

Hidden Linux Metrics with Prometheus eBPF Exporter

Building blocks for managing a fleet at scale

Alexander Huynh
R&D @ Cloudflare

Ivan Babrou
Performance @ Cloudflare

What does Cloudflare do



CDN

Moving content physically closer to visitors with our CDN.

Intelligent caching

Unlimited DDOS mitigation

Unlimited bandwidth at flat pricing with free plans

Edge access control

IPFS gateway

Onion service



Website Optimization

Making web fast and up to date for everyone.

TLS 1.3 (with 0-RTT)

HTTP/2 + QUIC

Server push

AMP

Origin load-balancing

Smart routing

Serverless / Edge Workers

Post quantum crypto



DNS

Cloudflare is the fastest managed DNS providers in the world.

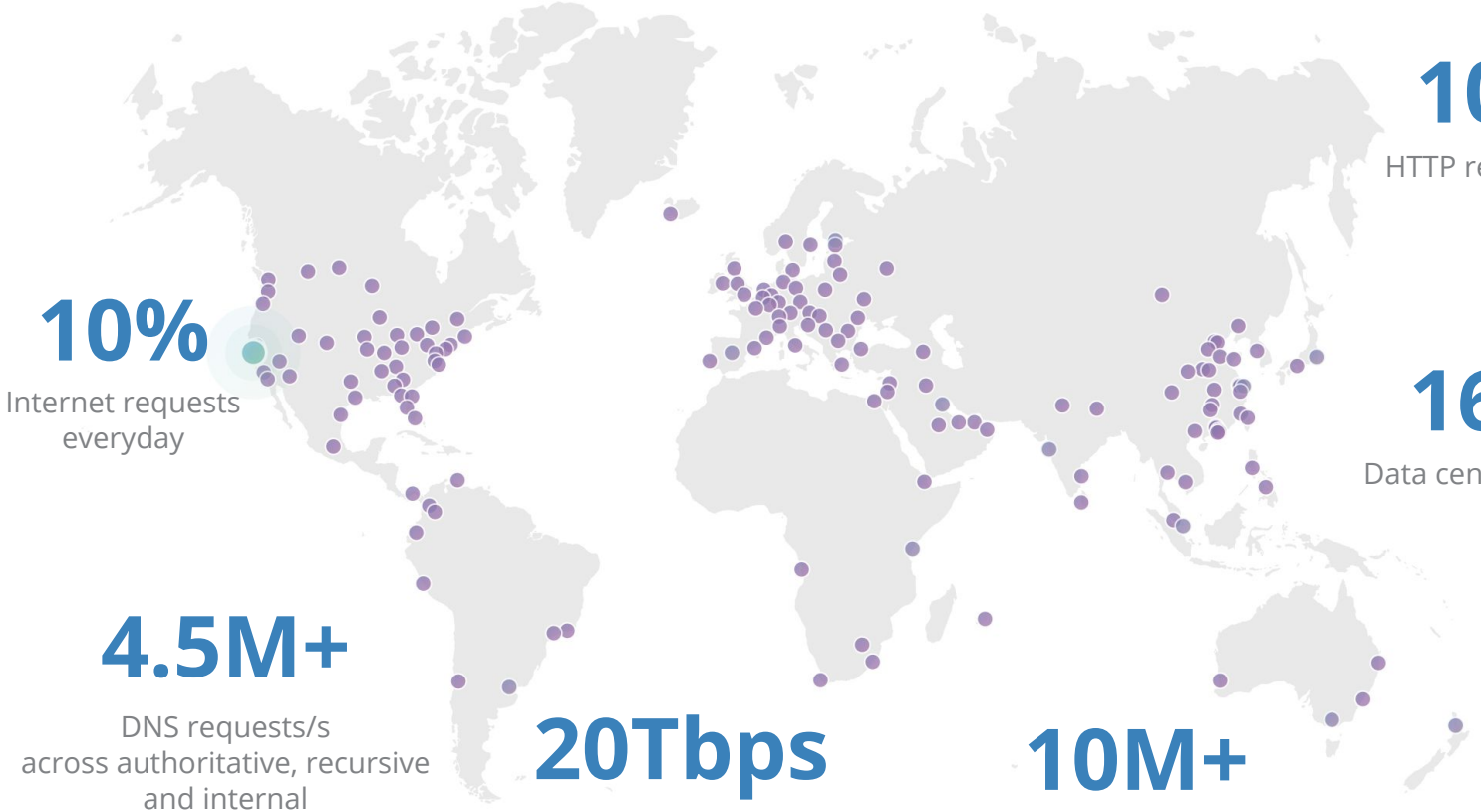
1.1.1.1

2606:4700:4700::1111

DNS over TLS

CLOUDFLARE

Cloudflare’s anycast network



10M+
HTTP requests/second

10%
Internet requests everyday

160+
Data centers globally

4.5M+
DNS requests/s across authoritative, recursive and internal

20Tbps
Network capacity

10M+
Websites, apps & APIs in 150 countries

Link to slides with speaker notes

Slideshare doesn’t allow links on the first 3 slides

eBPF overview

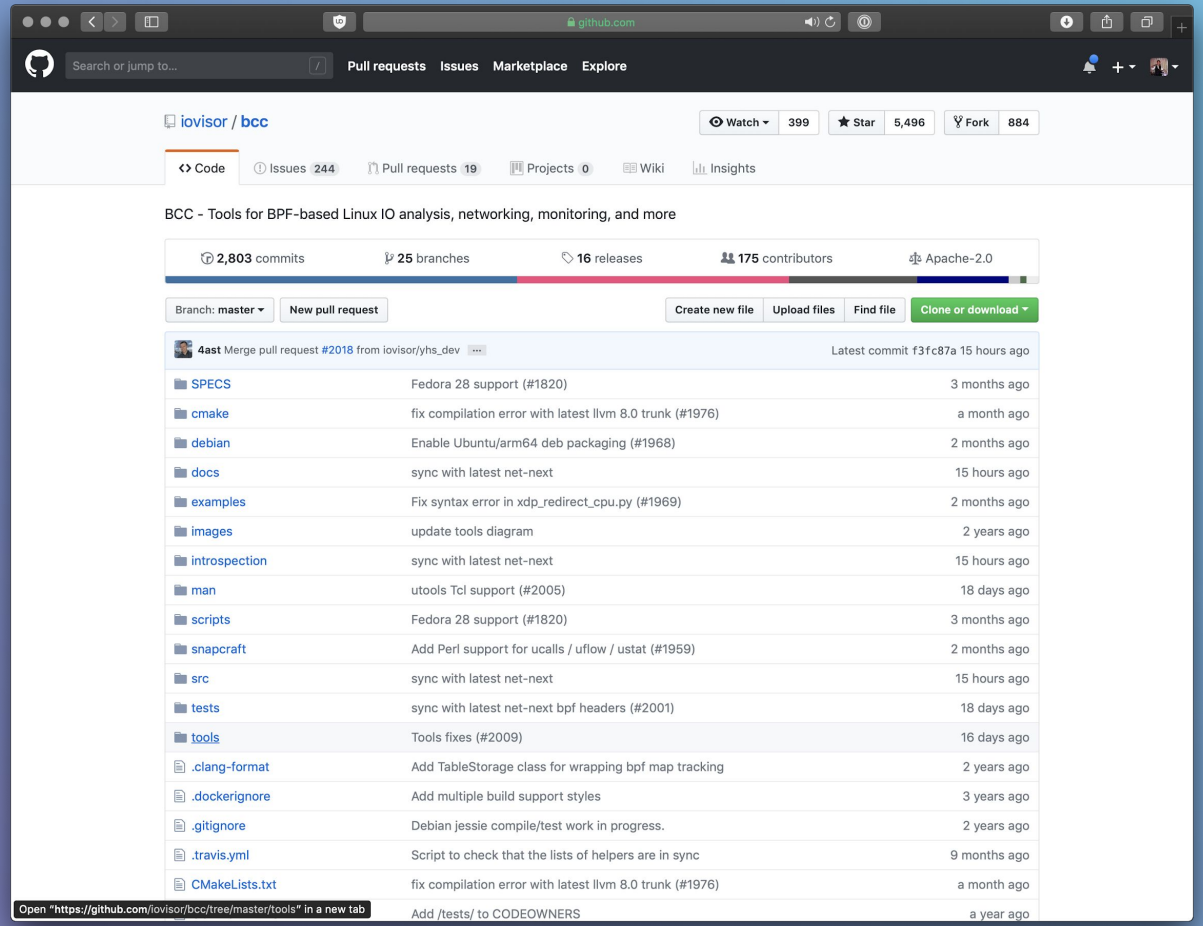
- Extended Berkeley Packet Filter
- In-kernel virtual machine with specific instruction set
- General tracepoints

- Further reading
  - http://www.brendangregg.com/ebpf.html
  - https://cilium.readthedocs.io/en/v1.1/bpf/

eBPF overview

- Builds on classical BPF
- Limited to 4096 instructions, 512 bytes stack, no loops
- Both kernel-level and process-level tracing

- Further reading
  - http://www.brendangregg.com/ebpf.html
  - https://cilium.readthedocs.io/en/v1.1/bpf/


github.com
Search or jump to...
Pull requests
Issues
Marketplace
Explore
iovisor / bcc
Watch 399
Star 5,496
Fork 884
Code
Issues 244
Pull requests 19
Projects 0
Wiki
Insights
BCC - Tools for BPF-based Linux IO analysis, networking, monitoring, and more
2,803 commits
25 branches
16 releases
175 contributors
Apache-2.0
Branch: master
New pull request
Create new file
Upload files
Find file
Clone or download
4ast Merge pull request #2018 from iovisor/yhs_dev
Latest commit f3fc87a 15 hours ago
SPECS Fedora 28 support (#1820) 3 months ago
cmake fix compilation error with latest llvm 8.0 trunk (#1976) a month ago
debian Enable Ubuntu/arm64 deb packaging (#1968) 2 months ago
docs sync with latest net-next 15 hours ago
examples Fix syntax error in xdp_redirect_cpu.py (#1969) 2 months ago
images update tools diagram 2 years ago
introspection sync with latest net-next 15 hours ago
man utools Tcl support (#2005) 18 days ago
scripts Fedora 28 support (#1820) 3 months ago
snapcraft Add Perl support for ucalls / uflow / ustat (#1959) 2 months ago
src sync with latest net-next 15 hours ago
tests sync with latest net-next bpf headers (#2001) 18 days ago
tools Tools fixes (#2009) 16 days ago
.clang-format Add TableStorage class for wrapping bpf map tracking 2 years ago
.dockerignore Add multiple build support styles 3 years ago
.gitignore Debian jessie compile/test work in progress. 2 years ago
.travis.yml Script to check that the lists of helpers are in sync 9 months ago
CMakeLists.txt fix compilation error with latest llvm 8.0 trunk (#1976) a month ago
Add /tests/ to CODEOWNERS a year ago
Open "https://github.com/iovisor/bcc/tree/master/tools" in a new tab

eBPF usage, current

- Many examples at bcc tools
  - Print function parameters
  - Histogram of I/O operation times
  - Tap network connections

