Débug, VM et Docker

M1 - CHPS

Architecture Interne des Systèmes d'exploitations (AISE)

Jean-Baptiste Besnard
<jbbesnard@paratools.com>

Spack

```
Naos  ~
 spack list | wc -w
3618
```

- Conçu pour les problématiques HPC
- Gestionnaire de paquets en userspace
- Supporte de nombreuses recettes HPC:
  - Plusieurs souches MPI
  - hwloc
- Prend en charge les variantes compil' et les flags
- Simple à installer:
  - Télécharger les sources: https://github.com/spack/spack
  - `export PATH=$PWD/spack/bin:$PATH`
  - Intégration: `source ./spack/share/spack/setup-env.sh`

Spac

- Co
- Ge
- Su
  -
  -
- Pre

```
└─ spack
usage: spack [-hkV] [--color {always,never,auto}] COMMAND ...

A flexible package manager that supports multiple versions,
configurations, platforms, and compilers.

These are common spack commands:

query packages:
  list                  list and search available packages
  info                  get detailed information on a particular package
  find                  list and search installed packages

build packages:
  install               build and install packages
  uninstall             remove installed packages
  spec                  show what would be installed, given a spec
```

```
└─ spack find
==> 85 installed packages
-- linux-linuxmint20-haswell / gcc@9.3.0 ------------------------
autoconf@2.69         czmq@4.1.1          gmp@6.2.1           libmd@1.0.3         m4@1.4.19
automake@1.15         diffutils@3.7       hwloc@2.5.0         libpciaccess@0.16   mpc@1.1.0
automake@1.16.3       diffutils@3.8       hwloc@2.6.0         libsigsegv@2.13     mpfr@3.1.6
automake@1.16.5       docbook-xml@4.5     hwloc@2.6.0         libsodium@1.0.18    ncurses@6.2
berkeley-db@18.1.40   docbook-xsl@1.79.2  jansson@2.13.1      libtool@2.4.6       numactl@2.0.14
binutils@2.37         expat@2.4.1         libbsd@0.11.3       libxml2@2.9.12      openmpi@4.1.1
boost@1.77.0          flux-core@0.31.0    libedit@3.1-20210216 libyaml@0.2.5      openssh@8.7p1
bzip2@1.0.8           flux-sched@0.20.0   libelf@0.8.13       libzmq@4.3.4        openssl@1.1.1l
cmake@3.21.2          gcc@9.3.0           libevent@2.1.12     lua@5.3.5           openssl@1.1.1l
cmake@3.21.4          gdbm@1.19           libffi@3.3          lua-luaposix@35.0   osu-micro-benchmarks@5.7.1
cuda@11.4.0           gettext@0.21        libiconv@1.16       lz4@1.9.3           pcre@8.44
```

Python & pyenv

- Python est maintenant prédominant en environnement HPC
  - Codage simple pour des non-experts
  - Large panel de fonctionnalités
  - Gestionnaire de paquets inclus
  - Langage orienté objet
  - Python 2 déprécié, focus sur python3 !
- Installer un paquet : pip
  - `pip install mpi4py (via pypi.org)`
  - `pip install .`

Pyenv

- Toujours essayer de développer dans des « environnements »
- Un environnement crée une installation virtuelle d'une souche Python
- La gestion de l'environnement Python peut être cloisonné
  - Pip install virtualenv
- Créer un env: **python3 -m virtualenv ./build**
- Charger un env: **source ./build/bin/activate**
- Décharger: **deactivate**
- Supprimer un env: **rm -rf ./build/**
- Comment gérer plusieurs installations Python par dessus !
  - **Pyenv !**

The Art of Debugging

Bug d’un programme

- Qu’est-ce qu’un « bug » dans un programme?

Bug d’un programme

- Qu’est-ce qu’un programme qui « bug » ?
  - Crash (SEGV par exemple)
  - Résultat différent de ce qui est attendu
  - Interblocage (deadlock)
- Idée : Suivre l’exécution du programme (flot & variables)
- L’outil du débutant en debug : printf
  - Avantages : Simple, aucune connaissance à priori
  - Inconvénients : recompilation, scories…

Le « vrai » debugging

- Contrôlé par un outil tiers : le débugger
- Large panel de fonctionnalités :
  - Suivre une variable
  - Mettre en pause le programme
  - Insérer des « points d’arrêt » (breakpoint)
  - Exécution du programme instruction par instruction
  - Explorer le binaire
  - Explorer la mémoire
  - …

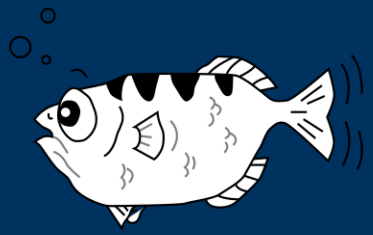
Exemple d’un debugger

```
└─▶ gcc segv.c -g && ./a.out
[1]    28842 segmentation fault (core dumped)  ./a.out
```

```
└─▶ gcc segv.c -g && gdb ./a.out
Reading symbols from ./a.out...done.
gdb >> run
Starting program: /home/adamj/Documents/cours/aise/Cours_5/debug/a.out
Missing separate debuginfos, use: dnf debuginfo-install glibc-2.27-37.fc28.x86_64

Program received signal SIGSEGV, Segmentation fault.
----------------------------------------------------------------------------------------------------------------------------------[regs]
  RAX: 0x0000000000000000  RBX: 0x0000000000000000  RBP: 0x00007FFFFFFFD990  RSP: 0x00007FFFFFFFD978  o d I t s z a p c
  RDI: 0x0000000000000001  RSI: 0x00007FFFFFFFDA78  RDX: 0x00007FFFFFFFDA88  RCX: 0x00007FFFF7DD0718  RIP: 0x0000000000000000
  R8 : 0x00007FFFF7DD1D80  R9 : 0x00007FFFF7DD1D80  R10: 0x0000000000000007  R11: 0x0000000000000002  R12: 0x00000000004003B0
  R13: 0x00007FFFFFFFDA70  R14: 0x0000000000000000  R15: 0x0000000000000000
  CS: 0033  DS: 0000  ES: 0000  FS: 0000  GS: 0000  SS: 002B                    Error while running hook_stop:
Cannot access memory at address 0x0
0x0000000000000000 in ?? ()
gdb >> backtrace
#0  0x0000000000000000 in ?? ()
#1  0x00000000004004ac in main (argc=0x1, argv=0x7fffffffda78) at segv.c:4
```

GDB



- « GNU Debugger »
- Le debugging n'est pas magique, des informations supplémentaires sont ajoutées au binaire via le flag **`-g`**
- Extra informations dans les sections dédiées (voir format DWARF)
- Commande interactive. Un prompt est ouvert en attente de commandes utilisateur (ou d'un script)
- Lancer GDB : **`gdb ./a.out`** (ou **`file ./a.out`** au prompt)
- Arguments : option **`--args`** ou **`set args`** au prompt
- Démarrer le programme : **`run`**
- Quitter : **`q[uit]`** (ou Ctrl + D)
- Aide : `help <cmd>`
- `Spack install gdb`

```
En-têtes de section :
  [Nr] Nom                Type
  [ 0]                    NULL
  [ 1] .interp            PROGBITS
  [ 2] .note.ABI-tag      NOTE
  [ 3] .note.gnu.build-id NOTE
  [ 4] .gnu.hash          GNU_HASH
  [ 5] .dynsym            DYNSYM
  [ 6] .dynstr            STRTAB
  [ 7] .gnu.version       VERSYM
  [ 8] .gnu.version_r     VERNEED
  [ 9] .rela.dyn          RELA
  [10] .init              PROGBITS
  [11] .text              PROGBITS
  [12] .fini              PROGBITS
  [13] .rodata            PROGBITS
  [14] .eh_frame_hdr      PROGBITS
  [15] .eh_frame          PROGBITS
  [16] .init_array        INIT_ARRAY
  [17] .fini_array        FINI_ARRAY
  [18] .dynamic           DYNAMIC
  [19] .got               PROGBITS
  [20] .got.plt           PROGBITS
  [21] .data              PROGBITS
  [22] .bss               NOBITS
  [23] .comment           PROGBITS
  [24] .debug_aranges     PROGBITS
  [25] .debug_info        PROGBITS
  [26] .debug_abbrev      PROGBITS
  [27] .debug_line        PROGBITS
  [28] .debug_str         PROGBITS
  [29] .symtab            SYMTAB
  [30] .strtab            STRTAB
  [31] .shstrtab          STRTAB
```

GDB

- Afficher du contenu : `print` (ou `p`) de tout type. Le type doit être connu de GDB pour être affiché. Peut couvrir quasiment toute expression C (ex: **`a->b.c`**)
- La variable i : `print i`
- Le contenu du pointeur p : `print *p`
- Le contenu à l'adresse (de type T) : `print {T}addr`
- Rappel : VOID n'est pas un type défini !
- Briser une ambiguïté de portée : `print main::i`
- Variables GDB, préfixées par « `$` » : registres, retour de print…

Format	
a	Pointer.
c	Read as integer, print as character.
d	Integer, signed decimal.
f	Floating point number.
o	Integer, print as octal.
s	Try to treat as C string.
t	Integer, print as binary (*t* = „two“).
u	Integer, unsigned decimal.
x	Integer, print as hexadecimal.

```
gdb >> x $rsp
0x7fffffffdfe8: 0x00007ffff7ddf0b3
gdb >> x/10a $rsp
0x7fffffffdfe8: 0x7ffff7ddf0b3 <__libc_start_main+243>  0x7ffff7ffc620 <_rtld_global_ro>
0x7fffffffdff8: 0x7fffffffe0d8  0x100000000
0x7fffffffe008: 0x555555555231 <main>   0x5555555552b0 <__libc_csu_init>
0x7fffffffe018: 0xb0c75bfdb89320cb      0x5555555550e0 <_start>
0x7fffffffe028: 0x7fffffffe0d0  0x0
gdb >> x/10zg $rsp
0x7fffffffdfe8: 0x00007ffff7ddf0b3      0x00007ffff7ffc620
0x7fffffffdff8: 0x00007fffffffe0d8      0x0000000100000000
0x7fffffffe008: 0x0000555555555231      0x00005555555552b0
0x7fffffffe018: 0xb0c75bfdb89320cb      0x00005555555550e0
0x7fffffffe028: 0x00007fffffffe0d0      0x0000000000000000
```

