sysdig

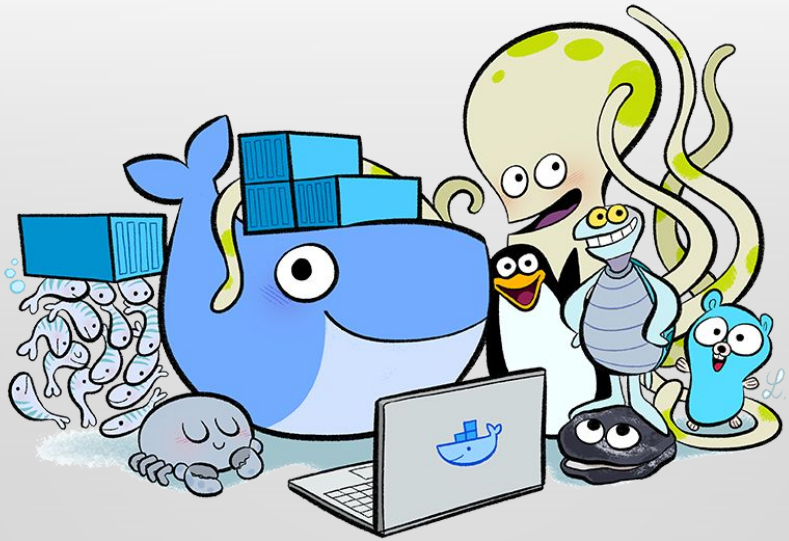
50 Shades of System Calls

Jorge Salamero Sanz
@bencerillo @sysdig

How Linux SysAdmins (used to) do (performance) troubleshooting?

- `uptime`
- `free -m`
- `ps aux`
- `vmstat 1`
- `netstat -putan`
- `top`
- `tcpdump`

But something changed



How developers are doing (distributed) tracing and profiling?

- OpenTracing and alternatives:
  - Zipkin, Dapper, HTrace, etc
- Commercial:
  - Lightstep, New Relic, AppDynamics, Dyn, SPM
- Self-brewed and hacks:
  - statsd, JMX
  - print
  - logs can bite you in the ass

What if we had something...

- Open source
- Simple and easy to use (trade-off vs eBPF/bcc)
- Lightweight
- Could work everywhere (including containers)

