オープンソース
Sysdig ユーザガイド

本文の内容は、オープンソースのSysdig User Guide(https://github.com/draios/sysdig/wiki/Sysdig-User-Guide)を元に日本語に翻訳・再構成した内容となっております。

基礎

sysdigを使用する最も簡単な方法は、引数なしで呼び出すことです。これを行うと、sdigがstraceと同様に、すべてのイベントをキャプチャして標準出力に書き込むようになります。

```
$ sysdig
34378 12:02:36.269753803 2 echo (7896) > close fd=3(/usr/lib/locale/locale-archive)
34379 12:02:36.269754164 2 echo (7896) < close res=0
34380 12:02:36.269781699 2 echo (7896) > fstat fd=1(/dev/pts/3)
34381 12:02:36.269783882 2 echo (7896) < fstat res=0
34382 12:02:36.269784970 2 echo (7896) > mmap
34383 12:02:36.269786575 2 echo (7896) < mmap
34384 12:02:36.269827674 2 echo (7896) > write fd=1(/dev/pts/3) size=12
34385 12:02:36.269839477 2 echo (7896) < write res=12 data=hello world.
34386 12:02:36.269843986 2 echo (7896) > close fd=1(/dev/pts/3)
34387 12:02:36.269844466 2 echo (7896) < close res=0
34388 12:02:36.269844816 2 echo (7896) > munmap
34389 12:02:36.269850803 2 echo (7896) < munmap
34390 12:02:36.269851915 2 echo (7896) > close fd=2(/dev/pts/3)
34391 12:02:36.269852314 2 echo (7896) < close res=0
```

デフォルトでは、sysdigは各イベントの情報を次の形式で1行に出力します。
`*%evt.num %evt.time %evt.cpu %proc.name (%thread.tid) %evt.dir %evt.type %evt.args`

どこ:

- evt.numは増分イベント番号です
- evt.timeはイベントのタイムスタンプです
- evt.cpuは、イベントがキャプチャされたCPU番号です
- proc.nameは、イベントを生成したプロセスの名前です。
- thread.tidは、イベントを生成したTIDであり、単一スレッドプロセスのPIDに対応します
- evt.dirはイベントの方向であり、>は開始イベント、<は終了イベントです。
- evt.typeはイベントの名前です。 'open'または 'read'
- evt.argsはイベント引数のリストです。システムコールの場合、これらはシステムコール引数に対応する傾向がありますが、常にそうであるとは限りません。一部のシステムコール引数は、単純化またはパフォーマンス上の理由から除外されています。

注意:現在、すべてのシステムコールがsysdigによってデコードされるわけではありません。 デコードされていないシステムコールは引き続き出力に表示されますが、引数はありません。

出力を見ると、この出力とstraceの出力の主な違いをすぐに見つけることができます:

- ほとんどのシステムコールでは、sysdigは2つの個別のエントリを表示します。1つは '>'でマークされたもので、もう1つは '<'でマークされたものです。 これにより、マルチプロセス環境での出力の追跡が容易になります。
- ファイル記述子が解決されます。 これは、可能な場合は常に、FD番号の後にFD自体の人間が読める形式(ネットワーク接続のタプル、ファイルの名前など)が続くことを意味します。

FDのレンダリングに使用される正確な形式は次のとおりです。
num(<type>resolved_string)ここで:

- numはFD番号です
- resolve_stringは、FDの解決された表現です。127.0.0.1:40370->127.0.0.1:80(TCPソケットの場合)
- typeはfdタイプの1文字のエンコーディングで、次のいずれかになります。
  - ファイルの場合はf
  - IPv4ソケットの場合は4
  - IPv6ソケットの場合は6
  - UNIXソケットの場合はu
  - シグナルFDの場合はs
  - イベントFDの場合はe
  - iotify FDの場合はi
  - タイマーFDの場合はt

この時点でsysdig出力の基本を理解できるはずですが、もちろんsysdigは多くの興味深いことを示すことができる強力なツールです。 これらの2つのブログ投稿は、詳細とコンテキストを提供します。

- Sysdig outputを解釈する
- 魅力的なLinuxシステムコールの世界

キャプチャーファイル

Sysdigでは、キャプチャーしたイベントをディスクに保存して、後で分析できるようにすることができます。 構文は次のとおりです。

```
$ sysdig –w myfile.scap
```

ファイルに保存されるイベントの数を100に制限する場合は、–nフラグを使用できます。

```
$ sysdig –n 100 –w myfile.scap
```

-Cコマンドラインフラグを使用すると、キャプチャを特定のサイズのファイルに分割できます。 たとえば、これを使用してサイズが1MBのファイルを生成します。

```
sudo sysdig -C 1 -w dump.scap
```

-Cフラグと-Wフラグを組み合わせて、保持するファイルの数をsysdigに指示できます。 たとえば、次のコマンドラインは、イベントをサイズが1MBのファイルにキャプチャし、最後の5つのファイルのみをディスクに保持します。

```
sudo sysdig -C 1 -W 5 -w dump.scap
```

継続的なキャプチャにファイルローテーションを使用する方法の詳細については、ここを参照してください。

以前に保存したキャプチャファイルの読み取りは、-rフラグを使用して実行できます。

```
$ sysdig –r myfile.scap
```

sysdigは各キャプチャファイル(実行中のプロセス、開いているファイル、ユーザー名など)にOSの完全なスナップショットを保存することに注意してください。これは、オフライン分析を行っても情報が失われないことを意味します。MACおよびWindowsバージョンのsysdigをダウンロードできることにも注意してください。ライブキャプチャはできませんが、Linuxで収集されたキャプチャファイルの分析に使用できます。

フィルタリング

基本事項をみて行きましたので、次は楽しんで行きましょう。Sysdigのフィルタリングシステムは強力で用途が広く、干し草の山から針を探すように設計されています。フィルターは、tcpdumpのようにコマンドラインの最後に指定し、ライブキャプチャまたはキャプチャファイルの両方に適用できます。 たとえば、特定のコマンド、この場合はcatのアクティビティを見てみましょう。

```
$ ./sysdig proc.name=cat
21368 13:10:15.384878134 1 cat (8298) < execve res=0 exe=cat args=index.html. tid=8298(cat) pid=8298(cat) ptid=1978(bash) cwd=/root fdlimit=1024
21371 13:10:15.384948635 1 cat (8298) > brk size=0
21372 13:10:15.384949909 1 cat (8298) < brk res=10665984
21373 13:10:15.384976208 1 cat (8298) > mmap
21374 13:10:15.384979452 1 cat (8298) < mmap
21375 13:10:15.384990980 1 cat (8298) > access
21376 13:10:15.384999211 1 cat (8298) < access
21377 13:10:15.385008602 1 cat (8298) > open
21378 13:10:15.385014374 1 cat (8298) < open fd=3(/etc/ld.so.cache) name=/etc/ld.so.cache flags=0(O_NONE) mode=0
21379 13:10:15.385015508 1 cat (8298) > fstat fd=3(/etc/ld.so.cache)
21380 13:10:15.385016588 1 cat (8298) < fstat res=0
21381 13:10:15.385017033 1 cat (8298) > mmap
21382 13:10:15.385019763 1 cat (8298) < mmap
21383 13:10:15.385020047 1 cat (8298) > close fd=3(/etc/ld.so.cache)
21384 13:10:15.385020556 1 cat (8298) < close res=0
```