GDB

- Le programme s'exécute comme s'il était hors de GDB. Un deadlock ou un SEGV peut donc se reproduire.
- Pour interrompre le programme pendant son exécution : **`Ctrl + C`**
- Principe de base : Une fois le programme stoppé, il est possible d'inspecter son contenu. Lorsqu'un programme est stoppé, il se trouve sur une instruction donnée, mappée dans l'espace mémoire du processus
- Idée : afficher la pile d'appels courante : **`bt`**
- Navigation entre les « frames » : **`up`** | **`down`** | **`frame #x`**

```
gdb >> bt
#0  0x00007ffff7bc78bd in __lll_lock_wait () from /lib64/libpthread.so.0
#1  0x00007ffff7bc0d05 in pthread_mutex_lock () from /lib64/libpthread.so.0
#2  0x000000000040070c in func (arg=0x7fffffffd96c) at threads.c:18
#3  0x00007ffff7bbe594 in start_thread () from /lib64/libpthread.so.0
#4  0x00007ffff78f1f4f in clone () from /lib64/libc.so.6
```

GDB

- Il est possible de définir statiquement des instructions où l'on souhaite stopper le programme pour pouvoir l'inspecter : le « point d'arrêt » ou **breakpoint**
- Peut être une adresse mémoire (0x…), un nom de fonction (symbole, par extension) ou un tuple (fichier, numéro de ligne), si l'option **`-g`** est passé à la compilation
- À chaque fois que le point d'arrêt est rencontré, le programme est suspendu
- Mettre un breakpoint : **`break <ref>`**
- Supprimer un breakpoint : **`delete <ref>`**
- Activer / désactiver un breakpoint : **`enable`** | **`disable <ref>`**
- Reprendre une exécution normale après un breakpoint : **`continue`**
- À considérer : `les watchpoints` (triggers sur changement de contenu)

```
gdb >> info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000000000400508 in main at sample.c:9
        breakpoint already hit 1 time
2       breakpoint     keep y   0x00000000004004f0 in mult at sample.c:5
        breakpoint already hit 1 time
```

GDB

- Comment régler la question : « *Ma fonction est appelée 100 fois, comment mettre un breakpoint seulement sur certaines occurrences ?* »
- Idée : Les conditions
- La condition doit toujours être une expression C valide et doit pouvoir être évaluée comme un booléen (vrai/faux)
- Le contexte de la condition est la frame unrollée et non sa parente. Les variables locales sont donc accessibles
- Exemple: `break mult if a == 10`

GDB

- Un breakpoint est souvent posé à priori, sans savoir ou le programme présente un bug. On peut donc se poser la question : « *Comment continuer le programme jusqu'au bug tout en inspectant chaque instruction ?* »
- Idée : le « pas-à-pas » (= *stepping*)
- Une fois le breakpoint atteint, il est possible d'avancer **bloc par bloc**, revenant à mettre un breakpoint sur chaque ligne (=**`tbreak`**)
- `step` : ligne de code suivante (peut prendre un nombre en argument)
- `next` : ligne de code suivante, sans descendre dans les fonctions appelées
- continue : reprendre une exécution normale
- Pour avoir un grain **par instruction**, utiliser les versions suffixées par « i » (**`stepi`**, **`nexti`**…)
- Terminer la fonction courante (jusqu'au **`return`**) : **`finish`**

GDB

- Obtenir des informations: `info`
- Gestion des signaux : `handle`
- Quel type de variable ? : **`whatis`**
- Code machine ? **`Disassemble`**

```
gdb >> disassemble /m mult
Dump of assembler code for function mult:
7       {
   0x00000000004004e6 <+0>:     push   %rbp
   0x00000000004004e7 <+1>:     mov    %rsp,%rbp
   0x00000000004004ea <+4>:     mov    %edi,-0x4(%rbp)
   0x00000000004004ed <+7>:     mov    %esi,-0x8(%rbp)

8               return a * b;
   0x00000000004004f0 <+10>:    mov    -0x4(%rbp),%eax
   0x00000000004004f3 <+13>:    imul   -0x8(%rbp),%eax

9       }
   0x00000000004004f7 <+17>:    pop    %rbp
   0x00000000004004f8 <+18>:    retq

End of assembler dump.
gdb >>
```

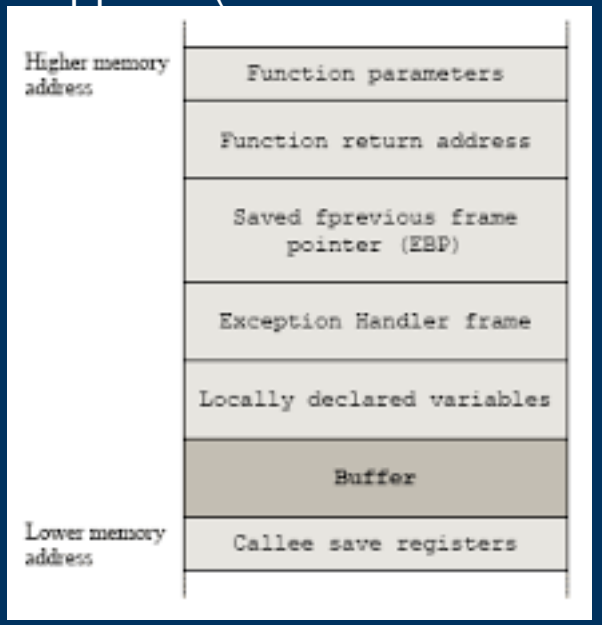
```
gdb >> help info
Generic command for showing things about the program being deb

List of info subcommands:

info address -- Describe where symbol SYM is stored
info all-registers -- List of all registers and their contents
info args -- Argument variables of current stack frame
info auto-load -- Print current status of auto-loaded files
info auxv -- Display the inferior's auxiliary vector
info bookmarks -- Status of user-settable bookmarks
info breakpoints -- Status of specified breakpoints (all user-
info checkpoints -- IDs of currently known checkpoints
info classes -- All Objective-C classes
info common -- Print out the values contained in a Fortran COM
info copying -- Conditions for redistributing copies of GDB
info dcache -- Print information on the dcache performance
info display -- Expressions to display when program stops
info exceptions -- List all Ada exception names
info extensions -- All filename extensions associated with a s
info files -- Names of targets and files being debugged
info float -- Print the status of the floating point unit
info frame -- All about selected stack frame
info frame-filter -- List all registered Python frame-filters
info functions -- All function names
info guile -- Prefix command for Guile info displays
info handle -- What debugger does when program gets various si
info inferiors -- IDs of specified inferiors (all inferiors if
info line -- Core addresses of the code for a source line
info locals -- Local variables of current stack frame
info macro -- Show the definition of MACRO
info macros -- Show the definitions of all macros at LINESPEC
info mem -- Memory region attributes
info os -- Show OS data ARG
info pretty-printer -- GDB command to list all registered pret
info probes -- Show available static probes
info proc -- Show /proc process information about any running
info program -- Execution status of the program
info record -- Info record options
info registers -- List of integer registers and their contents
info scope -- List the variables local to a scope
info selectors -- All Objective-C selectors
info set -- Show all GDB settings
```

Rappel: Layout de la pile

- La pile est une superposition couches appelées « frame », toutes identiques. Une stackframe est créée à chaque fois qu'une nouvelle fonction est appelée (instruction x86 **`call*`**)
- Dans une stack-frame est stockée :
  - Les arguments de fonctions
  - L'adresse de retour RIP dans la fonction parente (pour *Return Instruction Pointer*)
  - Le pointeur de pile **RBP** de la frame précédente
  - Les variables automatiques (dites « locales »)
- Il existe deux pointeurs de pointeurs de pile
  - RBP : « Base pointer » = l'adresse où commence la frame courante
  - RSP : « Stack Pointer » = l'adresse qui suit la dernière adresse accessible pour la frame courante




```
gdb >> disas main
Dump of assembler code for function main:
   0x000000000040052e <+0>:     push   %rbp
   0x000000000040052f <+1>:     mov    %rsp,%rbp
   0x0000000000400532 <+4>:     sub    $0x10,%rsp
```

Manipulation des Registres:

- **r* = 64 bits**
- **e* = 32 bits**
- **l* = 16 bits**

GDB : Threading

- Un des gros avantages d'un debugger est d'aider à gérer plusieurs flots d'exécution comme les pthreads.
- Lister les threads : **`info threads`**
- Par défaut, le thread #0 est celui actif
- Changer de thread: **`thread #x`**
- Exécuter une même commande sur plusieurs threads:

  `thread apply #x #y <cmd>`
- Par défaut, un breakpoint est mis à l'échelle d'un processus
- Un breakpoint thread-specific : **`break <ref> thread #x if …`**
- Attention, un thread doit exister pour être utilisé dans un breakpoint !

