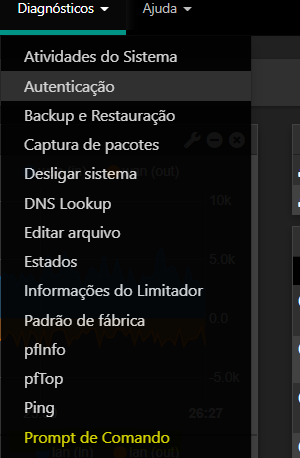
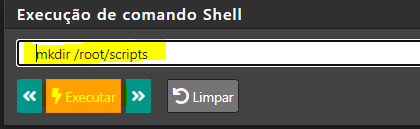
**Telegram**

**1-**

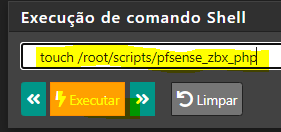


**2-**



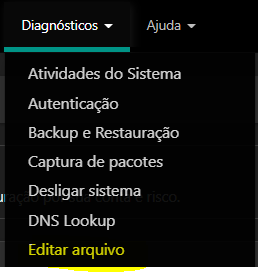
mkdir /root/scripts

**3-**

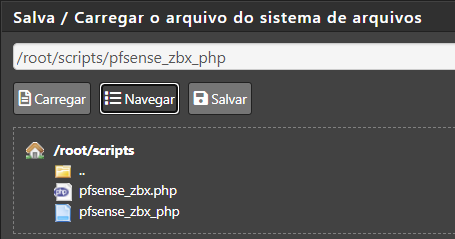
****

touch /root/scripts/pfsense\_zbx\_php

**4-**

****

**5- Colar script no arquivo “pfsense\_zbx\_php”**

****

**SCRIPT:**

<?php

/\*\*\*

pfsense\_zbx.php - Interface pfSense Zabbix

Versão 1.1.1 - 24-10-2021

Escrito por Riccardo Bicelli <r.bicelli@gmail.com>

Este programa está licenciado sob a Licença Apache 2.0

\*/

//Algumas definições úteis

define('SPEEDTEST\_INTERVAL', 8); //Intervalo de teste de velocidade (em horas)

require\_once('globals.inc');

require\_once('functions.inc');

require\_once('config.inc');

require\_once('util.inc');

//Para descoberta de interfaces

require\_once('interfaces.inc');

//Para descoberta OpenVPN

require\_once('openvpn.inc');

//Para descoberta de serviço

require\_once("service-utils.inc");

//Para o sistema

require\_once('pkg-utils.inc');

//Para DHCP

//Função de teste, para propósito de criação de template

função pfz\_test(){

        $linha = "-------------------\n";

        $ovpn\_servers = pfz\_openvpn\_get\_all\_servers();

        echo "Servidores OPEVPN:\n";

        print\_r($ovpn\_servers);

        echo $linha;

        $ovpn\_clients = openvpn\_get\_active\_clients();

        echo "Clientes OPENVPN:\n";

        print\_r($ovpn\_clients);

        echo $linha;

        $ifdescrs = get\_configured\_interface\_with\_descr(true);

        $ifaces=array();

        foreach ($ifdescrs as $ifdescr => $ifname){

          $ifinfo = get\_interface\_info($ifdescr);

          $ifaces[$ifname] = $ifinfo;

        }

        echo "Interfaces de rede:\n";

        print\_r($ifaces);

        print\_r(get\_interface\_arr());

        print\_r(get\_configured\_interface\_list());

        echo $linha;

        $serviços = get\_services();

        echo "Serviços: \n";

        print\_r($serviços);

        echo $linha;

        echo "IPsec:\n";

require\_once("ipsec.inc");

global $configuração;

init\_config\_arr(array('ipsec', 'phase1'));

init\_config\_arr(array('ipsec', 'phase2'));

$a\_phase2 = &$config['ipsec']['phase2'];

        $status = ipsec\_list\_sa();

echo "Status do IPsec: \n";

print\_r($status);

$a\_phase1 = &$config['ipsec']['phase1'];

$a\_phase2 = &$config['ipsec']['phase2'];

echo "Configuração de IPsec Fase 1: \n";

print\_r($a\_phase1);

echo "Configuração de IPsec Fase 2: \n";

print\_r($a\_phase2);

echo $linha;

//Pacotes

echo "Pacotes: \n";

require\_once("pkg-utils.inc");

$installed\_packages = get\_pkg\_info('all', false, true);

print\_r($installed\_packages);

