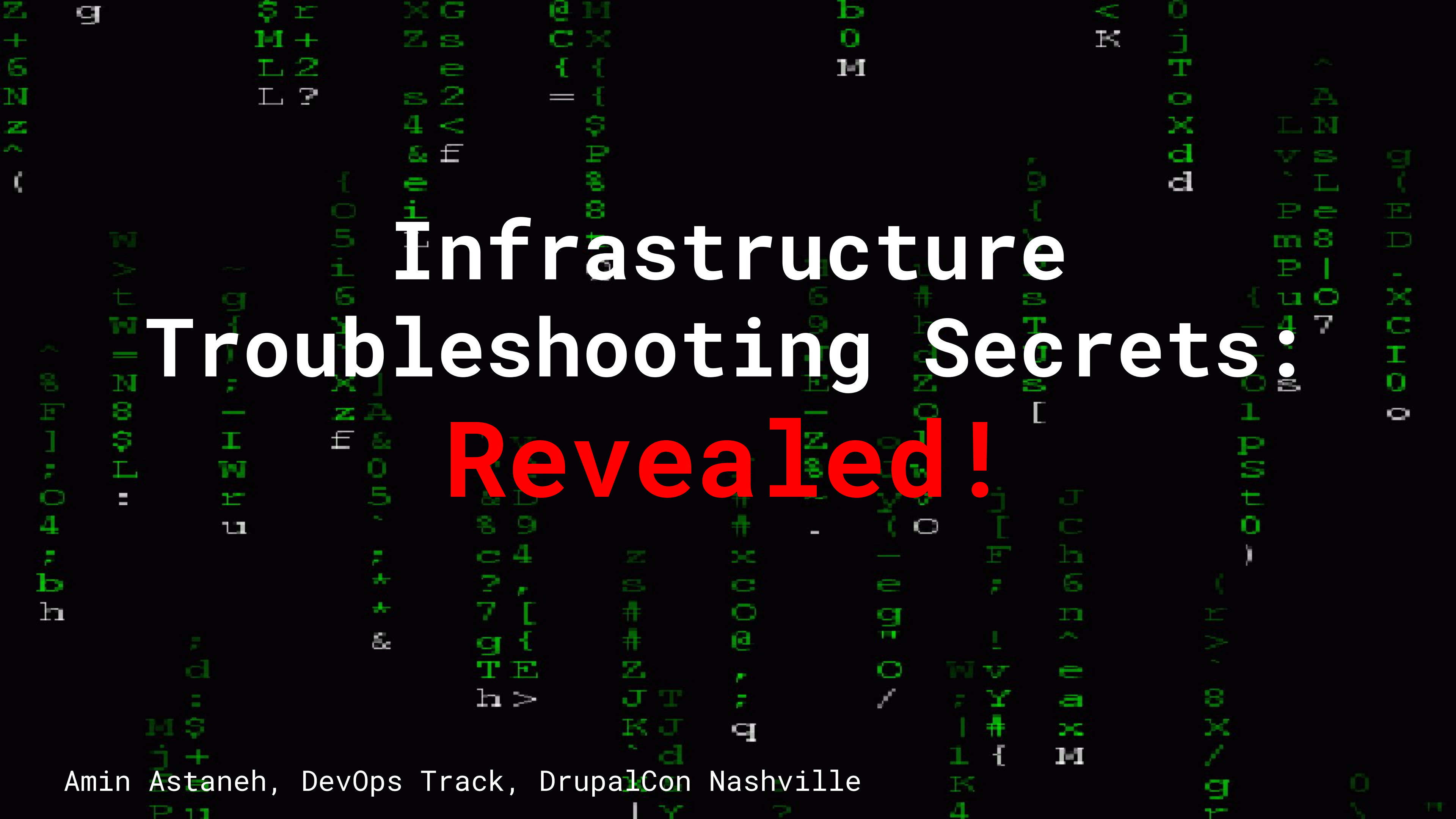


Infrastructure Troubleshooting Secrets: Revealed!

Amin Astaneh, DevOps Track, DrupalCon Nashville

Who Am I?

- Amin Astaneh
- Senior Manager, SRE at **Acquia**
- Served on Ops Team for 5 years
- Been on-call countless times
- Been paged countless times
- Heavily contributed to incident response process and tooling
- Built SRE competency, DevOps initiatives for 2 years

Agenda

- Intro
- The USE Method
- Hardware Resources
- Software Resources
- Process Introspection
- Outage Scenarios

DEF
ABC
TUV
WXY

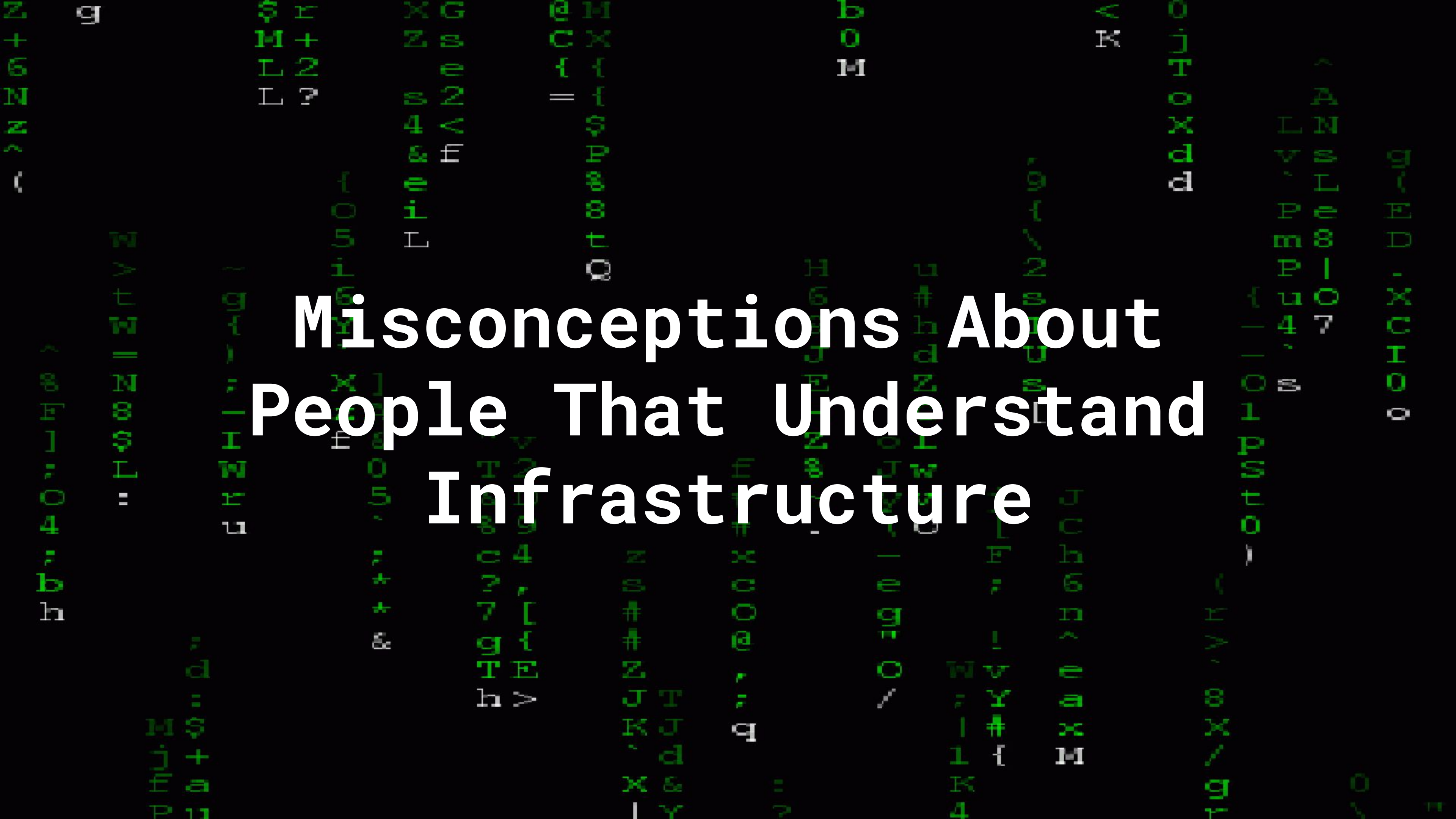
I have no idea what I'm doing
gifbin.com

Presentation Objectives

- Gain a basic understanding of the infrastructure level
- Learn a simple set of processes and tools to gather information about your infrastructure
- Learn how these tools can be used to identify current pain points in your Drupal availability/performance

5 years of Ops experience packed into less than 1 hour!

Slides will be uploaded after the presentation!



Misconceptions About People That Understand Infrastructure

The Big Secret

- They are **HUMAN**
- They have **tools**
- They have **processes**
- They have **heuristics** based on past experience

You can learn what they know!