GDB : Multi-processing

- Par défaut, GDB ne « suit » pas les processus descendants du processus courant. Configurable via la variable **`follow-fork-mode`**
  - Child: Debug du processus créé
  - Parent: Debug du processus initial (défaut)
- Mais qu'arrive-t-il à l'autre processus ? Configurable via la variable detach-on-fork :
  - On : détache le processus non sélectionné (défaut)
  - Off : GDB suit les deux processus. Celui non sélectionné est mis en suspens (défaut)
- Que se passe-t-il si l'application invoque **`exec*()`** ? **`Follow-exec-mode`**

GDB : Scripting

- Souvent en HPC, il n'est pas possible ou pratique d'avoir un prompt interactif (ex: app avec 64 processus)
- GDB fournit une interface légère de scripting soumis via la ligne de commande : **`-x file | -command=file`**
- Possible aussi via stdin : **`gdb < file`**
- Ou chargeable depuis le prompt : **`source file`**
- Prologue :
  - Set breakpoint pending on : Mise en place de breakpoints sans connaitre le mapping actuel en mémoire. GDB n'avertit pas lorsque le breakpoint n'a pas pu être posé
  - Set pagination off : Éviter la pagination (« Type <return> to continue… »)
  - Set logging on: conserver la sortie (set logging file)
- Epilogue : **`run`**

Valgrind

- `Spack install valgrind`
- Framework d'instrumentation: fournit les méthodes pour permettre à des outils d'analyse dynamiques
- **Memcheck**: problème de gestion mémoire
  - Accès mémoire invalides
  - Utilisation de mémoire non-initialisée
  - Fuites
  - Double frees & Co
  - Overlap de memcpy

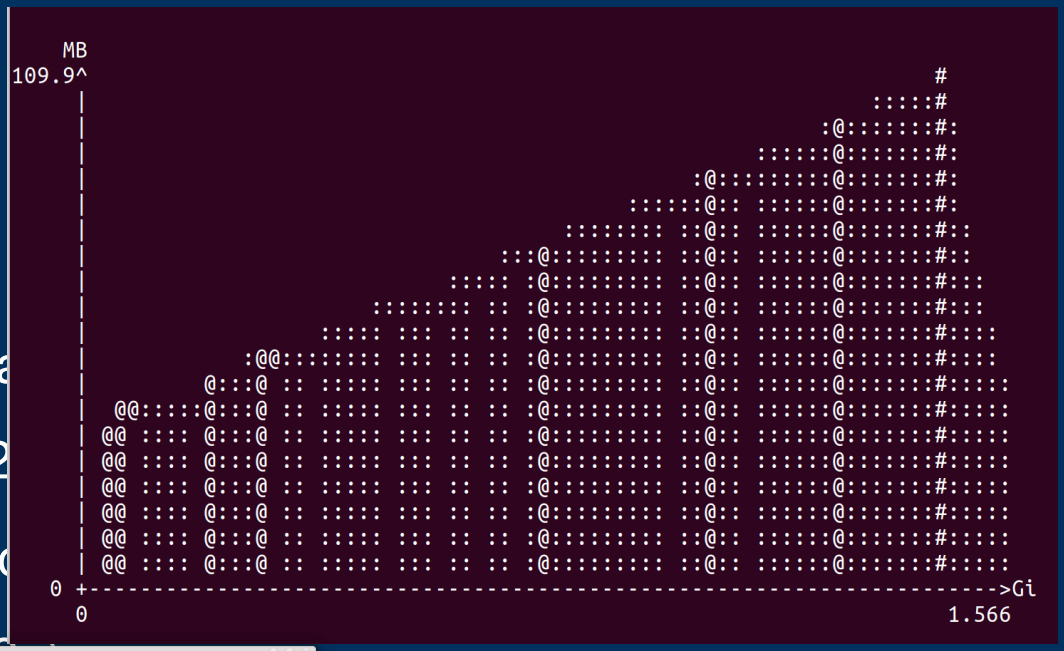
```
==1275213==
==1275213== HEAP SUMMARY:
==1275213==     in use at exit: 100 bytes in 1 blocks
==1275213==   total heap usage: 3 allocs, 2 frees, 2,148 bytes allocated
==1275213==
==1275213== 100 bytes in 1 blocks are definitely lost in loss record 1 of 1
==1275213==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpre
==1275213==    by 0x1092BB: main (in /home/adamj/a.out)
==1275213==
==1275213== LEAK SUMMARY:
==1275213==    definitely lost: 100 bytes in 1 blocks
==1275213==    indirectly lost: 0 bytes in 0 blocks
==1275213==      possibly lost: 0 bytes in 0 blocks
==1275213==    still reachable: 0 bytes in 0 blocks
==1275213==         suppressed: 0 bytes in 0 blocks
==1275213==
==1275213== For lists of detected and suppressed errors, rerun with: -s
==1275213== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

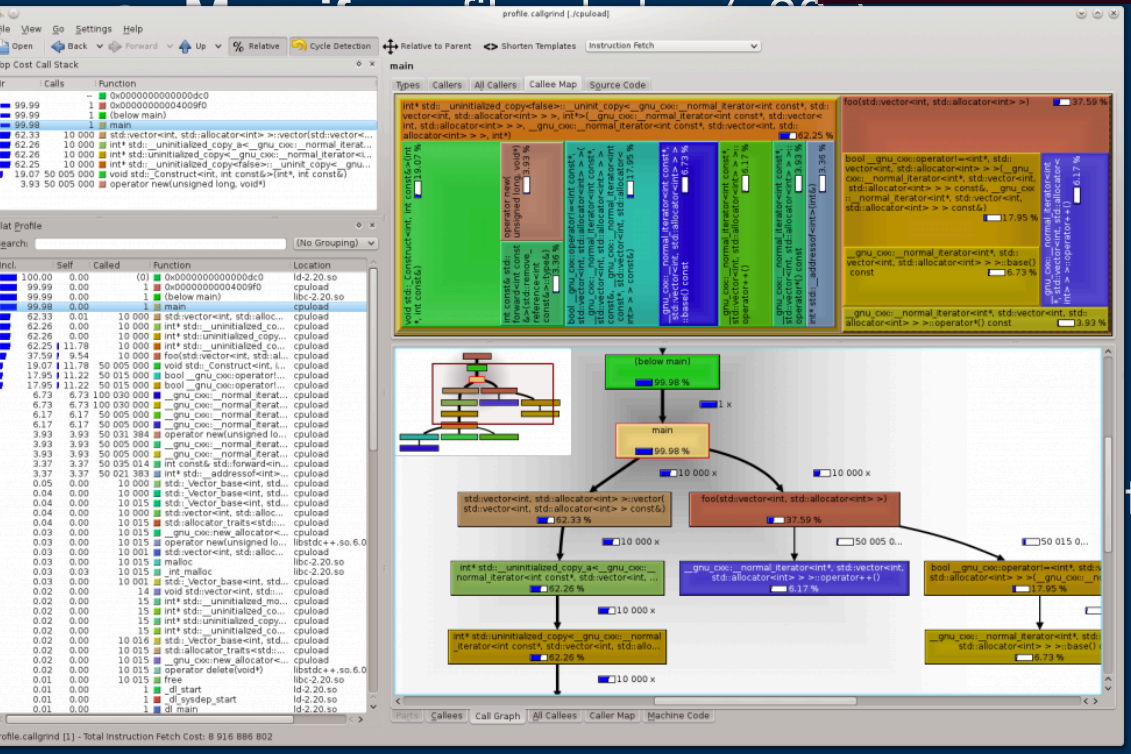
Valgrind

- Mais attention, instrumentation = overhead
- Codes sous memcheck ~20x plus lent
- **Cachegrind**: profiler d'accès aux caches (~70x)
- **Massif**: profiler de tas (~20x)
- **Callgrind**: graphe d'appel
- Helgrind: Threads
- …
- Certains outils ont une sortie analytique directe, d'autres ont besoin d'outils de visualisation pour être exploitable

Valgrind

- Mais attention, instrumenta
- Codes sous memcheck ~2
- **Cachegrind**: profiler d'acc





directe, d'autres ont besoin

table

Perf

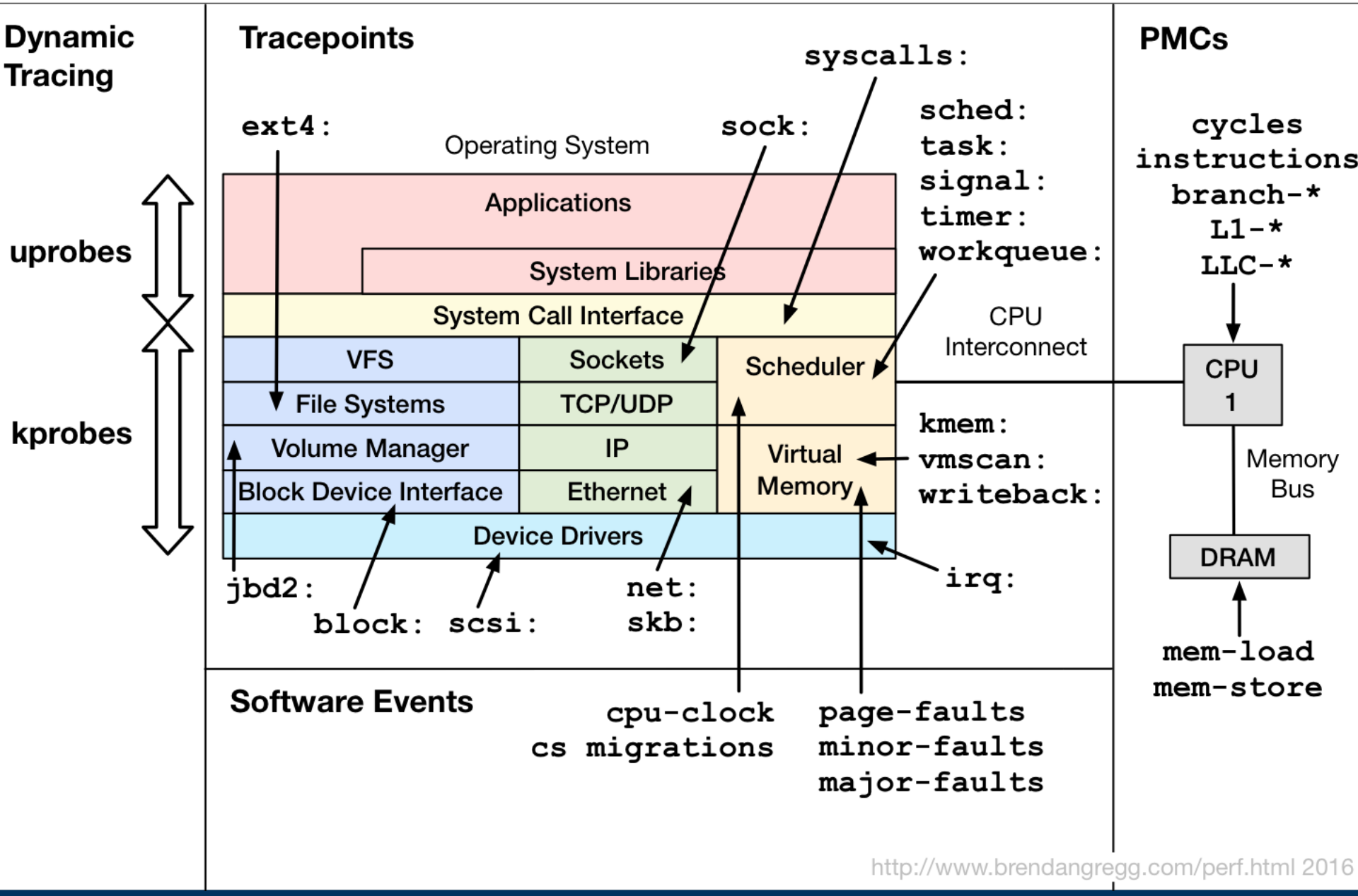
- Disponible depuis Linux kernel version 2.6.31
- Trace les compteurs relativement aux piles
- Très utilisé pour compter les cycles d'un app
- Mais peut faire beaucoup plus
- Une seule CLI: « perf »
- Supporte le « tracing » et le « sampling »

➡`yum install perf`

➡apt-get install linux-tools-common linux-tools-generic linux-tools-`uname -r`

➡git clone https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git

Linux perf_events Event Sources



Perf

- Se base beaucoup sur ptrace()
- Limitation des processus
- Limitation des capacités allouées par root

```
$ perf record -a

Error:
You may not have permission to collect system-wide stats.