BCC tools' trace

```
ubuntu:/usr/share/bcc/tools$ sudo ./trace 'sys_execve "%s", arg1'
PID     TID     COMM            FUNC             -
17366   17366   tmux: server    sys_execve       /bin/bash
17369   17369   bash            sys_execve       /usr/bin/locale-check
17371   17371   bash            sys_execve       /usr/bin/lesspipe
17372   17372   lesspipe        sys_execve       /usr/bin/basename
17374   17374   lesspipe        sys_execve       /usr/bin/dirname
17376   17376   bash            sys_execve       /usr/bin/dircolors
^C
```

CLOUDFLARE

BCC tools’ biolatency

```
ubuntu:/usr/share/bcc/tools$ sudo ./biolatency 10 1
Tracing block device I/O... Hit Ctrl-C to end.

     usecs               : count     distribution
...
        32 -> 63         : 0        |                                        |
        64 -> 127        : 2        |*                                       |
       128 -> 255        : 24       |***********************                 |
       256 -> 511        : 41       |****************************************|
       512 -> 1023       : 7        |******                                  |
      1024 -> 2047       : 1        |                                        |
```

CLOUDFLARE

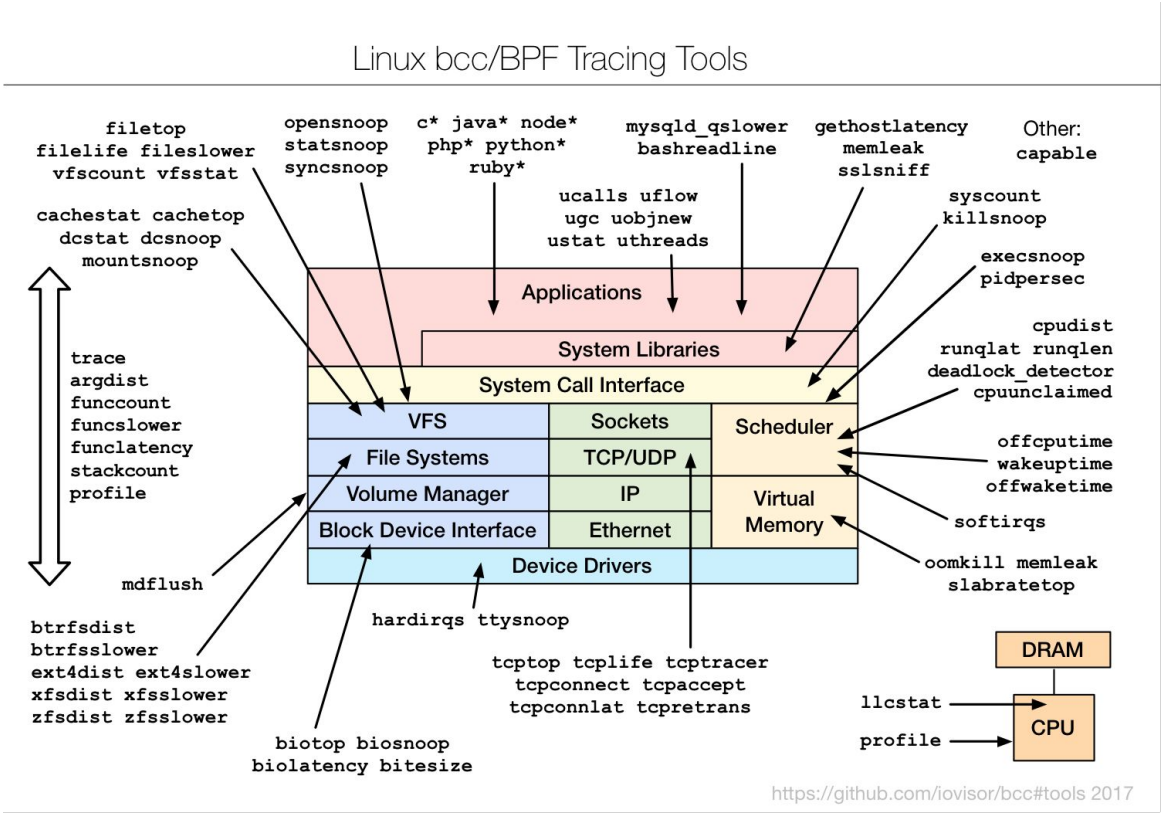
BCC tools' tcpconnect

```
ubuntu:/usr/share/bcc/tools$ sudo ./tcpconnect 2>/dev/null
PID    COMM         IP SADDR            DADDR            DPORT
6918   wget         4  10.0.2.15        1.1.1.1          443


ubuntu@ubuntu:~$ wget -qO /dev/null https://1.1.1.1/
```

CLOUDFLARE

Many more tools available

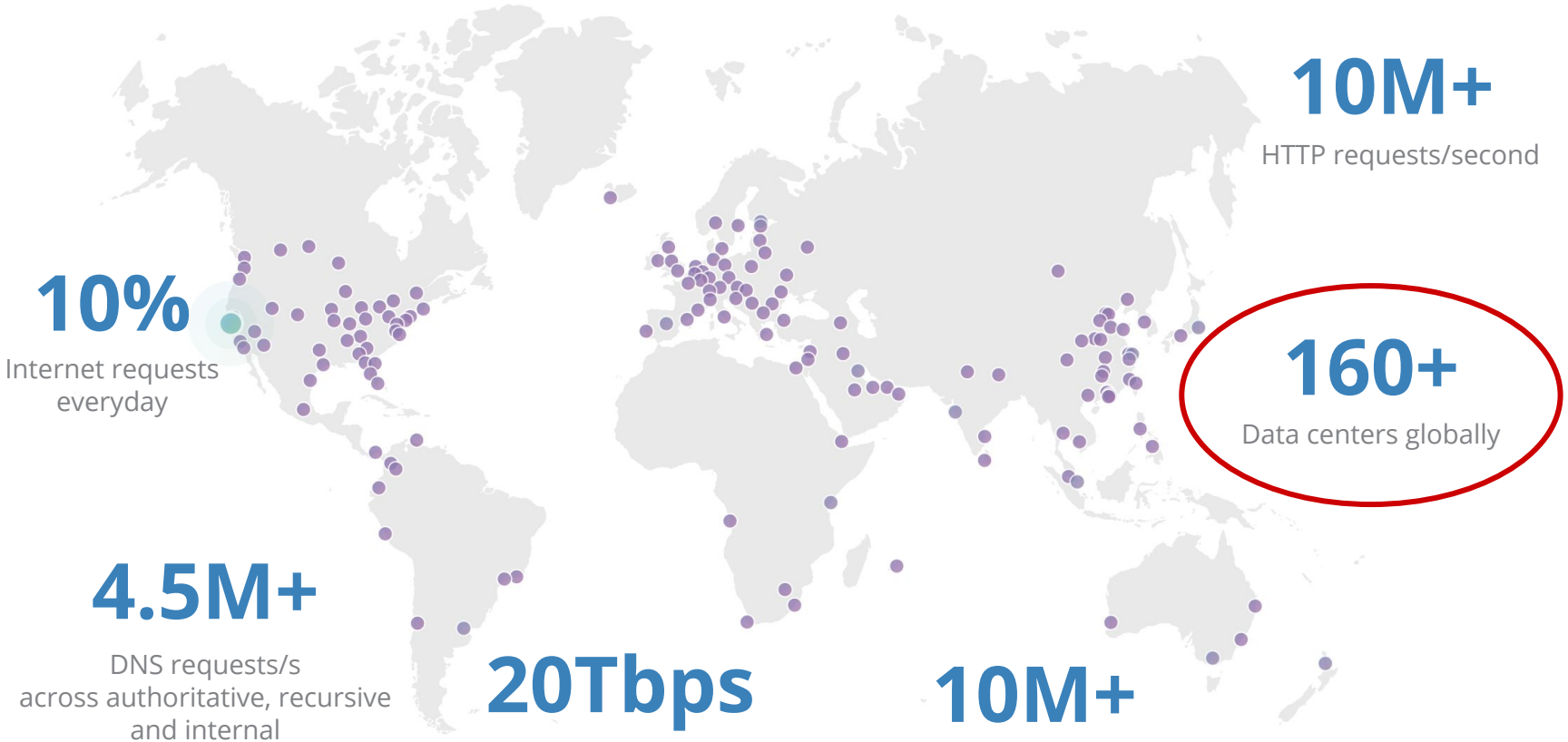


CLOUDFLARE

eBPF usage, current

- Limited to one system
- Human-driven
- Called as needed

Cloudflare’s anycast network



10M+
HTTP requests/second

10%
Internet requests everyday

160+
Data centers globally

4.5M+
DNS requests/s across authoritative, recursive and internal

20Tbps
Network capacity

10M+
Websites, apps & APIs in 150 countries

eBPF usage, future

- Passive collection
- Exposed for metrics consumption
- Stick to basic metrics to preserve performance

Two main options to collect system metrics



node_exporter

Gauges and counters for system metrics with lots of plugins:
cpu, diskstats, edac, filesystem, loadavg, meminfo, netdev, etc



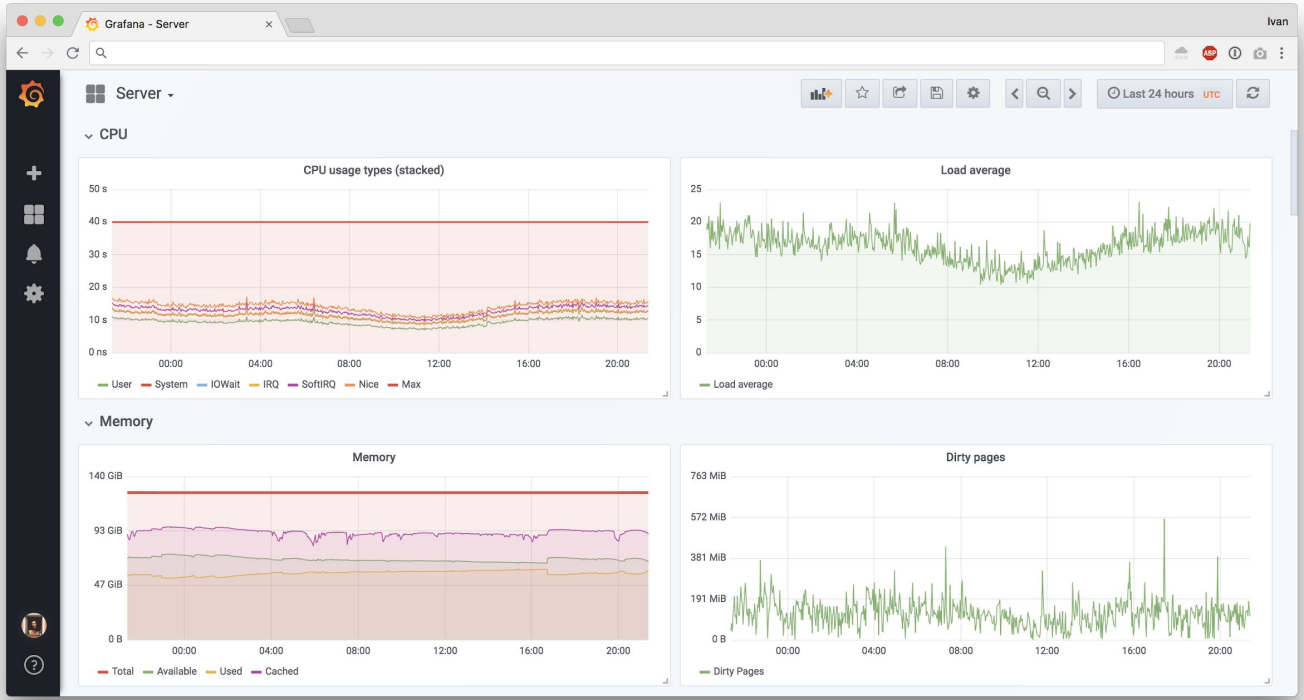
cAdvisor

Gauges and counters for container level metrics:
cpu, memory, io, net, delay accounting, etc.