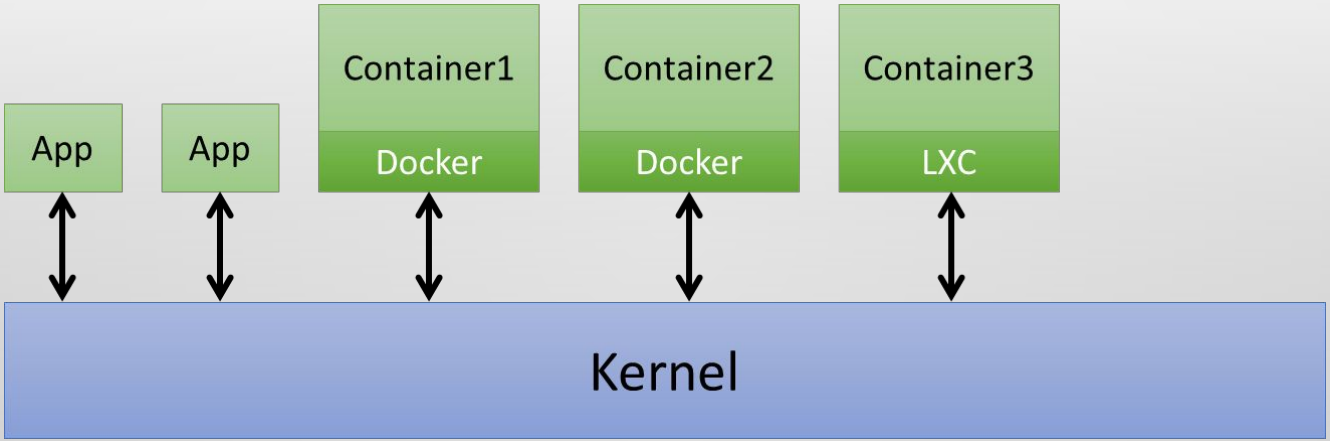
Sysdig

Open Source troubleshooting with native container support

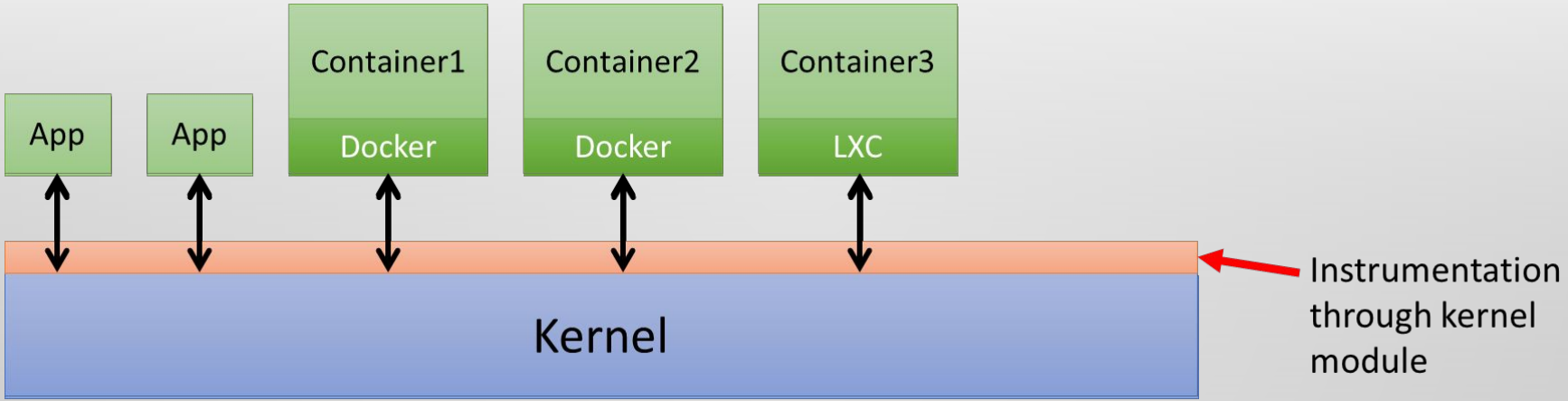
(`htop`, `vmstat`, `netstat`, `lsof`, `tcpdump`…)

