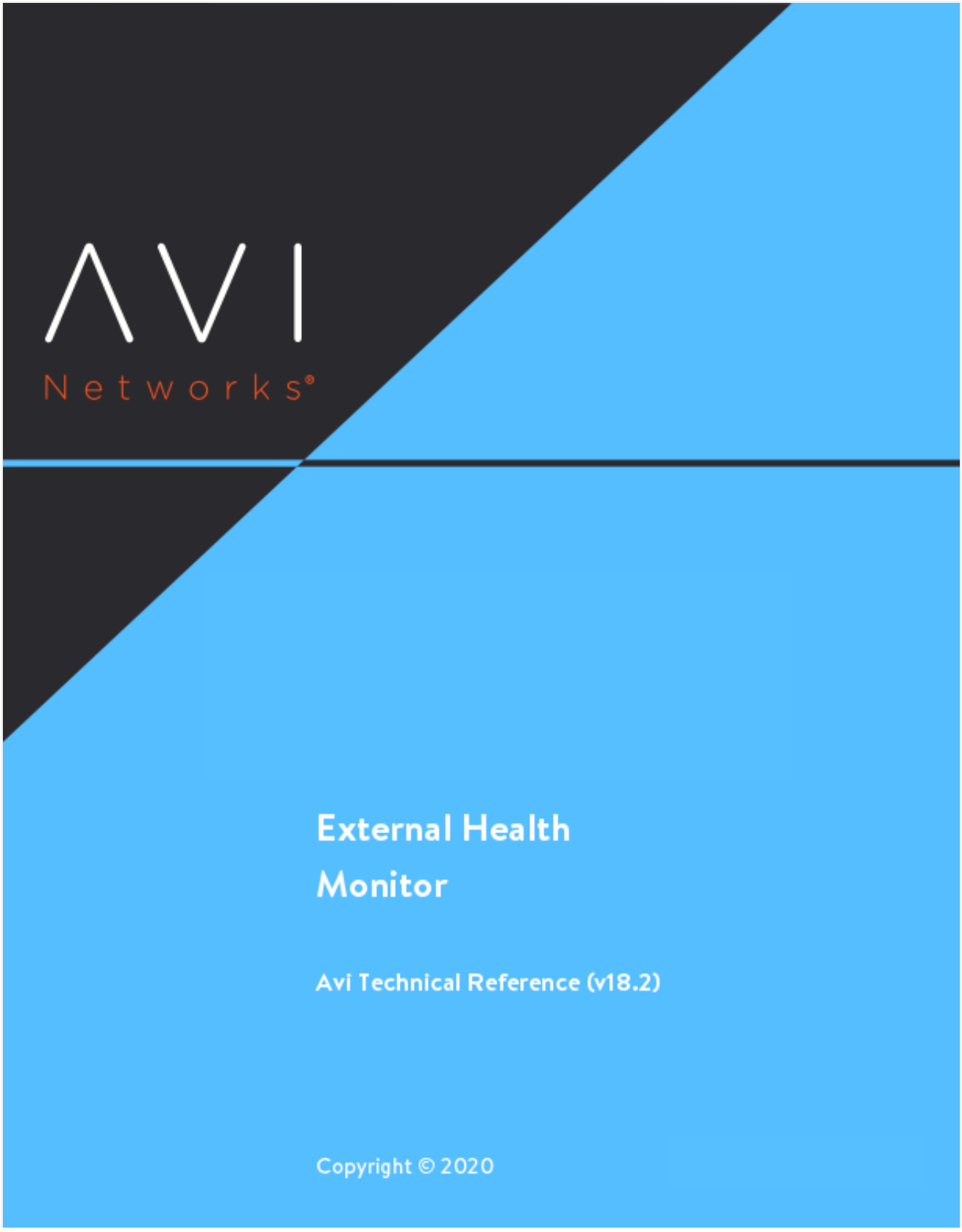

External Health Monitor

Avi Technical Reference (v18.2)

External Health Monitor

view online

Overview

This article covers the specific configuration for external health monitor type. Refer to Overview of Health Monitors article for general monitor information, implementation, and other monitor types.

The external monitor type allows scripts to be written to provide highly customized and granular health checks. The scripts may be Linux shell, Python, or Perl, which can be used to execute `wget`, `netcat`, `curl`, `snmpget`, `mysql-client`, or `dig`. External monitors have constrained access to resources, such as CPU and memory to ensure normal functioning of Avi Service Engines. As with any custom scripting, thoroughly validate the long term stability of the implemented script before pointing it at production servers.

Errors generated from the script may be viewed in the output of the Operations > Events log.

Avi Vantage includes three sample scripts via the System-Xternal Perl, Python and Shell monitors.

Note: Starting with release 18.2.5, Avi Vantage supports IPv6 external health monitors.

Configuring General Monitor

- Send Interval - Frequency at which the health monitor initiates a server check in seconds.
  - Best Practice - For busy Service Engines, keep the monitoring interval lower and receive timeout larger, since external checks tend to use more system resources than the system default monitors.
- Receive Timeout - Maximum amount of time before the server must return a valid response to the health monitor in seconds.
- Successful Checks - Number of consecutive health checks that must succeed before Avi Vantage marks a down server as being back up.
- Failed Checks - Number of consecutive health checks that must fail before Avi Vantage marks an up server as being down.

Configuring External Specific

As a best practice, clean up any temporary files created by scripts.

While building an external monitor, you need to manually test the successful execution of the commands. To test a command from an SE, it may be necessary to switch to the proper namespace or tenant. The production external monitor will correctly use the proper tenant. To manually switch tenants when testing a command from the SE CLI, follow the commands in the following article: Manually Validate Server Health.