Check out this issue about Prometheus friendliness.

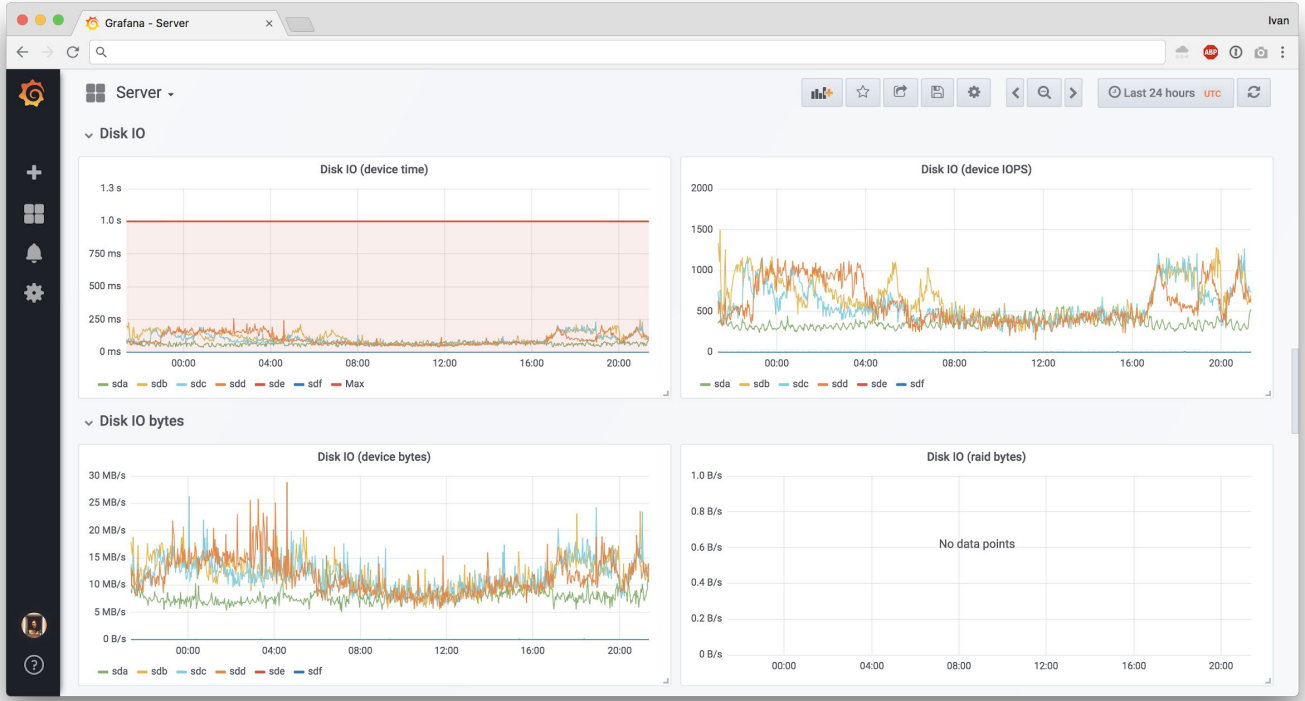
CLOUDFLARE

Example graphs from node_exporter



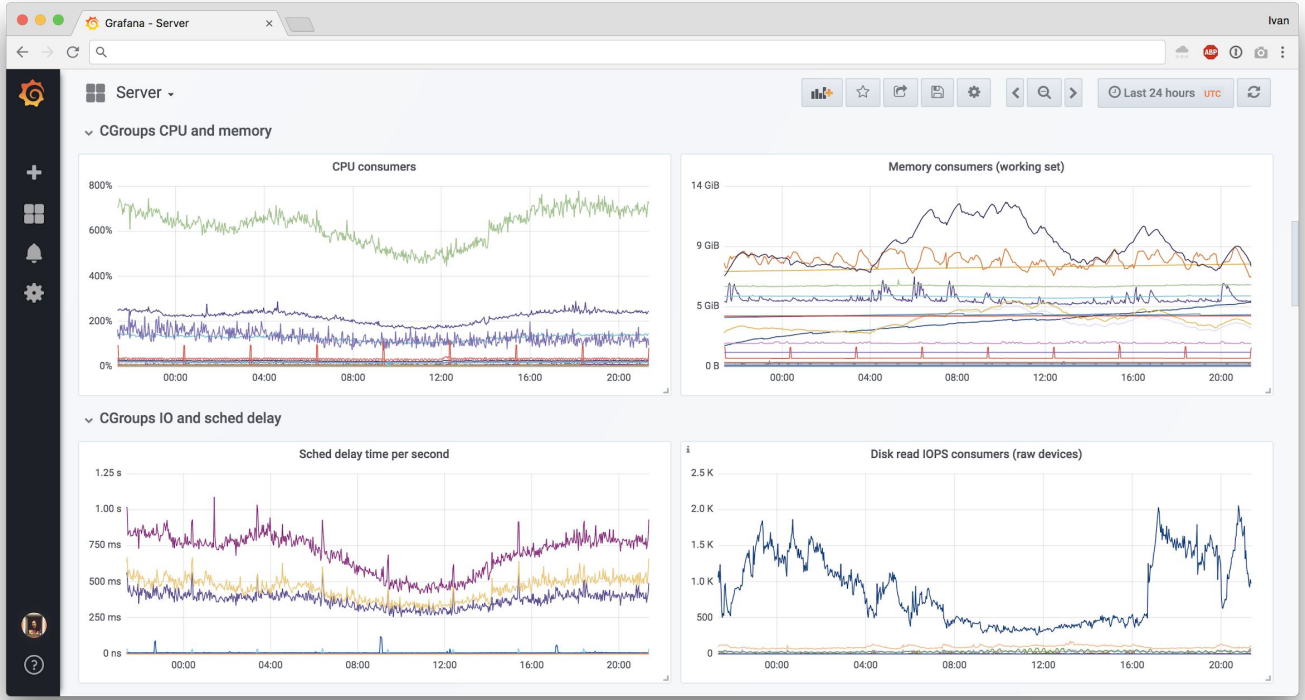
CLOUDFLARE

Example graphs from node_exporter



CLOUDFLARE

Example graphs from cAdvisor

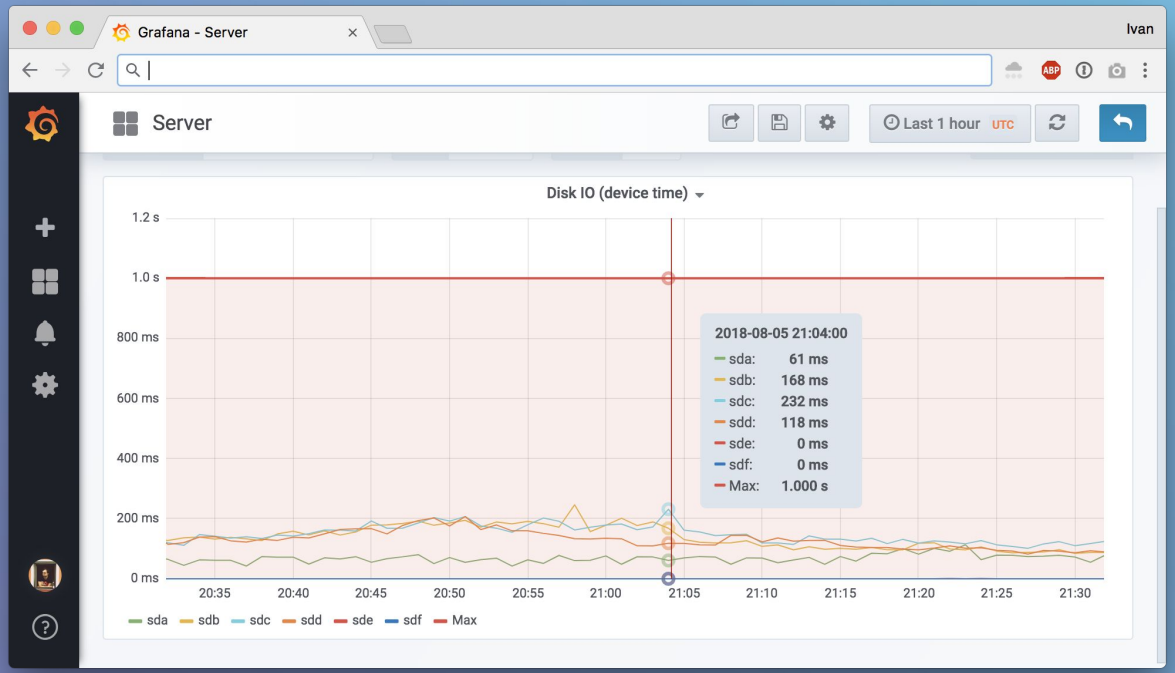


CLOUDFLARE

Counters are easy, but lack detail: e.g. IO

What's the distribution?

- Many fast IOs?
- Few slow IOs?
- Some kind of mix?
- Read vs write speed?



