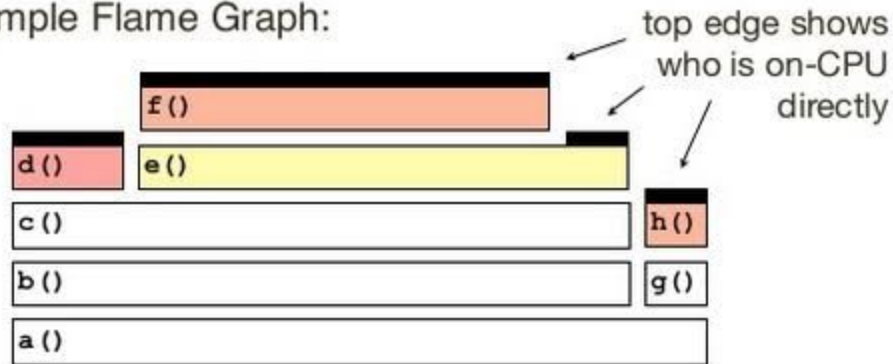


Function: `mysqld`do_select` (159,007 samples, 45.64%)

- Invented by Brendan Gregg (Sun, Oracle, Netflix, Intel) 2011
  - Presented at LISA 2013 [https://www.brendangregg.com/Slides/LISA13 Flame Graphs.pdf](https://www.brendangregg.com/Slides/LISA13%20Flame%20Graphs.pdf)
- Picture of relative time spent in a single image
- Visual, navigable, obvious
  - <https://raw.githubusercontent.com/brendangregg/FlameGraph/master/example-perf.svg>
- Low overhead (you will measure this!)

## Flame Graphs: How to Read

- A CPU Sample Flame Graph:



- Q: which function is on-CPU the most?
- A: `f()`

`e()` is on-CPU a little, but its runtime is mostly spent in `f()`, which is on-CPU directly



## Great Documentation

- USENIX ATC '17: Visualizing Performance with Flame Graphs
  - <https://www.youtube.com/watch?v=D53T1Ejig1Q>
- LISA13 - Blazing Performance with Flame Graphs
  - <https://www.youtube.com/watch?v=nZfNehCzGdw>
- <https://github.com/brendangregg/FlameGraph>
- <https://www.brendangregg.com/flamegraphs.html>

- Stack Sampling, sample as often or as little as your application dictates
  - **perf record -F 100 -g -- <application under measurement>**
- Can be generated by **any** profiler that captures stack back traces
  - Perf, Dtrace, Instruments, SystemTap, [VTune](#), ktap, Xperf
  - Better quality stack information, better FlameGraphs compile binaries with
    - **-g -fno-omit-frame-pointer**
    - Fedora enabling by default in 38
      - <https://fedoraproject.org/wiki/Changes/fno-omit-frame-pointer>
      - Brendan Gregg wants on by default in GCC
- Install Linux perf command
  - **apt install linux-tools-\$(uname -r) linux-tools-common**
  - When running in a container, tools in container must match host kernel!!



# Create a VM on the Google Cloud Console

Google Cloud

invertible-spot-755

Search (/) for resources, docs, products, and more

Search

S

Create an instance

HELP ASSISTANT

To create a VM instance, select one of the options:

New VM instance

Create a single VM instance from scratch

New VM instance from template

Create a single VM instance from an existing template

New VM instance from machine image

Create a single VM instance from an existing machine image

Marketplace

Deploy a ready-to-go solution onto a VM instance

Name \*

instance-2

Labels ?

+ ADD LABELS

Region \*

us-west1 (Oregon)

Region is permanent

Zone \*

us-west1-b

Zone is permanent

Machine configuration

Machine family

GENERAL-PURPOSE

COMPUTE-OPTIMIZED

MEMORY-OPTIMIZED

GPU

Machine types for common workloads, optimized for cost and flexibility

Series

T2D

Powered by AMD EPYC Milan CPU platform

Machine type

t2d-standard-4 (4 vCPU, 16 GB memory)

vCPU

4

Memory

16 GB

✓ CPU PLATFORM AND GPU

Display device

Enable to use screen capturing and recording tools.

☐ Enable display device

Confidential VM service ?