ご覧のとおり、sysdigはプロセスに接続していません。すべてをキャプチャして、不要なものを除外できます。フィルターステートメントでは、比較演算子(=、！=、<、<=、>、> =、contains、icontains、in、exists)を使用できます。 ブール演算子(and、or、not)と括弧を使用して組み合わせることができます。 例えば

```
$ sysdig proc.name=cat or proc.name=vi
```

catとviの両方の活動をキャプチャします。

```
$ sysdig proc.name!=cat and evt.type=open
```

catではないプログラムによって開かれたすべてのファイルを表示します。

接続ドメイン名によるフィルタリング

一部のフィルターは、dns情報などの外部情報を使用して、イベントとフィルター式を照合します。 詳細については、[このページ]を参照してください。

サポートされているすべてのフィルター

フィルターフィールドは’class.field’として表されます。 使用可能なクラスとそれらに含まれるフィールドのリストを取得する簡単な方法は、

```
$ sysdig -l
```

参考までに、使用できるフィールドのリストを次に示します(注意:リストは新しいリリースごとに変わるため、最新のバージョンのプログラムを使用してください)。

フィールドクラス: fd

fd.num	ファイル記述子を識別する一意の番号。
fd.type FDのタイプ	'file'、'directory'、'ipv4'、'ipv6'、'unix'、'pipe'、'event'、'signalfd'、'eventpoll'、'inotify'、または'signal fd'のいずれかです。
fd.typechar	単一文字としてのFDのタイプ。ファイルの場合は'f'、IPv4ソケットの場合は4、IPv6ソケットの場合は6、UNIXソケットの場合は'u'、パイプの場合は'p'、eventfdの場合は'e'、signalfdの場合は's'、eventpollの場合は'l'、'i' inotifyの場合は'o'、不明の場合は'o'。
fd.name	FDのフルネーム。 fdがファイルの場合、このフィールドにはフルパスが含まれます。 FDがソケットの場合、このフィールドには接続タプルが含まれます。
fd.directory	fdがファイルの場合、それを含むディレクトリ。
fd.filename	fdがファイルの場合、パスなしのファイル名。
fd.ip	(フィルターのみ)は、fdのIPアドレス(クライアントまたはサーバー)と一致します。
fd.cip	クライアントのIPアドレス。
fd.sip	サーバーのIPアドレス。
fd.lip	ローカルIPアドレス。
fd.rip	リモートIPアドレス。
fd.port	(フィルターのみ)は、fdのポート(クライアントまたはサーバー)と一致します。
fd.cport	TCP/UDP FD、クライアントポート
fd.sport	TCP/UDP FD、サーバポート
fd.lport	TCP/UDP FD、ローカルポート
fd.rport	TCP/UDP FD、リモートポート
fd.l4proto	ソケットのIPプロトコル。 'tcp'、'udp'、'icmp'、'raw'のいずれかです。
fd.sockfamily	ソケットイベントのソケットファミリ。 'ip'または'unix'を指定できます。
fd.sockfamily	このFDを所有するプロセスが接続のサーバーエンドポイントである場合は’true’
fd.uid	FDの一意の識別子で、FD番号とスレッドIDを連鎖させることによって作成されます。

fd.containername	コンテナIDとFD名の連鎖。 FDがどのコンテナに属しているかを識別しようとするときに役立ちます。
fd.containerdirectory	コンテナIDとディレクトリ名の連鎖。ディレクトリがどのコンテナに属しているかを識別しようとするときに役立ちます。
fd.proto	(フィルターのみ)は、fdのプロトコル(クライアントまたはサーバー)と一致します。
fd.cproto	TCP/UDP FD、クライアントプロトコル
fd.sproto	TCP/UDP FD、サーバープロトコル
fd.lproto	TCP/UDP FD、ローカルプロトコル
fd.rproto	TCP/UDP FD、リモートプロトコル
fd.net	(フィルターのみ)は、fdのIPネットワーク(クライアントまたはサーバー)と一致します。
fd.cnet	(フィルターのみ)は、fdのクライアントIPネットワークと一致します。
fd.snet	(フィルターのみ)は、fdのサーバーIPネットワークと一致します
fd.lnet	(フィルターのみ)は、fdのローカルIPネットワークと一致します
fd.rnet	(フィルターのみ)は、fdのリモートIPネットワークと一致します
fd.connected	TCP/UDP FD、ソケットが接続されている場合は'true'
fd.name_changed	イベントがこのイベントで使用されるfdの名前を変更するときに真。これは、接続タプルが変更されるudp接続などの場合に発生する可能性があります
fd.cip.name	クライアントのIPアドレスに関連付けられたドメイン名
fd.sip.name	サーバーのIPアドレスに関連付けられたドメイン名
fd.lip.name	ローカルIPアドレスに関連付けられたドメイン名
fd.rip.name	リモートIPアドレスに関連付けられたドメイン名

フィールドクラス: process

proc.pid	イベントを生成するプロセスのID。
proc.exe	最初のコマンドライン引数(通常は実行可能ファイル名またはカスタム名)。
proc.name	イベントを生成する実行可能ファイルの名前(パスを除く)。
proc.args	イベントを生成するプロセスを開始するときにコマンドラインで渡される引数。
proc.env	イベントを生成するプロセスの環境変数。
proc.cmdline	完全なプロセスコマンドライン、つまりproc.name + proc.args。
proc.exeline	フルプロセスコマンドライン、最初の引数としてexe、つまりproc.exe + proc.args。
proc.cwd	イベントの現在の作業ディレクトリ。
proc.nthreads	メインプロセススレッドを含め、イベントを生成するプロセスが現在持っているスレッドの数です。
proc.nchilds	イベントを生成するプロセスが現在持っている子スレッドの数。これには、メインプロセススレッドは含まれません。
proc.ppid	イベントを生成するプロセスの親のpid。
proc.pname	イベントを生成するプロセスの親の名前(パスを除く)。