- Script Code - Upload the script via copy/paste or uploading the file.
- Script Parameters - Enter any optional arguments to apply. These strings are passed in as arguments to the script, such as $1 = server IP, $2 = server port.
- Script Variables - Custom environment variables may be fed into the script to allow simplified re-usability. For instance, a script that authenticates to the server may have a variable set to USER=test.
- Script Success - If a script exits with any data, it is considered as success and marks as server up. If there is no data from the script, the monitor will mark the server down. In the SharePoint monitor example below, the script includes a `grep "200 OK"`. If this is found, this data is returned and the monitor exits as success. If the `grep` does not find this string, no data is returned and the monitor marks the server down.

MySQL Example Script

```
#!/bin/bash
#mysql --host=$IP --user=root --password=s3cret! -e "select 1"
```

SharePoint Example Script

```
#!/bin/bash
#curl http://$IP:$PORT/Shared%20Documents/10m.dat -I -L --ntlm -u $USER:$PASS -I -L > /run/hmuser/$HM_NAME.out 2>/dev/r
curl http://$IP:$PORT/Shared%20Documents/10m.dat -I -L --ntlm -u $USER:$PASS -I -L | grep "200 OK"
```

SharePoint Script Variables

```
USER='foo\administrator' PASS=foo123
```

Oracle Example Script

```
#!/usr/bin/python
import sys
import os
import cx_Oracle
IP=os.environ['IP']
conn_str='HR_user/HR_pw@' + IP  + '/hr_db'
connection = cx_Oracle.connect(conn_str)
cursor = connection.cursor()
cursor.execute('select * from JOBS')
for row in cursor:
    print row
connection.close()
```

Oracle Script Variables

```
LD_LIBRARY_PATH=/usr/lib/oracle/12.2/client64/lib
```

RADIUS Example Script

The below example performs an Access-Request using PAP authentication against the RADIUS pool member and checks for an Access-Accept response.

```
#!/usr/bin/python
import os
import radius
```

```
try:
    r = radius.Radius(os.environ['RAD_SECRET'],
                      os.environ['IP'],
                      port=int(os.environ['PORT']),
                      timeout=int(os.environ['RAD_TIMEOUT']))
    if r.authenticate(os.environ['RAD_USERNAME'], os.environ['RAD_PASSWORD']):
        print 'Access Accepted'
except:
    pass
```

RAD_SECRET, RAD_TIMEOUT, RAD_USERNAME and RAD_PASSWORD can be passed in the health monitor script variables, for example:

```
RAD_SECRET=foo123 RAD_USERNAME=avihealth RAD_PASSWORD=bar123 RAD_TIMEOUT=1
```

Applications like curl can have different syntax for v4 and v6 addresses. The external health monitor scripts should be aware of these syntax. Following are the examples:

Shell Script Example for IPV6 Support

```
#!/bin/bash
#curl -v $IP:$PORT >/run/hmuser/$HM_NAME.$IP.$PORT.out
if [[ $IP =~ : ]];
then curl -v [$IP]:$PORT;
else curl -v $IP:$PORT;
fi
```

perl Script Example for IPV6 Support

```
#!/usr/bin/perl -w
my $ip= $ARGV[0];
my $port = $ARGV[1];
my $curl_out;
if ($ip =~ /:/) {
$curl_out = `curl -v "[$ip]":"$port" 2>&1`;
} else {
$curl_out = `curl -v "$ip":"$port" 2>&1`;
}
if (index($curl_out, "200 OK") != -1) {
    print "Server is up";
}
```

List of SE Packages

Scripting Languages

- Bash (shell script)

- perl
- Python

Linux Packages (`apt`)

- curl
- snmp
- dnsutils
- libpython2.7
- python-dev
- mysql-client
- nmap
- freetds-dev
- freetds-bin
- ldapsearch

Python Packages (`pip`)

- pymssql
- cx_Oracle (and related libraries for Oracle Database 12c) ? Avi Vantage 17.1.3 onwards
- py-radius ? Avi Vantage 17.2.5 onwards

NTP Health Monitor Example using netcat program

```
nc -zuv pool.ntp.org 123 2>&1 | grep "(ntp) open"
```

The sample configuration for using a native perl script is as follows:

```
#!/usr/bin/perl
# ntpdate.pl

# this code will query a ntp server for the local time and display
# it.  it is intended to show how to use a NTP server as a time
# source for a simple network connected device.

#
# For better clock management see the offical NTP info at:
# http://www.eecis.udel.edu/~ntp/
#

# written by Tim Hogard (thogard@abnormal.com)
# Thu Sep 26 13:35:41 EAST 2002
# this code is in the public domain.
# it can be found here http://www.abnormal.com/~thogard/ntp/

$HOSTNAME=shift;
$HOSTNAME="192.168.1.254" unless $HOSTNAME ;    # our NTP server
$PORTNO=123;            # NTP is port 123
$MAXLEN=1024;           # check our buffers
```

```
use Socket;

#we use the system call to open a UDP socket
socket(SOCKET, PF_INET, SOCK_DGRAM, getprotobyname("udp")) or die "socket: $!";

#convert hostname to ipaddress if needed
$ipaddr   = inet_aton($HOSTNAME);
$portaddr = sockaddr_in($PORTNO, $ipaddr);

# build a message.  Our message is all zeros except for a one in the protocol version field
# $msg in binary is 00 001 000 00000000 ....  or in C msg[]={010,0,0,0,0,0,0,0,0,...}
#it should be a total of 48 bytes long
$MSG="\01
```

Note: The ntpdate or ntpq programs are not packaged in the Service Engine, and hence cannot be used at this point in time.