Monthly estimate

\$140.36

That's about \$0.19 hourly

Pay for what you use: No upfront costs and per second billing

Item	Monthly estimate
4 vCPU + 16 GB memory	\$123.36
100 GB SSD persistent disk	\$17.00
Use discount	-\$0.00
Total	\$140.36

Compute Engine pricing

LESS

## VM Recommendations

- Ubuntu 22.04 LTS
- t2d-standard-4 (4 physical cores, 16GB)
- 20GB+ PD-SSD for boot disk (IOPs scale with disk size)
- Allow HTTP and HTTPS (only for zero risk, ephemeral VMs)

## First Exercise

- Follow the README in this repo  
<https://github.com/jensengrey/flamegraph-container>

## Second Exercise

- Same VM or new VM
  - Take a VM Snapshot?
  - Brand new VM?
- Building and running two programs
  - Small Obfuscated Ray Tracer
  - POV-Ray

## Prepare your VM

```
# apt based (Ubuntu/Debian)
sudo sh -c 'apt update -y && apt upgrade -y'
sudo apt install build-essential -y
sudo apt install make tmux tree htop imagemagick nano vim git cmake -y
sudo apt install python3-dev python3-venv
```

```
# homebrew, https://brew.sh/
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

```
# rust, https://rustup.rs/
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

## Create Python Environment

```
#!/bin/bash
```

```
set -eux;
```

```
python3 -m venv test.env
```

```
source test.env/bin/activate
```

```
pip install -U pip setuptools wheel
```

## Install Perf Command

```
sudo apt install linux-tools-$(uname -r) linux-tools-common -y
```

## Install FlameGraph Tools

# the canonical tool for converting perf traces to svg  
git clone <https://github.com/brendangregg/FlameGraph>

# more performant in some scenarios

- <https://github.com/flamegraph-rs/flamegraph>
  - <https://github.com/jonhoo/inferno>



## Enable sudo-less event collection

```
sudo sh -c "echo 0 > /proc/sys/kernel/kptr_restrict"  
sudo sh -c "echo -1 > /proc/sys/kernel/perf_event_paranoid"
```

# perf top -e cpu-clock

```
notes -- arm@arm: ~ -- ssh arm@192.168.233.129 -- 121x40
~ -- zsh  ...S -- zsh  ...S -- zsh  ...233.129  ...S -- zsh  ...S -- zsh  ...S -- zsh  -zsh  +
Samples: 411K of event 'cpu-clock:pppH', 4000 Hz, Event count (approx.): 8834852570 lost: 0/0 drop: 0/0
Overhead Shared Object Symbol
75.05% miniray_4.6 [.] main
5.99% libc-2.31.so [.] __atan2_finite
4.67% libc-2.31.so [.] sinf32x
4.58% libc-2.31.so [.] cosf32x
3.23% libc-2.31.so [.] atan2f32x
0.30% perf [.] rb_next
0.29% [kernel] [k] __softirqentry_text_start
0.24% libc-2.31.so [.] sqrtf32x
0.23% libvmttools.so.0.0.0 [.] Backdoor_InOut
0.19% miniray_4.6 [.] sqrt@plt
0.16% miniray_4.6 [.] D
0.13% miniray_4.6 [.] putchar@plt
0.10% perf [.] hpp__sort_overhead
0.09% [kernel] [k] finish_task_switch
0.08% miniray_4.6 [.] __libc_csu_init
0.08% perf [.] dso__find_symbol
0.08% [kernel] [k] __schedule
0.07% perf [.] hist_entry__sort
0.07% libc-2.31.so [.] strcmp
0.07% perf [.] perf_hpp__is_dynamic_entry
0.06% libc-2.31.so [.] pow
0.05% miniray_4.6 [.] __FRAME_END__+0xffff55552818202c
0.05% [kernel] [k] filldir64
0.05% containerd [.] _start
0.05% miniray_4.6 [.] 0x000000000000279c
0.04% [kernel] [k] __d_lookup
0.04% [kernel] [k] el0_svc_common.constprop.0
0.04% perf [.] hists__decay_entry
0.04% [kernel] [k] update_blocked_averages
0.04% [kernel] [k] rebalance_domains
0.04% libc-2.31.so [.] cfree
0.04% libc-2.31.so [.] malloc
0.04% libc-2.31.so [.] 0x00000000000076b74
0.03% [kernel] [k] do_task_stat
0.03% [kernel] [k] do_idle
0.03% perf [.] output_resort
For a higher level overview, try: perf top --sort comm,dso
[0] 0:htop 1:bash 2:bash 3:python3 4:bash 5:perf+ "arm" 18:05 18-Jan-23
```