proc.pcmdline	イベントを生成するプロセスの親の完全なコマンドライン(proc.name + proc.args)。
proc.apid	プロセスの祖先の1つのpid。例えば。proc.apid [1]は親pidを返し、proc.apid [2]は祖父母pidを返します。proc.apid [0]は現在のプロセスのPIDです。引数なしのproc.apidはフィルターでのみ使用でき、プロセスの祖先のいずれかに一致します。例 proc.apid=1234。
proc.aname	プロセスの祖先の1つの名前(パスを除く)。例えば、proc.aname [1]は親の名前を返し、proc.aname [2]は祖父母の名前を返します。 proc.aname [0]は現在のプロセスの名前です。引数なしのproc.anameを使用できますフィルターのみで、プロセスの祖先のいずれかに一致します。 例 proc.aname=bash。
proc.loginshellid	現在のプロセスの祖先がある場合、その祖先の中で最も古いシェルのpid。このフィールドは、異なるユーザーセッションを分離するために使用でき、spy_userなどのchiselと組み合わせて使用すると便利です。
proc.duration	プロセスが開始してからのナノ秒数。
proc.fdopencount	プロセスの開いているFDの数
proc.fdlimit	プロセスが開くことができるFDの最大数。
proc.fdusage	プロセスで開いているFDと使用可能な最大FDの比率。
proc.vmsize	プロセスの合計仮想メモリ(KB単位)。
proc.vmrss	プロセス用の常駐非スワップメモリ(KBとして)。
proc.vmswap	プロセス用にスワップしたメモリ(kb)。
thread.pfmajor	スレッド開始以降のメジャーページフォールトの数。
thread.pfminor	スレッド開始以降のマイナーページフォールトの数。
thread.tid	イベントを生成するスレッドのID。
thread.ismain	イベントを生成するスレッドがプロセスのメインスレッドである場合は'true'。
thread.exectime	最後にスケジュールされたスレッドが費やしたCPU時間(ナノ秒単位)。スイッチイベントによってのみエクスポートされます。
thread.totexectime	キャプチャの開始以降の現在のスレッドの合計CPU時間(ナノ秒単位)。スイッチイベントによってのみエクスポートされます。
thread.cgroups	スレッドが属するすべてのcgroupで、単一の文字列に集約されます。
thread.cgroup	特定のサブシステムについて、スレッドが属するcgroup。例えば、thread.cgroup.cpuacct。
thread.vtid	現在のPID namespaceから見た、イベントを生成するスレッドのID。
proc.vpid	現在のPID namespaceから見た、イベントを生成するプロセスのID。
thread.cpu	最後の1秒間にスレッドによって消費されたCPU。
thread.cpu.user	最後の1秒間にスレッドによって消費されたユーザーCPU。
thread.cpu.system	最後の1秒間にスレッドによって消費されたシステムCPU。
thread.vmsize	プロセスのメインスレッドの場合、これはプロセスの合計仮想メモリ(KB)です。他のスレッドの場合、このフィールドはゼロです。
thread.vmrss	プロセスのメインスレッドの場合、これはプロセスの常駐非スワップメモリ(kb)です。他のスレッドの場合、このフィールドはゼロです。
proc.sidイ	ベントを生成するプロセスのセッションID。
proc.sname	現在のプロセスのセッションリーダーの名前。これは、pid = proc.sidを持つプロセスか、または、現在のプロセスと同じsid。
proc.tty	プロセスの制御端末。端末のないプロセスの場合は0。

proc.exepath	プロセスの完全な実行可能パス。
proc.vpgid	現在のPID namespaceから見た、イベントを生成しているプロセスのプロセスグループID。

フィールドクラス: evt

evt.num	イベント番号。
evt.time	ナノ秒部分を含む時間文字列としてのイベントタイムスタンプ。
evt.time.s	ナノ秒のない時間文字列としてのイベントタイムスタンプ。
evt.datetime	日付を含む時間文字列としてのイベントタイムスタンプ。
evt.rawtime	絶対イベントタイムスタンプ、つまりエポックからのナノ秒。
evt.rawtime.s	イベントのタイムスタンプの整数部分(たとえば、エポックからの秒数)。
evt.rawtime.ns	絶対イベントタイムスタンプの小数部分。
evt.reltime	キャプチャの開始からのナノ秒数。
evt.reltime.s	キャプチャの開始からの秒数。
evt.reltime.ns	キャプチャの開始からの時間の小数部(ns)。
evt.latency	終了イベントと対応する開始イベントの間のデルタ(ナノ秒単位)。
evt.latency.s	イベントレイテンシデルタの整数部分。
evt.latency.ns	イベントレイテンシデルタの小数部分。
evt.latency.human	人間が読める文字列としての、exitイベントと対応するenterイベントの間のデルタ(例:10.3ms)。
evt.deltatime	このイベントと前のイベントの間のデルタ(ナノ秒単位)。
evt.deltatime.s	このイベントと前のイベントの間のデルタの整数部分。
evt.deltatime.ns	このイベントと前のイベントの間のデルタの小数部分。
evt.outputtime	これは-t paramに依存します。デフォルトは%evt.time('h')です。
evt.dir	イベントの方向は、Enterイベントの場合は'>'、Exitイベントの場合は'<'のいずれかです。
evt.type	イベントの名前('open'など)。
evt.type.is	イベントタイプを指定でき、そのタイプのイベントに対して1を返します。たとえば、evt.type.is.openは、オープンイベントの場合は1、その他のイベントの場合は0を返します。
syscall.type	システムコールイベントの場合、システムコールの名前('open'など)。他のイベント(スイッチまたはsysdig内部イベントなど)の場合は設定解除します。filtered/printed値が実際にシステムコールであることを確認する必要がある場合は、evt.typeの代わりにこのフィールドを使用します。
evt.category	イベントのカテゴリ。値の例は、'file'(オープンやクローズなどのファイル操作の場合)、'net'(ソケットやバインドなどのネットワーク操作の場合)、メモリー(brkやmmapなどの場合)などです。
evt.cpu	このイベントが発生したCPUの番号。
evt.args	単一の文字列に集約されたすべてのイベント引数です。
evt.arg	名前または番号で指定されたイベント引数の1つ。一部のイベント(戻りコードやFDなど)は、可能な場合、テキスト表現に変換されます。例えば。 'evt.arg.fd'または 'evt.arg [0]'。
evt.rawarg	名前で指定されたイベント引数の1つ。例えば。 'evt.rawarg.fd'。

