

Viajante dos logs

O plugin do Nagios `check_logfiles` ajuda a monitorar arquivos de log – mesmo quando há rodízio e mudanças de nome.

por **Gerhard Lausser**

A ferramenta de monitoramento Nagios é, na realidade, uma plataforma bastante genérica para observação. Ele permite o monitoramento de computadores, processos, dispositivos e serviços de rede. Outra coisa que ele pode acompanhar são arquivos de log. Os plugins `check_log` e `check_log2`, por exemplo, são populares com muitos administradores; entretanto, esses plugins às vezes têm problemas em situações nas quais um aplicativo ou script esteja fazendo o rodízio dos logs. As ferramentas tendem a escorregar ocasionalmente e perder algumas linhas, o que não se pode permitir quando se precisa de 100% de cobertura.

Para cobrir essas falhas, o plugin `check_logfiles`[1] foi desenvolvido para verificar cada uma das entradas – mesmo que um log seja movido, mude de nome ou desapareça num arquivo compactado durante o período de monitoramento.

Quadro 1: Opções de configuração

Na execução do `configure`, antes da instalação, há algumas opções importantes para regular o funcionamento do `check_logfiles`:

- ▶ `--with-perl` deve ser usado caso já exista um interpretador `Perl` instalado no sistema.
- ▶ `--prefix` especifica o diretório `home` da instalação do Nagios. O plugin é instalado no subdiretório `libexec/`.
- ▶ `--with-sockfiles-dir` especifica o diretório onde serão armazenadas as informações de estado entre as execuções do programa.

Mas isso não é tudo: vários outros recursos sofisticados fazem esse plugin se destacar de seus antecessores. Por exemplo, o `check_logfiles` consegue lidar com múltiplas chaves de busca, com exceções que identificam um subconjunto especial de uma chave de busca como inofensiva, aplicar limites que disparam alertas após um número mínimo de ocorrências e integrar programas externos.

Este artigo mostrará como começar a monitorar arquivos de log com o plugin do Nagios `check_logfiles`. Para começar, vamos supor que o leitor tenha um conhecimento básico do Nagios. Quem estiver em busca desses conhecimentos pode consultar a edição 31 da *Linux Magazine*[2], que teve como tema de capa justamente o monitoramento de redes com Nagios.

Instalação

O plugin `check_logfiles` está disponível como um tarball[1]. Após ser baixado e descompactado, basta entrar no diretório `check_logfiles-2.3.1.1/` e seguir os passos padrão de instalação: `configure; make; make install`. Há várias opções disponíveis na etapa do `configure` (veja o **quadro 1**).

Primeiro caso

Com o plugin instalado e configurado, já é possível usá-lo para monitorar logs.

Para um primeiro contato com o plugin em ação, considere o exemplo a seguir, que realiza uma busca simples pelo texto `BIGERROR` num arquivo

chamado `rhubarbomat.log`. A chamada ao plugin é semelhante a:

```
check_logfiles \  
-criticalpattern='BIGERROR' \  
-logfile=rhubarbomat.log
```

Se a string `BIGERROR` ocorrer em uma linha que tenha sido adicionada após a última execução do `check_logfiles`, o plugin retorna um status `CRITICAL`; caso contrário, ele retorna `OK`. O texto é, na verdade, uma expressão regular.

Em vez de `-criticalpattern`, seria possível usar `-warningpatter`, como em `--warningpattern='SMALLERROR'`. O código de saída para uma busca com êxito é 1 para `WARNING`. Obviamente, nada impede o uso das duas opções ao mesmo tempo.

Esse primeiro exemplo não leva em consideração o rodízio de logs. Embora o plugin identifique a chave de busca, ele não procuraria em arquivos de log que tivessem sido rotacionados, atuando apenas no arquivo mais recente.

Para permitir que a busca cubra logs rotacionados entre duas chamadas ao `check_logfiles` e, portanto, para evitar descontinuidades, o plugin precisa de uma dica de onde encontrar os arquivos mais antigos.