What you see when perf is restricted

```
sudo sh -c "echo 1 > /proc/sys/kernel/kptr_restrict"
```

```
sudo sh -c "echo 1 > /proc/sys/kernel/perf_event_paranoid"
```

Firefox Developer Edition

https://ssh.cloud.google.com/v2/ssh/projects/invertible-spot-755/zones/us-west1-b/instances/tde

SSH-in-browser

UPLOAD FILE

DOWNLOAD FILE

```
Error:
Access to performance monitoring and observability operations is limited.
Consider adjusting /proc/sys/kernel/perf_event_paranoid setting to open
access to performance monitoring and observability operations for processes
without CAP_PERFMON, CAP_SYS_PTRACE or CAP_SYS_ADMIN Linux capability.
More information can be found at 'Perf events and tool security' document:
https://www.kernel.org/doc/html/latest/admin-guide/perf-security.html
perf_event_paranoid setting is 1:
  -1: Allow use of (almost) all events by all users
      Ignore mlock limit after perf_event_mlock_kb without CAP_IPC_LOCK
  >= 0: Disallow raw and ftrace function tracepoint access
  >= 1: Disallow CPU event access
  >= 2: Disallow kernel profiling
To make the adjusted perf_event_paranoid setting permanent preserve it
in /etc/sysctl.conf (e.g. kernel.perf_event_paranoid = <setting>)

Press any key...
```

[0] 0:htop- 1:perf\* "tdelme" 21:52 30-Jan-23

## Enable sudo-less event collection

```
sudo sh -c "echo 0 > /proc/sys/kernel/kptr_restrict"  
sudo sh -c "echo -1 > /proc/sys/kernel/perf_event_paranoid"
```

Samples per second

Call graph recording

`perf record -F 100 -a -g -- <application under measurement>`

All cores



## First Test Workload

```
git clone https://github.com/mzucker/miniray  
cd miniray  
mkdir build
```

Edit ../CmakeLists.txt

```
cmake_minimum_required(VERSION 2.8.12)
```

```
set(GCC_COVERAGE_COMPILE_FLAGS "-g3 -fno-omit-frame-pointer")
```

```
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${GCC_COVERAGE_COMPILE_FLAGS}")
```

```
set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} ${GCC_COVERAGE_COMPILE_FLAGS}")
```

```
project(miniray)
```

```
set(EXECUTABLE_OUTPUT_PATH ${PROJECT_BINARY_DIR})
```

```
add_subdirectory(src)
```



## Run cmake

```
cmake ..  
-- The C compiler identification is GNU 11.3.0  
-- The CXX compiler identification is GNU 11.3.0  
-- Detecting C compiler ABI info  
-- Detecting C compiler ABI info - done  
-- Check for working C compiler: /usr/bin/cc - skipped  
-- Detecting C compile features  
-- Detecting C compile features - done  
-- Detecting CXX compiler ABI info  
-- Detecting CXX compiler ABI info - done  
-- Check for working CXX compiler: /usr/bin/c++ - skipped  
-- Detecting CXX compile features  
-- Detecting CXX compile features - done  
-- Configuring done  
-- Generating done  
-- Build files have been written to: /home/sean_jensengrey/miniray/build
```

## Build the code

```
$ make
[  3%] Built target encode
[  6%] Built target miniray_4.6
[ 10%] Built target miniray_4.5
[ 13%] Built target miniray_1.3
[ 16%] Built target miniray_4.6_commented
[ 20%] Built target miniray_4.0
[ 23%] Built target miniray_2.4
...
# src/miniray_0.2.cpp:73:3: error: narrowing conversion of '2169135176' from
'unsigned int' to 'i' {aka 'int'} [-Wnarrowing]
```

## Render First Image

```
$ perf stat ./miniray_4.6 > test2.ppm
```

```
Performance counter stats for './miniray_4.6':
```

17,380.65 msec	task-clock	#	0.998 CPUs utilized
2,498	context-switches	#	0.144 K/sec
3	cpu-migrations	#	0.000 K/sec
52	page-faults	#	0.003 K/sec
<not supported>	cycles		
<not supported>	instructions		
<not supported>	branches		
<not supported>	branch-misses		

```
17.415531414 seconds time elapsed
```

```
17.378465000 seconds user
```

```
0.000000000 seconds sys
```