Before We Begin

- **LAMP (GNU/Linux)**
- You know CLI basics
- You have SSH access to your infrastructure

The USE Method

Origin of USE Method

Brendan Gregg, Performance Engineer at Netflix:

“I developed the USE Method to teach others how to solve common performance issues quickly, without overlooking important areas.. it is intended to be **simple**, **straightforward**, **complete**, and **fast**.”

http://www.brendangregg.com/usemethod.html

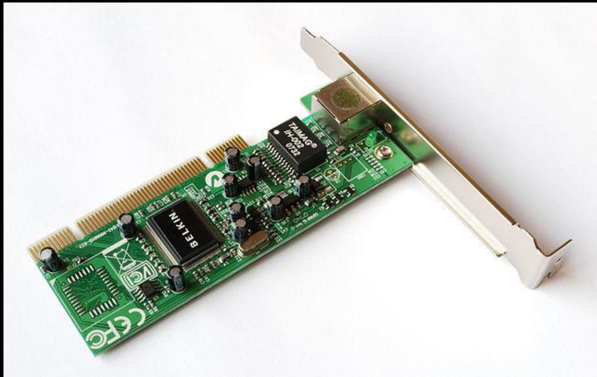
The USE Method

For every resource, check:

- **U**tilization
- **S**aturation
- **E**rrors

Resources

- All **physical server** functional components
  - CPU(s), Memory, Disk(s), Network Adapter(s)
- All **software** functional components
  - PHP Proc Pool, MySQL innodb_buffer_pool, Varnish cache
- All **OS** functional components
  - Max processes, max open files, max tcp connections



Utilization

The average time that a resource was busy doing work.

Usually represented as a percentage over an interval.

Eg: 75% of available memory was being used on Server X over the last 5 seconds.

Saturation

The degree to which the resource has extra work which it can't service, often queued.

Eg: queue_wait values in the Drupal request log are increasing due to all PHP processes handling requests.

This can be measured or observed via other signals (logs, error messages, etc)

Errors

The total count of a resource demonstrating that it is not functioning as designed or intended (error events).

Eg: The CLI printed ‘Input/output error’ when I tried to read a file from disk.

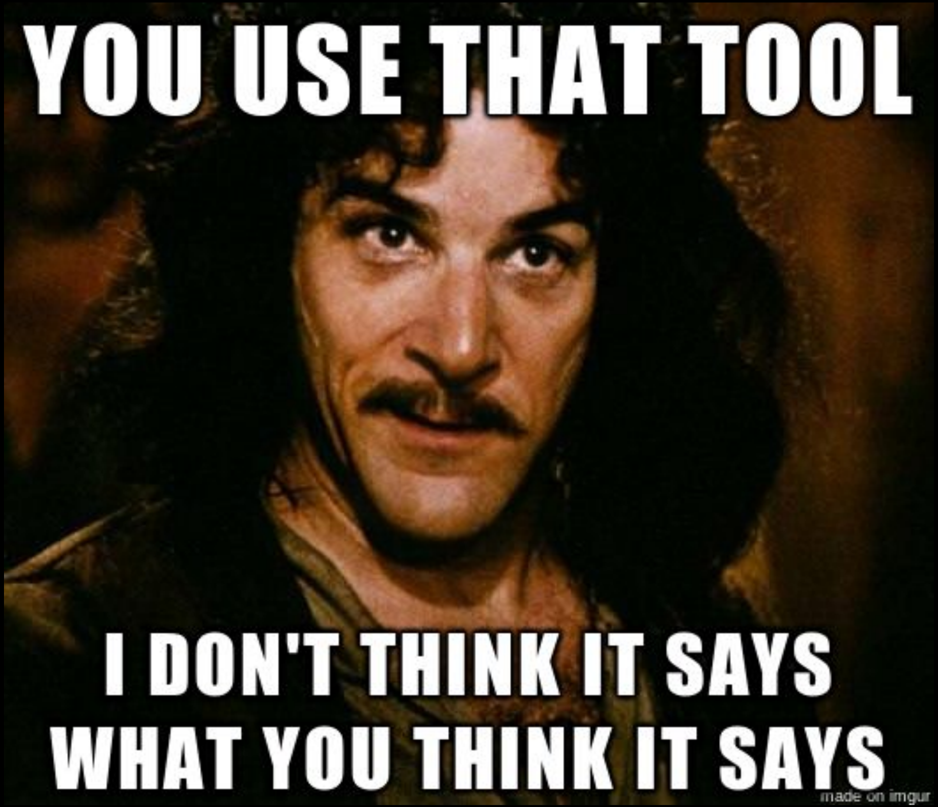
This can also be measured or observed via other signals (logs, error messages, etc)

Hardware Resources

Main Hardware Resources

- CPU
- Memory
- Storage (Capacity, I/O)
- Network I/O

A Word On `top`


YOU USE THAT TOOL
I DON'T THINK IT SAYS WHAT YOU THINK IT SAYS
made on imgur

A Word On `top`

Start with single-purpose tools first before using the all-in-one tools like top and its brethren.

CPU

There are several types of CPU Utilization.

Let’s discuss the common ones:

- **USR**: Time spent in user apps (Eg: Drupal, Cron)
- **SYS**: Time spent in the kernel (Eg: reading/writing to the network device)
- **IOWAIT**: Time spent waiting on storage devices (Eg: reading/writing to disks)
- **IDLE**: Time spent not doing anything. (0%=saturation)

You can observe these metrics in aggregate or per CPU core, which is important when considering single-threaded processes (not common).

Measuring CPU

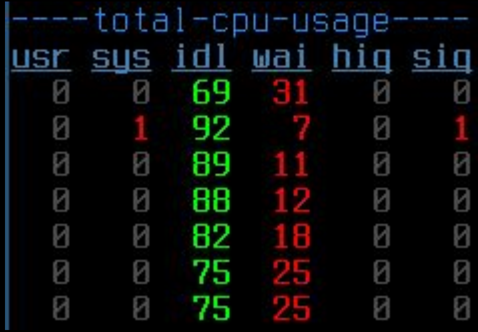
Simple:

- **`dstat -c`: Recent, colorized**
- `mpstat 1`: Older, non-colorized

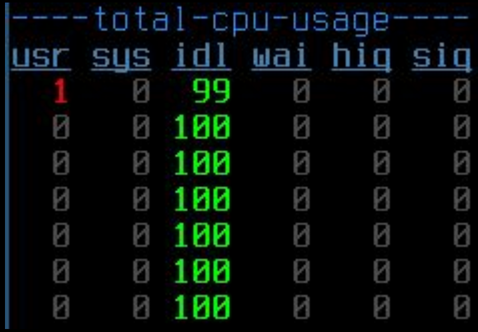
Complex:

- **`htop`: colorized**
- `top`: classic and ubiquitous
- `atop`: supports process accounting

Example `dstat` Output

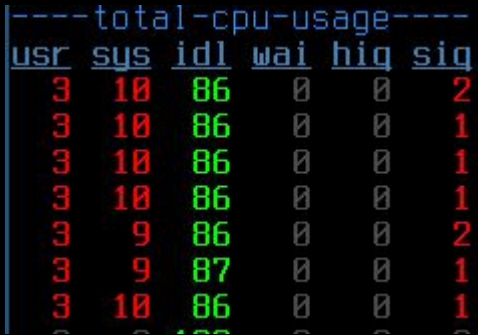


```
----total-cpu-usage----
usr sys idl wai hiq siq
  0   0  69  31   0   0
  0   1  92   7   0   1
  0   0  89  11   0   0
  0   0  88  12   0   0
  0   0  82  18   0   0
  0   0  75  25   0   0
  0   0  75  25   0   0
```

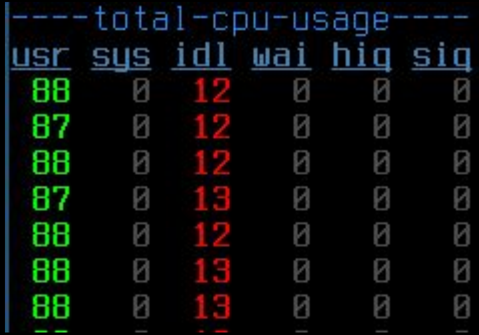


```
----total-cpu-usage----
usr sys idl wai hiq siq
  1   0  99   0   0   0
  0   0 100   0   0   0
  0   0 100   0   0   0
  0   0 100   0   0   0
  0   0 100   0   0   0
  0   0 100   0   0   0
  0   0 100   0   0   0
```

Can you speculate about what is happening for each set of metrics?

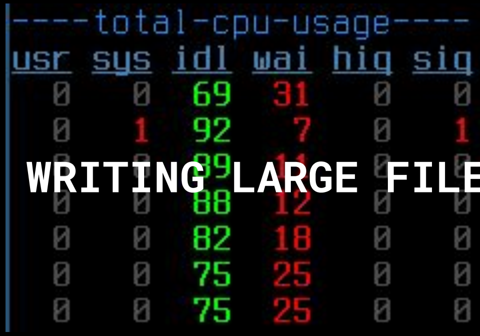


```
----total-cpu-usage----
usr sys idl wai hiq siq
  3  10  86   0   0   2
  3  10  86   0   0   1
  3  10  86   0   0   1
  3  10  86   0   0   1
  3   9  86   0   0   2
  3   9  87   0   0   1
  3  10  86   0   0   1
```



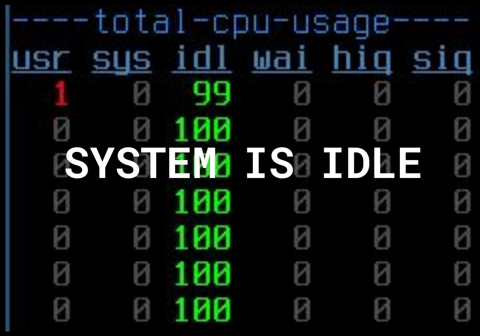
```
----total-cpu-usage----
usr sys idl wai hiq siq
 88   0  12   0   0   0
 87   0  12   0   0   0
 88   0  12   0   0   0
 87   0  13   0   0   0
 88   0  12   0   0   0
 88   0  13   0   0   0
 88   0  13   0   0   0
```

Example `dstat` Output



```
----total-cpu-usage----
usr sys idl wai hiq siq
  0   0  69  31   0   0
  0   1  92   7   0   1
  0   0  89  11   0   0
  0   0  88  12   0   0
  0   0  82  18   0   0
  0   0  75  25   0   0
  0   0  75  25   0   0
```