O parâmetro que lida com isso é o `-rotation`, que passa o nome do novo arquivo ou contém uma expressão regular que coincide com os nomes de arquivos rotacionados (**figura 1**). Primeiro, suponha que o

arquivo `rhubarbomat.log` seja automaticamente renomeado para `rhubarbomat.log.0` diariamente e que seja criado um arquivo vazio `rhubarbomat.log`. Depois disso, o antigo arquivo `rhubarbomat.log.0` é renomeado para `rhubarbomat.log.1`, o `rhubarbomat.log.1` vira `rhubarbomat.log.2` e assim por diante. Nesse caso, o parâmetro `-logfile=/var/log/rhubarbomat.log` `-rotation='rhubarbomat\log\%d'` faria o plugin encontrar e efetuar buscas nas versões anteriores do log. Como alternativa, pode-se especificar explicitamente o nome de arquivo `rhubarbomat.log.0`.

O plugin `check_logfiles` investiga somente as linhas do arquivo que tenham sido alteradas desde sua última execução, o que significa que reexecuções imediatas produzirão resultados diferentes. Após retornar um resultado `CRITICAL`, a próxima chamada produz `OK`, como mostra o [exemplo 1](#). Uma definição de serviço para o Nagios é fornecida no [exemplo 2](#).

Arquivo de configuração

O `check_logfiles` realmente se destaca quando é usado um arquivo de configuração em vez de parâmetros de linha de comando. O exemplo anterior precisaria do seguinte arquivo de configuração:

```
$ cat rhubarb.cfg
@searches = ({
    tag => '0815',
    logfiles => '/var/log/
➤ rhubarbomat.log',
    criticalpatterns =>
➤ '*.0815.*',
    rotation => 'loglog0log1',
    options => 'noprotocon'
});
```

O plugin é executado por uma linha de comando como `check_logfiles -f nome_do_arquivo`. É fácil ver que o

arquivo de configuração é composto por código Perl. Os elementos no vetor `@searches` (referido apenas como “a busca” daqui em diante) são referências a *hashes* que combinam o log e a chave de busca. A tag é um identificador único para essa combinação. O plugin precisa disso para evitar ambigüidades nos arquivos que armazenam as informações de status para a próxima execução do `check_logfiles`. Um vetor possibilita a busca em múltiplos arquivos de log com uma única chamada ao plugin.

O código em Perl para essa configuração é mostrado no [exemplo 3](#).

Por outro lado, se for desejável que o plugin soe um alarme caso uma chave de busca esteja ausente do log, o padrão de busca precisa iniciar com um ponto de exclamação

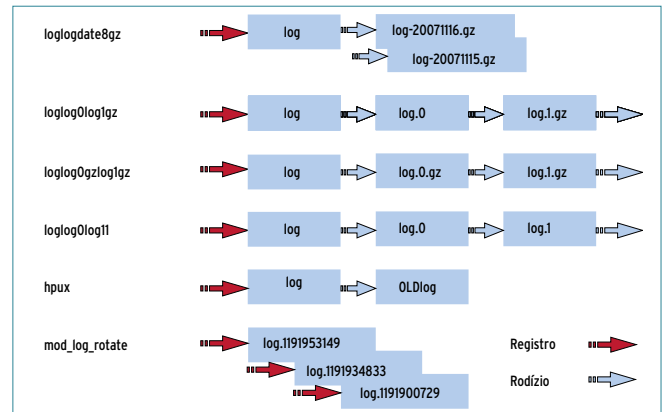


Figura 1 O rodízio de arquivos de log dificulta o trabalho de alguns plugins. As entradas podem acabar sendo perdidas em virtude das mudanças de nomes de arquivos.

(!). A seguinte sintaxe informa se o becape da noite passada foi completado sem erro:

```
criticalpatterns => '!backup
➤ successful']
```

Além disso, é possível definir exceções que se parecem com a mensagem buscada, mas representam um caso especial:

Exemplo 1: Execuções recorrentes

```
01 $ logger "test1 this is 0815"
02 $ logger "this isn't because its 0916"
03 $
04 $ check_logfiles -logfile=/var/log/ rhubarbomat.log --tag=0815
➤-criticalpattern= '*.0815.*' --rotation='loglog0log1'
05
06 CRITICAL - (1 errors in check_logfiles.protocol-2007-10-10-15-10-02) -
Oct 10 15:09:56 localhost lausser: test1 this is 0815 |0815_lines=2
07 0815_warnings=0 0815_criticals=1 0815_unknowns=0
08 $
09 $ echo $?
10 2
11 $
12 $ logger "rhubarb"
13 $ check_logfiles -logfile=/var/log/ rhubarbomat.log --tag=0815
➤-criticalpattern= '*.0815.*' --rotation='loglog0log1'
14
15 OK - no errors or warnings |0815_lines=1 0815_warnings=0 0815_criticals=0
16 0815_unknowns=0
17 $ echo $?
18 0
```

```
criticalpatterns => ['SCSI Error'].
criticalexceptions => ['SCSI
↳ Error. *disk0 .*'],
```