## Render First Image

```
$ perf stat ./miniray_4.6 > miniray_46.ppm  
$ convert miniray_46.ppm miniray_46.jpg
```



## Peruse File System

```
python3 -m http.server 9999
# get your external ip
https://www.google.com/search?hl=en&q=my%20ip%20address
# add a firewall rule for your IP, allow all
gcloud compute --project=<your project> \
  firewall-rules create devaccess \
  --direction=INGRESS \
  --priority=400 \
  --network=default \
  --action=ALLOW \
  --rules=tcp \
  --source-ranges=<your ip> \
  --enable-logging
```

```
$ perf record -F 10000 -g -- ./miniray_4.6 > miniray_4.6.ppm
```

```
$ cd ~/FlameGraph;
```

```
$ perf script | ./stackcollapse-perf.pl | ./flamegraph.pl > perf.svg
```

Reset Zoom

Flame Graph

Search

ic



## Exercise 1: Generate FlameGraph

Follow the instructions on the previous slide and generate the flamegraph.svg

Exercise 2: What is the impact on wall clock  
time for including debug information?

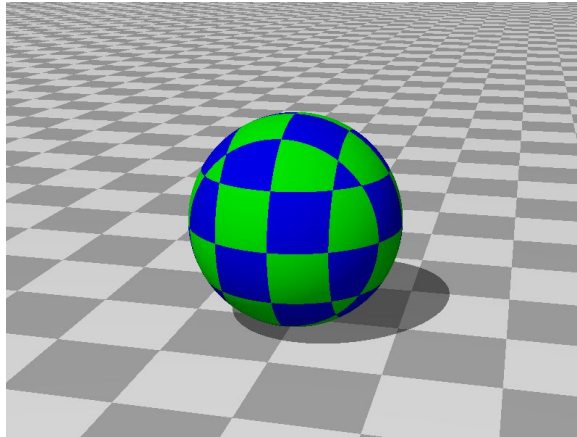


## Install Docker on Ubuntu

- Bring up an Ubuntu VM
- Install Docker
- Install devtools

## Build Container and Render Test Image

```
git clone https://github.com/jensengrey/workshop-containers-af99  
cd workshop-containers-af99/povray  
sudo docker build -t povtest:1 - < povray.dockerfile  
sudo docker run -t --rm -v $PWD:/povfiles --user $(id -u):$(id -g) povtest:1 povray  
/povfiles/debug.pov
```