evt.info	ほとんどのイベントでこのフィールドはevt.argsと同じ値を返します。ただし、一部のイベント(/dev/logへの書き込みなど)では、引数のデコードから得られるより高レベルの情報が提供されます。
evt.buffer	read()、recvfrom()などの1つを持つイベントのバイナリデータバッファー。このフィールドを 'contains'を含むフィルターで使用して、I/Oデータバッファーを検索します。
evt.buflen	read()、recvfrom()などのイベントがある場合のバイナリデータバッファーの長さ
evt.res	文字列としてのイベントの戻り値。イベントが失敗した場合、結果はエラーコード文字列('ENOENT'など)です。それ以外の場合、結果は文字列 'SUCCESS'です。
evt.rawres	イベントの戻り値(数値-2など)。範囲比較に役立ちます。
evt.failed	エラーステータスを返したイベントの場合は’true’。
evt.is_io	read()、send、recvfrom()など、FDに対して読み取りまたは書き込みを行うイベントの場合は’true’。
evt.is_io_read	FDから読み取るイベント(read()、recv()、recv from()など)の場合は'true'
evt.is_io_write	FDに書き込むイベントの場合、(write()、send()など)の場合は'true'
evt.io_dir	'r'は、read()のように、FDから読み取るイベントに使用します。 write()のように、FDに書き込むイベントの場合は'w'
evt.is_wait	スレッドを待機させるイベントの場合は'true'。 sleep()、select()、poll()。
evt.wait_latency	スレッドを待機させるイベント(sleep()、select()、poll()など)の場合、これは、イベントが返されるのを待機するのに費やされた時間(ナノ秒単位)です。
evt.is_syslog	/dev/logへの書き込みであるイベントの場合は’true’。
evt.count	このフィルターフィールドは常に1を返し、chisel内部からのイベントをカウントするために使用できます。
evt.count.error	このフィルターフィールドは、エラーで返されたイベントに対して1を返し、chisel内からのイベントの失敗をカウントするために使用できます。
evt.count.error.file	このフィルターフィールドは、エラーで返され、ファイルI/Oに関連するイベントに対して1を返し、chisel内からのイベントの失敗をカウントするために使用できます。
evt.count.error.net	このフィルターフィールドは、エラーで返され、ネットワークI/Oに関連するイベントに対して1を返し、chisel内からのイベントの失敗をカウントするために使用できます。
evt.count.error.memory	このフィルターフィールドは、エラーで返され、メモリ割り当てに関連するイベントに対して1を返し、chisel内からのイベントの失敗をカウントするために使用できます。
evt.count.error.other	このフィルターフィールドは、エラーで返され、前のカテゴリのいずれにも関連しないイベントに対して1を返し、chiselの内部からのイベントの失敗をカウントするために使用できます。
evt.count.exitこ	のフィルターフィールドは、終了イベントに対して1を返し、chisel内部からの単一のイベントをカウントするために使用できます。
evt.around	(フィルターのみ)指定された時間間隔でイベントを受け入れます。構文はevt.around[T]=Dです。Tは%evt.rawtimeによってイベントに対して返された値で、Dはミリ秒単位のデルタです。たとえば、evt.around

	[1404996934793590564]=1000は、タイムスタンプの1秒前と1秒後の合計2秒のキャプチャで、タイムスタンプ付きのイベントを返します。
evt.abspath	renameatやsymlinkatのようなシステムコール中にdirfdとnameから計算された絶対パス。複数のパスをサポートするシステムコールには、'evt.abspath.src'または'evt.abspath.dst'を使用します。
evt.is_open_read	パスが読み取り用に開かれたopen/openatイベントの場合は'true'
evt.is_open_write	パスが書き込み用に開かれたopen/openatイベントの場合は'true'

フィールドクラス: user

user.uid	ユーザーID
user.name	ユーザー名
user.homedir	ユーザーのホームディレクトリ
user.shell	ユーザーのシェル
user.loginuid	auditユーザーID(auid)
user.loginname	auditユーザー名(auid)

フィールドクラス: group

group.gid	グループID
group.name	グループ名

フィールドクラス:syslog

syslog.facility.str	文字列としての機能
syslog.facility	数値としての機能(0〜23)
syslog.severity.str	文字列としての重大度。 次のいずれかの値をとることができます:emerg、alert、crit、err、warn、notice、info、debug
syslog.severity	数値としての重大度(0から7)
syslog.message	syslogに送信されたメッセージ

フィールドクラス: container

container.id	コンテナID。
container.name	コンテナ名。
container.image	コンテナイメージの名前(例:Docdigの場合はsysdig/sysdig:latest)。
container.image.id	コンテナイメージID(6f7e2741b66bなど)。
container.type	コンテナのタイプ。例:dockerまたはrkt
container.privileged	特権として実行されているコンテナの場合はtrue、それ以外の場合はfalse
container.mounts	スペースで区切られたマウント情報のリスト。リストの各項目の形式は、<source>:<dest>:<mode>:<rdrw>:<propagation>です。

container.mount	単一のマウントに関する情報。番号(container.mount [0]など)またはマウントソース(container.mount [/usr/local])で指定します。パス名はglob(container.mount [/usr/local/*])にすることができます。その場合、最初に一致したマウントが返されます。情報の形式は、<source>:<dest>:<mode>:<rdrw>:<propagation>です。指定されたインデックスを持つマウントまたは提供されたソースと一致するマウントがない場合、NULL値の代わりに文字列'none'を返します。
container.mount.source	番号で指定されたマウントソース(例:container.mount.source [0])またはマウント先(container.mount.source [/host/lib/modules])。パス名はglobにすることができます。
container.mount.dest	番号で指定されたマウント先(例:container.mount.dest [0])またはマウントソース(container.mount.dest [/lib/modules])。パス名はglobにすることができます。
container.mount.mode	マウントモード。番号で指定します(例:container.mount.mode [0])またはマウントソース(container.mount.mode [/usr/local])。パス名はglobにすることができます。
container.mount.rdwr	マウントrdwr値。番号で指定します(例:container.mount.rdwr [0])またはマウントソース(container.mount.rdwr [/usr/local])。パス名はglobにすることができます。
container.mount.propagation	番号(例:container.mount.propagation [0])またはマウントソース(container.mount.propagation [/usr/local])で指定されたマウント伝播値パス名はglobにすることができます。
container.image.repository	コンテナイメジリポジトリ(sysdig/sysdigなど)。
container.image.tag	コンテナイメージタグ(例:stable、latest)。
container.image.digest	コンテナイメージレジストリダイジェスト(例:sha256:d977378f890d445c15e51795296e4e5062f109ce6da83e0a355fc4ad8699d27)。

フィールドクラス: fdlist

fdlist.nums	ポーリングイベントの場合、これは'fds'引数のFD番号のコンマ区切りのリストであり、文字列として返されます。
fdlist.names	ポーリングイベントの場合、これは 'fds'引数内のFD名のコンマ区切りのリストであり、文字列として返されます。
fdlist.cips	ポーリングイベントの場合、これは'fds'引数内のクライアントIPアドレスのカンマ区切りのリストであり、文字列として返されます。
fdlist.sips	ポーリングイベントの場合、これは'fds'引数内のサーバーIPアドレスのカンマ区切りのリストであり、文字列として返されます。
fdlist.cports	TCP/UDP FDの場合、ポーリングイベントの場合、これは'fds'引数のクライアントTCP/UDPポートのカンマ区切りのリストであり、文字列として返されます。
fdlist.sports	ポーリングイベントの場合、これは'fds'引数内のサーバーTCP / UDPポートのカンマ区切りのリストであり、文字列として返されます。