Essas entradas soariam o alarme do Nagios para a linha:

```
SCSI Error /dev/disk5 I/O Timeout
```

mas fariam o plugin ignorar a linha:

```
SCSI Error /dev/disk0 I/O Timeout
```

Rodízio

No final de cada rodada, o plugin guarda a última posição lida no log, juntamente com a data de alteração e o número do inode do arquivo. Essa informação é armazenada no que se conhece como arquivo de busca, ou *seek file*. O `check_logfiles` gera o nome do arquivo de busca a partir do nome do arquivo de log e do dia.

Na próxima vez que o plugin for executado, ele comparará esses dados com as propriedades do arquivo de log atual e verificará se o arquivo foi expandido, deletado, rotacionado ou criado como um arquivo novo.

Dependendo do intervalo desde a última execução do plugin, vários rodízios podem ter ocorrido. O plugin usa a hora e o parâmetro `rotation` para encontrar arquivos correspondentes.

Na maioria dos casos, o arquivo de log tem apenas algumas linhas novas. O `check_logfiles` continua na posição que marcou no fim da última execução e lê as linhas seguintes até chegar ao fim do arquivo. Esse princípio oferece ao plugin uma enorme vantagem de velocidade

quando comparado a outras técnicas que se baseiam no *diff* para precisar as diferenças entre o arquivo de log atual e uma cópia gravada, principalmente no caso de crescimento rápido do arquivo.

Digitando `./configure with-seek-file-dir`, pode-se especificar um diretório para os arquivos de busca, ou pode-se alterar o caminho dinamicamente com a variável `$seekfilesdir` no arquivo de configuração.

O plugin usa `/tmp/` por padrão; porém, é bom alterar isso para `/var/tmp/`, pois alguns sistemas operacionais não mantêm o conteúdo de `/tmp/` após a reinicialização.

Tipos

Além do parâmetro `rotation`, é possível procurar no parâmetro `type`, que especifica o tipo de arquivo de log. Se existir o parâmetro de rodízio, `type` supõe um valor para `rotation`. Isso significa que os arquivos compactados são relevantes para a busca. Se o parâmetro de rotação não existir, `type` supõe o valor `simple`. Essa configuração faz sentido se um aplicativo gerar continuamente novos arquivos de log e apagar os arquivos existentes, ou se o administrador estiver preparado para aceitar o fato de que as últimas poucas linhas de um arquivo de log rotatório não serão levadas em consideração.

Exemplo 2: Definição de serviço

```
01 define service {
02   service_description check_
↳ 0815msgs
03   host_name logserver
04   max_check_attempts 1
05   is_volatile 1
06   check_command
07   check_logfiles_critical!0815!/
↳ var/log/ rhubarbomat.
↳ log!loglog0log1!.!0815.*
08 }
09 define command {
10   command_name check_logfiles_
↳ critical
11   command_line $USER1$/check_
↳ logfiles $$
12   --logfile="$ARG2$"
13   --criticalpattern="$ARG4$"
↳ --tag="$ARG1$" $$
14   --rotation="$ARG3$"
15 }
```

Exemplo 3: Exemplo de configuração

```
01 @searches = (
02 {
03   tag => 'lamp-apache'
04   logfile => '/var/log/apache/
↳ error.log',
05   criticalpatterns => ['.*error.*',
↳ '.*fatal.*'],
06   rotation => 'solaris'
07 },
08 {
09   tag => 'lamp-mysql',
10   logfile => '/var/log/mysql.log',
11   criticalpatterns =>
↳ ['corruption','you hit a bug']
12 }
```

Exemplo 4: Pesquisa em tipos de log virtual

```
01 @searches = (
02 {
03   tag => 'host0',
04   logfile => '/sys/class/scsi_host/host0/state',
05   type => "virtual",
06   criticalpatterns => [
07   'Link [^Up]+' # Soar o alarme se "Link Up" ausente
08 ],
09   options => 'noprotocol',
10 },
11 );
```

O log *virtual* é outro tipo em que o `check_logfiles` buscará. Esse tipo é usado, por exemplo, para o sistema de arquivos `/proc/` em máquinas Linux (e oferece a opção de configurar facilmente o monitoramento de hardware).