}

// Descoberta de interface

// Performance melhorada

function pfz\_interface\_discovery($is\_wan=falso,$is\_cron=falso) {

    $ifdescrs = get\_configured\_interface\_with\_descr(true);

    $ifaces = get\_interface\_arr();

    $ifcs=array();

    $if\_ret=array();

    $json\_string = '{"data":[';

    foreach ($ifdescrs as $ifname => $ifdescr){

          $ifinfo = get\_interface\_info($ifname);

          $ifinfo["descrição"] = $ifdescr;

      $ifcs[$ifname] = $ifinfo;

    }

    foreach ($ifaces as $hwif) {

        $ifdescr = $hwif;

        $has\_gw = false;

        $is\_vpn = false;

        $has\_public\_ip = false;

        foreach($ifcs as $ifc=>$ifinfo){

                if ($ifinfo["hwif"] == $hwif){

                        $ifdescr = $ifinfo["descrição"];

                        if (array\_key\_exists("gateway",$ifinfo)) $has\_gw=true;

                        // Edição nº 81 - https://stackoverflow.com/a/13818647/15093007

                        if (filter\_var($ifinfo["ipaddr"], FILTER\_VALIDATE\_IP, FILTER\_FLAG\_IPV4 | FILTER\_FLAG\_NO\_PRIV\_RANGE | FILTER\_FLAG\_NO\_RES\_RANGE)) $has\_public\_ip=true;

                        if (strpos($ifinfo["if"],"ovpn")!==false) $is\_vpn=true;

                        pausa;

                }

        }

if (($is\_wan==falso) || (($is\_wan==true) && (($has\_gw==true) || ($has\_public\_ip==true)) && ($is\_vpn==false)) ) {

    $if\_ret[]=$hwif;

    $json\_string .= '{"{#IFNAME}":"' . $hwif . '"';

    $json\_string .= ',"{#IFDESCR}":"' . $ifdescr . '"';

    $json\_string .= '},';

        }

    }

    $json\_string = rtrim($json\_string,",");

    $json\_string .= "]}";

if ($is\_cron) return $if\_ret;

    echo $json\_string;

}

//Teste de velocidade da interface

function pfz\_interface\_speedtest\_value($ifname, $value){

    $tvalor = explodir(".", $valor);

    if (count($tvalue)>1) {

     $valor = $tvalor[0];

     $subvalor = $tvalor[1];

    }

//Se a interface possui um gateway é considerado WAN, então vamos fazer o speedtest

$filename = "/tmp/speedtest-$ifname";

if (file\_exists($filename)) {

$speedtest\_data = json\_decode(file\_get\_contents($filename), true);

if (array\_key\_exists($value, $speedtest\_data)) {

if ($subvalor == falso)

echo $speedtest\_data[$valor];

senão

echo $speedtest\_data[$value][$subvalue];

}

}

}

// Isso deve ser executado via cron job

function pfz\_speedtest\_cron(){

require\_once("services.inc");

$ifdescrs = get\_configured\_interface\_with\_descr(true);

    $ifaces = get\_interface\_arr();

    $pf\_interface\_name='';

    $subvalor=falso;

    $ifcs = pfz\_interface\_discovery(true, true);

    foreach ($ifcs as $ifname) {

          foreach ($ifdescrs as $ifn => $ifd){

      $ifinfo = get\_interface\_info($ifn);

      if($ifinfo['hwif']==$ifname) {

      $pf\_interface\_name = $ifn;

      pausa;

      }

       }

  pfz\_speedtest\_exec($ifname, $ifinfo['ipaddr']);

    }

}

// instala um cron job para testes de velocidade

function pfz\_speedtest\_cron\_install($enable=true){

//Instala o Cron Job

$command = "/usr/local/bin/php " . \_\_ARQUIVO\_\_ . " speedtest\_cron";

install\_cron\_job($comando, $habilitar, $minuto = "\*/15", "\*", "\*", "\*", "\*", "root", true);

}

function pfz\_speedtest\_exec ($ifname, $ipaddr){

$filename = "/tmp/speedtest-$ifname";

$filetemp = "$filename.tmp";

$filerun = "/tmp/speedtest-run";

// Edição #82

// Sleep com atraso aleatório para evitar problemas quando 2 pfSense na mesma linha de Internet

dormir (rand ( 1, 90));

if ( (time()-filemtime($filename) > SPEEDTEST\_INTERVAL \* 3600) || (file\_exists($filename)==false) ) {