フィールドクラス: k8s

k8s.pod.name	Kubernetesポッド名。
k8s.pod.id	KubernetesポッドID。
k8s.pod.label	Kubernetesポッドラベル。例えば、 'k8s.pod.label.foo'。
k8s.pod.labels	Kubernetesポッドのコンマ区切りのキー/値ラベル。例えば、 'foo1：bar1、foo2：bar2'。
k8s.rc.name	Kubernetesレプリケーションコントローラ名。
k8s.rc.id	KubernetesレプリケーションコントローラID。
k8s.rc.label	Kubernetesレプリケーションコントローラのラベル。例えば、 'k8s.rc.label.foo'。
k8s.rc.labels	Kubernetesレプリケーションコントローラのカンマ区切りのキー/値ラベル。例えば、 'foo1：bar1、foo2：bar2'。
k8s.svc.name	Kubernetesサービス名（連結された複数の値を返すことができます）。
k8s.svc.id	KubernetesサービスID（連結された複数の値を返すことができます）。
k8s.svc.label	Kubernetesサービスラベル。例えば、 'k8s.svc.label.foo'（複数の値を連結して返すことができます）。
k8s.svc.labels	Kubernetesサービスのコンマ区切りのキー/値ラベル。例えば、 'foo1：bar1、foo2：bar2'。
k8s.ns.name	Kubernetesネームスペース名。
k8s.ns.id	KubernetesネームスペースID。
k8s.ns.label	Kubernetesネームスペースラベル。例えば、 'k8s.ns.label.foo'。
k8s.ns.labels	Kubernetesネームスペースのコンマ区切りのキー/値ラベル。例えば、 'foo1：bar1、foo2：bar2'。
k8s.rs.name	Kubernetesレプリカセット名。
k8s.rs.id	KubernetesレプリカセットID。
k8s.rs.label	Kubernetesレプリカセットラベル。例えば、 'k8s.rs.label.foo'。
k8s.rs.labels	Kubernetesレプリカセットのコンマ区切りのキー/値ラベル。例えば、 'foo1：bar1、foo2：bar2'。
k8s.deployment.name	Kubernetesデプロイメント名。
k8s.deployment.id	KubernetesデプロイメントID。
k8s.deployment.label	Kubernetesデプロイメントラベル、例えば。 'k8s.rs.label.foo'。
k8s.deployment.labels	Kubernetesデプロイメントのカンマ区切りのキー/値ラベル。例えば、 'foo1：bar1、foo2：bar2'。

フィールドクラス: mesos

mesos.task.name	Mesosタスク名。
mesos.task.id	MesosタスクID。
mesos.task.label	Mesosタスクラベル。 例えば。 'mesos.task.label.foo'。
mesos.task.labels	Mesosタスクのコンマ区切りのキー/値ラベル。 例えば。 'foo1：bar1、foo2：bar2'。
mesos.framework.name	Mesosフレームワーク名。
mesos.framework.id	MesosフレームワークID。

marathon.app.name	マラソンアプリ名。
marathon.app.id	マラソンアプリID。
marathon.app.label	マラソンアプリのラベル。 例えば、 'marathon.app.label.foo'。
marathon.app.labels	マラソンアプリのコンマ区切りのキー/値ラベル。 例えば、 'foo1：bar1、foo2：bar2'。
marathon.group.name	マラソングループ名。
marathon.group.id	マラソングループID。

フィールドクラス: span

span.id	スパンのID。これは、このスパンの開始および終了トレーサーイベントを照合するために使用される一意の識別子です。また、トレースに属する異なるスパンを一致させるためにも使用できます。
span.time	ナノ秒の部分を含む人間が読み取れる文字列としてのスパンのトレーサの開始時間。
span.ntags	このスパンが持つタグの数。
span.nargs	このスパンが持つ引数の数。
span.tags	スパンのすべてのタグのドット区切りリスト。
span.tag	スパンのタグの1つ。0ベースのオフセットで指定されます。'span.tag [1]'。負のオフセットを使用して、タグリストの最後から要素を選択できます。たとえば、'span.tag [-1]'は最後のタグを返します。
span.args	スパンの引数のコンマ区切りリスト。
span.arg	スパン引数の1つ。名前または0ベースのオフセットで指定します。例えば。'span.arg.xxx'または'span.arg [1]'。負のオフセットを使用して、タグリストの最後から要素を選択できます。たとえば、'span.arg [-1]'は最後の引数を返します。
span.enterargs	スパンの開始トレーサイベント引数のカンマ区切りリスト。 開始トレーサーの場合、これはevt.argsと同じです。終了トレーサーの場合、これは対応する開始トレーサーのevt.argsです。
span.duration	このスパンの終了トレーサーイベントと開始トレーサーイベントの間のデルタ。
span.duration.human	人間が読める文字列としての、このスパンの終了トレーサーイベントと開始イベントの間のデルタ（例：10.3ms）。

フィールドクラス: evtin

evtin.span.id	指定されたIDを持つスパンの開始トレーサーと終了トレーサーの間にあり、トレーサーを生成したのと同じスレッドによって生成されたすべてのイベントを受け入れます。

evtin.span.ntags　指定された数のタグを持つスパンの開始トレーサーと終了トレーサーの間にあり、トレーサーを生成したのと同じスレッドによって生成されるすべてのイベントを受け入れます。

evtin.span.nargs　指定された数の引数を持つスパンの開始トレーサーと終了トレーサーの間にあり、トレーサーを生成したのと同じスレッドによって生成されるすべてのイベントを受け入れます。

evtin.span.tagsは、指定されたタグを持つスパンの開始トレーサーと終了トレーサーの間にあり、トレーサーを生成したのと同じスレッドによって生成されるすべてのイベントを受け入れます。

evtin.span.tag　指定されたタグを持つスパンの開始トレーサーと終了トレーサーの間にあり、トレーサーを生成した同じスレッドによって生成されたすべてのイベントを受け入れます。このフィールドで受け入れられる構文については、span.tagの説明を参照してください。

evtin.span.args　指定された引数を持つスパンの開始トレーサーと終了トレーサーの間にあり、トレーサーを生成したのと同じスレッドによって生成されるすべてのイベントを受け入れます。

evtin.span.arg　指定された引数を持つスパンの開始トレーサーと終了トレーサーの間にあり、トレーサーを生成したのと同じスレッドによって生成されたすべてのイベントを受け入れます。このfが受け入れる構文については、span.argの説明を参照してください。

evtin.span.p.id　evtin.span.idと同じですが、スパンを生成した同じプロセス内の他のスレッドによって生成されたイベントも受け入れます。

evtin.span.p.ntags　evtin.span.ntagsと同じですが、スパンを生成した同じプロセス内の他のスレッドによって生成されたイベントも受け入れます。

evtin.span.p.nargs　evtin.span.nargsと同じですが、スパンを生成した同じプロセスの他のスレッドによって生成されたイベントも受け入れます。

evtin.span.p.tags　evtin.span.tagsと同じですが、スパンを生成した同じプロセス内の他のスレッドによって生成されたイベントも受け入れます。

evtin.span.p.tag　evtin.span.tagと同じですが、スパンを生成した同じプロセス内の他のスレッドによって生成されたイベントも受け入れます。

evtin.span.p.args　evtin.span.argsと同じですが、スパンを生成した同じプロセス内の他のスレッドによって生成されたイベントも受け入れます。

evtin.span.p.arg　evtin.span.argと同じですが、スパンを生成した同じプロセスの他のスレッドによって生成されたイベントも受け入れます。

evtin.span.s.id　evtin.span.idと同じですが、スパンを生成したスクリプトによって生成されたイベント、つまり、親PIDがスパンを生成しているプロセスと同じであるプロセスによって生成されたイベントも受け入れます。