Os arquivos desse sistema de arquivos não crescem; em vez disso, é preciso tratá-los como se tivessem sido criados imediatamente antes de serem lidos. O plugin sempre investiga esses arquivos a partir da primeira linha (veja o [exemplo 4](#)).

Além disso, o tipo `errpt` procura o relatório de erros do AIX. Isso informa ao plugin que ele deve procurar padrões na saída do comando `errpt`, exatamente como se fosse um arquivo de log normal. O tipo `psloglist` ainda é experimental; ele permite que o plugin faça buscas no log de eventos de máquinas Windows.

Parâmetros de busca

Vários parâmetros estão disponíveis para padrões para buscas nos logs. Os parâmetros mais importantes

estão listados no [quadro 2](#), enquanto que os parâmetros globais são descritos no [quadro 3](#). Uma lista completa de todos os parâmetros possíveis pode ser encontrada online [\[1\]](#).

Os parâmetros são usados no arquivo de configuração conforme mostrado no trecho do [exemplo 5](#).

Saída e desempenho

A saída do `check_logfiles` contém referências às descobertas, como por exemplo:

```
CRITICAL - (3 errors in check_
↳logfiles.protocol-2007-10-10-16-
↳21-09) InnoDB: Database pae
↳corruption on disk or a failed
↳...|mysql_lines=12 mysql_
↳warnings=0 mysql_criticals=3
↳mysql_unknowns=0
```

Exemplo 5: Parâmetros do arquivo de configuração

```
01 $ cat rhubarb.cfg
02 @searches = ({
03   tag => '0815',
04   logfile => '/var/log/rhubarbomat.log',
05   archivedir => '/var/log/archives',
06   rotation => 'loglog0gzloglgz',
07   criticalpatterns => '.*0815.*',
08   criticalexceptions => '.*0815 macht aber nix.*',
09   warningpatterns => ['.*failure.*','!successful'],
10   warningthreshold => 10,
11   okpatterns => '.*cleared.*',
12   options => 'case,noprocol,script'
13   script => 'restart_rhubarbomat'
14 });
```

Além do código típico de saída do Nagios, pode-se ver três pontos (...), o que indica que há mais linhas coincidentes.

Para cada busca ou dia, o plugin também retorna um conjunto de quatro estatísticas de desempenho:

↳ `_lines`: o número de linhas buscadas no log.

Quadro 2: Parâmetros de busca

Uma busca eficiente nos arquivos de log depende do bom uso dos parâmetros disponíveis. Conheça todos eles:

- ↳ `tag` Um curto descritor único para a busca.
- ↳ `logfile` Nome do arquivo de log a ser varrido.
- ↳ `archivedir` Diretório com os logs rotacionados.
- ↳ `rotation` Expressão regular usada para localizar arquivos de log rotacionados. Há valores pré-definidos para os padrões mais comuns.
- ↳ `criticalpatterns` Um único padrão que o plugin busca no log. Se for necessário pesquisar múltiplos padrões em uma categoria, é preciso especificá-los como elementos de um vetor (veja o [exemplo 6](#)).
- ↳ `criticalexceptions` Suporta uma especificação de padrões mais granular: o parâmetro ignora exceções que não sejam contabilizadas como erros.
- ↳ `warningthreshold` Limites (*thresholds*) são usados sempre que se deseja contar um certo número de ocorrências antes de emitir um alerta:
- ↳ `warningthreshold=>n` Significa que somente uma ocorrência será contabilizada a cada *n*.
- ↳ `okpatterns` Reinicia o contador e deleta todas as ocorrências dos tipos *warning* e *critical* encontradas anteriormente.
- ↳ `nologfilenocry` Ignora arquivos de log inexistentes; caso contrário, se o arquivo não estiver presente, o plugin retorna um status *UNKNOWN* (desconhecido).
- ↳ `syslogserver` Se o log contiver mensagens de múltiplos servidores, o plugin usa essa opção para procurar somente as mensagens vindas da máquina local.
- ↳ `syslogclient=nome` O mesmo que a opção acima, mas procura apenas em mensagens de um cliente específico. Essa opção é interessante para servidores *Syslog* centralizados.
- ↳ `nocase` Ignora a caixa nas expressões regulares.
- ↳ `options` Múltiplas opções separadas por vírgulas oferecem um controle mais fino sobre as ações do plugin. Um prefixo `no` inverte o significado da opção.
- ↳ `nocase` Torna os padrões insensíveis à caixa.
- ↳ `noprocol` Impede que o plugin crie um arquivo de protocolo. Normalmente, qualquer linha do log que contenha o padrão de busca é escrita no arquivo de protocolo. Essa opção economiza processamento trabalhoso em caso de alertas.