Histograms to the rescue

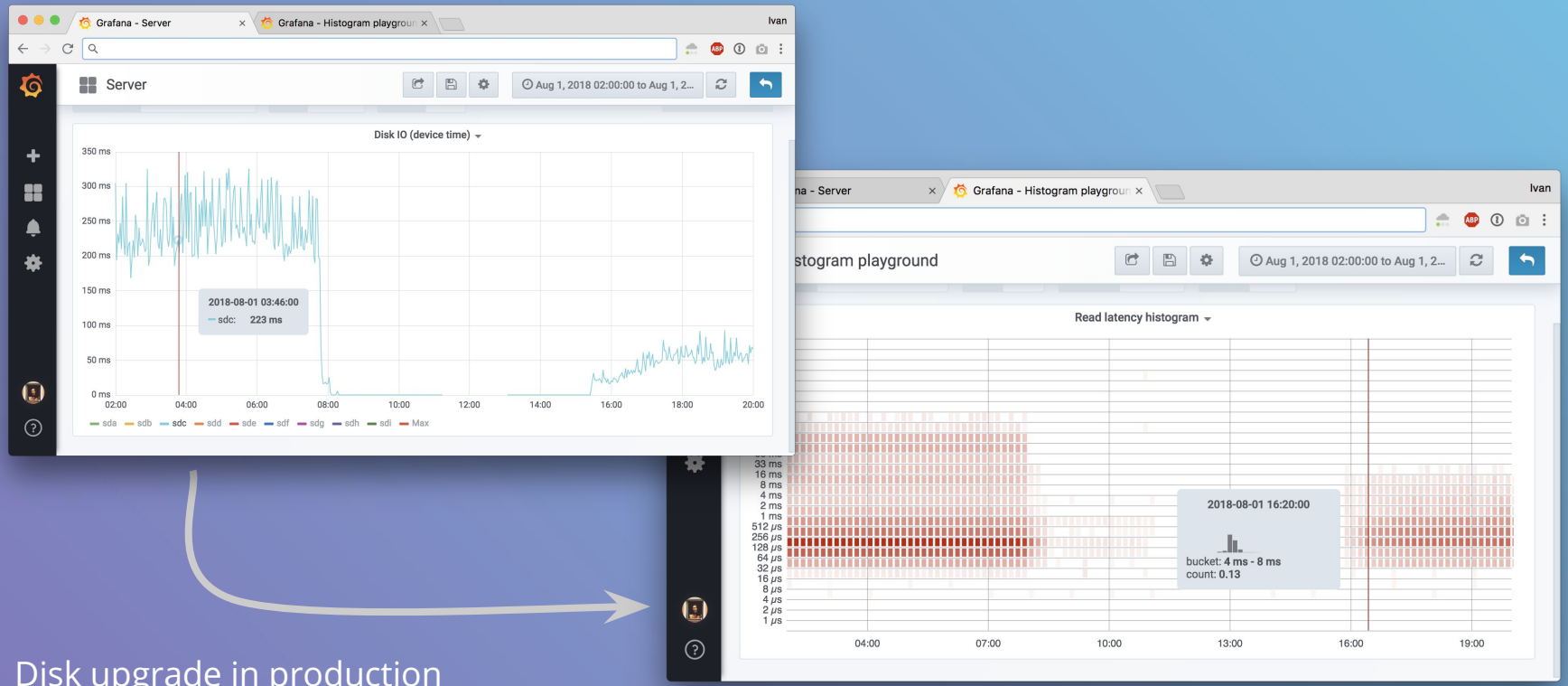
- Counter:

```
node_disk_io_time_ms{instance="foo", device="sdc"} 39251489
```

- Histogram:

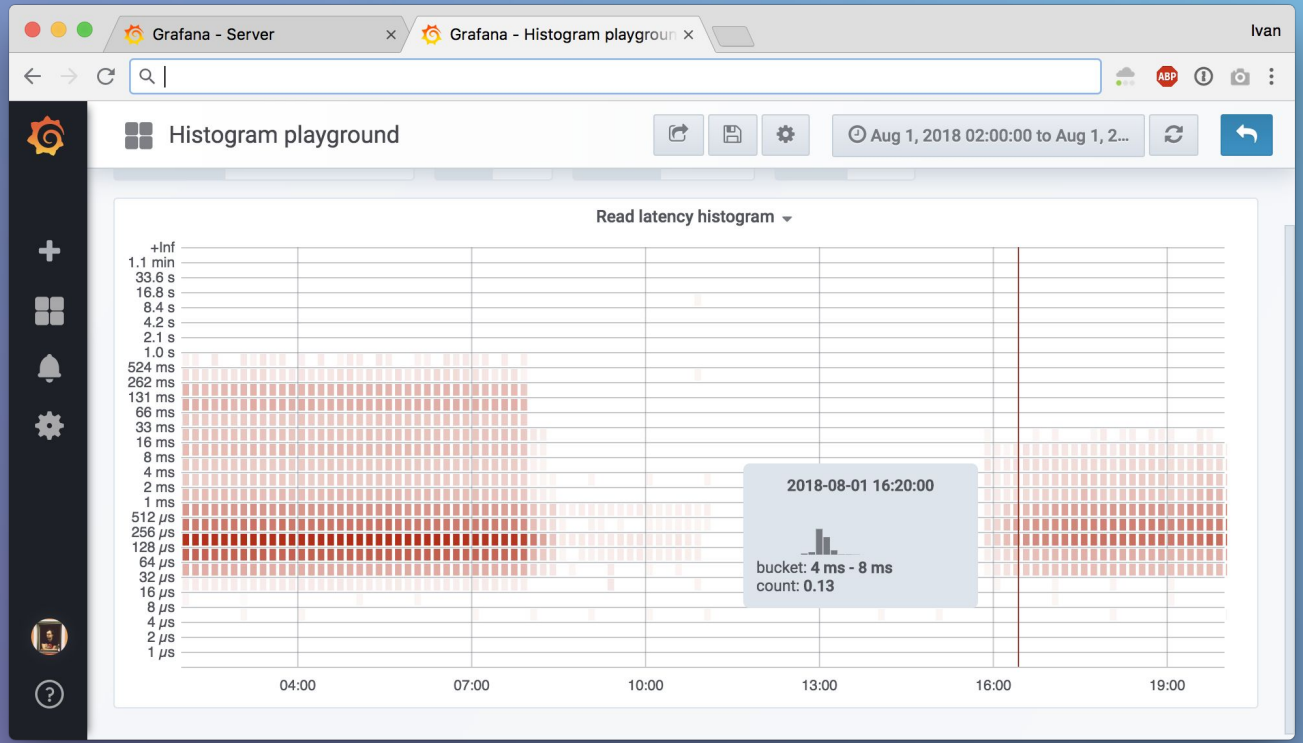
```
bio_latency_seconds_bucket{instance="foo", device="sdc", le="+Inf"} 53516704
bio_latency_seconds_bucket{instance="foo", device="sdc", le="67.108864"} 53516704
...
bio_latency_seconds_bucket{instance="foo", device="sdc", le="0.001024"} 51574285
bio_latency_seconds_bucket{instance="foo", device="sdc", le="0.000512"} 46825073
bio_latency_seconds_bucket{instance="foo", device="sdc", le="0.000256"} 33208881
bio_latency_seconds_bucket{instance="foo", device="sdc", le="0.000128"} 9037907
bio_latency_seconds_bucket{instance="foo", device="sdc", le="6.4e-05"} 239629
bio_latency_seconds_bucket{instance="foo", device="sdc", le="3.2e-05"} 132
bio_latency_seconds_bucket{instance="foo", device="sdc", le="1.6e-05"} 42
bio_latency_seconds_bucket{instance="foo", device="sdc", le="8e-06"} 29
bio_latency_seconds_bucket{instance="foo", device="sdc", le="4e-06"} 2
bio_latency_seconds_bucket{instance="foo", device="sdc", le="2e-06"} 0
```