Consider tweaking /proc/sys/kernel/perf_event_paranoid,
which controls use of the performance events system by
unprivileged users (without CAP_SYS_ADMIN).

The current value is 1:

  -1: Allow use of (almost) all events by all users
>= 0: Disallow raw tracepoint access by users without CAP_IOC_LOCK
>= 1: Disallow CPU event access by users without CAP_SYS_ADMIN
>= 2: Disallow kernel profiling by users without CAP_SYS_ADMIN
```

Perf

- Stats globales du programme:

```
$perf stat date
Thu May 24 17:44:55 CEST 2018

 Performance counter stats for 'date':

          0.889236      task-clock:u (msec)       #    0.311 CPUs utilized
                 0      context-switches:u        #    0.000 K/sec
                 0      cpu-migrations:u          #    0.000 K/sec
               156      page-faults:u             #    0.175 M/sec
            452575      cycles:u                  #    0.509 GHz
            211872      instructions:u            #    0.47  insn per cycle
             47041      branches:u                #   52.900 M/sec
              4193      branch-misses:u           #    8.91% of all branches

       0.002858312 sec
```

```
$perf stat -d date
Thu May 24 17:46:55 CEST 2018

 Performance counter stats for 'date':

          0.888090      task-clock:u (msec)       #    0.639 CPUs utilized
                 0      context-switches:u        #    0.000 K/sec
                 0      cpu-migrations:u          #    0.000 K/sec
               156      page-faults:u             #    0.176 M/sec
                 0      cycles:u                  #    0.000 GHz
            211872      instructions:u
             47041      branches:u                #   52.969 M/sec
              4175      branch-misses:u           #    8.88% of all branches
             55889      L1-dcache-loads:u         #   62.932 M/sec
              6000      L1-dcache-load-misses:u   #   10.74% of all L1-dcache hits
   <not supported>      LLC-loads:u
   <not supported>      LLC-load-misses:u

       0.001389281 seconds time elapsed
```

Perf

- Plusieurs exécutions pour lisser le bruit: -r <rep>

```
Performance counter stats for 'date' (1000 runs):

        0,285749      task-clock (msec)         #    0,643 CPUs utilized            ( +-  0,06% )
               2      context-switches          #    0,007 M/sec                    ( +-  0,18% )
               0      cpu-migrations            #    0,000 K/sec
             186      page-faults               #    0,651 M/sec                    ( +-  0,00% )
         855 254      cycles                    #    2,993 GHz                      ( +-  1,39% )  (0,00%)
         810 631      instructions              #    0,95  insn per cycle           ( +-  0,04% )
         158 770      branches                  #  555,629 M/sec                    ( +-  0,03% )
           6 739      branch-misses             #    4,24% of all branches          ( +-  0,07% )
         216 810      L1-dcache-loads           #  758,743 M/sec                    ( +-  0,03% )
    <not counted>     L1-dcache-load-misses                                        ( +-  4,20% )  (23,72%)
    <not counted>     LLC-loads                                                    (0,00%)
    <not counted>     LLC-load-misses                                              (0,00%)

      0,000444485 seconds time elapsed                                          ( +-  0,11% )
```

perf list

- Liste les évènements disponibles:

$ perf list -v # List all events you can use (-v for verbose desc.)

$ perf list -v float # List all events linked to FP operations

$ perf list -v l3 # List all events linked to the L3 cache

perf record

Récupère les stats d’un code

```
$ perf record -g date # Record cycles for « date »
```

« -g » capture les call stacks

Résultats dans « perf.data »

```
$perf record -g -e L1-dcache-load-misses date
```

Selection d’events avec « -e », il vaut mieux limiter leur nombre pour éviter un salin trop important

perf report

```
$ perf record -g date
$ perf report --stdio --sort parent
```

```
100.00%   100.00%  [other]
          |
          |--37.76%--_dl_sysdep_start
          |          dl_main
          |          |
          |           --29.08%--_dl_relocate_object
          |                     _dl_lookup_symbol_x
          |
          |--20.32%--_dl_addr
          |
          |--20.32%--0x2342
          |
          |--13.85%--0x1c33a
          |          _dl_map_object
          |          open_path
          |          open_verify
          |
          |--4.46%--strlen
          |
          |--2.05%--_dl_start_user
          |          _dl_start
          |          async_page_fault
          |
           --1.06%--_dl_start
                     |
                      --0.75%--async_page_fault
```

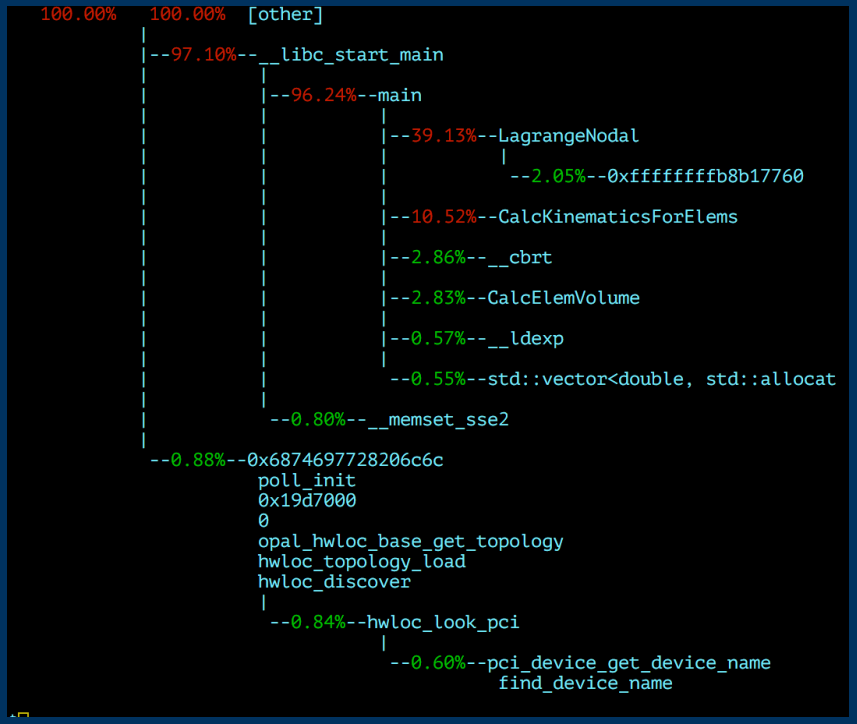
perf report « lulesh »

```
$ perf record  -e cycles -g ./lulesh2.0 -i 50
$ perf report  -g --sort parent
```

```
Samples: 3K of event 'cycles:u', Event count (approx.): 3025875782
  Children      Self  Parent symbol
-  100,00%   100,00%  [other]
   - 97,10% __libc_start_main
      - 96,24% main
         - 39,13% LagrangeNodal
              0xffffffffb8b17760
           10,52% CalcKinematicsForElems
           2,86% __cbrt
           2,83% CalcElemVolume
           0,57% __ldexp
           0,55% std::vector<double, std::allocator<double> >::resize
        0,80% __memset_sse2
   - 0,88% 0x6874697728206c6c
        poll_init
        0x19d7000
        0
        opal_hwloc_base_get_topology
        hwloc_topology_load
      - hwloc_discover
         - 0,84% hwloc_look_pci
            - 0,60% pci_device_get_device_name
                 find_device_name
```

perf report « lulesh »

```
$ perf report  -g --sort parent --stdio
```



perf top

```
$ yes > /dev/null&
[1] 13755
$ perf top -p 13755
```

```
Samples: 94K of event 'cycles:u', Event count (approx.): 53656556765
Overhead  Shared Object      Symbol
  32,84%  libc-2.17.so       [.] _IO_file_xsputn@@GLIBC_2.2.5
  22,74%  libc-2.17.so       [.] fputs_unlocked
  13,46%  libc-2.17.so       [.] __strlen_sse2
  12,45%  libc-2.17.so       [.] __GI___mempcpy
   2,60%  yes                [.] 0x00000000000015af
   2,59%  yes                [.] 0x00000000000015c5
   2,55%  yes                [.] 0x00000000000015a0
   2,52%  yes                [.] 0x00000000000012f0
   2,15%  yes                [.] 0x0000000000001590
   2,13%  yes                [.] 0x0000000000001628
   2,11%  yes                [.] 0x000000000000159c
   0,44%  yes                [.] 0x00000000000015c8
   0,39%  yes                [.] 0x0000000000001595
   0,39%  yes                [.] 0x00000000000015d5
   0,38%  yes                [.] 0x000000000000162f
   0,04%  libc-2.17.so       [.] _IO_do_write@@GLIBC_2.2.5
For a higher level overview, try: perf top --sort comm,dso
```

Trace de programme en temps réel

perf top -a

```
# perf top -a -e syscalls:sys_enter_open
```

Trace global (à l'échelle du système) du syscall **`open`** (kernel tracepoint)

REQUIRES ROOT

```
syscalls:sys_enter_accept4                [Tracepoint event]
syscalls:sys_enter_access                 [Tracepoint event]
syscalls:sys_enter_acct                   [Tracepoint event]
syscalls:sys_enter_add_key                [Tracepoint event]
syscalls:sys_enter_adjtimex               [Tracepoint event]
syscalls:sys_enter_alarm                  [Tracepoint event]
syscalls:sys_enter_bind                   [Tracepoint event]
syscalls:sys_enter_brk                    [Tracepoint event]
syscalls:sys_enter_capget                 [Tracepoint event]
syscalls:sys_enter_capset                 [Tracepoint event]
syscalls:sys_enter_chdir                  [Tracepoint event]
syscalls:sys_enter_chmod                  [Tracepoint event]
syscalls:sys_enter_chown                  [Tracepoint event]
syscalls:sys_enter_chroot                 [Tracepoint event]
syscalls:sys_enter_clock_adjtime          [Tracepoint event]
syscalls:sys_enter_clock_getres           [Tracepoint event]
```

Perf est **beaucoup** plus performant en root, car un grand nombre d'events ne sont disponibles qu'avec un utilisateur privilégié

Perf et programmes MPI

Perf est orienté processus. Pour fonctionner avec un code MPI, il est nécessaire de séparer la sortie dans des fichiers séparés. Ici, un simple script fait l'affaire.