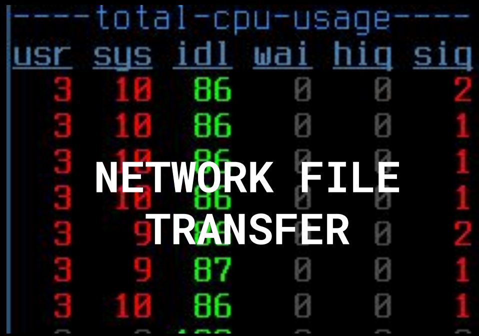
WRITING LARGE FILE



```
----total-cpu-usage----
usr sys idl wai hiq siq
  1   0  99   0   0   0
  0   0 100   0   0   0
  0   0 100   0   0   0
  0   0 100   0   0   0
  0   0 100   0   0   0
  0   0 100   0   0   0
  0   0 100   0   0   0
```

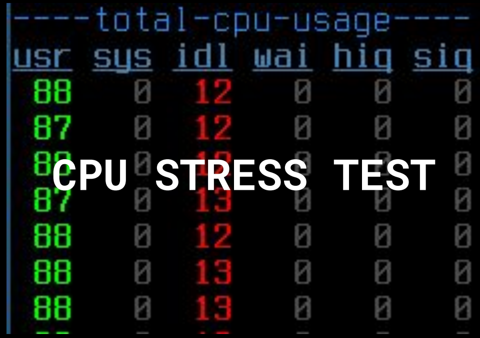
SYSTEM IS IDLE

Can you speculate about what is happening for each set of metrics?



```
----total-cpu-usage----
usr sys idl wai hiq siq
  3  10  86   0   0   2
  3  10  86   0   0   1
  3  10  86   0   0   1
  3  10  86   0   0   1
  3   9  86   0   0   2
  3   9  87   0   0   1
  3  10  86   0   0   1
```

NETWORK FILE TRANSFER



```
----total-cpu-usage----
usr sys idl wai hiq siq
 88   0  12   0   0   0
 87   0  12   0   0   0
 88   0  12   0   0   0
 87   0  13   0   0   0
 88   0  12   0   0   0
 88   0  13   0   0   0
 88   0  13   0   0   0
```

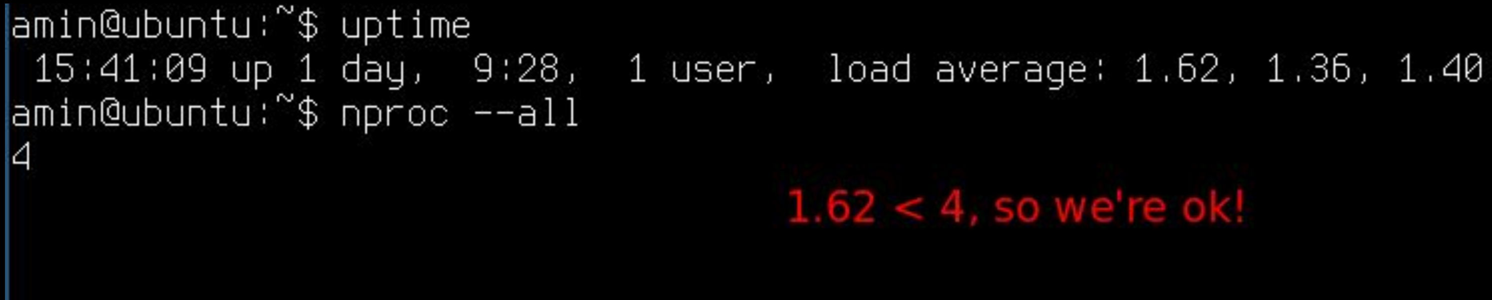
CPU STRESS TEST

Let’s Talk About Load Averages

`uptime` and `top` displays the load average, which is basically the number of processes competing for CPU resources over 1m, 5m, and 15m.

A general rule: If the load average >= the number of server cores, that is a sign of saturation.

(You can easily find number of cores with `nproc --all`.)

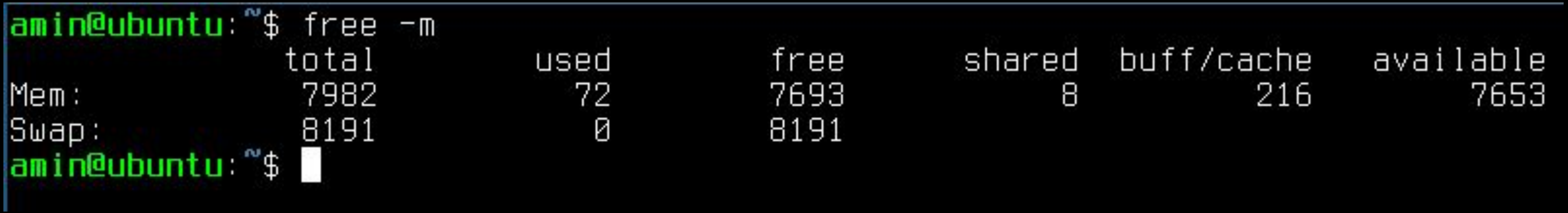


```
amin@ubuntu:~$ uptime
 15:41:09 up 1 day,  9:28,  1 user,  load average: 1.62, 1.36, 1.40
amin@ubuntu:~$ nproc --all
4
```

1.62 < 4, so we're ok!

Memory

Servers have a pool of RAM used for running applications. You can check its utilization with `free -m`:



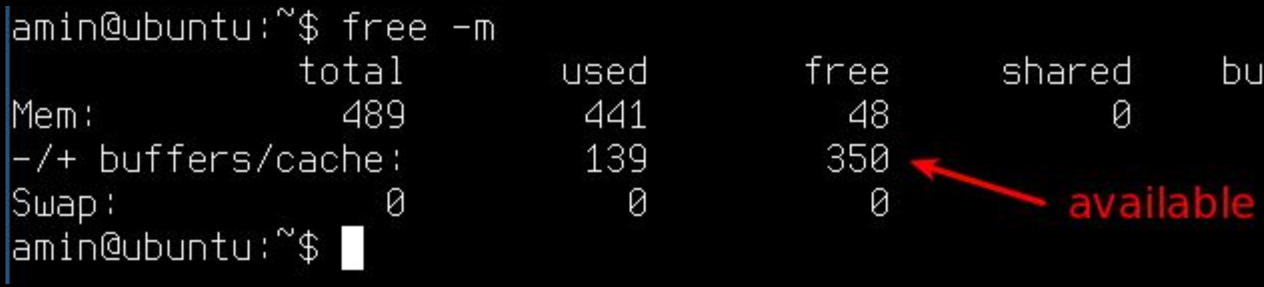
```
amin@ubuntu:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:           7982          72        7693           8         216        7653
Swap:          8191           0        8191
amin@ubuntu:~$ 
```

- **Used**: memory used by actual processes
- **Shared**: memory shared between processes
- **Buffers**: used for reading/writing to devices
- **Cache**: stores copies of files in memory for fast access
- **Available**: the actual amount of memory free for use

The metric you will usually care about is ‘**available**’.

Memory, cont.

You might see output from `free -m` that looks like this. Here's now to determine how much memory is available on a system:



```
amin@ubuntu:~$ free -m
             total       used       free     shared    buffers     cached
Mem:           489        441         48          0         90        211
-/+ buffers/cache:        139        350
Swap:            0          0          0
amin@ubuntu:~$ 
```

An entertaining reference: https://www.linuxatemyram.com/

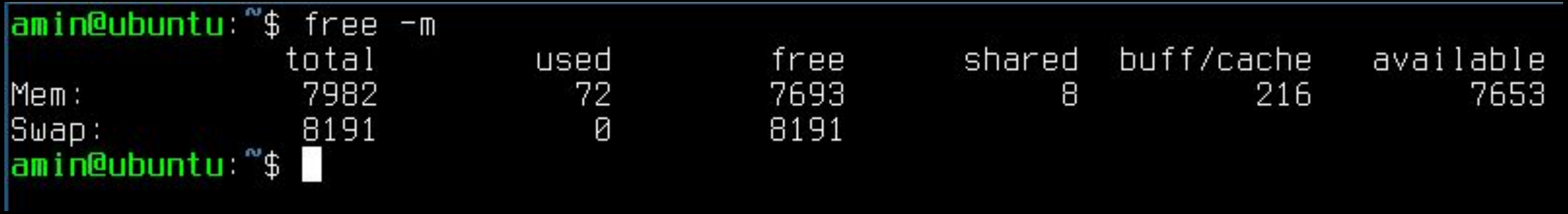
Memory Saturation

What happens when you start to run out of memory?

Swapping.

Contents of RAM will get stored in the swap partition or file, if configured. Hard disk storage is several orders of magnitude slower than RAM, so performance will suffer.

You can check with `free -m`.



```
amin@ubuntu:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:           7982          72        7693           8         216        7653
Swap:          8191           0        8191
amin@ubuntu:~$ 
```

Memory Saturation

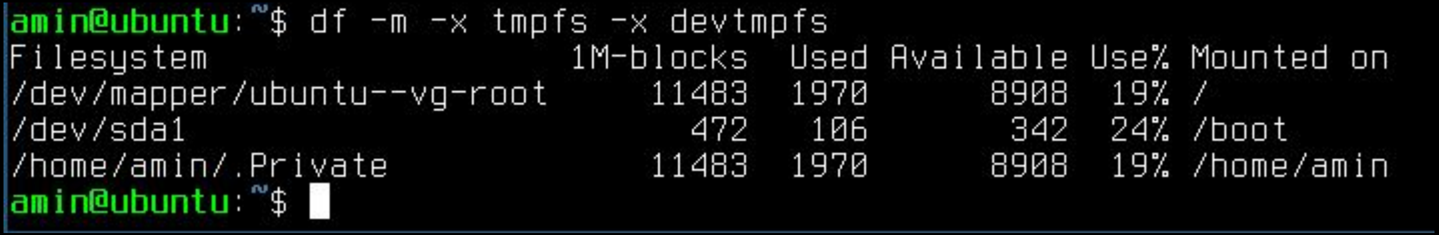
When memory is completely exhausted, the Linux Kernel’s OOM-killer will kill processes to free up memory.