Can be nicely visualized with new Grafana



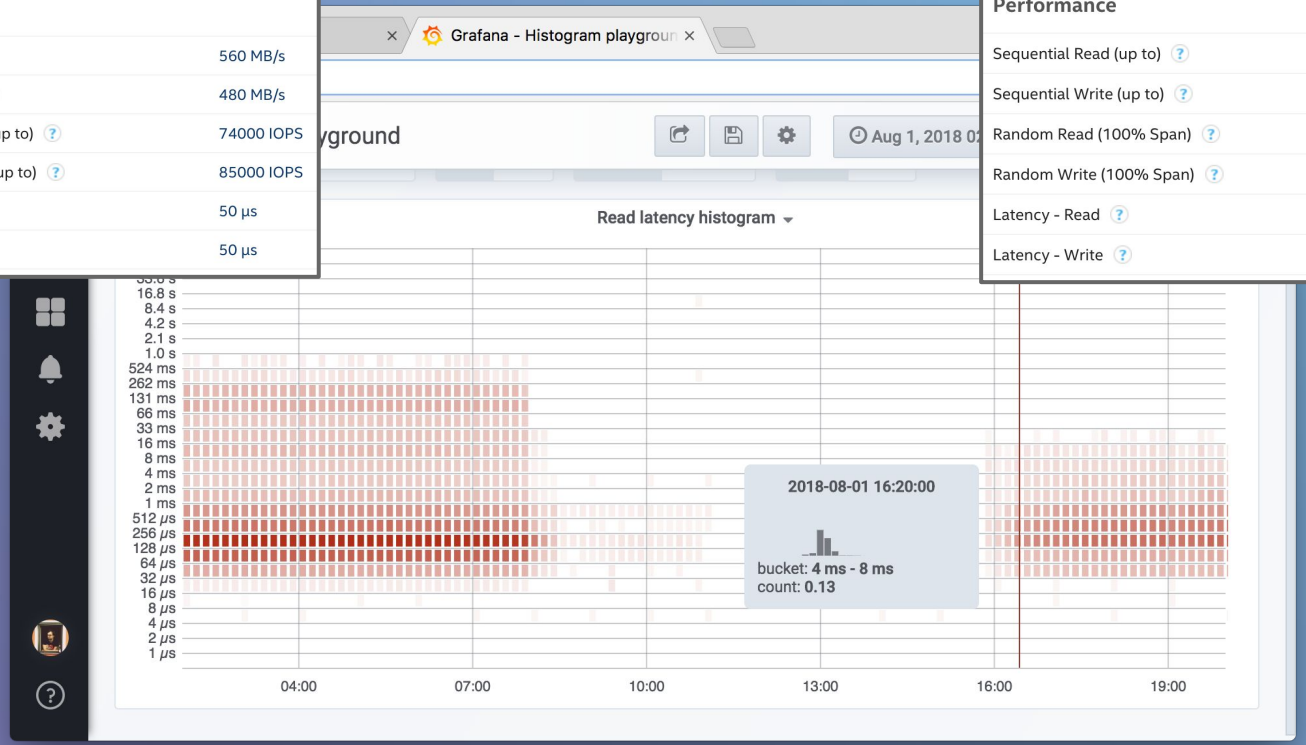
Disk upgrade in production

Larger view to see in detail



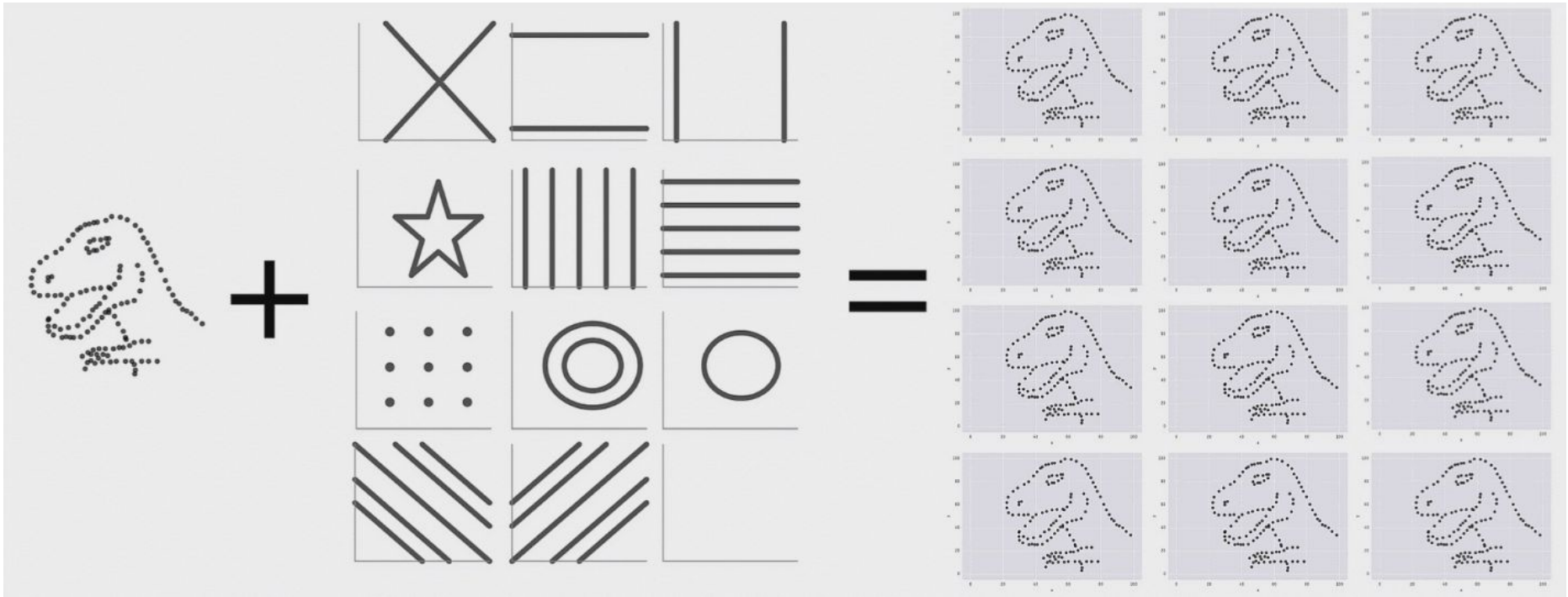
So much for holding up to spec

Performance	
Sequential Read (up to)	560 MB/s
Sequential Write (up to)	480 MB/s
Random Read (8GB Span) (up to)	74000 IOPS
Random Write (8GB Span) (up to)	85000 IOPS
Latency - Read	50 µs
Latency - Write	50 µs



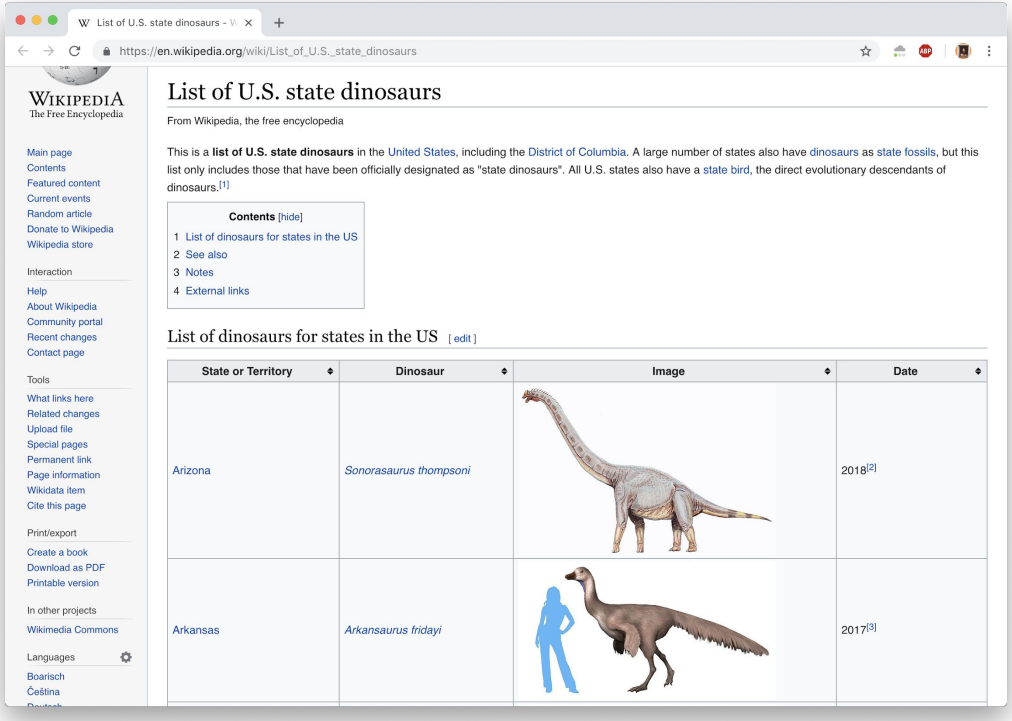
Performance	
Sequential Read (up to)	500 MB/s
Sequential Write (up to)	330 MB/s
Random Read (100% Span)	72000 IOPS
Random Write (100% Span)	20000 IOPS
Latency - Read	36 µs
Latency - Write	36 µs

Autodesk research: Datasaurus (animated)



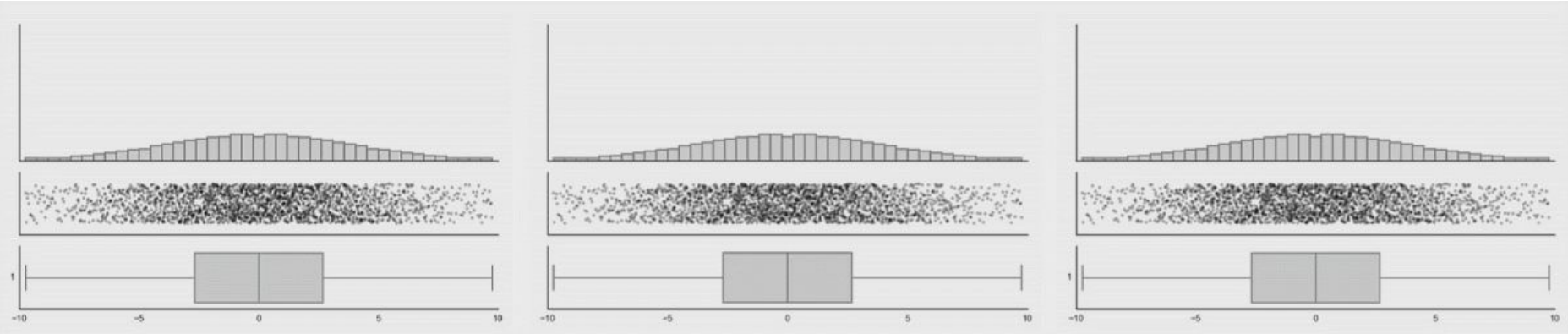
CLOUDFLARE

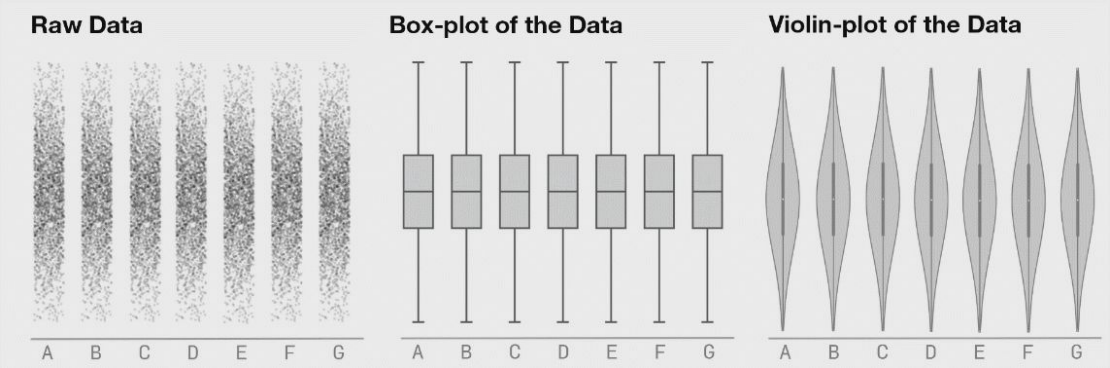
Fun fact: US states have official dinosaurs



CLOUDFLARE

Autodesk research: Datasaurus (animated)





CLOUDFLARE

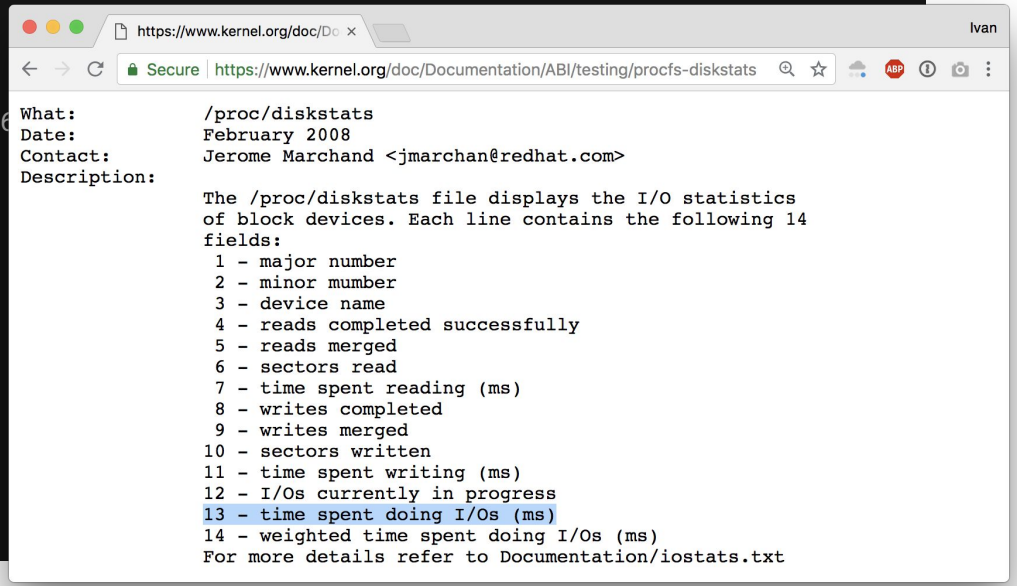
You need individual events for histograms

- Solution has to be low overhead (no blktrace)
- Solution has to be universal (not just IO tracing)
- Solution has to be supported out of the box (no modules or patches)
- Solution has to be safe (no kernel crashes or loops)

Monitoring, current

```
# counter-based, field 10 of https://www.kernel.org/doc/Documentation/iostats.txt
node_disk_io_time_ms{device="sdc"} 39251489

# counter-based, field 11
node_disk_io_time_weighted{device="sdc"} 110260706
```