evtin.span.s.ntags　evtin.span.idと同じですが、スパンを生成したスクリプトによって生成されたイベント、つまり、親PIDがスパンを生成しているプロセスと同じであるプロセスによって生成されたイベントも受け入れます。

evtin.span.s.nargs　evtin.span.idと同じですが、スパンを生成したスクリプトによって生成されたイベント、つまり、親PIDがスパンを生成しているプロセスと同じであるプロセスによって生成されたイベントも受け入れます。

evtin.span.s.tags	evtin.span.idと同じですが、スパンを生成したスクリプトによって生成されたイベント、つまり、親PIDがスパンを生成しているプロセスと同じであるプロセスによって生成されたイベントも受け入れます。
evtin.span.s.tag	evtin.span.idと同じですが、スパンを生成したスクリプトによって生成されたイベント、つまり、親PIDがスパンを生成しているプロセスと同じであるプロセスによって生成されたイベントも受け入れます。
evtin.span.s.args	evtin.span.idと同じですが、スパンを生成したスクリプトによって生成されたイベント、つまり、親PIDがスパンを生成しているプロセスと同じであるプロセスによって生成されたイベントも受け入れます。
evtin.span.s.arg	evtin.span.idと同じですが、スパンを生成したスクリプトによって生成されたイベント、つまり、親PIDがスパンを生成しているプロセスと同じであるプロセスによって生成されたイベントも受け入れます。
evtin.span.m.id	evtin.span.idと同じですが、他のスレッドや他のプロセスを含め、スパン中にマシンで生成されたすべてのイベントを受け入れます。
evtin.span.m.ntags	evtin.span.idと同じですが、スパン中にマシン上で生成された、他のスレッドや他のプロセスを含むすべてのイベントを受け入れます。
evtin.span.m.nargs	evtin.span.idと同じですが、スパン中にマシン上で生成された、他のスレッドや他のプロセスを含むすべてのイベントを受け入れます。
evtin.span.m.tags	evtin.span.idと同じですが、スパン中にマシン上で生成された、他のスレッドや他のプロセスを含むすべてのイベントを受け入れます。
evtin.span.m.tag	evtin.span.idと同じですが、スパン中にマシン上で生成された、他のスレッドや他のプロセスを含むすべてのイベントを受け入れます。
evtin.span.m.args	evtin.span.idと同じですが、スパン中にマシン上で生成された、他のスレッドや他のプロセスを含むすべてのイベントを受け入れます。
evtin.span.m.arg	evtin.span.idと同じですが、スパン中にマシン上で生成された、他のスレッドや他のプロセスを含むすべてのイベントを受け入れます。

ご覧のとおり、多くのクリエイティブな深堀において十分な仕組みです。たとえば、sysdigがファイル記述子を解決するという仕組みを活用して、次のようなことができます。

```
$ sysdig evt.type=accept and proc.name!=apache
```

Apache以外のプロセスが受信した受信ネットワーク接続を確認します。
しかし、それだけではありません。
追加の説明に値するいくつかの特別なフィールド、evt.argとevt.rawargがあります。sysdigがキャプチャーするすべてのイベントには、タイプ（例: 'open'、 'read' ...）と、標準化タイプシステムを使用してエンコードされたパラメーターのセット（例: 'fd'、 'name' ...）があります。 これは退屈に聞こえると思いますので、その利点についてお話ししましょう。どのイベントのどのパラメーターもフィルターで使用できます。 たとえば、次のコマンドラインは、対話型ユーザーによって実行されるプログラムを示しています。

```
$ sysdig evt.type=execve and evt.arg.ptid=bash
```

フィルターは、プログラムの実行に使用されるexecveシステムコールを受け入れますが、親プロセス名が'bash'の場合のみです。evt.argとevent.rawargの違いは、2番目はPID、FD、エラーコードなどの解決を行わず、引数をそのままの数値形式のままにすることです。 たとえば、

```
$ sysdig evt.arg.res=ENOENT
```

特定のI/Oエラーコードでフィルタリングするか、エラーコードがネガティブであるため、これは

```
$ sysdig " evt.rawarg.res<0 or evt.rawarg.fd<0"
```

エラーが発生したすべてのシステムコールが表示されます。フィルターで使用できるすべてのイベントとそのパラメーターのリストを取得するには、次のように入力します。

```
$ sysdig -L
```