  // o arquivo é mais antigo que SPEEDTEST\_INTERVAL

  if ( (time()-filemtime($filerun) > 180 ) ) @unlink($filerun);

if (file\_exists($filerun)==false) {

  touch($filerun);

  $st\_command = "/usr/local/bin/speedtest --source $ipaddr --json > $filetemp";

exec ($st\_command);

renomear($arquivotemp,$arquivo);

@unlink($filerun);

}

}

retorne verdadeiro;

}

// Descoberta do servidor OpenVPN

function pfz\_openvpn\_get\_all\_servers(){

     $servidores = openvpn\_get\_active\_servers();

     $sk\_servers = openvpn\_get\_active\_servers("p2p");

     $servidores = array\_merge($servidores,$sk\_servers);

     return ($servidores);

}

function pfz\_openvpn\_serverdiscovery() {

     $servidores = pfz\_openvpn\_get\_all\_servers();

     $json\_string = '{"data":[';

     foreach ($servidores como $servidor){

          $nome = trim(preg\_replace('/\w{3}(\d)?\:\d{4,5}/i', '', $server['nome']));

          $json\_string .= '{"{#SERVER}":"' . $server['vpnid'] . '"';

          $json\_string .= ',"{#NAME}":"' . $name . '"';

          $json\_string .= '},';

     }

     $json\_string = rtrim($json\_string,",");

     $json\_string .= "]}";

     echo $json\_string;

}

// Obtém o valor do servidor OpenVPN

function pfz\_openvpn\_servervalue($server\_id,$valuekey){

     $servidores = pfz\_openvpn\_get\_all\_servers();

     foreach($servidores como $servidor) {

          if($server['vpnid']==$server\_id){

               $valor = $servidor[$valorchave];

               if ($valuekey=="status") {

                    if ( ($server['mode']=="server\_user") || ($server['mode']=="server\_tls\_user") || ($server['mode']=="server\_tls") ){

                         if ($value=="") $value="server\_user\_listening";

                    } else if ($server['mode']=="p2p\_tls"){

                        // Para p2p\_tls, certifique-se de que temos um cliente e retorne se for o caso

                        if ($valor=="")

                            $value=(is\_array($server["conns"]) && count($server["conns"]) > 0) ? "cima baixo";

                    }

               }

          }

     }

     switch ($valuekey){

          caso "cones":

               //Conexões do cliente: é um array, portanto é suficiente contar elementos

               if (is\_array($valor))

                    $valor = contagem($valor);

               senão

                    $valor = "0";

               pausa;

          caso "estado":

               $valor = pfz\_valuemap("openvpn.server.status", $valor);

               pausa;

          caso "modo":

               $valor = pfz\_valuemap("openvpn.server.mode", $valor);

               pausa;

     }

     //se ($valor=="") $valor="nenhum";

     echo $valor;

}

//Servidor OpenVPN/Descoberta de autenticação de usuário

function pfz\_openvpn\_server\_userdiscovery(){

     $servidores = pfz\_openvpn\_get\_all\_servers();

     $json\_string = '{"data":[';

     foreach ($servidores como $servidor){

          if (($server['mode']=='server\_user') || ($server['mode']=='server\_tls\_user') || ($server['mode']=='server\_tls') ) {

               if (is\_array($server['conns'])) {

                    $nome = trim(preg\_replace('/\w{3}(\d)?\:\d{4,5}/i', '', $server['nome']));

                    foreach($server['conns'] as $conn) {

                     $common\_name = pfz\_replacespecialchars($conn['common\_name']);

                        $json\_string .= '{"{#SERVERID}":"' . $server['vpnid'] . '"';

                        $json\_string .= ',"{#SERVERNAME}":"' . $name . '"';

                        $json\_string .= ',"{#UNIQUEID}":"' . $server['vpnid'] . '+' . $common\_name . '"';

                        $json\_string .= ',"{#USERID}":"' . $conn['common\_name'] . '"';

                        $json\_string .= '},';

                    }

               }

          }

     }

     $json\_string = rtrim($json\_string,",");

     $json\_string .= "]}";

     echo $json\_string;

}

// Obtém o valor conectado ao usuário do OpenVPN

function pfz\_openvpn\_server\_uservalue($unique\_id, $valuekey, $default=""){

$unique\_id = pfz\_replacespecialchars($unique\_id,true);