CLOUDFLARE

Monitoring, future

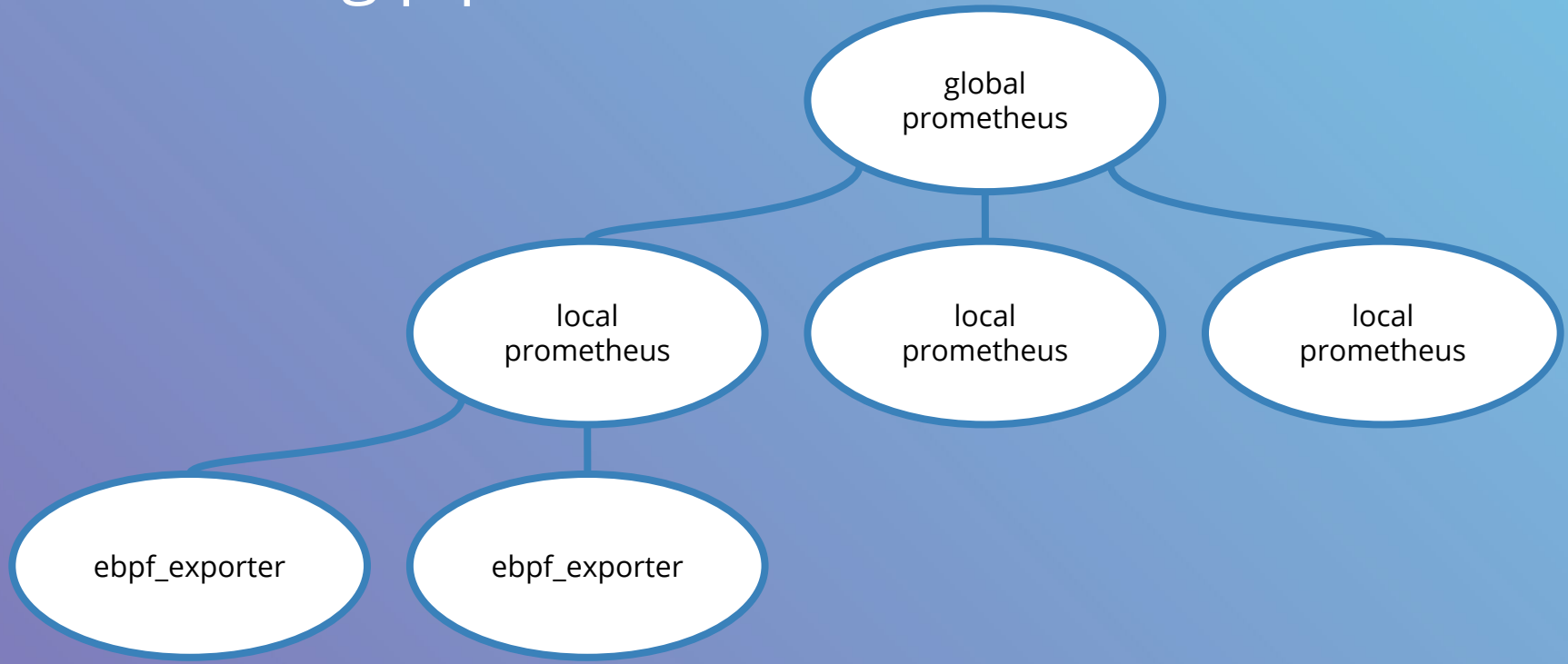
```
# histogram-based
bio_latency_seconds_bucket{device="sdc", le="+Inf"} 53516704
bio_latency_seconds_bucket{device="sdc", le="67.108864"} 53516704
...
bio_latency_seconds_bucket{device="sdc", le="0.000512"} 46825073
bio_latency_seconds_bucket{device="sdc", le="0.000256"} 33208881
bio_latency_seconds_bucket{device="sdc", le="0.000128"} 9037907
bio_latency_seconds_bucket{device="sdc", le="6.4e-05"} 239629
bio_latency_seconds_bucket{device="sdc", le="3.2e-05"} 132
bio_latency_seconds_bucket{device="sdc", le="1.6e-05"} 42
bio_latency_seconds_bucket{device="sdc", le="8e-06"} 29
bio_latency_seconds_bucket{device="sdc", le="4e-06"} 2
bio_latency_seconds_bucket{device="sdc", le="2e-06"} 0
```

The future is now

- ebpf_exporter
- Uses iovisor/gobpf and prometheus/client_golang
- Configured via YAML

- https://github.com/cloudflare/ebpf_exporter

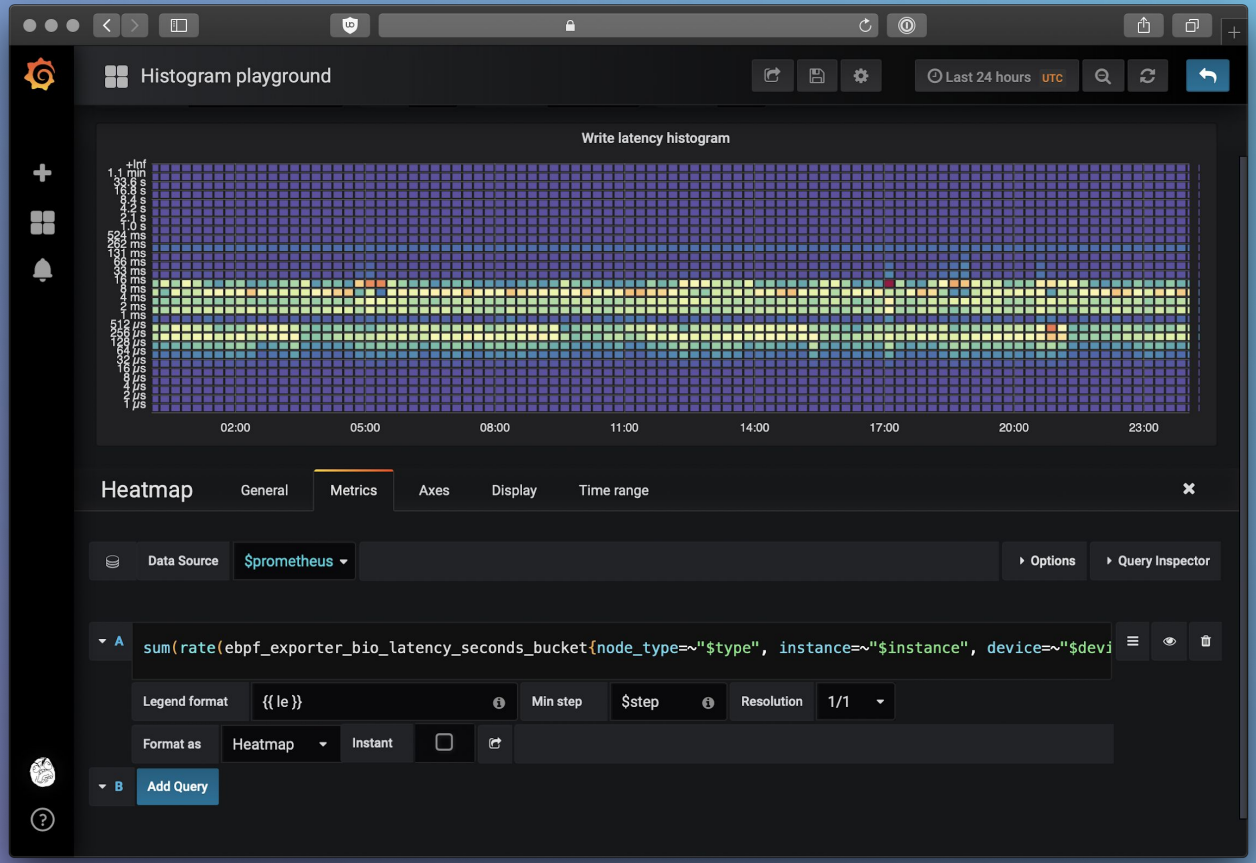
Monitoring pipeline



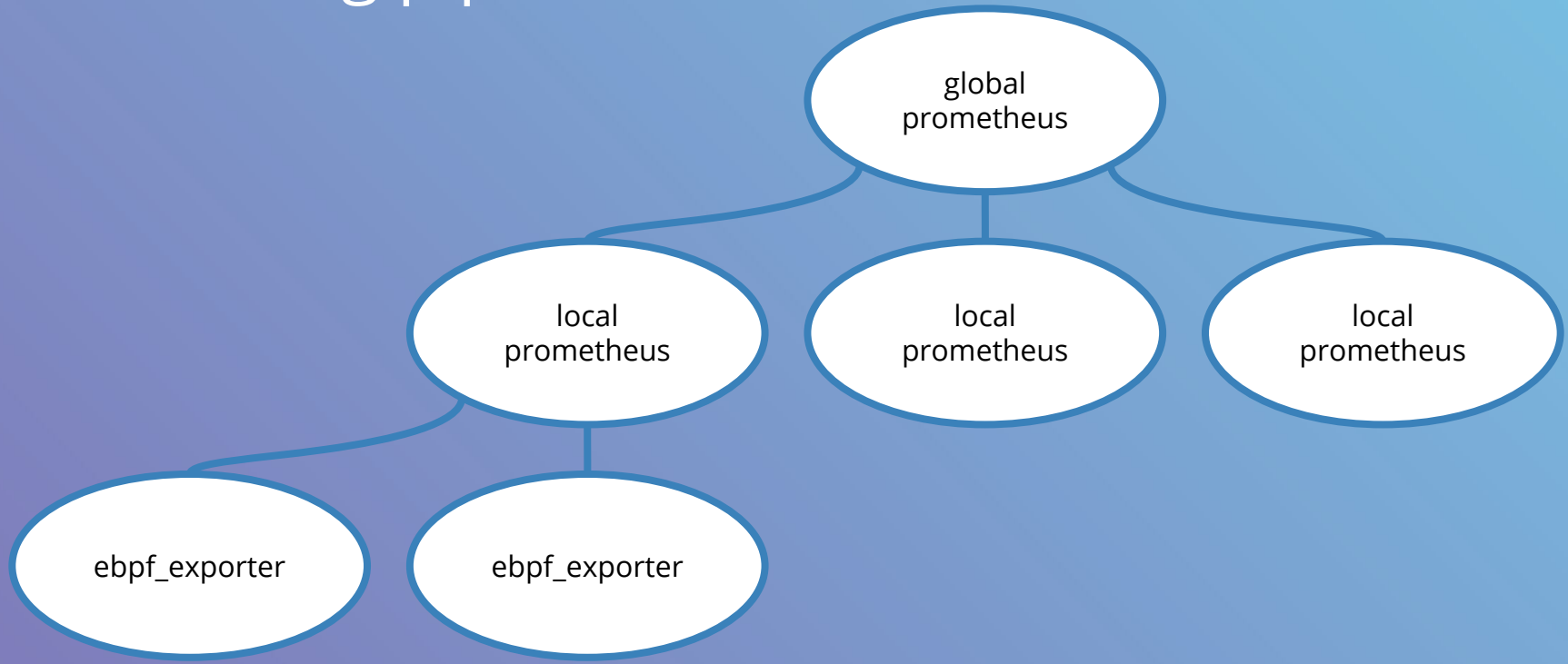
Monitoring, future

```
# histogram-based
bio_latency_seconds_bucket{device="sdc", le="+Inf"} 53516704
bio_latency_seconds_bucket{device="sdc", le="67.108864"} 53516704
...
bio_latency_seconds_bucket{device="sdc", le="0.000512"} 46825073
bio_latency_seconds_bucket{device="sdc", le="0.000256"} 33208881
bio_latency_seconds_bucket{device="sdc", le="0.000128"} 9037907
bio_latency_seconds_bucket{device="sdc", le="6.4e-05"} 239629
bio_latency_seconds_bucket{device="sdc", le="3.2e-05"} 132
bio_latency_seconds_bucket{device="sdc", le="1.6e-05"} 42
bio_latency_seconds_bucket{device="sdc", le="8e-06"} 29
bio_latency_seconds_bucket{device="sdc", le="4e-06"} 2
bio_latency_seconds_bucket{device="sdc", le="2e-06"} 0
```