```
#!/bin/sh
H=`hostname`
P=$$
perf record -o perf-${H}-${P}.data  $@
```

```
$ srun -n 8 script.sh -g ./matmult
```

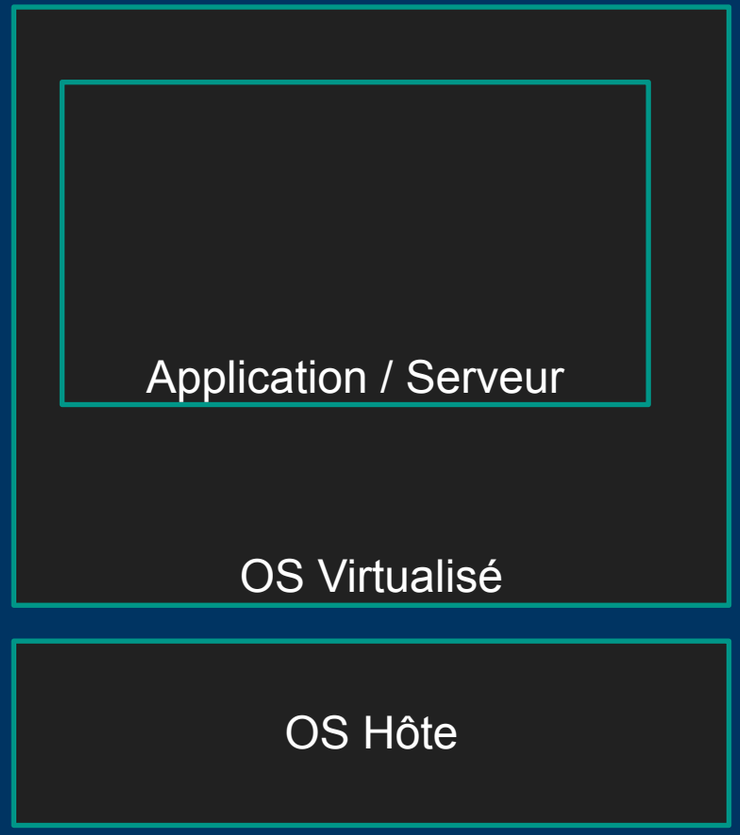
```
$ perf report -i <one-perf-file>.data -g
```

Machines Virtuelles

Virtualisation

Une machine virtuelle:

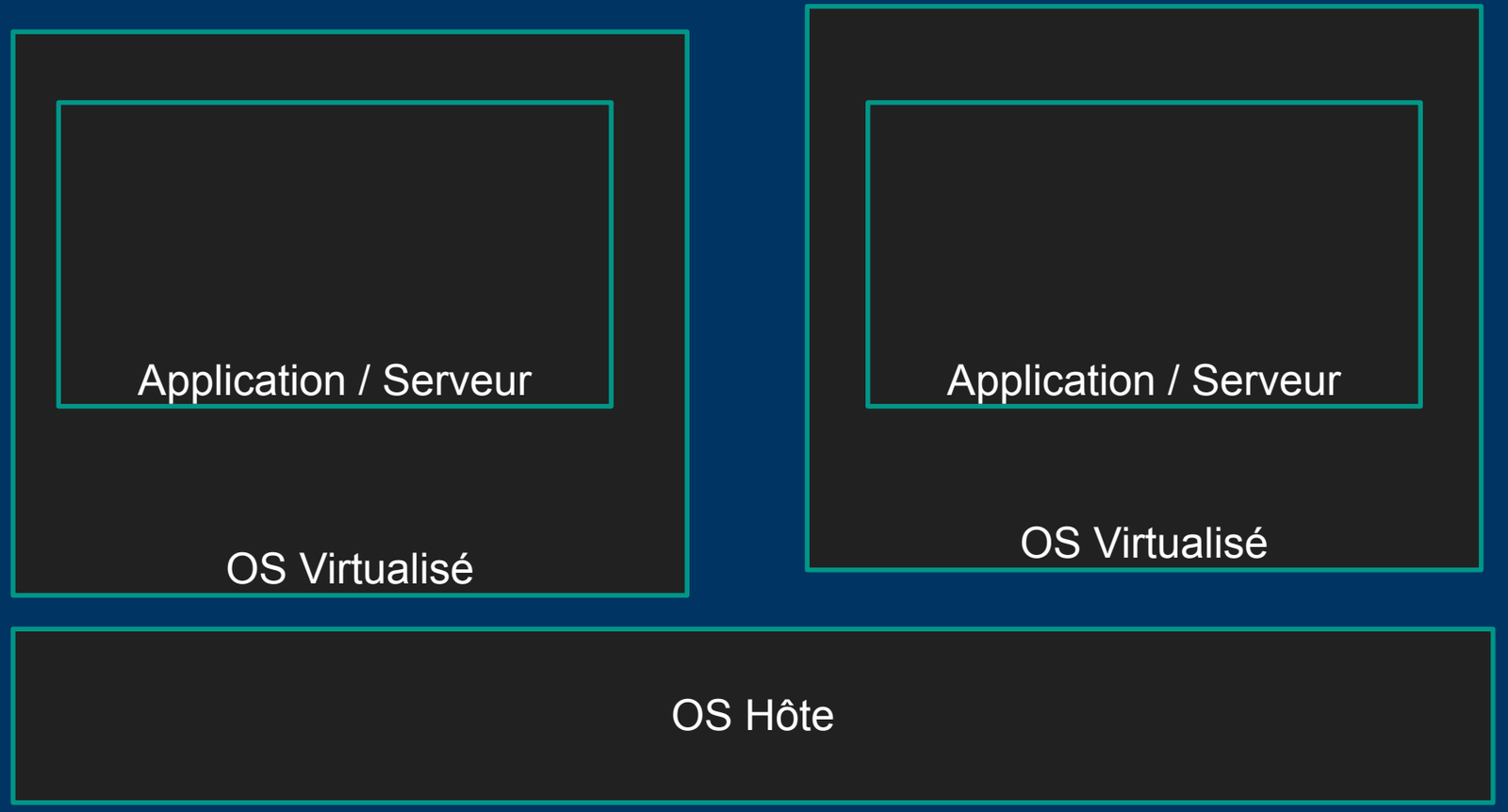
- Émule une machine de manière logicielle pour permettre l'exécution isolée d'un programme;
- Execute du code dans un contexte spécifique (souvent avec l'aide du matériel) pour le contraindre en terme d'accès;



Virtualisation

Une machine virtuelle:

- Émule une machine de manière logicielle pour permettre l'exécution isolée d'un programme;
- Execute du code dans un contexte spécifique (souvent avec l'aide du matériel) pour le contraindre en terme d'accès;



Avantages de la Virtualisation

- Regroupement des serveurs sur une même machine. De plus certains serveurs sont faibles en consommation CPU.
- Isolation FORTE des serveurs avec des **systèmes d'exploitation différents et des systèmes de fichiers distincts**;
- **Isolation** y compris vis à vis du matériel (carte réseau virtuelle) et contraintes mémoire cpu explicite (exposition partielle des resources) — **gestion dynamique des resources** (CPU Hotplug);
- **Réplication** et **sauvegarde** facilitée (on sauve l'image disque dans sa totalité), rétablir le système c'est rétablir une image plus récente;
- Facilité d'administration il devient possible de **migrer** un serveur/service donné.

Inconvénients de la Virtualisation

- Les abstractions matérielles ont un coût en performance non négligeable;
- L'OS est totalement répliqué en stockage et en mémoire dans les différentes VMs;
- Si un serveur avec de nombreuses VMs tombe toutes les VMs associées sont inopérantes (besoin de redondance);