Monitoring, alerting, troubleshooting tool for Docker, Kubernetes

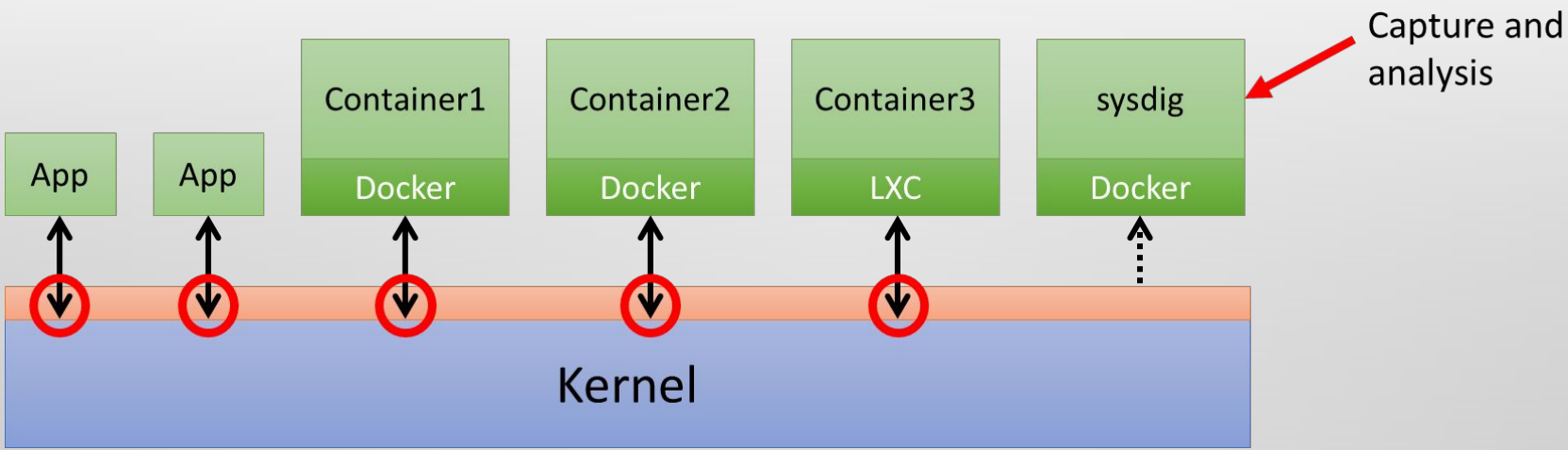
Kernel-level instrumentation



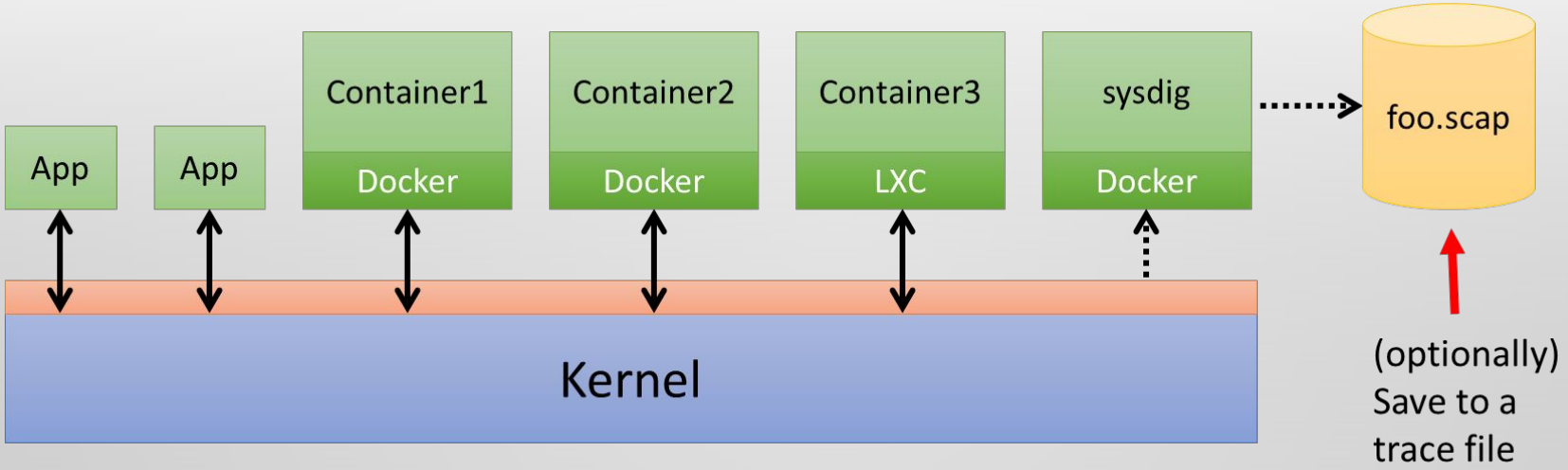
Kernel-level instrumentation



Kernel-level instrumentation



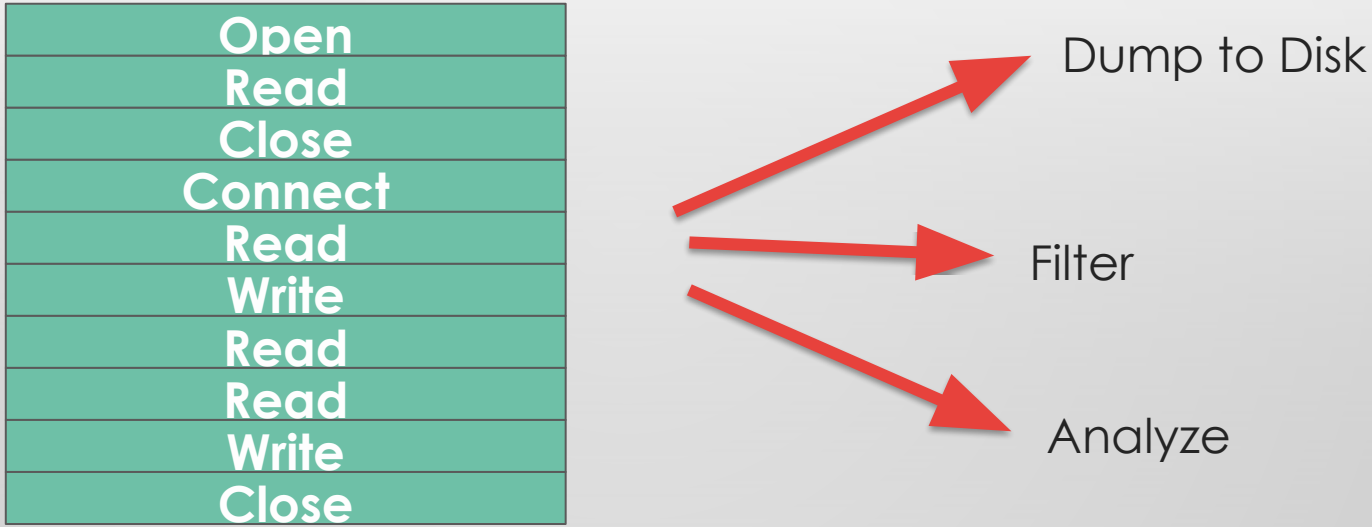
Kernel-level instrumentation



Sysdig tool

- Capture system events, filter and run scripts
- Trace dumps for analysis
- Container support
- CLI or curses interface