You can check for these events by looking at the kernel log or running `dmesg`:

```
Mar 15 10:10:26 ubuntu kernel: mysqld invoked oom-killer: gfp_mask=0x201da,
order=0, oom_score_adj=-1000
```

Disk Storage

To measure utilization of storage capacity of your local disks and network-attached storage, use `df -m`.



```
amin@ubuntu:~$ df -m -x tmpfs -x devtmpfs
Filesystem                    1M-blocks  Used Available Use% Mounted on
/dev/mapper/ubuntu--vg-root       11483  1970      8908  19% /
/dev/sda1                           472   106       342  24% /boot
/home/amin/.Private               11483  1970      8908  19% /home/amin
amin@ubuntu:~$ 
```

When Use% is at 100%, the disk is full (saturated).

Pretty straightforward, right?

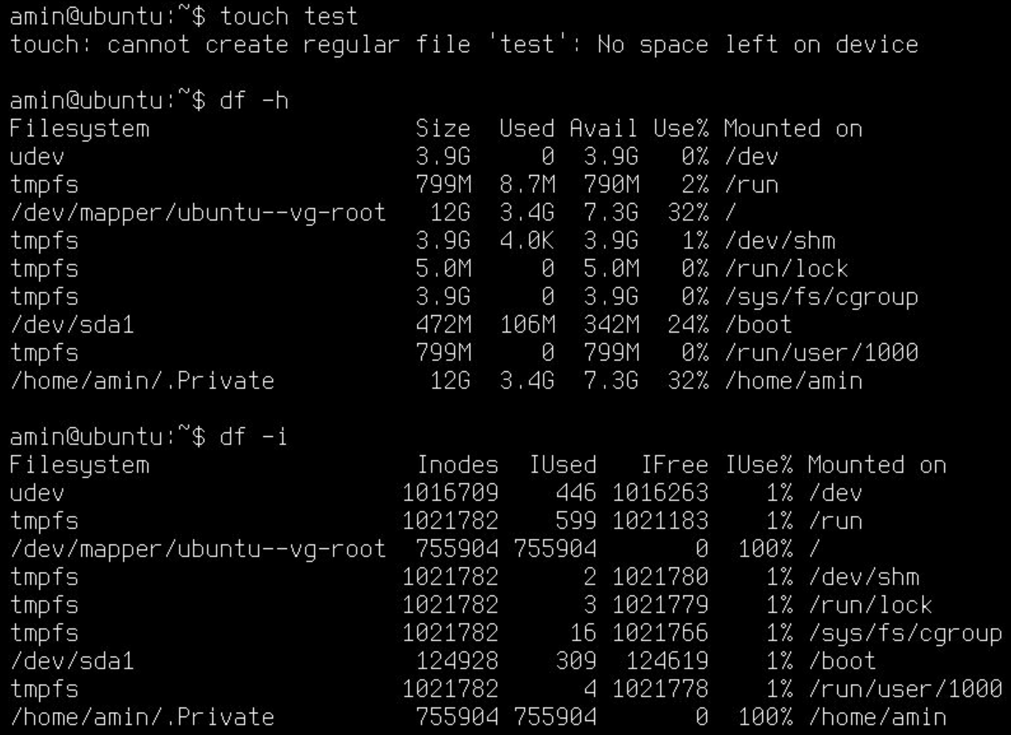
Disk Storage

.. or is it?

Another important thing to measure is the number of **inodes** (or loosely, the total number of files) on the filesystem.

Filesystems have a max number of inodes they can store that cannot be changed.

Watch out for this! Run `**df -i**`!

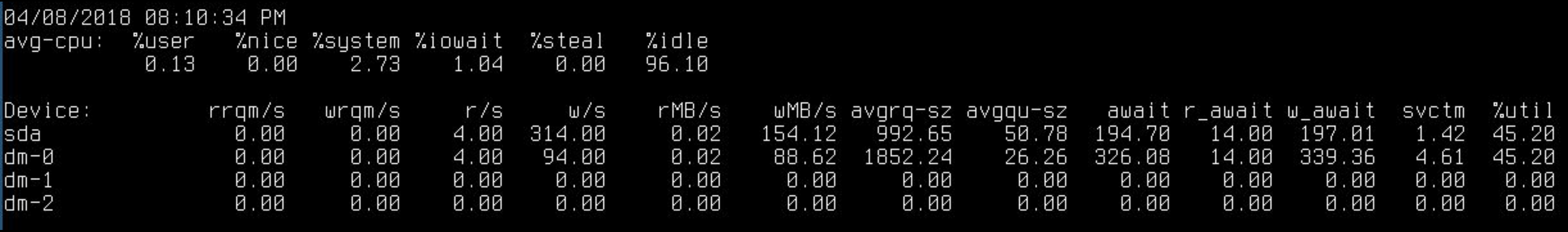


```
amin@ubuntu:~$ touch test
touch: cannot create regular file 'test': No space left on device

amin@ubuntu:~$ df -h
Filesystem                     Size  Used Avail Use% Mounted on
udev                           3.9G     0  3.9G   0% /dev
tmpfs                          799M  8.7M  790M   2% /run
/dev/mapper/ubuntu--vg-root     12G  3.4G  7.3G  32% /
tmpfs                          3.9G  4.0K  3.9G   1% /dev/shm
tmpfs                          5.0M     0  5.0M   0% /run/lock
tmpfs                          3.9G     0  3.9G   0% /sys/fs/cgroup
/dev/sda1                      472M  106M  342M  24% /boot
tmpfs                          799M     0  799M   0% /run/user/1000
/home/amin/.Private             12G  3.4G  7.3G  32% /home/amin

amin@ubuntu:~$ df -i
Filesystem                     Inodes  IUsed   IFree IUse% Mounted on
udev                          1016709    446 1016263    1% /dev
tmpfs                         1021782    599 1021183    1% /run
/dev/mapper/ubuntu--vg-root    755904 755904       0  100% /
tmpfs                         1021782      2 1021780    1% /dev/shm
tmpfs                         1021782      3 1021779    1% /run/lock
tmpfs                         1021782     16 1021766    1% /sys/fs/cgroup
/dev/sda1                      124928    309  124619    1% /boot
tmpfs                         1021782      4 1021778    1% /run/user/1000
/home/amin/.Private            755904 755904       0  100% /home/amin
```

Disk I/O

The only command you’ll ever need: **`iostat -mxt 1`:**

Every second, print eXtended statistics in megabytes.



```
04/08/2018 08:10:34 PM
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.13    0.00    2.73    1.04    0.00   96.10

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
sda               0.00     0.00    4.00  314.00     0.02   154.12   992.65    50.78  194.70   14.00  197.01   1.42  45.20
dm-0              0.00     0.00    4.00   94.00     0.02    88.62  1852.24    26.26  326.08   14.00  339.36   4.61  45.20
dm-1              0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
dm-2              0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
```

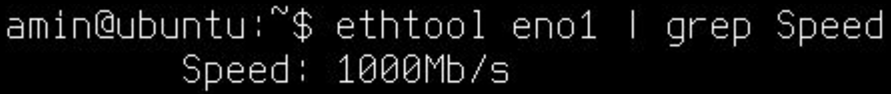
Let’s discuss what’s happening here! Key metrics are:

- rMB/s and wMB/s: read and write throughput in megabytes
- r_await/w_await: average time to service read and write requests. Sustained high values (> 1000) indicate saturation.

Network I/O

Most systems have gigabit network adapters.

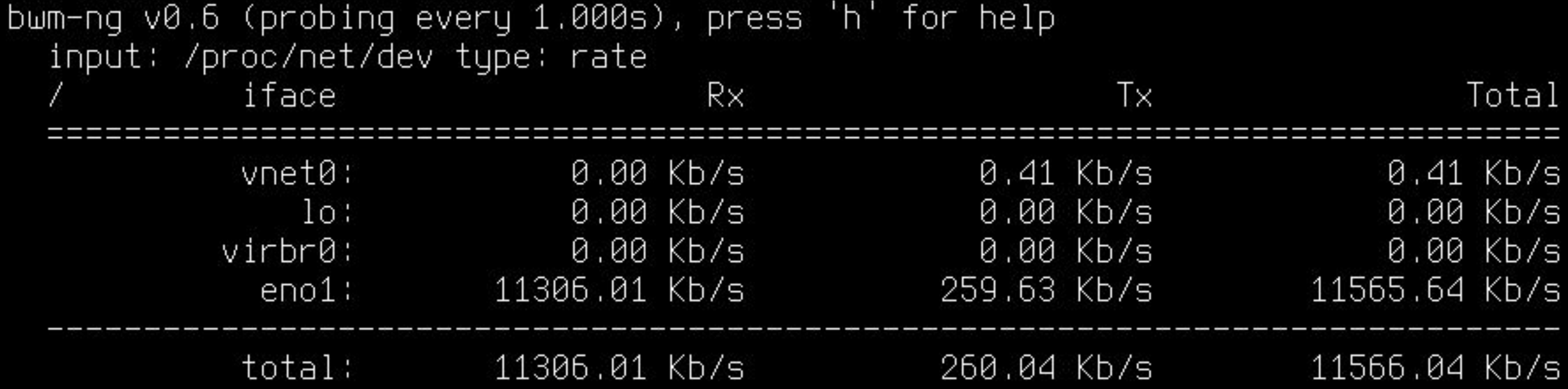
You can check the theoretical maximum your network interface can support with ethtool:



```
amin@ubuntu:~$ ethtool eno1 | grep Speed
        Speed: 1000Mb/s
```