- Il y a un overhead d'administration important du fait de la complexité additionnelles des machines séparées.

Image Debian avec Docker

Récupérez une image avec debian:

root MDP toto
chps MDP toto

https://france.paratools.com/chps.qcow2

CTRL+ALT pour sortir la souris

Le Conteneur

Conteneurs ?

- C’est beaucoup de chose, cela veut principalement dire que les ressources d’un processus sont isolées;
- Ceci utilise la notion de namespace:
  - ➡ Mount namespaces
  - ➡ User-namespaces
  - ➡ Network namespaces
  - ➡ (…)
- En général le conteneur consiste en un changement de système de fichier tout en gardant le même noyau à la différence d’une VM par exemple.

Liste des Namespaces

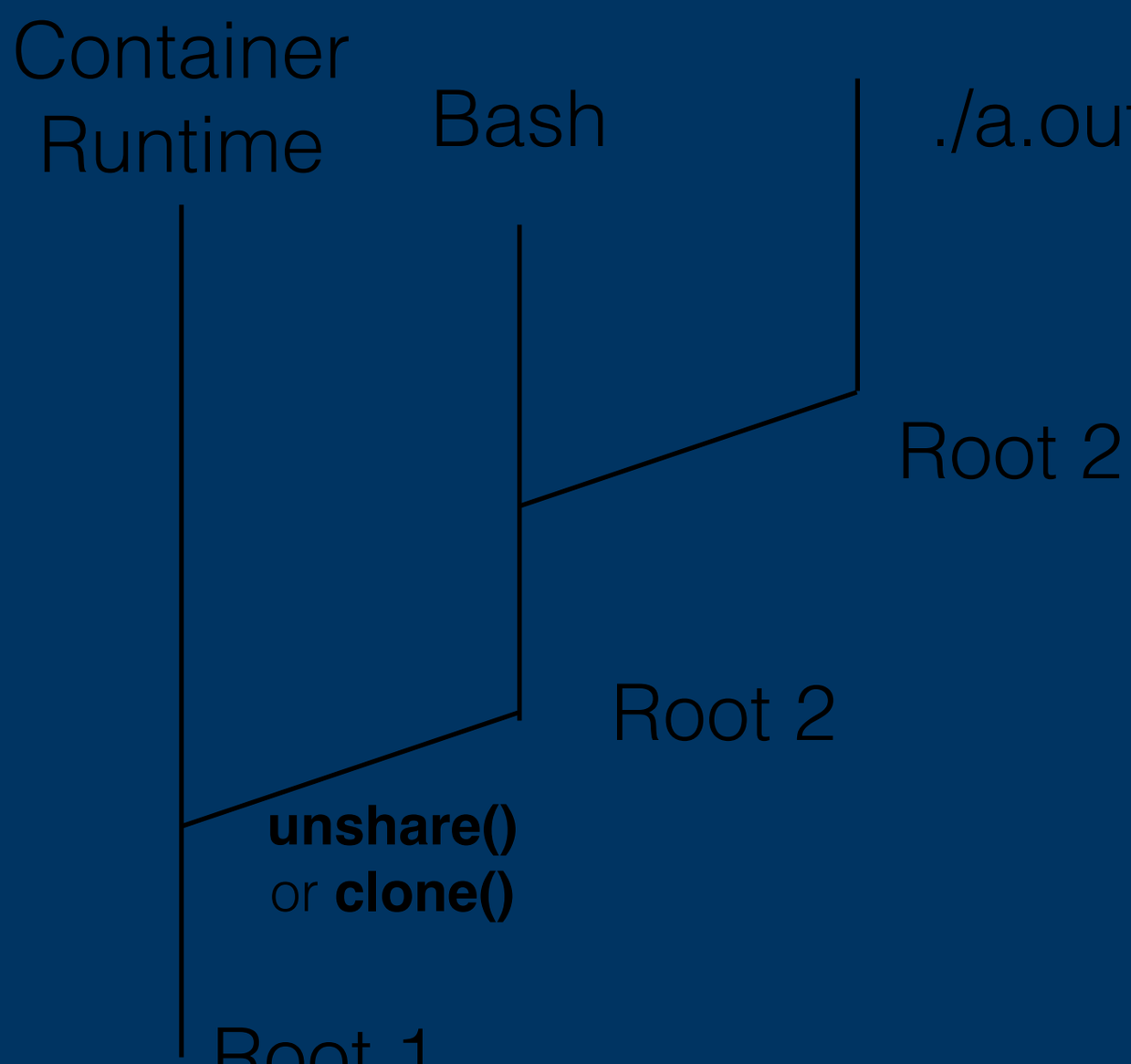
« man namespaces »

Tableau 1

Namespace	Constant	Isolates
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

Un Conteneur

Aspects Runtime



Le Conteneur vu de l'utilisateur

RUN <IMAGE> <COMMAND>

RUN <IMAGE>

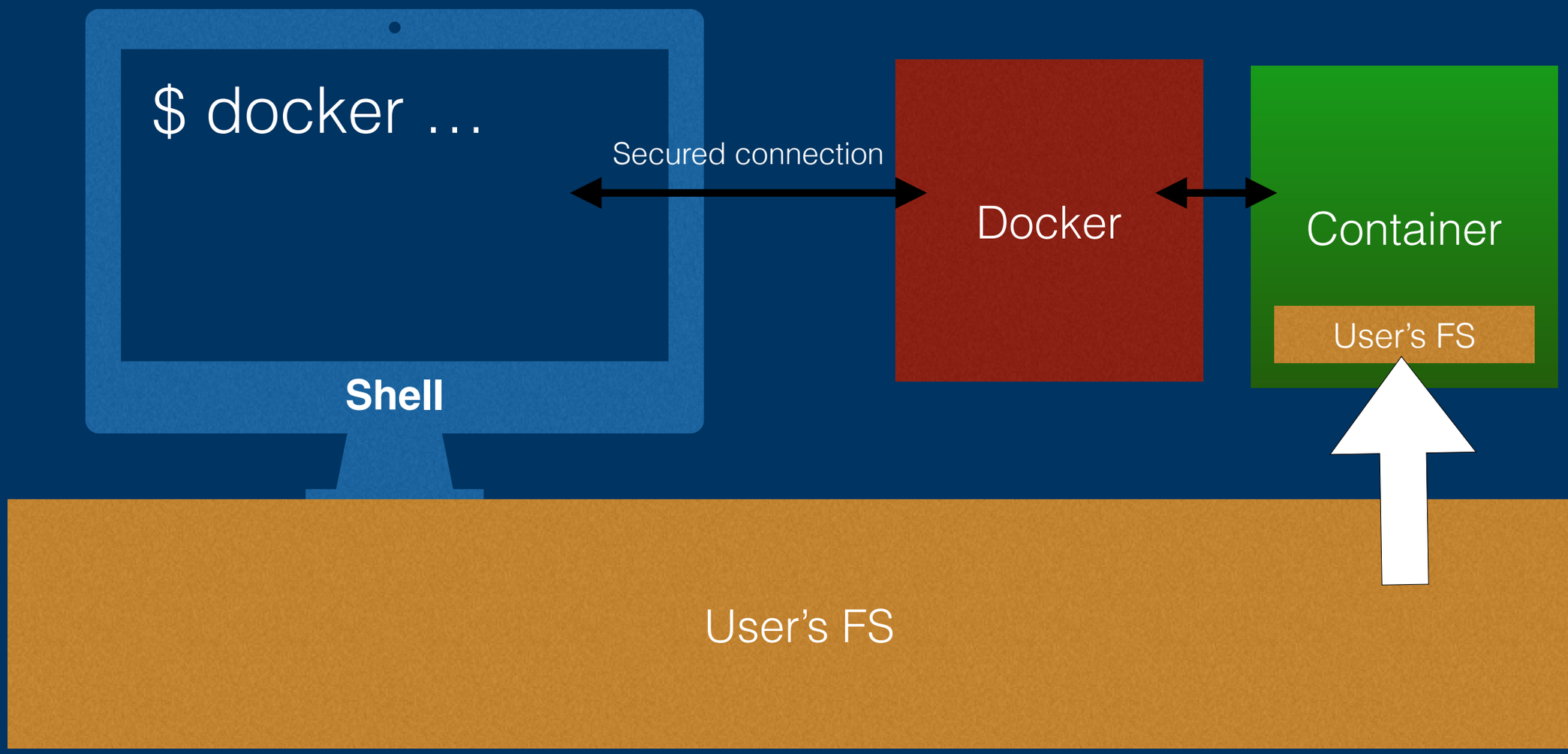
Une image contient toutes les dépendances pour lancer un programmes (binaires bibliothèques …) mais PAS le kernel.

Docker

Commencer avec Docker

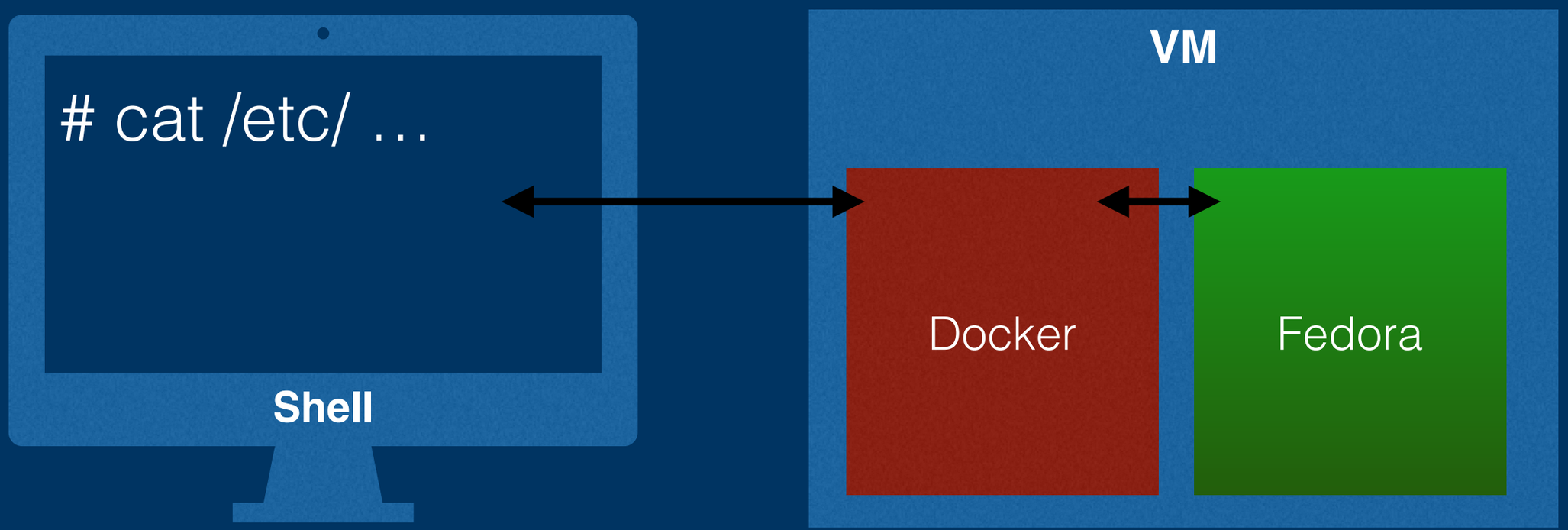
```
$ docker version
Client: Docker Engine - Community
 Version:           18.09.5
(…)
```

Le Démon Docker



First Run inside Docker

```
$ docker run -ti fedora
(Entering container … )
# cat /etc/redhat-release
Fedora release 31 (Thirty One)
```



Syntaxe Docker Run

```
docker run -ti --rm [IMAGE] [COMMAND] [ARGS]
```

- -i : mode interactif (par défaut Docker tourne en arrière plan);
- -t : pour ouvrir un TTY et donc avoir un support terminal complet pour VIM par exemple;
- --rm: supprimer le conteneur à la sortie du programme.

Lister les Images Docker

Pour voir les images disponibles allez sur Docker HUB
https://hub.docker.com/search?q=&type=image