# perf report --no-inline

```
2023-01-30 -- sean_jensengrey@flamegraph-demo: ~ -- ssh - gcloud.py compute ssh --zone us-west1-b sean_jensengrey@flamegraph-demo --project invertible-spot-755 -- 171x47
Samples: 16K of event 'cpu-clock:ppph', Event count (approx.): 4138250000
Children  Self  Command  Shared Object  Symbol
+ 98.28%  0.00%  povray   libc.so.6      [.] send_vc
+ 98.27%  0.00%  povray   libc.so.6      [.] __strncat_ssse3
+ 98.27%  0.00%  povray   libboost_thread.so.1.74.0 [.] 0x00007f59aa6e80ca
+ 56.67%  0.00%  povray   povray         [.] pov::Task::TaskThread
+ 56.66%  0.59%  povray   povray         [.] pov::TraceTask::SimpleSamplingM0
+ 56.65%  0.00%  povray   povray         [.] pov::TraceTask::Run
+ 53.71%  1.45%  povray   povray         [.] pov::TracePixel::operator()
+ 48.47%  0.88%  povray   povray         [.] pov::Trace::TraceRay
+ 41.54%  0.00%  povray   povray         [.] vfe::vfeSession::WorkerThread
+ 34.95%  2.73%  povray   povray         [.] pov::Trace::ComputeTextureColour
+ 31.75%  0.33%  povray   povray         [.] pov::Trace::ComputeOneTextureColour
+ 31.40%  4.40%  povray   povray         [.] pov::Trace::ComputeLightedTexture
+ 24.74%  0.01%  povray   povray         [.] POVMS_ProcessMessages
+ 24.14%  0.62%  povray   povray         [.] pov::Trace::ComputeDiffuseLight
+ 23.52%  4.91%  povray   povray         [.] pov::Trace::ComputeOneDiffuseLight
+ 20.83%  6.82%  povray   povray         [.] pov::Intersect_BBBox_Tree
+ 19.67%  0.01%  povray   povray         [.] POVMS_Receive
+ 19.65%  0.02%  povray   povray         [.] POVMS_MessageReceiver::ReceiveHandler
+ 19.05%  0.02%  povray   povray         [.] pov_frontend::RenderFrontend<vfe::vfeParserMessageHandler, pov_frontend::FileMessageHandler, vfe::vfeRenderMe
+ 19.03%  5.85%  povray   povray         [.] pov_frontend::ImageMessageHandler::DrawPixelBlockSet
+ 16.84%  0.00%  povray   povray         [.] vfe::vfeSession::ProcessFrontend
+ 16.84%  0.01%  povray   povray         [.] vfe::VirtualFrontEnd::Process
+ 16.72%  0.00%  povray   povray         [.] pov_frontend::ImageProcessing::WriteImage[abi:cxx11]
+ 16.72%  0.31%  povray   povray         [.] pov_base::Png::Write
+ 13.00%  12.99% povray   libm.so.6      [.] _eqtf2
+ 12.94%  1.19%  povray   povray         [.] pov::Trace::TraceShadowRay
+ 12.69%  4.19%  povray   povray         [.] pov_base::GetEncodedRGBValue
+ 12.40%  0.31%  povray   povray         [.] pov::Trace::FindIntersection
+ 11.01%  1.53%  povray   povray         [.] pov::Trace::TracePointLightShadowRay
+ 6.92%  2.69%  povray   povray         [.] pov::Find_Intersection
+ 5.49%  5.49%  povray   povray         [.] pov::Check_And_Enqueue
+ 4.87%  4.81%  povray   povray         [.] POVMSStream_Read
+ 3.71%  1.58%  povray   libpng16.so.16.37.0 [.] png_write_row
+ 3.55%  0.50%  povray   povray         [.] pov::Box::All_Intersections
+ 3.55%  3.53%  povray   povray         [.] pov::Trace::ComputeOneWhiteLightRay
+ 3.19%  2.65%  povray   povray         [.] pov::TracePixel::CreateCameraRay
+ 2.94%  0.56%  povray   povray         [.] pov_base::SRGBGammaCurve::Encode
+ 2.49%  2.30%  povray   povray         [.] POVMSStream_Write
+ 2.33%  0.00%  povray   povray         [.] POVMS_Object::Write
+ 2.19%  0.48%  povray   povray         [.] pov::ViewData::CompletedRectangle
+ 2.11%  0.00%  povray   libpng16.so.16.37.0 [.] 0x00007f59aa736436
+ 2.11%  0.00%  povray   libz.so.1.2.11    [.] deflate
+ 1.88%  0.00%  povray   [kernel.kallsyms] [k] entry_SYSCALL_64_after_hwframe
Cannot load tips.txt file, please install perf!
[0] 0:htop- 1:python3 2:python3 3:bash 4:bash 5:bash 6:bash 7:perf* 8:bash 9:bash *flamegraph-demo* 09:45 31-Jan-23
```

## Hacking Dockerfiles

- Don't suffer in docker build loop
- Spin up a VM with the same base image as your container, debug and then port into Dockerfile

## Single Best Docker+Perf Tutorial

- <https://gendignoux.com/blog/2019/11/09/profiling-rust-docker-perf.html>

## Docker Locks Down Specific Syscalls

- <https://docs.docker.com/engine/security/seccomp/>

## Modify Build Command to Support Perf

```
cd povray/unix && \  
    ./prebuild.sh && \  
    cd .. && \  
    CXXFLAGS="-fno-omit-frame-pointer" ./configure --enable-debug --enable-profile  
COMPILED_BY="povray-perf-builder" && \  
    make -j4 && \  
    make install
```

## Generate Perf Trace

```
sudo docker run -t -v $PWD:/povfiles --security-opt seccomp=unconfined povtest-perf:1 perf  
record -F 100 -a -g -- povray /povfiles/debug.pov;
```





## Grafana Phlare

- <https://grafana.com/oss/flare/>
- <https://github.com/grafana/flare>
- <https://grafana.com/docs/flare/latest/>
- <https://grafana.com/docs/flare/latest/operators-guide/deploy-kubernetes/>
- <https://grafana.com/tutorials/>
- <https://www.youtube.com/watch?v=ORINvZAurIY>