Exemplo 6: Busca de múltiplos padrões

```
01 @searches = ({
02   tag => 'minor_errors',
03   type => 'errpt',
04   criticalpatterns =>
05     ['ADAPTER ERROR',
06     'The largest dump device
07     is too small.',
08     'The copy directory is too
09     small.',
10     'Kernel heap use exceeds
11     allocation count',
12     'Kernel heap use exceeds
13     percentage thres',
14     'LINK ERROR',
15     'SCSI BUS OR DEVICE ERROR',
16     'SCSI DEVICE OR MEDIA
17     ERROR',
18     'Possible malfunction on
19     local adapter',
20     'ETHERNET DOWN',
21     'UNABLE TO ALLOCATE SPACE
22     IN KERNEL HEAP'
23   ],,);
```

▸ **_warnings**: o número de linhas que contêm padrões de alerta.

▸ **_criticals**: o número de linhas que contêm padrões críticos.

Esses parâmetros oferecem uma rápida indicação da densidade do problema.

Ações

A opção `script` pode executar um programa no caso de uma coincidência específica:

```
script => 'nome_do_programa'
```

ou na versão mais recente:

```
script => sub { código_em_perl }
```

As ações incluem reiniciar um aplicativo ou enviar *traps* SNMP e mensagens NSCA. Isso significa que o `check_logfiles` pode rodar

como aplicativo *standalone* sem depender do tratamento de eventos do Nagios.

Os parâmetros `scriptparams` e `scriptstdin` permitem que os usuários rodem scripts externos com parâmetros de linha de comando – e até passem a entrada a partir de STDIN. O **exemplo 7** ilustra essa possibilidade.

No **exemplo 7**, sempre que aparecer uma mensagem de erro numa linha do arquivo `messages`, a linha é enviada para o servidor Nagios com o comando `send_nsca` como resultado passivo de um serviço.

No caso mais simples, o código de saída retornado pelo script externo será irrelevante e não influenciará o código de saída do `check_logfiles`. Por exemplo, mesmo que a aplicação *Rhubarbomat* do **exemplo 5** reinicie com sucesso, o `check_logfiles` ainda retornará um estado *CRITICAL* para o Nagios.

A opção `smartscrip` passa o código de saída do script externo para o `check_logfile`. O plugin age como se tivesse descoberto outra linha após aquela que o disparou, o que permite o disparo de um erro,

Exemplo 7: Execução de scripts

```
01 $scriptpath = '/usr/bin/nagios/libexec: /usr/local/nagios/contrib';
02 $MACROS = {
03   CL_NSCA_HOST_ADDRESS => "lpmo1.muc",
04   CL_NSCA_PORT => 5778
05 };
06
07 @searches =(
08   {
09     tag => 'rhubarb',
10     logfile => '/var/log/rhubarbomat.log',
11     criticalpatterns => ['ERROR', 'crashed'],
12     script => 'reinicia_rhubarbomat',
13     scriptparams => '--rhubarbprefix=bla',
14     options => 'script'
15   },
16   {
17     tag => 'san',
18     logfile => '/var/adm/messages',
19     criticalpatterns => [
20       'Link Down Event received',
21       'Loop OFFLINE',
22       'fctl:.*disappeared from fabric',
23       '.*Lun.*disappeared.*'
24     ],
25     options => 'script',
26     script => 'send_nsca',
27     scriptparams => '-H $CL_NSCA_HOST_ADDRESS$ -p $CL_NSCA_PORT$
28     -to $CL_NSCA_TO_SEC$ -c $CL_NSCA_CONFIG_FILE$',
29     scriptstdin => '$CL_HOSTNAME$\t$CL_SERVICEDESC$\t$CL_
30     SERVICESTATEID$\t$CL_SERVICEOUTPUT$\n',
31   });
```