```
$ docker image list
REPOSITORY TAG    IMAGE ID     CREATED       SIZE
debian     latest b5d2d9b1597b 11 days ago   114MB
alpine     latest cc0abc535e36 2 weeks ago   5.59MB
ubuntu     latest 549b9b86cb8d 2 weeks ago   64.2MB
fedora     latest f0858ad3febd 2 months ago 194MB
```

Lister les Conteneurs

```
$ docker image list
REPOSITORY                TAG                 IMAGE ID            CREATED             SIZE
jupyter/scipy-notebook    latest              295a5802d799        2 days ago          3.45GB
pcocc/umoci               v0.1                27530b76a409        7 months ago        11.5MB
pcocc/umoci               v0.1-amd64          27530b76a409        7 months ago        11.5MB
pcocc/tzdata              v0.1                4a126134da60        7 months ago        1.53MB
pcocc/tzdata              v0.1-amd64          4a126134da60        7 months ago        1.53MB
pcocc/squashfs-tools      v0.1                0da08da32ae5        7 months ago        2.11MB
pcocc/squashfs-tools      v0.1-amd64          0da08da32ae5        7 months ago        2.11MB
(…)
```

Lancer un Conteneur Détaché

```
$ docker run -ti -d  --rm ubuntu sleep 10000

$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
8aa628c5982a        ubuntu              "sleep 100000"      About a minute ago   Up About a minute                       recursing_herschel
$ docker attach recursing_herschel
(…)
# CTRL + p puis CTRL + q
read escape sequence
```

Pour se détacher CTRL+p puis CTRL+q
Uniquement si lancé avec -ti !!!!

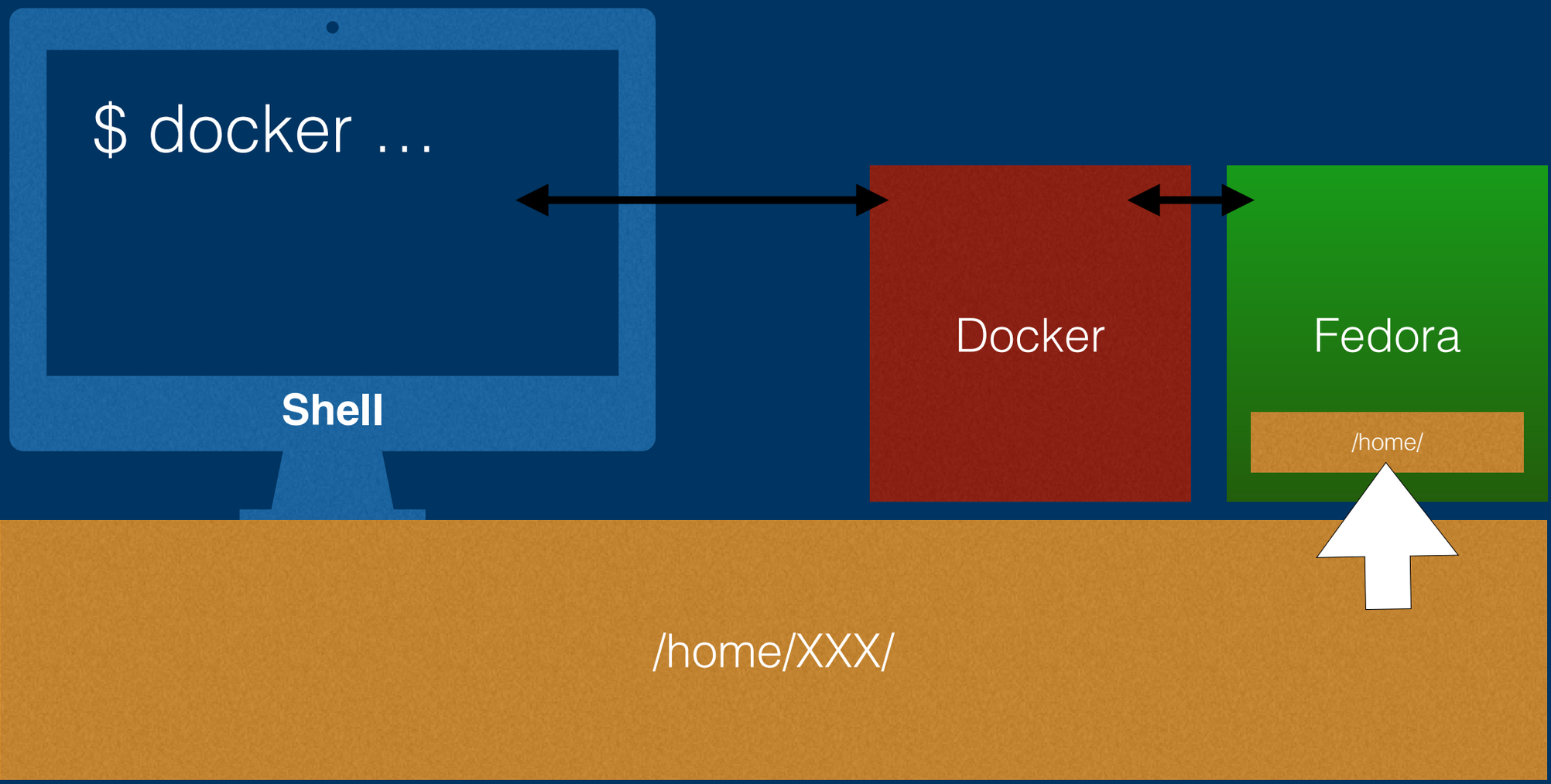
Monter le $HOME

```
cd $HOME; touch ./hello_docker
docker run -ti \
   --rm -v $HOME:/home -w /home fedora
ls ./hello_docker
./hello_docker
```

- -v : Volume (mounting dir & files)
  [FROM]:[TO]

- -w : work directory (the CWD of the command being run) here bash by default.

Monter le $HOME



Voir la Commande par Défaut

$ docker inspect nginx

```
"Env": [
    "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
    "NGINX_VERSION=1.17.0",
    "NJS_VERSION=0.3.2",
    "PKG_RELEASE=1~stretch"
],
"Cmd": [
    "/bin/sh",
    "-c",
    "#(nop) ",
    "CMD [\"nginx\" \"-g\" \"daemon off;\"]"
],
```

Altérer la Commande par Défaut

```
$ docker run [OPTIONS] IMAGE[:TAG] [COMMAND] [ARG...]
```

$ docker run -ti --rm nginx [CMD] [ARGS]

$ docker run -ti --rm nginx /bin/bash

```
$ docker run -ti  --rm nginx /bin/bash
root@2fa50e296016:/#
```

Exit ou CTRL + D pour quitter

Editer une image existante

```
$ docker run -ti \
    --name mycont -v $HOME:/home/ fedora
cp /home/my_data /data
CTRL+D
```

```
$ docker ps -a #-a for terminated cont.
```

```
CONTAINER ID        IMAGE COMMAND CREATED           STATUS                      PORTS               NAMES
044ed20ab55b        fedora   « /bin/bash" 32 seconds ago      Exited (0) 20 seconds ago  hungry_rosalind
```

```
$ docker commit hungry_rosalind myfed
```

```
CONTAINER ID        IMAGE COMMAND CREATED           STATUS                      PORTS               NAMES

044ed20ab55b        fedora   « /bin/bash" 32 seconds ago      Exited (0) 20 seconds ago  hungry_rosalind
```

```
$ docker image list
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
myfed	**latest**	**93006aa38912**	**7 seconds ago**	**194MB**
fedora	**latest**	**f0858ad3febd**	**2 months ago**	**194MB**

Lancer un Serveur

```
docker run -ti --rm -p 8080:80 nginx
```

Redirige 8080 vers le port 80 du conteneur.

```
$ docker run -d  -p 8080:80  nginx

$ curl localhost:8080
(…)
<title>Welcome to nginx!</title>
(…)

$ docker ps
CONTAINER ID        IMAGE               COMMAND                  PORTS                  NAMES
2211a94ebf61        nginx               "nginx -g 'daemon of…"   0.0.0.0:8080->80/tcp   magical_nash

$ docker kill magical_nash
```

Dockerfile
Créez vos propre conteneurs

Le Dockerfile

- Décrit la recette d’un conteneur
- Vise à être reproductible (partager la recette et non le conteneur)
- Est la base du partage sur le HUB Docker:

1	**ADD file ... in /**	**24.65 MB**
2	CMD ["bash"]	0 B
3	LABEL maintainer=NGINX Docker Maintainers	0 B
4	ENV NGINX_VERSION=1.17.8	0 B
5	ENV NJS_VERSION=0.3.8	0 B
6	ENV PKG_RELEASE=1~buster	0 B
7	/bin/sh -c set -x	22.72 MB
8	/bin/sh -c ln -sf /dev/stdout	202 B
9	EXPOSE 80	0 B
10	STOPSIGNAL SIGTERM	0 B
11	CMD ["nginx" "-g" "daemon	0 B

FROM

```
FROM [--platform=<platform>] <image> [AS <name>]
```

Or

```
FROM [--platform=<platform>] <image>[:<tag>] [AS <name>]
```

Or

```
FROM [--platform=<platform>] <image>[@<digest>] [AS <name>]
```

The `FROM` instruction initializes a new build stage and sets the *Base Image* for subsequent instructions. As such, a valid `Dockerfile` must start with a `FROM` instruction. The image can be any valid image – it is especially easy to start by **pulling an image** from the *Public Repositories*.

- `ARG` is the only instruction that may precede `FROM` in the `Dockerfile`. See Understand how ARG and FROM interact.
- `FROM` can appear multiple times within a single `Dockerfile` to create multiple images or use one build stage as a dependency for another. Simply make a note of the last image ID output by the commit before each new `FROM` instruction. Each `FROM` instruction clears any state created by previous instructions.
- Optionally a name can be given to a new build stage by adding `AS name` to the `FROM` instruction. The name can be used in subsequent `FROM` and `COPY --from=<name|index>` instructions to refer to the image built in this stage.
- The `tag` or `digest` values are optional. If you omit either of them, the builder assumes a `latest` tag by default. The builder returns an error if it cannot find the `tag` value.