そして、参考までに、ここにリストがあります。ここで、'>'は開始イベントを示し、'<'は終了イベントを示します(注意：リストは新しいリリースごとに変更されるため、必ず最新のプログラムを使用してください):

```
> 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)
> send(FD fd, UINT32 size)
< send(ERRNO res, BYTEBUF data)
> sendto(FD fd, UINT32 size, SOCKTUPLE tuple)
< sendto(ERRNO res, BYTEBUF data)
> recv(FD fd, UINT32 size)
< recv(ERRNO res, BYTEBUF data)
> recvfrom(FD fd, UINT32 size)
< recvfrom(ERRNO res, BYTEBUF data, SOCKTUPLE tuple)
> shutdown(FD fd, FLAGS8 how)
< shutdown(ERRNO res)
> getsockname()
< getsockname()
> getpeername()
< getpeername()
> socketpair(FLAGS32 domain, UINT32 type, UINT32 proto)
< socketpair(ERRNO res, FD fd1, FD fd2, UINT64 source, UINT64 peer)
> setsockopt()
< setsockopt()
> getsockopt()
< getsockopt()
```

```
> sendmsg(FD fd, UINT32 size, SOCKTUPLE tuple)
< sendmsg(ERRNO res, BYTEBUF data)
> sendmmsg()
< sendmmsg()
> recvmsg(FD fd)
< recvmsg(ERRNO res, UINT32 size, BYTEBUF data, SOCKTUPLE tuple)
> recvmmsg()
< recvmmsg()
> creat()
< creat(FD fd, FSPATH name, UINT32 mode)
> pipe()
< pipe(ERRNO res, FD fd1, FD fd2, UINT64 ino)
> eventfd(UINT64 initval, FLAGS32 flags)
< eventfd(FD res)
> futex(UINT64 addr, FLAGS16 op, UINT64 val)
< futex(ERRNO res)
> stat()
< stat(ERRNO res, FSPATH path)
> lstat()
< lstat(ERRNO res, FSPATH path)
> fstat(FD fd)
< fstat(ERRNO res)
> stat64()
< stat64(ERRNO res, FSPATH path)
> lstat64()
< lstat64(ERRNO res, FSPATH path)
> fstat64(FD fd)
< fstat64(ERRNO res)
> epoll_wait(ERRNO maxevents)
< epoll_wait(ERRNO res)
> poll(FDLIST fds, INT64 timeout)
< poll(ERRNO res, FDLIST fds)
> select()
< select(ERRNO res)
> select()
< select(ERRNO res)
> lseek(FD fd, UINT64 offset, FLAGS8 whence)
< lseek(ERRNO res)
> llseek(FD fd, UINT64 offset, FLAGS8 whence)
< llseek(ERRNO res)
> getcwd()
< getcwd(ERRNO res, CHARBUF path)
> chdir()
< chdir(ERRNO res, CHARBUF path)
> fchdir(FD fd)
< fchdir(ERRNO res)
> mkdir(FSPATH path, UINT32 mode)
< mkdir(ERRNO res)
> rmdir(FSPATH path)
< rmdir(ERRNO res)
> openat(FD dirfd, CHARBUF name, FLAGS32 flags, UINT32 mode)
< openat(FD fd)
> link(FSPATH oldpath, FSPATH newpath)
< link(ERRNO res)
> linkat(FD olddir, CHARBUF oldpath, FD newdir, CHARBUF newpath)
< linkat(ERRNO res)
> unlink(FSPATH path)
```

```
< unlink(ERRNO res)
> unlinkat(FD dirfd, CHARBUF name)
< unlinkat(ERRNO res)
> pread(FD fd, UINT32 size, UINT64 pos)
< pread(ERRNO res, BYTEBUF data)
> pwrite(FD fd, UINT32 size, UINT64 pos)
< pwrite(ERRNO res, BYTEBUF data)
> readv(FD fd)
< readv(ERRNO res, UINT32 size, BYTEBUF data)
> writev(FD fd, UINT32 size)
< writev(ERRNO res, BYTEBUF data)
> preadv(FD fd, UINT64 pos)
< preadv(ERRNO res, UINT32 size, BYTEBUF data)
> pwritev(FD fd, UINT32 size, UINT64 pos)
< pwritev(ERRNO res, BYTEBUF data)
> dup(FD fd)
< dup(FD res)
> signalfd(FD fd, UINT32 mask, FLAGS8 flags)
< signalfd(FD res)
> kill(PID pid, SIGTYPE sig)
< kill(ERRNO res)
> tkill(PID tid, SIGTYPE sig)
< tkill(ERRNO res)
> tgkill(PID pid, PID tid, SIGTYPE sig)
< tgkill(ERRNO res)
> nanosleep(RELTIME interval)
< nanosleep(ERRNO res)
> timerfd_create(UINT8 clockid, FLAGS8 flags)
< timerfd_create(FD res)
> inotify_init(FLAGS8 flags)
< inotify_init(FD res)
> getrlimit(FLAGS8 resource)
< getrlimit(ERRNO res, INT64 cur, INT64 max)
> setrlimit(FLAGS8 resource)
< setrlimit(ERRNO res, INT64 cur, INT64 max)
> prlimit(PID pid, FLAGS8 resource)
< prlimit(ERRNO res, INT64 newcur, INT64 newmax, INT64 oldcur, INT64 oldmax)
> fcntl(FD fd, FLAGS8 cmd)
< fcntl(FD res)
> switch(PID next, UINT64 pgft_maj, UINT64 pgft_min, UINT32 vm_size, UINT32 vm_rss, UINT32
vm_swap)
> brk(UINT64 addr)
< brk(UINT64 res, UINT32 vm_size, UINT32 vm_rss, UINT32 vm_swap)
> mmap(UINT64 addr, UINT64 length, FLAGS32 prot, FLAGS32 flags, FD fd, UINT64 offset)
< mmap(UINT64 res, UINT32 vm_size, UINT32 vm_rss, UINT32 vm_swap)
> mmap2(UINT64 addr, UINT64 length, FLAGS32 prot, FLAGS32 flags, FD fd, UINT64 pgoffset)
< mmap2(UINT64 res, UINT32 vm_size, UINT32 vm_rss, UINT32 vm_swap)
> munmap(UINT64 addr, UINT64 length)
< munmap(ERRNO res, UINT32 vm_size, UINT32 vm_rss, UINT32 vm_swap)
> splice(FD fd_in, FD fd_out, UINT64 size, FLAGS32 flags)
< splice(ERRNO res)
> ptrace(FLAGS16 request, PID pid)
< ptrace(ERRNO res, DYNAMIC addr, DYNAMIC data)
> ioctl(FD fd, UINT64 request, UINT64 argument)
< ioctl(ERRNO res)
> rename()
< rename(ERRNO res, FSPATH oldpath, FSPATH newpath)
```

```
> renameat()
< renameat(ERRNO res, FD olddirfd, CHARBUF oldpath, FD newdirfd, CHARBUF newpath)
> symlink()
< symlink(ERRNO res, CHARBUF target, FSPATH linkpath)
> symlinkat()
< symlinkat(ERRNO res, CHARBUF target, FD linkdirfd, CHARBUF linkpath)
> procexit(ERRNO status)
> sendfile(FD out_fd, FD in_fd, UINT64 offset, UINT64 size)
< sendfile(ERRNO res, UINT64 offset)
> quotactl(FLAGS16 cmd, FLAGS8 type, UINT32 id, FLAGS8 quota_fmt)
> setresuid(UID ruid, UID euid, UID suid)
< setresuid(ERRNO res)
> setresgid(GID rgid, GID egid, GID sgid)
< setresgid(ERRNO res)
> setuid(UID uid)
< setuid(ERRNO res)
> setgid(GID gid)
< setgid(ERRNO res)
> getuid()
< getuid(UID uid)
> geteuid()
< geteuid(UID euid)
> getgid()
< getgid(GID gid)
> getegid()
< getegid(GID egid)
> getresuid()
< getresuid(ERRNO res, UID ruid, UID euid, UID suid)
> getresgid()
< getresgid(ERRNO res, GID rgid, GID egid, GID sgid)
> clone()
< clone(PID res, CHARBUF exe, BYTEBUF args, PID tid, PID pid, PID ptid, CHARBUF cwd, INT64 fdlimit,
UINT64 pgft_maj, UINT64 pgft_min, UINT32 vm_size, UINT32 vm_rss, UINT32 vm_swap, CHARBUF
comm, BYTEBUF cgroups, FLAGS32 flags, UINT32 uid, UINT32 gid, PID vtid, PID vpid)
> fork()
< fork(PID res, CHARBUF exe, BYTEBUF args, PID tid, PID pid, PID ptid, CHARBUF cwd, INT64 fdlimit,
UINT64 pgft_maj, UINT64 pgft_min, UINT32 vm_size, UINT32 vm_rss, UINT32 vm_swap, CHARBUF
comm, BYTEBUF cgroups, FLAGS32 flags, UINT32 uid, UINT32 gid, PID vtid, PID vpid)
> vfork()
< vfork(PID res, CHARBUF exe, BYTEBUF args, PID tid, PID pid, PID ptid, CHARBUF cwd, INT64 fdlimit,
UINT64 pgft_maj, UINT64 pgft_min, UINT32 vm_size, UINT32 vm_rss, UINT32 vm_swap, CHARBUF
comm, BYTEBUF cgroups, FLAGS32 flags, UINT32 uid, UINT32 gid, PID vtid, PID vpid)
> execve()
< execve(ERRNO res, CHARBUF exe, BYTEBUF args, PID tid, PID pid, PID ptid, CHARBUF cwd,
UINT64 fdlimit, UINT64 pgft_maj, UINT64 pgft_min, UINT32 vm_size, UINT32 vm_rss, UINT32 vm_swap,
CHARBUF comm, BYTEBUF cgroups, BYTEBUF env)
> signaldeliver(PID spid, PID dpid, SIGTYPE sig)
> getdents(FD fd)
< getdents(ERRNO res)
> getdents64(FD fd)
< getdents64(ERRNO res)
> setns(FD fd, FLAGS32 nstype)
< setns(ERRNO res)
> flock(FD fd, FLAGS32 operation)
< flock(ERRNO res)
> cpu_hotplug(UINT32 cpu, UINT32 action)
> accept()
```

```
< accept(FD fd, SOCKTUPLE tuple, UINT8 queuepct, UINT32 queuelen, UINT32 queuemax)
> accept(INT32 flags)
< accept(FD fd, SOCKTUPLE tuple, UINT8 queuepct, UINT32 queuelen, UINT32 queuemax)
> semop(INT32 semid)
< semop(ERRNO res, UINT32 nsops, UINT16 sem_num_0, INT16 sem_op_0, FLAGS16 sem_flg_0,
UINT16 sem_num_1, INT16 sem_op_1, FLAGS16 sem_flg_1)
> semctl(INT32 semid, INT32 semnum, FLAGS16 cmd, INT32 val)
< semctl(ERRNO res)
> ppoll(FDLIST fds, RELTIME timeout, SIGSET sigmask)
< ppoll(ERRNO res, FDLIST fds)
> mount(FLAGS32 flags)
< mount(ERRNO res, CHARBUF dev, FSPATH dir, CHARBUF type)
> umount(FLAGS32 flags)
< umount(ERRNO res, FSPATH name)
> semget(INT32 key, INT32 nsems, FLAGS32 semflg)
< semget(ERRNO res)
> access(FLAGS32 mode)
< access(ERRNO res, FSPATH name)
> chroot()
< chroot(ERRNO res, FSPATH path)
> tracer(INT64 id, CHARBUFARRAY tags, CHARBUF_PAIR_ARRAY args)
< tracer(INT64 id, CHARBUFARRAY tags, CHARBUF_PAIR_ARRAY args)
> setsid()
< setsid(PID res)
> mkdir(UINT32 mode)
< mkdir(ERRNO res, FSPATH path)
> rmdir()
< rmdir(ERRNO res, FSPATH path)
> notification(CHARBUF id, CHARBUF desc)
> execve()
< execve(ERRNO res, CHARBUF exe, BYTEBUF args, PID tid, PID pid, PID ptid, CHARBUF cwd,
UINT64 fdlimit, UINT64 pgft_maj, UINT64 pgft_min, UINT32 vm_size, UINT32 vm_rss, UINT32 vm_swap,
CHARBUF comm, BYTEBUF cgroups, BYTEBUF env, INT32 tty)
> unshare(FLAGS32 flags)
< unshare(ERRNO res)
```