Quadro 3: Parâmetros globais

Além dos parâmetros relacionados a uma única entrada de busca, outras variáveis globais são lidas por todas as buscas e definem o comportamento do plugin, independentemente de uma busca individual.

- ▶ `$seekfilesdir` Especifica o diretório onde os arquivos com informações de status serão salvos.
- ▶ `$scriptpath` Uma lista de caminhos que o plugin varre em busca de plugins externos, que ele ativa com o parâmetro `script`.
- ▶ `$prescript` Especifica um programa externo para ser executado no início.
- ▶ `$postscript` Especifica um programa externo para ser executado antes do término.
- ▶ `$protocolsdir` Diretório onde o `check_logfiles` guarda os arquivos de protocolo com as linhas retornadas nas buscas.

Exemplo 8: Script supersmart

```
01 @searches =(
02 {
03   tag => 'rhubarb',
04   logfile => '/var/log/rhubarbomat.log',
05   criticalpatterns => ['ERROR', 'crashed'],
06   script => sub {
07     if (`reinicia_rhubarbomat` =~ /successful/) {
08       if ($ENV{CHECK_LOGFILES_SERVICEOUTPUT} =~ / ERROR/) {
09         printf "OK - rhubarbomat reiniciado\n";
10         return 0;
11       } else {
12         printf "WARNING - rhubarbomat travado reiniciado\n";
13         return 1;
14       }
15     } else {
16       printf "CRITICAL - impossivel reiniciar rhubarbomat\n";
17       return 2;
18     }
19   },
20   options => 'supersmartscrip'
21 },
```

mas não a reversão da mensagem original do log ou a reavaliação da mensagem.

A terceira opção é `supersmart script`. Scripts desse tipo sobrescrevem a entrada coincidente no arquivo de log com seus códigos e textos de saída em vez de adicionar uma entrada. Diversas variáveis de ambiente estão disponíveis para esses scripts:

- ▶ `CHECK_LOGFILES_SERVICEOUTPUT` – conteúdo da linha de ativação
- ▶ `CHECK_LOGFILES_SERVICESTATE` – `WARNING`, `CRITICAL` ou `UNKNOWN`
- ▶ `CHECK_LOGFILES_SERVICESTATEID` – 1, 2 ou 3

Com o uso dessa informação e outros dados – assim como a hora do dia ou os resultados do reinício do aplicativo –, a mensagem de erro depois pode ser reavaliada. Isso permite que o verificador de arquivo

de log troque um status `CRITICAL` para `WARNING` ou retorne um código de saída de 0 e, portanto, cancele o alerta. O exemplo 8 dá um exemplo.

Scripts

As ações também podem ser ativadas antes de se começar a procurar em um arquivo de log, ou após terminar todas as buscas.

O parâmetro `$prescript`, que aponta para um script externo ou subrotina em Perl, ajuda a chamar uma ação. Os `prescripts supersmart` cancelam a execução do `check_logfiles` caso o código de saída seja maior que zero. Isso possibilita a verificação do estado de execução de um processo.

Se o processo não estiver rodando, por que se preocupar em verificar seus logs respectivos? Os `prescripts` também podem forçar aplicativos a escreverem (fazer um `flush`) em

seus logs, certificando-se de que os dados estejam atualizados.

Os `postscripts supersmart` podem substituir completamente os resultados do `check_logfiles`, não importa quantas mensagens de erro eles contivessem originalmente.

Ou, se o formato de saída padrão do `check_logfiles` não for agradável, pode-se executar um `postscript supersmart` para modificá-lo, a fim de obter uma melhor ênfase. ■

Mais informações

[1] Check_logfiles: <http://www.console.com/opensource/nagios/check-logfiles/>

[2] Julian Hein, "Nagios: O verdadeiro grande irmão": <http://www.linuxmagazine.com.br/article/1011>