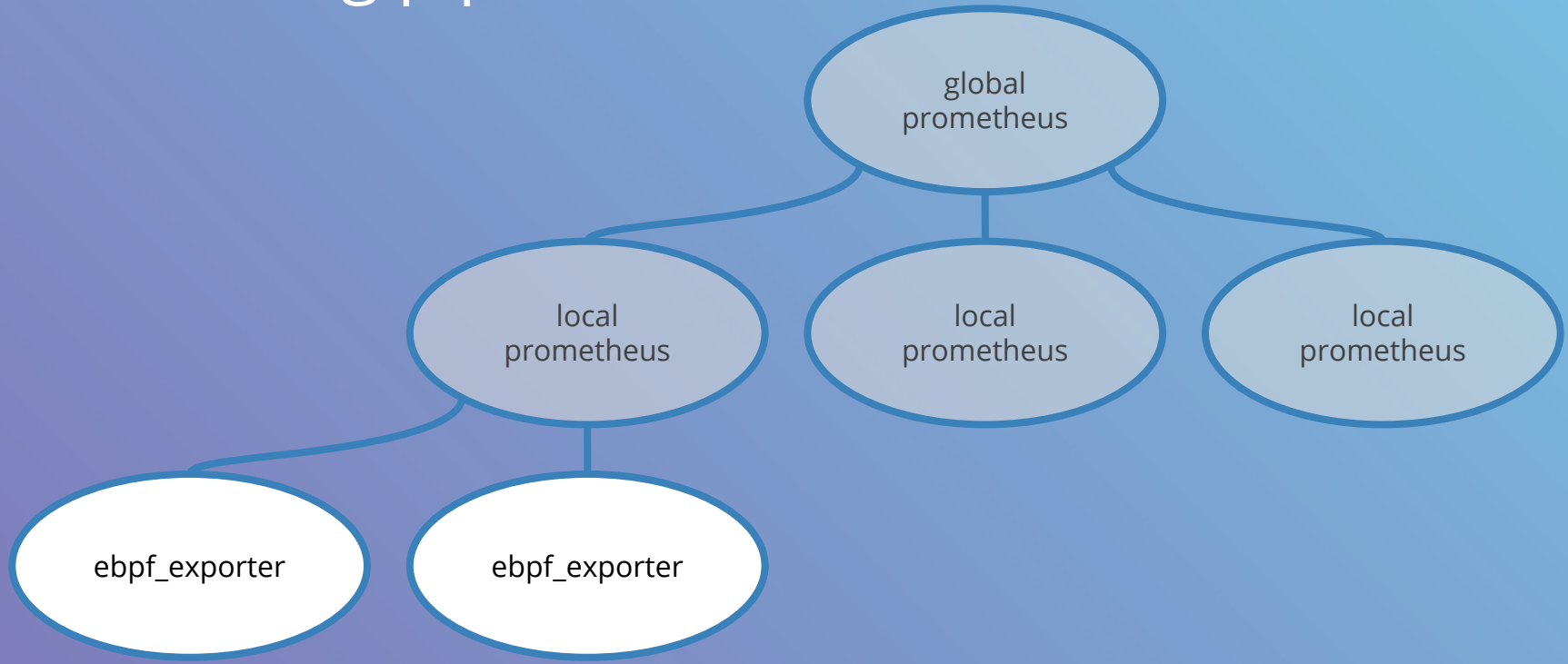
Graphs!


Histogram playground
Last 24 hours UTC
Write latency histogram
+Inf
1.1 min
33.6 s
16.8 s
8.4 s
4.2 s
2.1 s
1.0 s
524 ms
262 ms
131 ms
66 ms
33 ms
16 ms
8 ms
4 ms
2 ms
1 ms
512 µs
256 µs
128 µs
64 µs
32 µs
16 µs
8 µs
4 µs
2 µs
1 µs
02:00
05:00
08:00
11:00
14:00
17:00
20:00
23:00
Heatmap
General
Metrics
Axes
Display
Time range
Data Source
$prometheus
Options
Query Inspector
A
sum(rate(ebpf_exporter_bio_latency_seconds_bucket{node_type=~"$type", instance=~"$instance", device=~"$devi
Legend format
{{ le }}
Min step
$step
Resolution
1/1
Format as
Heatmap
Instant
B
Add Query

Monitoring pipeline



Monitoring pipeline



Platform setup

```
ubuntu:~$ # bionic
ubuntu:~$ cat /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=18.04
DISTRIB_CODENAME=bionic
DISTRIB_DESCRIPTION="Ubuntu 18.04.1 LTS"
```

CLOUDFLARE

Kernel configuration

```
ubuntu:~$ grep -E 'CONFIG[^=]*_BPF' /boot/config-$(uname -r)
CONFIG_BPF=y
CONFIG_BPF_SYSCALL=y
CONFIG_NET_CLS_BPF=m
CONFIG_NET_ACT_BPF=m
CONFIG_BPF_JIT=y
CONFIG_BPF_EVENTS=y
```

Dependencies

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

# nice to have utilities, like `perf`
ubuntu:~$ sudo apt-get install -y linux-tools-$(uname -r)

# ebpf_exporter dependencies
ubuntu:~$ sudo apt-get install -y git golang
```

CLOUDFLARE

Installing ebpf_exporter

```
ubuntu:~$ mkdir /tmp/ebpf_exporter
ubuntu:~$ cd /tmp/epbf_exporter
ubuntu:/tmp/ebpf_exporter$ GOPATH=$(pwd) go get github.com/cloudflare/ebpf_exporter/...
```

Build on BCC tools' biolatency

```
ubuntu:~$ sudo /usr/share/bcc/tools/biolatency 10 1
Tracing block device I/O... Hit Ctrl-C to end.

     usecs               : count     distribution
...
        32 -> 63         : 0        |                                        |
        64 -> 127        : 2        |*                                       |
       128 -> 255        : 24       |***********************                 |
       256 -> 511        : 41       |****************************************|
       512 -> 1023       : 7        |******                                  |
      1024 -> 2047       : 1        |                                        |
```

CLOUDFLARE

Generate exportable metrics

```
ubuntu:/tmp/ebpf_exporter$ sudo ./bin/ebpf_exporter \
    --config.file=src/github.com/cloudflare/ebpf_exporter/examples/bio-tracepoints.yaml
2018/10/19 23:38:13 Starting with 1 programs found in the config
2018/10/19 23:38:13 Listening on :9435


ubuntu@ubuntu:~$ wget -qO - http://localhost:9435/metrics | \
    grep '^ebpf_exporter_bio_latency_seconds_bucket' | head -n 10 | tail -n -5
ebpf_exporter_bio_latency_seconds_bucket{device="sda",operation="write",le="3.2e-05"} 0
ebpf_exporter_bio_latency_seconds_bucket{device="sda",operation="write",le="6.4e-05"} 7
ebpf_exporter_bio_latency_seconds_bucket{device="sda",operation="write",le="0.000128"} 41
ebpf_exporter_bio_latency_seconds_bucket{device="sda",operation="write",le="0.000256"} 106
ebpf_exporter_bio_latency_seconds_bucket{device="sda",operation="write",le="0.000512"} 141
```

CLOUDFLARE

Example 1: timers

```
ubuntu@ubuntu:/tmp/ebpf_exporter$ cat src/github.com/cloudflare/ebpf_exporter/examples/timers.yaml
programs:
  - name: timers
    metrics:
      counters:
        - name: timer_start_total
          help: Timers fired in the kernel
          table: counts
          labels:
            - name: function
              size: 8
              decoders:
                - name: ksym
    tracepoints:
      timer:timer_start: tracepoint__timer__timer_start
    code: |
      BPF_HASH(counts, u64);

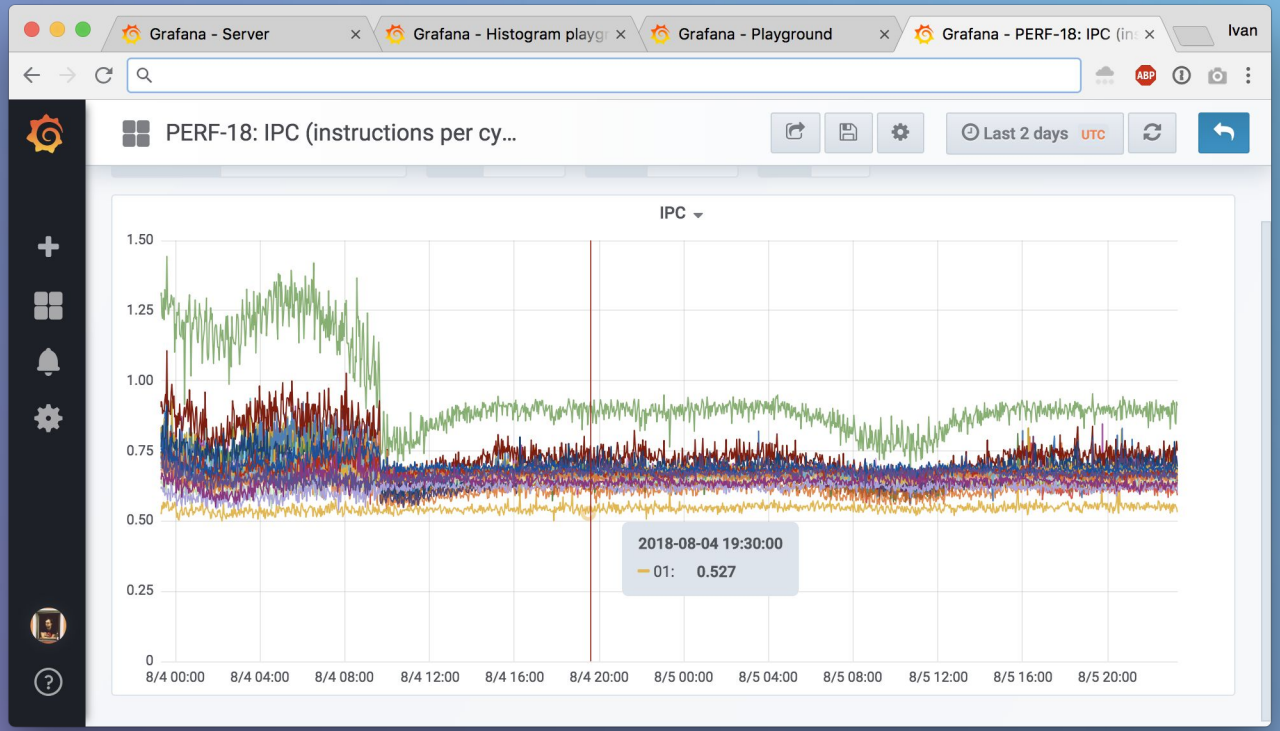
      // Generates function tracepoint__timer__timer_start
      TRACEPOINT_PROBE(timer, timer_start) {
          counts.increment((u64) args->function);
          return 0;
      }
```

CLOUDFLARE

Example 1: timers