Network I/O

You can observe per-second data rates from all network interfaces with bwm-ng. This link is 1.1% utilized.



```
bwm-ng v0.6 (probing every 1.000s), press 'h' for help
  input: /proc/net/dev type: rate
  /         iface                   Rx                   Tx                Total
  ==============================================================================
           vnet0:           0.00 Kb/s            0.41 Kb/s            0.41 Kb/s
              lo:           0.00 Kb/s            0.00 Kb/s            0.00 Kb/s
          virbr0:           0.00 Kb/s            0.00 Kb/s            0.00 Kb/s
            eno1:       11306.01 Kb/s          259.63 Kb/s        11565.64 Kb/s
  ------------------------------------------------------------------------------
           total:       11306.01 Kb/s          260.04 Kb/s        11566.04 Kb/s
```

`sudo bwm-ng -t 1000 -u bits`

(`dstat -n` is useful as well)

Software Resources

Common Types of Software Resources

All software services (Eg: Apache, MySQL, etc) have some form of tunable resources that introduce constraints.

- Process pools
- Connection limits
- Memory allocations

We’ll discuss the common ones and how to detect saturation.

PHP’s memory_limit

This limits the amount of memory that a single PHP execution can use.

Saturation can be checked in the webserver error logs:

```
“Fatal error: Allowed memory size of 134217728 bytes exhausted (tried to
allocate 44 bytes) in /var/www/html/test.php on line 36”
```

PHP-FPM’s pm.max_children

This limits the number of simultaneous requests that PHP-FPM will handle.

Similar to “FcgidMaxProcessesPerClass” from mod_fcgid.

Saturation can be checked in the webserver logs:

```
“WARNING: [pool www] server reached pm.max_children setting (5),
consider raising it.”
```

MySQL’s max_connections

This limits the number of concurrent connections that MySQL will handle.

Saturation can be checked in the webserver error logs:

```
“SQLSTATE[08004] [1040] Too many connections”
```

Apache’s MaxRequestWorkers

This limits the number of simultaneous requests that Apache will handle.

Formerly known as MaxClients prior to 2.3.13.

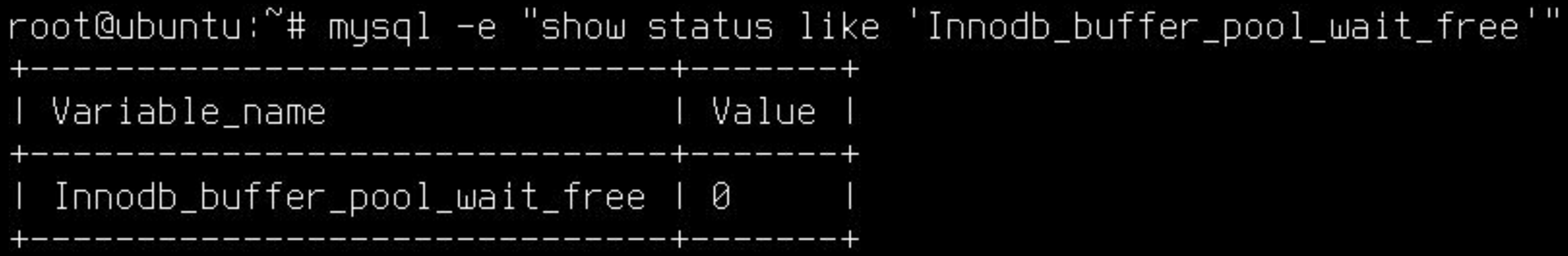
Saturation can be checked in the Apache error logs:

“server reached MaxRequestWorkers setting”

MySQL’s innodb_buffer_pool_size

The InnoDB buffer pool is a cache for your data and indexes in MySQL, which speeds up read requests.

Saturation can be checked by seeing how often MySQL performs cache evictions by flushing to disk.



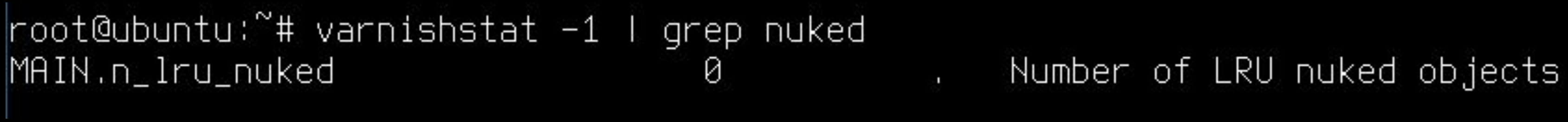
```
root@ubuntu:~# mysql -e "show status like 'Innodb_buffer_pool_wait_free'"
+------------------------------+-------+
| Variable_name                | Value |
+------------------------------+-------+
| Innodb_buffer_pool_wait_free | 0     |
+------------------------------+-------+
```

https://dev.mysql.com/doc/refman/5.7/en/server-status-variables.html#statvar_Innodb_buffer_pool_wait_free

Varnish Cache Size

Varnish deflects backend requests to Drupal by caching and serving previous requests, which improves performance.

Saturation can be checked by seeing the rate that Varnish performs cache evictions by rate of change to the **n_lru_nuked** counter.



```
root@ubuntu:~# varnishstat -1 | grep nuked
MAIN.n_lru_nuked                  0          .   Number of LRU nuked objects
```

Don’t just increase settings!

A common urge is to just increase connections and process limits. **Resist the temptation.**

For example:

Blindly increasing FPM’s pm.max_children may saturate available memory and make a performance problem even worse.

Custom ini_set() of memory_limit to a large value will produce similar results.

Process Introspection

SF

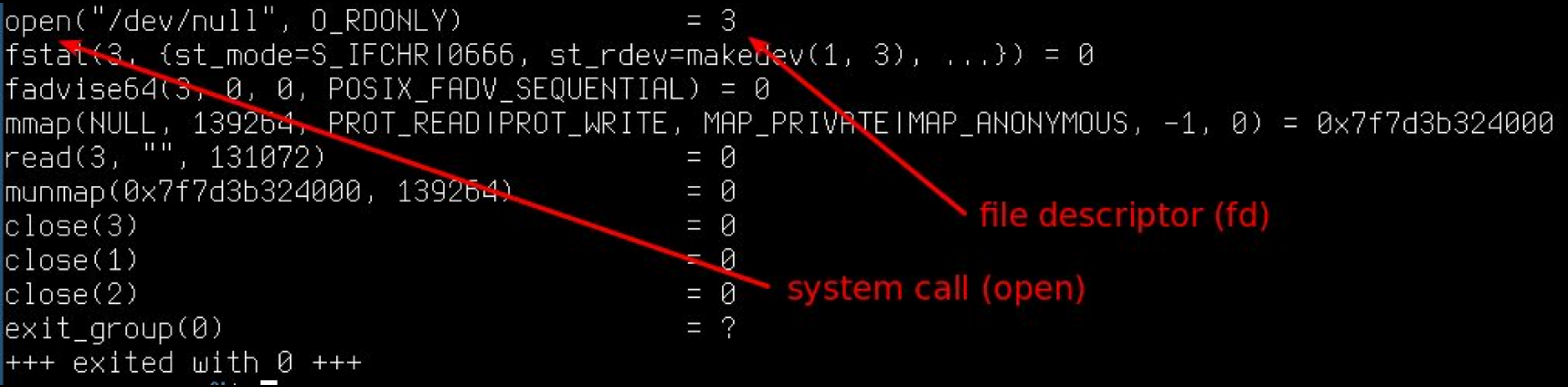
Yes, you can actually do this.

(though it doesn’t look as impressive as it does in *Hackers*...)