Event stream



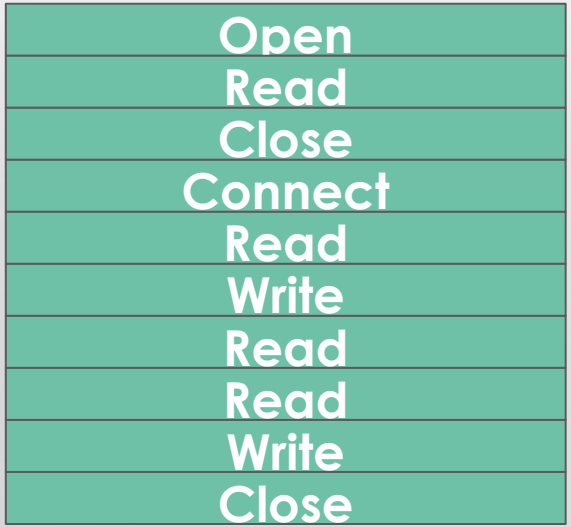
Sysdig Tracers:

System Call tracing

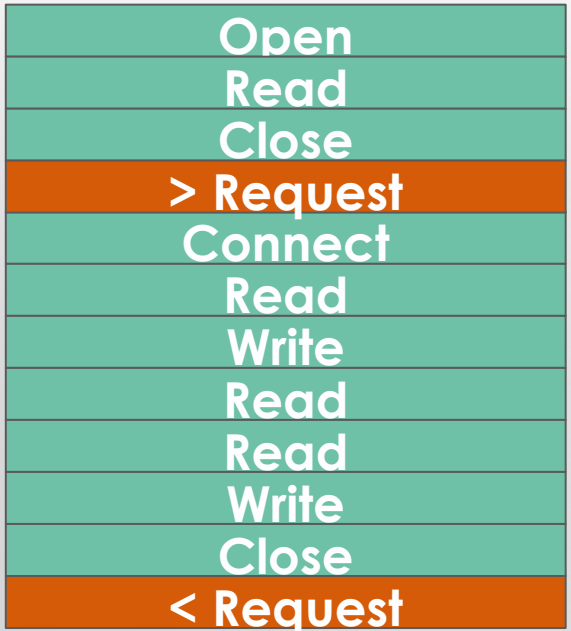
Sysdig Tracers: System Call tracing

- Inject markers inside Sysdig event stream
- Mark being and end of **spans**
  - Function calls
  - Network request
  - Arbitrary piece of code
- From any language (write to /dev/null)
- Low overhead (<1us)