```
ubuntu@ubuntu:/tmp/ebpf_exporter$ wget -qO - http://localhost:9436/metrics \
    | grep '^ebpf_exporter_timer_start_total' | head -n 10
ebpf_exporter_timer_start_total{function="__prandom_timer"} 42
ebpf_exporter_timer_start_total{function="blk_rq_timed_out_timer"} 513
ebpf_exporter_timer_start_total{function="clocksource_watchdog"} 4924
ebpf_exporter_timer_start_total{function="commit_timeout"} 319
ebpf_exporter_timer_start_total{function="delayed_work_timer_fn"} 5268
ebpf_exporter_timer_start_total{function="idle_worker_timeout"} 7
ebpf_exporter_timer_start_total{function="io_watchdog_func"} 8550
ebpf_exporter_timer_start_total{function="mce_timer_fn"} 8
ebpf_exporter_timer_start_total{function="neigh_timer_handler"} 303
ebpf_exporter_timer_start_total{function="pool_mayday_timeout"} 7
```

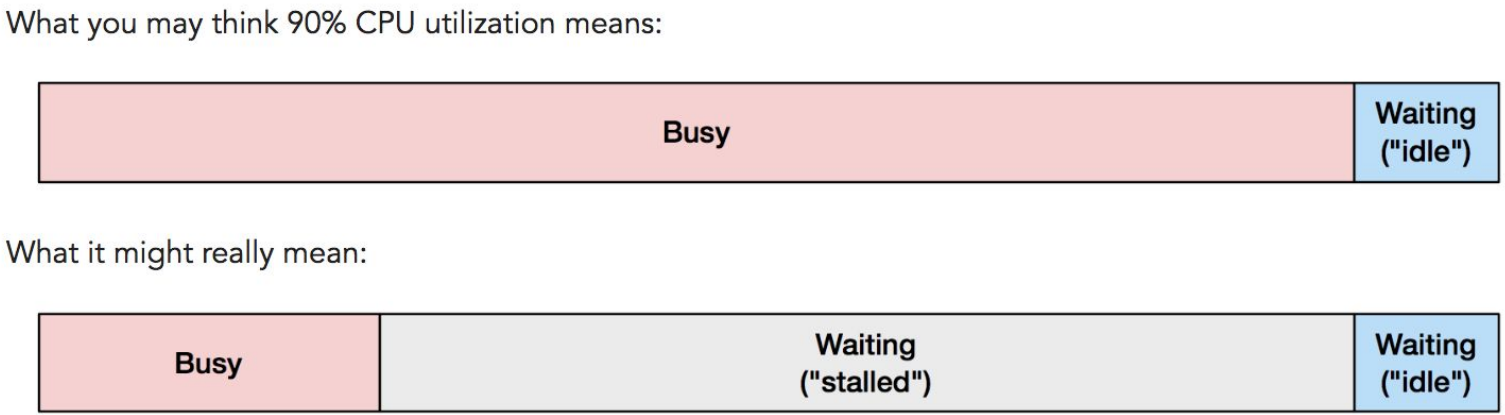
Other bundled examples: IPC



Why can instructions per cycle be useful?

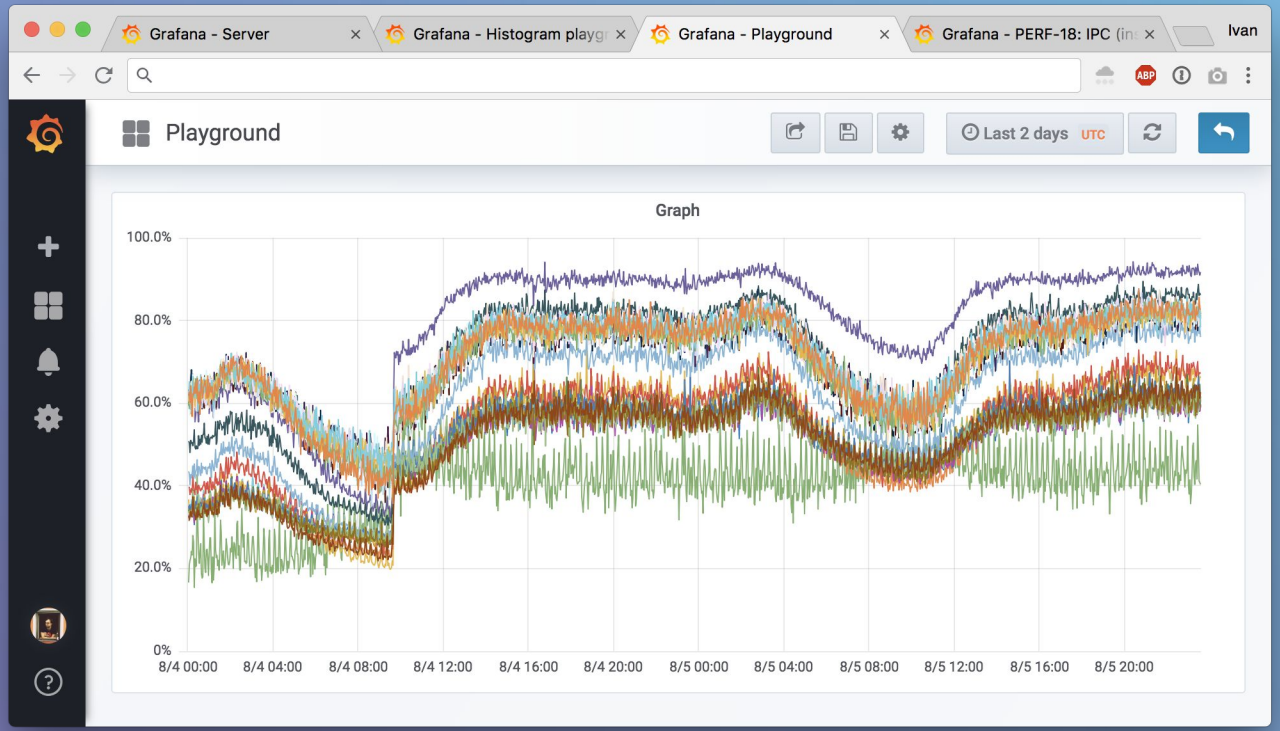
See Brendan Gregg's blog post: "CPU Utilization is Wrong".

TL;DR: same CPU% may mean different throughput in terms of CPU work done. IPC helps to understand the workload better.



CLOUDFLARE

Other bundled examples: LLC (L3 Cache)



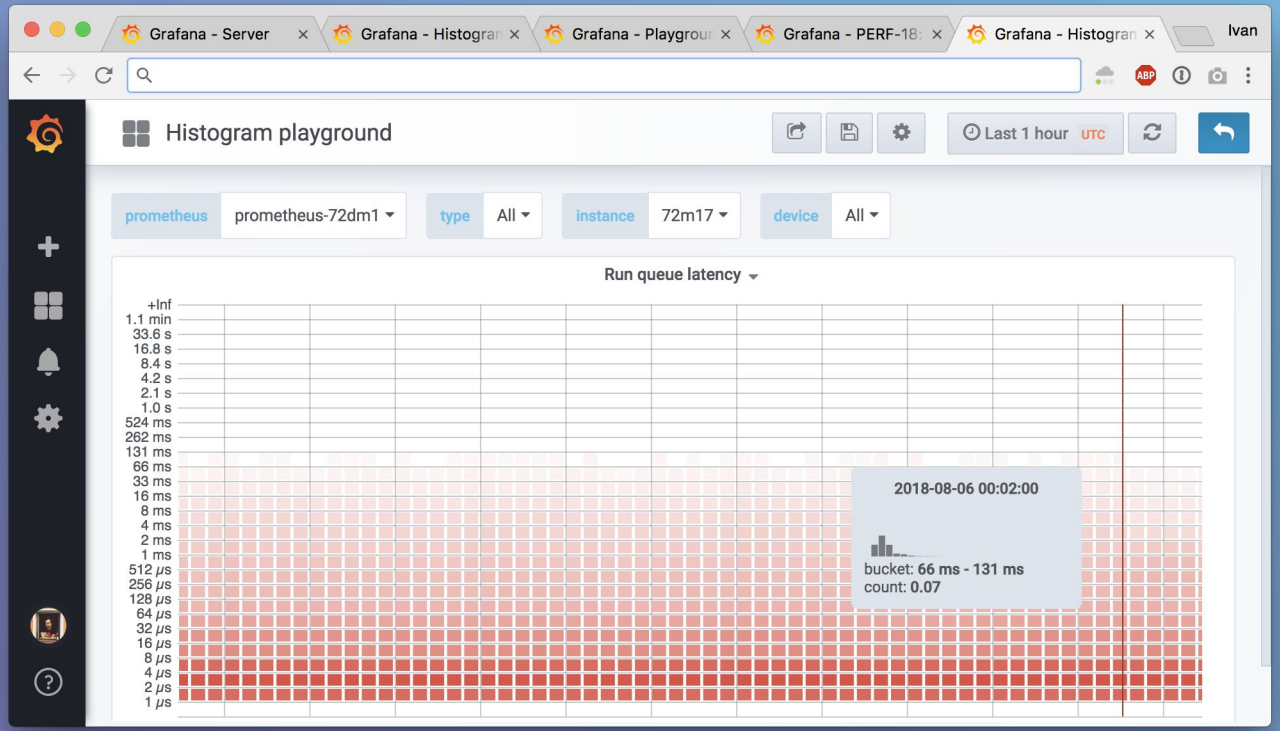
Why can LLC hit rate be useful?

You can answer questions like:

- Do I need to pay more for a CPU with bigger L3 cache?
- How does having more cores affect my workload?

LLC hit rate usually follows IPC patterns as well.

Other bundled examples: run queue delay



Why can run queue latency be useful?

See: “perf sched for Linux CPU scheduler analysis” by Brendan G.

You can see how contended your system is, how effective is the scheduler and how changing sysctls can affect that.

It’s surprising how high delay is by default.

From Scylla: “Reducing latency spikes by tuning the CPU scheduler”.

How many things can you measure?

Numbers are from a production Cloudflare machine running Linux 4.14:

- 501 hardware events and counters:
  - `sudo perf list hw cache pmu | grep '^  [a-z]' | wc -l`
- 1853 tracepoints:
  - `sudo perf list tracepoint | grep '^  [a-z]' | wc -l`
- Any non-inlined kernel function (there's like a bazillion of them)
- No support for USDT or uprobes yet

Run through runqlat example

eBPF overhead numbers for kprobes

You should always measure yourself (system CPU is the metric).

Here's what we've measured for getpid() syscall:

Case	ns/op	overhead ns/op	ops/s	overhead percent
no probe	316	0	3,164,556	0%
simple	424	108	2,358,490	34%
complex	647	331	1,545,595	105%

CLOUDFLARE

Where should you run ebpf_exporter

Anywhere where overhead is worth it.

- Simple programs can run anywhere
- Complex programs (run queue latency) can be gated to:
  - Canary machines where you test upgrades
  - Timeline around updates

At Cloudflare we do exactly this, except we use canary datacenters.

Thank you

- https://github.com/cloudflare/ebpf_exporter
- Link to this deck, which supplements
  - Debugging Linux issues with eBPF

- Always looking for interested people