strace: a system call tracer

- Attaches to running programs and shows in real time their activity
- System calls are basically how a program asks the OS to do something (file or network read/write, memory mgmt)
- Does slow down execution

strace basic example



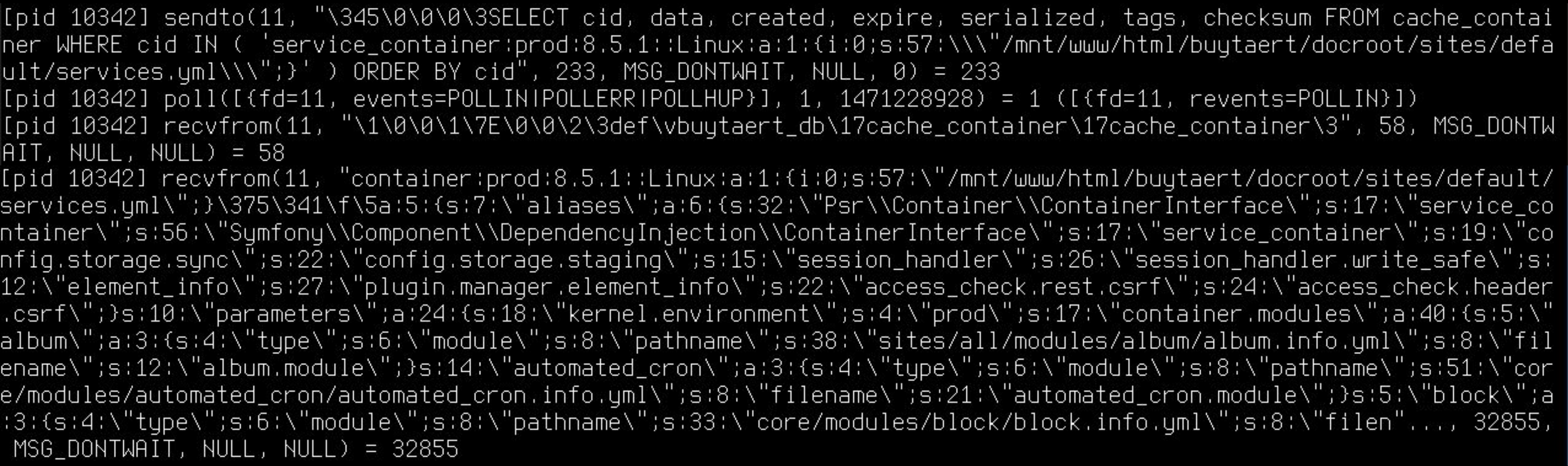
```
open("/dev/null", O_RDONLY)                 = 3
fstat(3, {st_mode=S_IFCHR|0666, st_rdev=makedev(1, 3), ...}) = 0
fadvise64(3, 0, 0, POSIX_FADV_SEQUENTIAL) = 0
mmap(NULL, 139264, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f7d3b324000
read(3, "", 131072)                         = 0
munmap(0x7f7d3b324000, 139264)              = 0
close(3)                                    = 0
close(1)                                    = 0
close(2)                                    = 0
exit_group(0)                               = ?
+++ exited with 0 +++
```

`strace cat /dev/null`

There’s a manual page for each syscall, too!

`man 2 <syscall>`

Now a more interesting example..



```
[pid 10342] sendto(11, "\345\0\0\0\3SELECT cid, data, created, expire, serialized, tags, checksum FROM cache_contai
ner WHERE cid IN ( 'service_container:prod:8.5.1::Linux:a:1:{i:0;s:57:\\\"/mnt/www/html/buytaert/docroot/sites/defa
ult/services.yml\\\";}' ) ORDER BY cid", 233, MSG_DONTWAIT, NULL, 0) = 233
[pid 10342] poll([{fd=11, events=POLLIN|POLLERR|POLLHUP}], 1, 1471228928) = 1 ([{fd=11, revents=POLLIN}])
[pid 10342] recvfrom(11, "\1\0\0\1\7E\0\0\2\3def\vbuytaert_db\17cache_container\17cache_container\3", 58, MSG_DONTW
AIT, NULL, NULL) = 58
[pid 10342] recvfrom(11, "container:prod:8.5.1::Linux:a:1:{i:0;s:57:\"/mnt/www/html/buytaert/docroot/sites/default/
services.yml\";}\375\341\f\5a:5:{s:7:\"aliases\";a:6:{s:32:\"Psr\\Container\\ContainerInterface\";s:17:\"service_co
ntainer\";s:56:\"Symfony\\Component\\DependencyInjection\\ContainerInterface\";s:17:\"service_container\";s:19:\"co
nfig.storage.sync\";s:22:\"config.storage.staging\";s:15:\"session_handler\";s:26:\"session_handler.write_safe\";s:
12:\"element_info\";s:27:\"plugin.manager.element_info\";s:22:\"access_check.rest.csrf\";s:24:\"access_check.header
.csrf\";}s:10:\"parameters\";a:24:{s:18:\"kernel.environment\";s:4:\"prod\";s:17:\"container.modules\";a:40:{s:5:\"
album\";a:3:{s:4:\"type\";s:6:\"module\";s:8:\"pathname\";s:38:\"sites/all/modules/album/album.info.yml\";s:8:\"fil
ename\";s:12:\"album.module\";}s:14:\"automated_cron\";a:3:{s:4:\"type\";s:6:\"module\";s:8:\"pathname\";s:51:\"cor
e/modules/automated_cron/automated_cron.info.yml\";s:8:\"filename\";s:21:\"automated_cron.module\";}s:5:\"block\";a
:3:{s:4:\"type\";s:6:\"module\";s:8:\"pathname\";s:33:\"core/modules/block/block.info.yml\";s:8:\"filen"..., 32855,
 MSG_DONTWAIT, NULL, NULL) = 32855
```

Output from `strace -f -p <PID> -s 1024`, tracing an PHP-FPM parent and its children for https://dri.es

Let’s break it down..

- -f: follows child processes
- -p: process ID, or PID
- -s 1024: print up to 1024 characters of output from each syscall

Extra flags:

- -e 'trace=sendto,recvfrom': only prints those syscalls
- -e 'trace=!gettimeofday': excludes syscalls
- -T: print time spent in each syscall

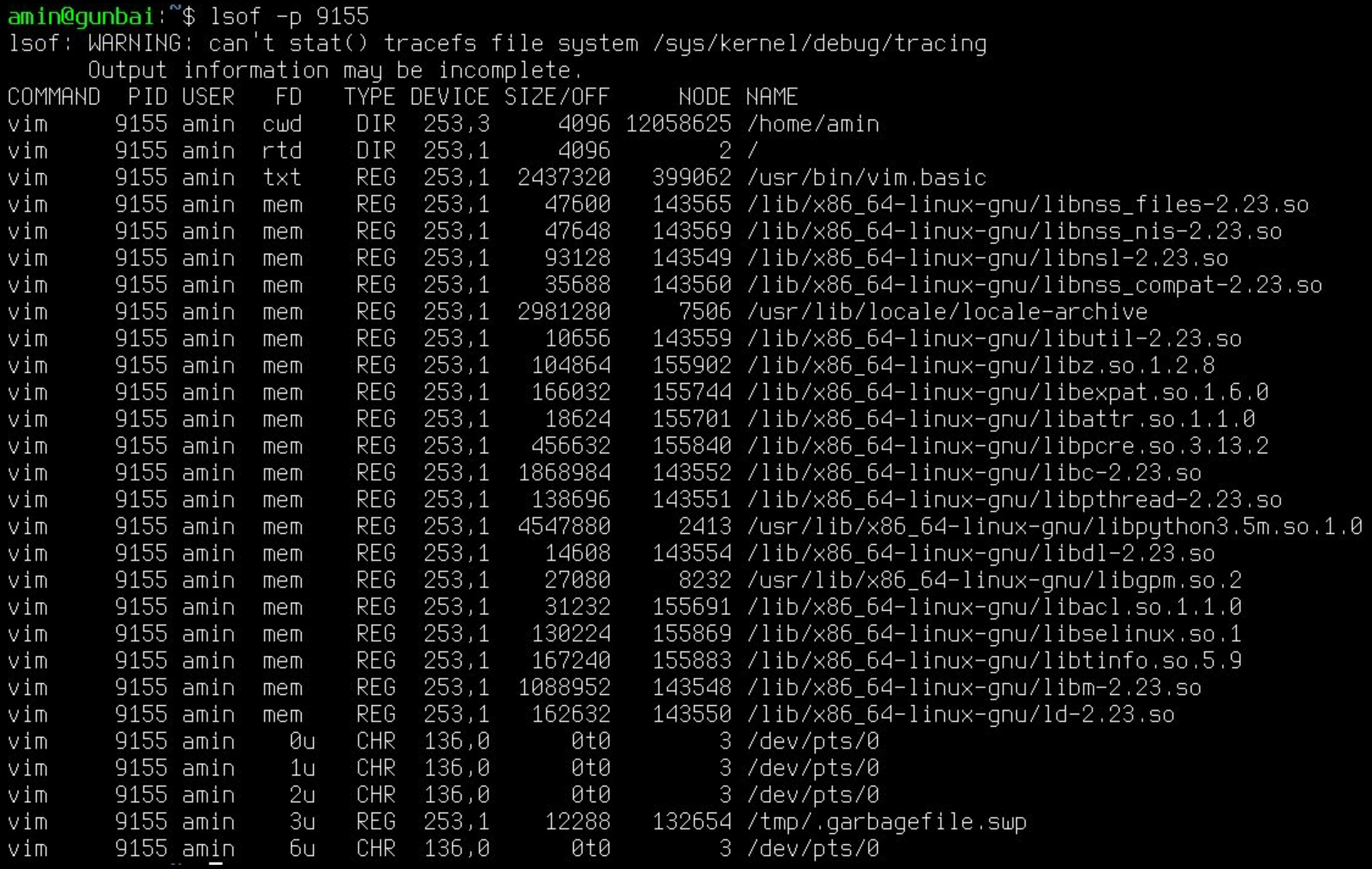
So what can I do with it?