The optional `--platform` flag can be used to specify the platform of the image in case `FROM` references a multi-platform image. For example, `linux/amd64`, `linux/arm64`, or `windows/amd64`. By default, the target platform of the build request is used. Global build arguments can be used in the value of this flag, for example automatic platform ARGs allow you to force a stage to native build platform (`--platform=$BUILDPLATFORM`), and use it to cross-compile to the target platform inside the stage.

https://docs.docker.com/engine/reference/builder/

RUN

RUN has 2 forms:

- `RUN <command>` (*shell* form, the command is run in a shell, which by default is `/bin/sh -c` on Linux or `cmd /S /C` on Windows)
- `RUN ["executable", "param1", "param2"]` (*exec* form)

The `RUN` instruction will execute any commands in a new layer on top of the current image and commit the results. The resulting committed image will be used for the next step in the `Dockerfile`.

Layering `RUN` instructions and generating commits conforms to the core concepts of Docker where commits are cheap and containers can be created from any point in an image's history, much like source control.

The *exec* form makes it possible to avoid shell string munging, and to `RUN` commands using a base image that does not contain the specified shell executable.

The default shell for the *shell* form can be changed using the `SHELL` command.

In the *shell* form you can use a `\` (backslash) to continue a single RUN instruction onto the next line. For example, consider these two lines:

```
RUN /bin/bash -c 'source $HOME/.bashrc; \
echo $HOME'
```

Together they are equivalent to this single line:

```
RUN /bin/bash -c 'source $HOME/.bashrc; echo $HOME'
```

EXPOSE

`EXPOSE <port> [<port>/<protocol>...]`

The `EXPOSE` instruction informs Docker that the container listens on the specified network ports at runtime. You can specify whether the port listens on TCP or UDP, and the default is TCP if the protocol is not specified.

The `EXPOSE` instruction does not actually publish the port. It functions as a type of documentation between the person who builds the image and the person who runs the container, about which ports are intended to be published. To actually publish the port when running the container, use the `-p` flag on `docker run` to publish and map one or more ports, or the `-P` flag to publish all exposed ports and map them to high-order ports.

By default, `EXPOSE` assumes TCP. You can also specify UDP:

```
EXPOSE 80/udp
```

To expose on both TCP and UDP, include two lines:

```
EXPOSE 80/tcp
EXPOSE 80/udp
```

$ docker run -d -P nginx

$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
bdc0d96b31b7 nginx "nginx -g 'daemon of…" 14 seconds ago Up 13 seconds 0.0.0.0:32768->80/tcp boring_brattain

https://docs.docker.com/engine/reference/builder/#expose

ENV

`ENV <key> <value>`
`ENV <key>=<value> ...`
The `ENV` instruction sets the environment variable `<key>` to the value `<value>`. This value will be in the environment for all subsequent instructions in the build stage and can be replaced inline in many as well.

The `ENV` instruction has two forms. The first form, `ENV <key> <value>`, will set a single variable to a value. The entire string after the first space will be treated as the `<value>` - including whitespace characters. The value will be interpreted for other environment variables, so quote characters will be removed if they are not escaped.

The second form, `ENV <key>=<value> ...`, allows for multiple variables to be set at one time. Notice that the second form uses the equals sign (=) in the syntax, while the first form does not. Like command line parsing, quotes and backslashes can be used to include spaces within values.

For example:

```
ENV myName="John Doe" myDog=Rex\ The\ Dog \
    myCat=fluffy
```

and

```
ENV myName John Doe
ENV myDog Rex The Dog
ENV myCat fluffy
```

will yield the same net results in the final image.

The environment variables set using `ENV` will persist when a container is run from the resulting image. You can view the values using `docker inspect`, and change them using `docker run --env <key>=<value>`.

https://docs.docker.com/engine/reference/builder/#env

COPY

COPY has two forms:

- `COPY [--chown=<user>:<group>] <src>... <dest>`
- `COPY [--chown=<user>:<group>] ["<src>",... "<dest>"]` (this form is required for paths containing whitespace)

The `COPY` instruction copies new files or directories from `<src>` and adds them to the filesystem of the container at the path `<dest>`.

Multiple `<src>` resources may be specified but the paths of files and directories will be interpreted as relative to the source of the context of the build.

https://kapeli.com/cheat_sheets/Dockerfile.docset/Contents/Resources/Documents/index

ADD

ADD has two forms:

- `ADD [--chown=<user>:<group>] <src>... <dest>`
- `ADD [--chown=<user>:<group>] ["<src>",... "<dest>"]` (this form is required for paths containing whitespace)

The `ADD` instruction copies new files, directories or remote file URLs from `<src>` and adds them to the filesystem of the image at the path `<dest>`.

Multiple `<src>` resources may be specified but if they are files or directories, their paths are interpreted as relative to the source of the context of the build.

Identique à COPY mais:

- **Supporte les URLs**
- **Extrait les archives !**

https://docs.docker.com/engine/reference/builder/#add

CMD

The `CMD` instruction has three forms:

- `CMD ["executable","param1","param2"]` (*exec* form, this is the preferred form)
- `CMD ["param1","param2"]` (as *default parameters to ENTRYPOINT*)
- `CMD command param1 param2` (*shell* form)

There can only be one `CMD` instruction in a `Dockerfile`. If you list more than one `CMD` then only the last `CMD` will take effect.

The main purpose of a `CMD` is to provide defaults for an executing container. These defaults can include an executable, or they can omit the executable, in which case you must specify an `ENTRYPOINT` instruction as well.

https://docs.docker.com/engine/reference/builder/#cmd

ENTRYPOINT

ENTRYPOINT has two forms:

- `ENTRYPOINT ["executable", "param1", "param2"]` (*exec* form, preferred)
- `ENTRYPOINT command param1 param2` (*shell* form)

An `ENTRYPOINT` allows you to configure a container that will run as an executable.

Un conteneur exécute:
[ENTRYPOINT] [CMD]

https://docs.docker.com/engine/reference/builder/#entrypoint

WORKDIR

`WORKDIR /path/to/workdir`

The `WORKDIR` instruction sets the working directory for any `RUN`, `CMD`, `ENTRYPOINT`, `COPY` and `ADD` instructions that follow it in the `Dockerfile`. If the `WORKDIR` doesn't exist, it will be created even if it's not used in any subsequent `Dockerfile` instruction.

https://docs.docker.com/engine/reference/builder/#workdir

Exemple de Dockerfile

Alpine avec VIM

```
from alpine
RUN apk add vim
CMD ["vim"]
```

$ docker build . -t alpvim

```
Sending build context to Docker daemon  3.072kB
Step 1/3 : from alpine
 ---> 055936d39205
Step 2/3 : RUN apk add vim
 ---> Running in 21c3189729ac
fetch http://dl-cdn.alpinelinux.org/alpine/v3.9/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.9/community/x86_64/APKINDEX.tar.gz
(1/5) Installing lua5.3-libs (5.3.5-r2)
(2/5) Installing ncurses-terminfo-base (6.1_p20190105-r0)
(3/5) Installing ncurses-terminfo (6.1_p20190105-r0)
(4/5) Installing ncurses-libs (6.1_p20190105-r0)
(5/5) Installing vim (8.1.1365-r0)
Executing busybox-1.29.3-r10.trigger
OK: 40 MiB in 19 packages
Removing intermediate container 21c3189729ac
 ---> e47638747864
Step 3/3 : CMD ["vim"]
 ---> Running in ac86ae31b7a3
Removing intermediate container ac86ae31b7a3
 ---> ee258ef6e23e
Successfully built ee258ef6e23e
Successfully tagged alpvim:latest
```

$ docker run -ti alpvim

Ubuntu Serveur Web

```
from ubuntu

RUN apt-get update && \
    apt-get install nginx -y && \
    apt-get clean

EXPOSE 80/tcp

CMD ["nginx", "-g", "daemon off;"]
```

```
$ docker build . -t unginx
$ docker run -d -P --rm unginx
$ docker ps

CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS                     NAMES
5dd75be0ed8a   unginx    "nginx -g 'daemon of…"   18 seconds ago   Up 16 seconds   0.0.0.0:32771->80/tcp     exciting_ride
```

Debian Serveur Web

```
from debian

RUN apt-get update && \
    apt-get install nginx -y && \
    apt-get clean

EXPOSE 80/tcp

CMD ["nginx", "-g", "daemon off;"]
```

```
$ docker build . -t unginx
$ docker run -d -P --rm unginx
$ docker ps

CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS                     NAMES
5dd75be0ed8a   unginx    "nginx -g 'daemon of…"   18 seconds ago   Up 16 seconds   0.0.0.0:32771->80/tcp     exciting_ride
```

La Composition De Conteneurs « docker-compose »

Considérons La Configuration Suivante

Comment démarrer deux conteneurs qui doivent partager des services ?

Docker Compose

docker-compose.yml

```yaml
version: '3.3'

services:
   myredis:
     image: redis
     restart: always
   web:
     depends_on:
       - myredis
     image: mywebsebver
ports:
  - « 8080:80"
restart: always
volumes:
        - ./html/:/var/www/html/
```

Syntaxe YAML !

https://fr.wikipedia.org/wiki/YAML

Serveur Web et Redis

```yaml
version: '3.3'

services:
   myredis:
     image: redis
     restart: always
   web:
     depends_on:
       - myredis
     image: mywebsebver
     ports:
       - « 8080:80"
     restart: always
     volumes:
             - ./html/:/var/www/html/
```

Conteneur base de donnée @ myredis

Conteneur « web » @ web

https://cheatography.com/tasjaevan/cheat-sheets/redis/

Redis

- Stockage clef-valeur avec des structure de donnée prédéfinies:
  - ➡ List
  - ➡ Hash
  - ➡ Sets
  - ➡ …
- Liste de commandes :
  https://cheatography.com/tasjaevan/cheat-sheets/redis/
- Interface extrêmement simple:
- Via netcat 127.0.0.1 6379
- Via redis-cli (du paquet redis-tools)

Serveur Web et Redis

```yaml
version: '3.3'

services:
  myredis:
    image: redis
    restart: always
    ports:
      - "6379:6379"
  web:
    depends_on:
      - myredis
    image: mywebsebver
    ports:
      - "8080:80"
    restart: always
    volumes:
            - ./html/:/var/www/html/
```

Conteneur base de donnée @ myredis

On expose le redis

Conteneur « web » @ web