Event stream

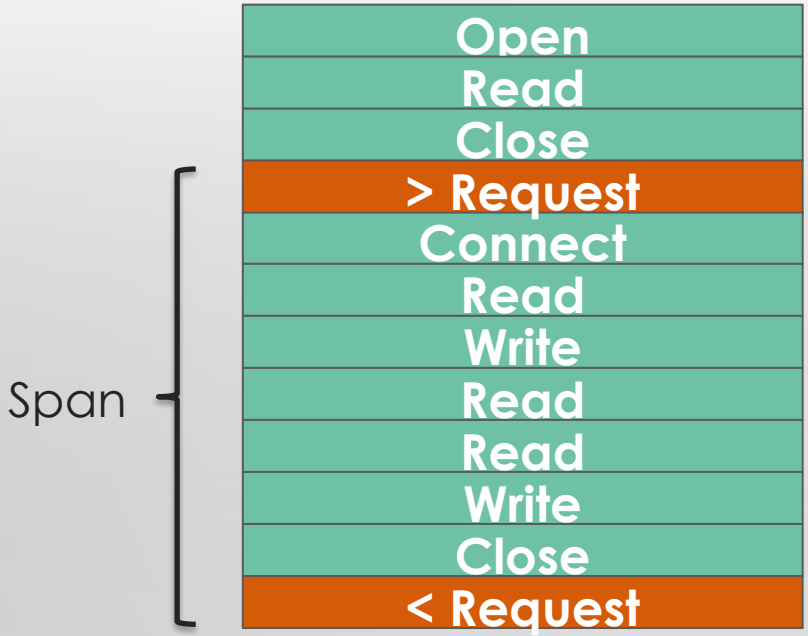


Event stream

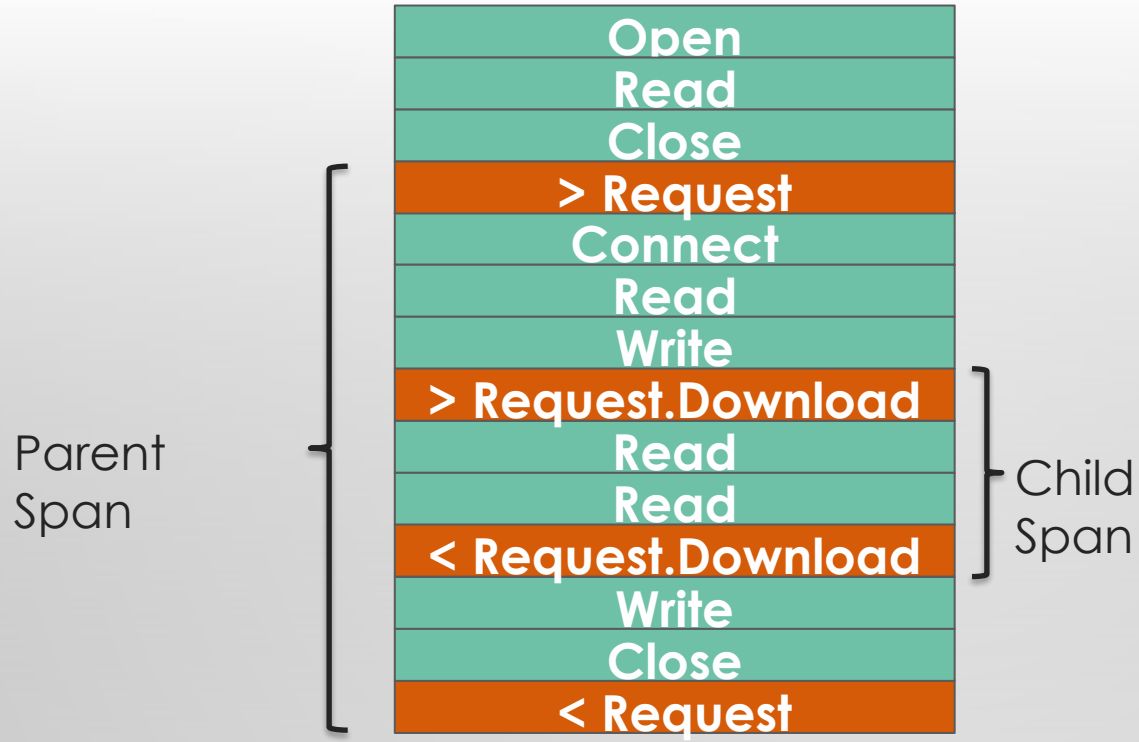


Event stream

Event stream



Event stream



Format

Simple format:

```
<dir>:<ID>:<tag1>.<tag2>...:<arg1>=<val1>,<arg2>=<val2>...
```

JSON Format:

```
["<direction>", <ID>, ["tag1", "tag2"...],
[{"arg1":"val1"}, {"arg2":"val2"}...]]
```

Example

```
while :
do
    # Start a trace named 'website-latency'
    echo ">::website-latency::" > /dev/null
    # Download the sysdig home page
    curl -s http://sysdig.org > /dev/null
    # End the the trace
    echo "<::website-latency::" > /dev/null
done
```

Fun things you can do

- Measure latencies in your code
- Save and filter traces with sysdig
- Analyze traces with csysdig
- Inspect system activity inside execution spans
- Trace-aware log monitoring
- Export trace latencies using statsd

Demo time!

Further reading

- Documentation:
  - https://github.com/draios/sysdig/wiki/Tracers
  - https://sysdig.com/blog/tracking-down-application-bottlenecks-with-tracers/
- Integrations
  - Python https://github.com/draios/tracer-py
  - Node: https://https://github.com/tj/node-trace
  - Go: https://github.com/tj/go-trace

Thank You!

@bencerillo
@sysdig
sysdig.com | sysdig.org