When tracing a PHP process:

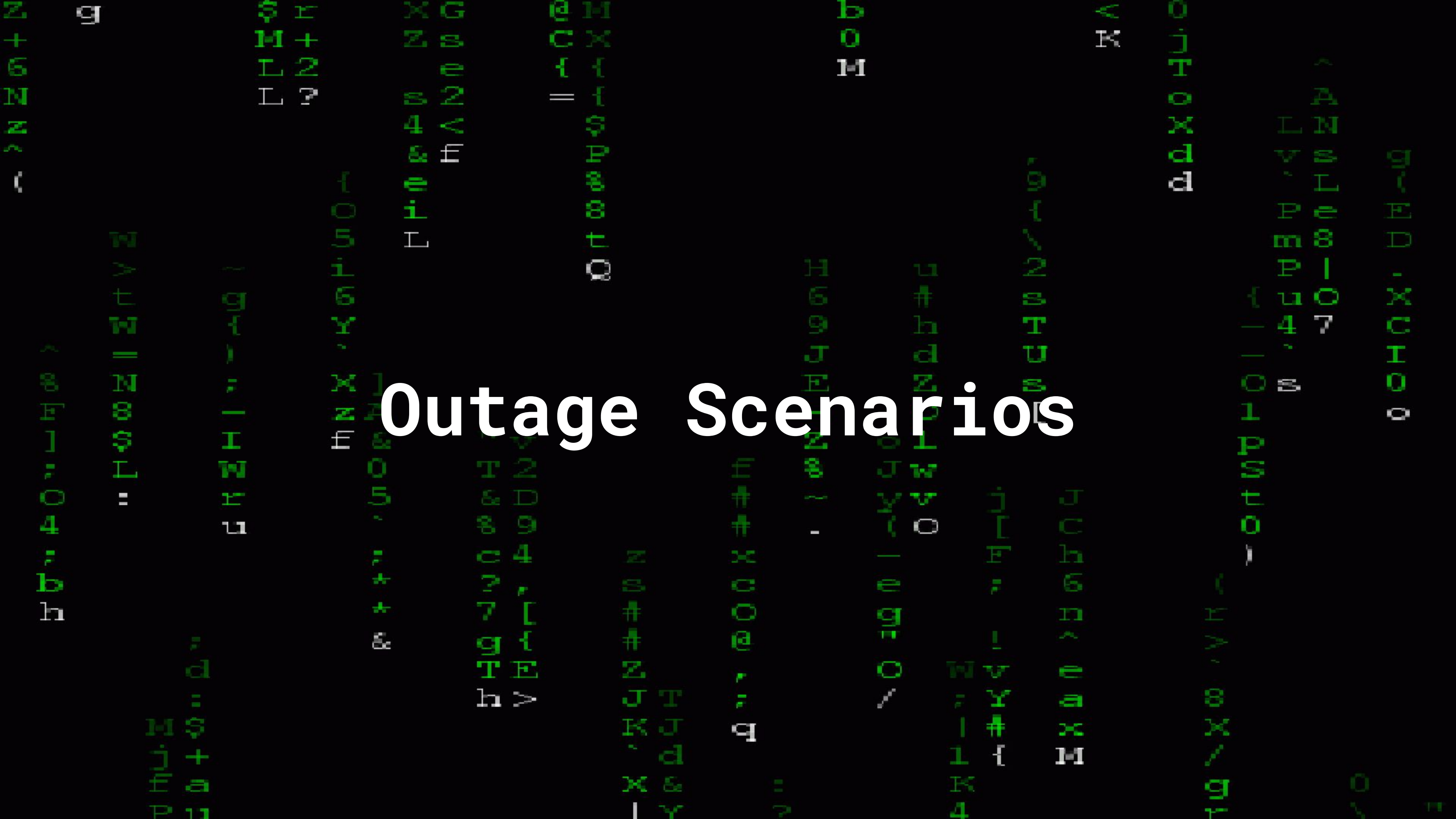
- Observe MySQL statements
- Observe Memcached statements
- Observe HTTP responses
- Observe file accesses
- Measure time spent in each syscall

lsof: list open files

- Prints open files and network connections for all running processes or for a single process (-p PID)
- Lists the file descriptor ids, enabling cross-referencing with strace



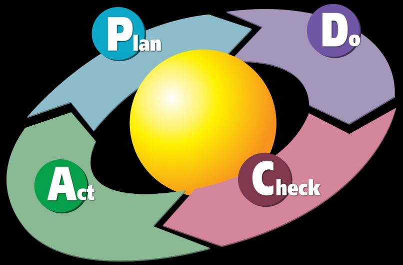
```
amin@gunbai:~$ lsof -p 9155
lsof: WARNING: can't stat() tracefs file system /sys/kernel/debug/tracing
      Output information may be incomplete.
COMMAND  PID USER   FD   TYPE DEVICE SIZE/OFF     NODE NAME
vim     9155 amin  cwd    DIR  253,3     4096 12058625 /home/amin
vim     9155 amin  rtd    DIR  253,1     4096        2 /
vim     9155 amin  txt    REG  253,1  2437320   399062 /usr/bin/vim.basic
vim     9155 amin  mem    REG  253,1    47600   143565 /lib/x86_64-linux-gnu/libnss_files-2.23.so
vim     9155 amin  mem    REG  253,1    47648   143569 /lib/x86_64-linux-gnu/libnss_nis-2.23.so
vim     9155 amin  mem    REG  253,1    93128   143549 /lib/x86_64-linux-gnu/libnsl-2.23.so
vim     9155 amin  mem    REG  253,1    35688   143560 /lib/x86_64-linux-gnu/libnss_compat-2.23.so
vim     9155 amin  mem    REG  253,1  2981280     7506 /usr/lib/locale/locale-archive
vim     9155 amin  mem    REG  253,1    10656   143559 /lib/x86_64-linux-gnu/libutil-2.23.so
vim     9155 amin  mem    REG  253,1   104864   155902 /lib/x86_64-linux-gnu/libz.so.1.2.8
vim     9155 amin  mem    REG  253,1   166032   155744 /lib/x86_64-linux-gnu/libexpat.so.1.6.0
vim     9155 amin  mem    REG  253,1    18624   155701 /lib/x86_64-linux-gnu/libattr.so.1.1.0
vim     9155 amin  mem    REG  253,1   456632   155840 /lib/x86_64-linux-gnu/libpcre.so.3.13.2
vim     9155 amin  mem    REG  253,1  1868984   143552 /lib/x86_64-linux-gnu/libc-2.23.so
vim     9155 amin  mem    REG  253,1   138696   143551 /lib/x86_64-linux-gnu/libpthread-2.23.so
vim     9155 amin  mem    REG  253,1  4547880     2413 /usr/lib/x86_64-linux-gnu/libpython3.5m.so.1.0
vim     9155 amin  mem    REG  253,1    14608   143554 /lib/x86_64-linux-gnu/libdl-2.23.so
vim     9155 amin  mem    REG  253,1    27080     8232 /usr/lib/x86_64-linux-gnu/libgpm.so.2
vim     9155 amin  mem    REG  253,1    31232   155691 /lib/x86_64-linux-gnu/libacl.so.1.1.0
vim     9155 amin  mem    REG  253,1   130224   155869 /lib/x86_64-linux-gnu/libselinux.so.1
vim     9155 amin  mem    REG  253,1   167240   155883 /lib/x86_64-linux-gnu/libtinfo.so.5.9
vim     9155 amin  mem    REG  253,1  1088952   143548 /lib/x86_64-linux-gnu/libm-2.23.so
vim     9155 amin  mem    REG  253,1   162632   143550 /lib/x86_64-linux-gnu/ld-2.23.so
vim     9155 amin    0u   CHR  136,0      0t0        3 /dev/pts/0
vim     9155 amin    1u   CHR  136,0      0t0        3 /dev/pts/0
vim     9155 amin    2u   CHR  136,0      0t0        3 /dev/pts/0
vim     9155 amin    3u   REG  253,1    12288   132654 /tmp/.garbagefile.swp
vim     9155 amin    6u   CHR  136,0      0t0        3 /dev/pts/0
```

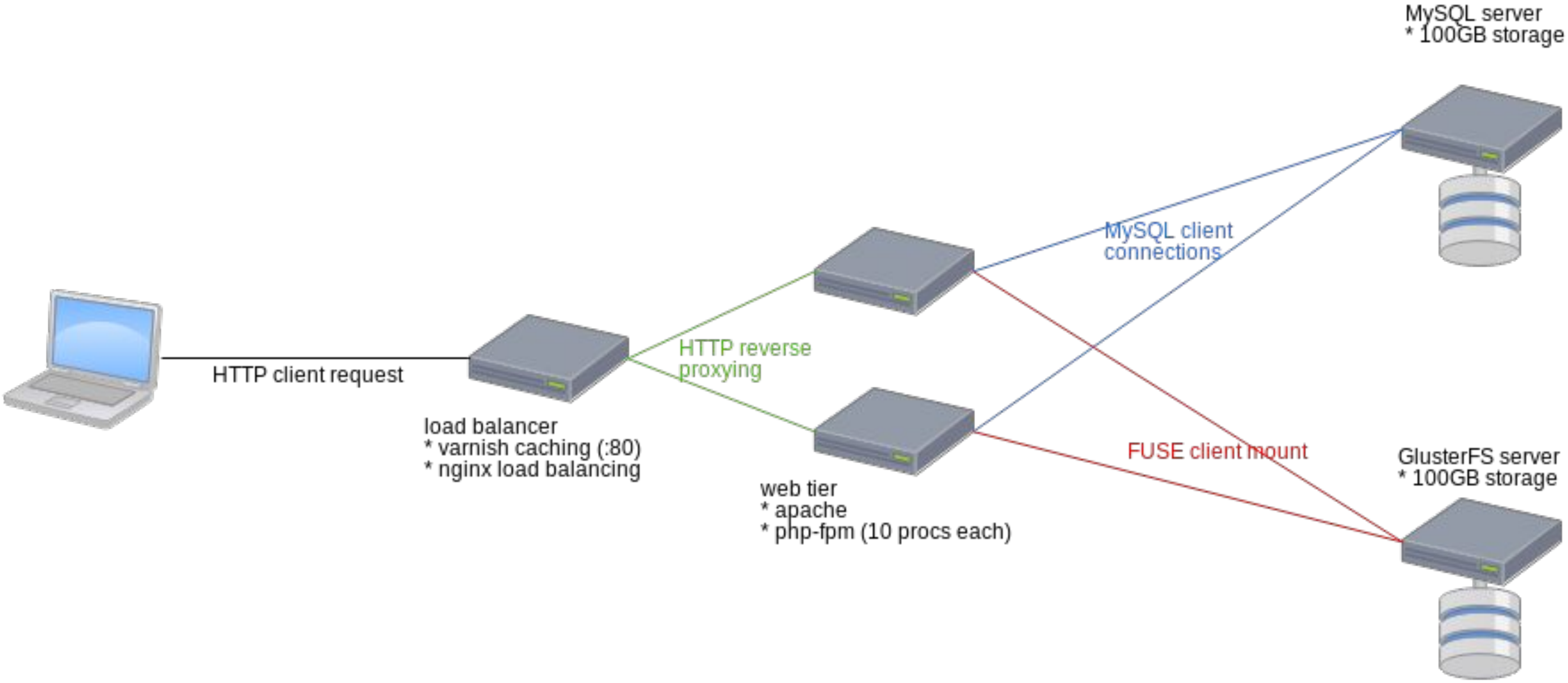


Outage Scenarios

My Troubleshooting ‘Kata’