出力フォーマット

フィルタリングとフィルターフィールドを試すのに少し時間がかかりましたか？ これで、同じフィールドを使用して、sysdigが画面に出力する内容をカスタマイズする方法を学習します。sysdigがフィールドをエンコードするために使用する型システムのもう1つの本当に良い利点は、それらすべてを使用してプログラム出力をカスタマイズできることです。出力のカスタマイズは、–pコマンドラインフラグで行われ、Cのprintf構文と多少似ています。 次に例を示します。

```
$ sysdig -p"user:%user.name dir:%evt.arg.path" evt.type=chdir
user:ubuntu dir:/root
user:ubuntu dir:/root/tmp
user:ubuntu dir:/root/Download
```

このワンライナーは、chdirシステムコール（ユーザーがcdを実行するたびに呼び出されるコール）をフィルタリングし、ユーザー名とユーザーが移動するディレクトリを出力します。基本的に、ファイルシステム内を移動するユーザーを追跡できます。

–pフォーマット構文に関する注意事項：

- フィールドの前に%を付ける必要があります
- Cのprintfとまったく同じように、文字列に任意のテキストを追加できます。
- デフォルトでは、行に出力されるのは、-pで指定されたすべてのフィールドがイベントに存在する場合のみです。 ただし、文字列の前に*を付けて、何が印刷されるようにすることもできます。 その場合、欠落しているフィールドは<NA>としてレンダリングされます

例えば、

```
$ sysdig -p"%evt.type %evt.dir %evt.arg.name" evt.type=open
```

このように、exit openイベントのみを出力します

```
open < /proc/1285/task/1399/stat
open < /proc/1285/task/1400/io
open < /proc/1285/task/1400/statm
```

Enterイベントには名前引数が含まれていないため、

```
open > <NA>
open < /proc/1285/task/1399/stat
open > <NA>
open < /proc/1285/task/1400/io
open > <NA>
open < /proc/1285/task/1400/statm
open > <NA>
```

フィルタリングと出力フォーマットを組み合わせると、sysdigは非常に柔軟で強力なツールになります。 ここではいくつかの例を示します。

```
$ sysdig -A -s 65000 -p"%evt.buffer" "proc.name=cat and evt.type=write and fd.num=1"
```

プロセス（この場合はcat）の標準出力を出力します。 -Aスイッチを使用して結果を人間が読める文字列としてレンダリングする方法と、-sスイッチを使用して通常の80バイトを超える各書き込みをキャプチャする方法に注意してください。注意: -sを使用すると、巨大なキャプチャファイルが生成される可能性があります。

```
$ sysdig -p"%user.name) %proc.name %proc.args" evt.type=execve and evt.arg.ptid=bash
```

実際のユーザー（つまりbashから）によって起動されたすべてのプログラムのユーザー、コマンド名、および引数を表示します。

```
$ sysdig -p"%user.name) %evt.arg.path" "evt.type=chdir"
```

インタラクティブなユーザーがアクセスするディレクトリを示します。

```
$ sysdig -p"user:%user.name process:%proc.name file:%fd.name" "evt.type=write and fd.name contains /etc/"
```

/etcディレクトリへのすべてのアクセスのユーザー、プロセス、およびファイル名を出力します。

```
$ sysdig -p"%fd.name" "proc.name=apache and evt.type=accept"
```

Apacheが受信したすべての接続のTCP/IPエンドポイント情報をリストします。私は何度も続けることができますが、まあ、あなたはアイデアを得ます。:-)より有用なsysdigワンライナーの学習に興味がありますか？sysdigのWebサイトにアクセスするか、Twitterでフォローしてください。 新しいワンライナーを定期的に投稿します。

Chisels

SysdigのChisels(ノミ)は小さなスクリプトで、sysdigイベントストリームを分析して有用なアクションを実行します。本質的に、それらはあなたがあなたのsysdigデータで本当にクールなことをすることを可能にします。データを掘り下げ、Chisels(ノミ)を使用してそれを美しいものに成形します。 わかりましたか？ すばらしい！

Chisels(ノミ)は、よく知られている強力で非常に効率的なスクリプト言語であるLuaで書かれています。

完全なチュートリアルについては、ここにアクセスしてください。