- USE Method: Identify all saturated resources (constraints)
- **Plan**: Choose the main constraint and decide how to address it
- **Do**: Implement the change
- **Check**: USE Method: Is the resource still a constraint?
- **Act**
  - If site is back up: SUCCESS
  - If improvement but still unresolved: Keep change, plan with new main constraint
  - If unchanged or worse: undo change and plan again




MySQL server
* 100GB storage
MySQL client
connections
HTTP reverse
proxying
HTTP client request
load balancer
* varnish caching (:80)
* nginx load balancing
FUSE client mount
GlusterFS server
* 100GB storage
web tier
* apache
* php-fpm (10 procs each)

Scenario 1

- Our site either loads slowly or times out with a 503 when requesting an uncached page.
- We apply USE Method to the balancers and find no saturation.
- We apply USE Method to the web servers, and find:
  - All PHP-FPM processes are in use (pm.max_children warnings)
  - CPU is mostly idle. When running top/ps, the PHP processes aren’t the top consumers.
  - lsof on all of the php-fpm processes shows this output:

```
php-fpm     1161        drupal   10u      IPv4        126303135   0t0         TCP
server-123.custom.domain.tld:23319->ec2-50-123-321-2.compute-1.amazonaws.com:https (ESTABLISHED)
```

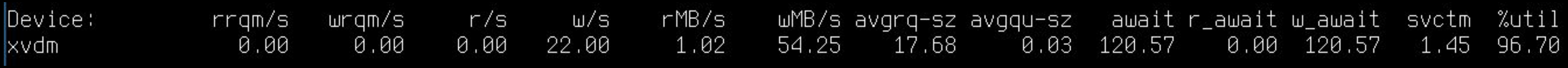
Can you guess what’s happening?

Scenario 1

- In Acquia Operations, we call this scenario an ‘external call’, where a Drupal site is making a call to a 3rd party service.
- If the third party service is slow/down, it can directly impact performance of your site as your code is waiting for a response.
- **We have even seen instances of sites making calls to itself!**
- The solution:
  - remove dependence on 3rd party services where possible
  - program defensively to gracefully degrade when it is unavailable.

Scenario 2

- Our site either loads slowly or times out with a 503 when requesting an uncached page.
- We apply USE Method to the balancers and find no saturation.
- We apply USE Method to the web servers, and find:
  - All PHP-FPM processes are in use (pm.max_children warnings)
  - CPU is 50% utilized by PHP-FPM processes in USR.
- We apply USE Method to the database server, and find this metric for the database volume by running iostat:



```
Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
xvdm              0.00     0.00    0.00   22.00     1.02    54.25    17.68     0.03  120.57    0.00  120.57   1.45  96.70
```

What’s happening here?

Scenario 2

We suspect very high write operations on the database, and decide to print MySQL’s processlist. (`mytop -d mysql`). We see a large quantity of statements that look like this:

```
   12514     drupal   web-123  drupal       3        Query INSERT INTO watchdog
(uid, type, message, variables, severity, link, location, referer, hostname,
timestamp) VALUES ('0', 'stuff
```

What did we discover?

Scenario 2

- The site had the **dblog module** enabled.
- In situations where a site is emitting a lot of code errors, massive write operations will happen to the database, saturating the underlying storage.
- Solution: **don’t use the dblog module**. Use syslog instead.

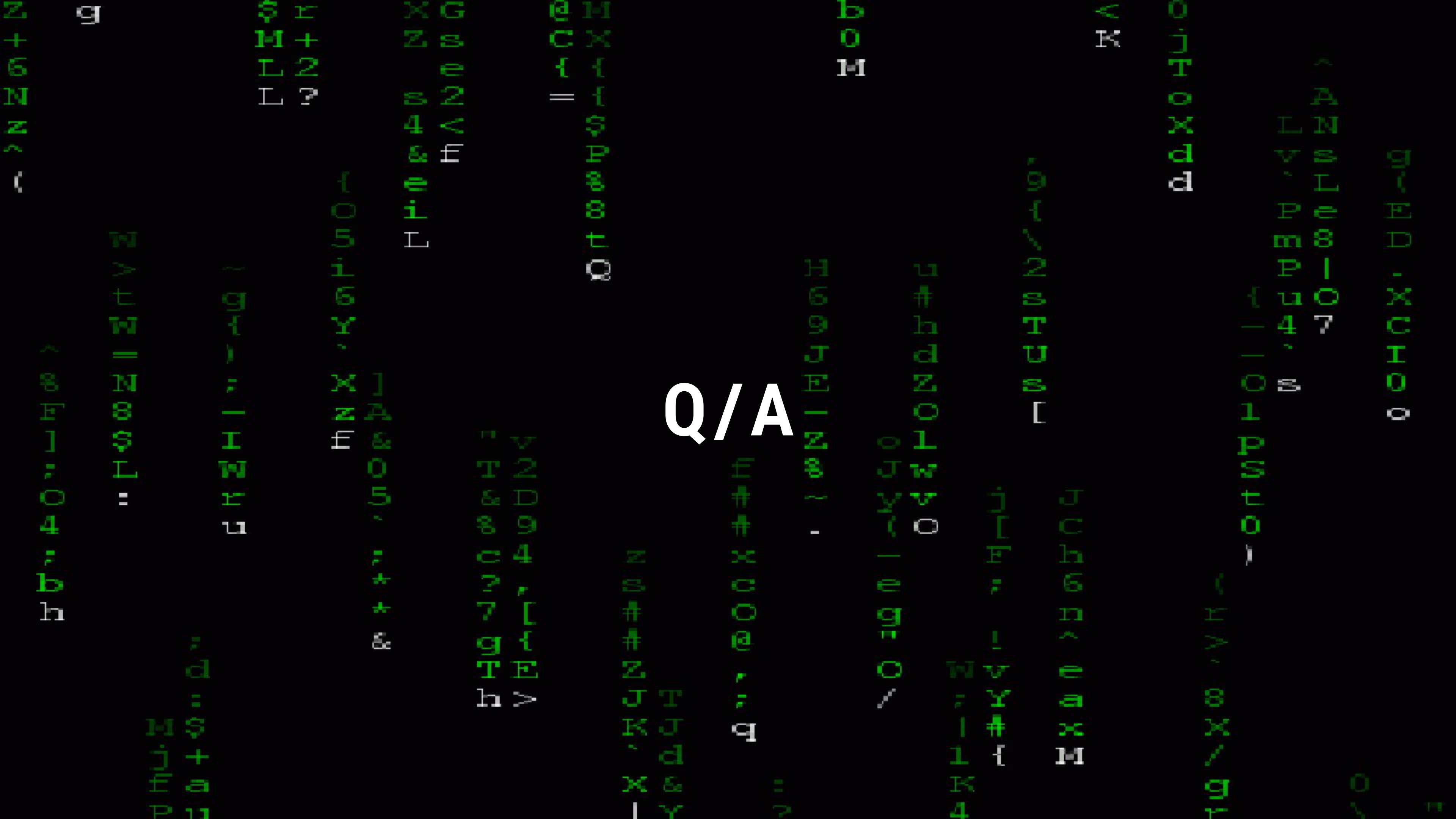
Scenario 2

- The site had the **dblog module** enabled.
- In situations where a site is emitting a lot of code errors, massive write operations will happen to the database, saturating the underlying storage.
- Solution: **don’t use the dblog module**. Use syslog instead.

Let’s Recap!

Let’s Recap!

- Troubleshooting infrastructure is accessible to mortals
- The USE Method
- Hardware Resources
- Software Resources
- Process Introspection
- PDCA as a process for improving performance



Q/A

What did you think?

Locate this session at the DrupalCon Nashville website:

http://nashville2018.drupal.org/schedule

Take the Survey!

https://www.surveymonkey.com/r/DrupalConNashville

Join us for
contribution sprints

Friday, April 13, 2018

Mentored
Core sprint

9:00-18:00
Room: 103

First time
sprinter workshop

9:00-12:00
Room: 101

General
sprint

9:00-18:00
Room: 104

#drupalsprint

Media Credits

- *The Fellowship of the Ring* (New Line Cinema)
- *Ghost In The Shell* (Kodansha, Bandai Visual, Manga Entertainment)
- *Hackers* (United Artists)
- *Zelda II: The Adventure of Link* (Nintendo)
- *The Princess Bride* (Act III Communications)
- *Superman* (Max Fleischer Studios, Paramount Pictures)
- PDCA Diagram (Karn G. Bulsuk, http://www.bulsuk.com)

Amin Astaneh
T: @aastaneh
IRC: amin
amin@aminastaneh.net