     $atpos=strpos($unique\_id,'+');

     $server\_id = substr($unique\_id,0,$atpos);

     $user\_id = substr($unique\_id,$atpos+1);

     $servidores = pfz\_openvpn\_get\_all\_servers();

     foreach($servidores como $servidor) {

          if($server['vpnid']==$server\_id) {

               foreach($server['conns'] as $conn) {

                    if ($conn['common\_name']==$user\_id){

                         $valor = $conn[$valorchave];

                    }

               }

          }

     }

     if ($valor=="") $valor = $default;

     echo $valor;

}

// Descoberta de cliente OpenVPN

function pfz\_openvpn\_clientdiscovery() {

     $clientes = openvpn\_get\_active\_clients();

     $json\_string = '{"data":[';

     foreach ($clientes como $cliente){

          $nome = trim(preg\_replace('/\w{3}(\d)?\:\d{4,5}/i', '', $client['nome']));

          $json\_string .= '{"{#CLIENT}":"' . $client['vpnid'] . '"';

          $json\_string .= ',"{#NAME}":"' . $name . '"';

          $json\_string .= '},';

     }

     $json\_string = rtrim($json\_string,",");

     $json\_string .= "]}";

     echo $json\_string;

}

function pfz\_replacespecialchars($inputstr,$reverse=false){

$specialchars = ",',\",`,\*,?,[,],{,},~,$,!,&,;,(,),<,>,|,#,@,0x0a" ;

$specialchars = explodir(",",$specialchars);

$resultstr = $inputstr;

for ($n=0;$n<count($specialchars);$n++){

if ($reverso==falso)

$resultstr = str\_replace($specialchars[$n],'%%' . $n . '%',$resultstr);

senão

$resultstr = str\_replace('%%' . $n . '%',$specialchars[$n],$resultstr);

}

return ($resultstr);

}

function pfz\_openvpn\_clientvalue($client\_id, $valuekey, $default="none"){

     $clientes = openvpn\_get\_active\_clients();

     foreach($clientes as $cliente) {

          if($client['vpnid']==$client\_id)

               $valor = $cliente[$valorchave];

     }

     switch ($valuekey){

          caso "estado":

               $valor = pfz\_valuemap("openvpn.client.status", $valor);

               pausa;

     }

     if ($valor=="") $valor=$default;

     echo $valor;

}

// Descoberta de serviços

// 2020-03-27: Adicionado espaço substituído por \_\_ para o problema nº 12

function pfz\_services\_discovery(){

     $serviços = get\_services();

     $json\_string = '{"data":[';

     foreach ($serviços como $serviço){

          if (!empty($service['name'])) {

               $status = get\_service\_status($service);

               if ($status="") $status = 0;

               $id="";

               //id para OpenVPN

               if (!empty($service['id'])) $id = "." . $serviço["id"];

               //zona para Portal Cativo

               if (!empty($service['zone'])) $id = "." . $service["zona"];

               $json\_string .= '{"{#SERVICE}":"' . str\_replace(" ", "\_\_", $service['name']) . $id . '"';

               $json\_string .= ',"{#DESCRIPTION}":"' . $service['description'] . '"';

               $json\_string .= '},';

          }

     }

     $json\_string = rtrim($json\_string,",");

     $json\_string .= "]}";

     echo $json\_string;

}

// Obtém o valor do serviço

// 2020-03-27: Adicionado substituição de espaço no nome do serviço para o problema nº 12

// 28-09-2020: Substituição de espaço corrigida

function pfz\_service\_value($nome,$valor){

     $serviços = get\_services();

     $nome = str\_replace("\_\_"," ",$nome);

     //Lista de serviços parados no CARP Slave.

     //Por enquanto esta é a melhor maneira que encontrei para filtrar os gatilhos

     //Aguardando uma forma no Zabbix de usar o Global Regexp em triggers com descoberta de itens

     $stopped\_on\_carp\_slave = array("haproxy","radvd","openvpn.","openvpn","avahi");

     foreach ($serviços como $serviço){

          $namecfr = $service["nome"];

          $carpcfr = $service["nome"];

          //Abrir VPN

          if (!empty($service['id'])) {

               $namecfr = $service['name'] . "." . $serviço["id"];

               $carpcfr = $service['name'] . ".";

          }

          //Portal cativo

          if (!empty($service['zone'])) {

               $namecfr = $service['name'] . "." . $service["zona"];

               $carpcfr = $service['name'] . ".";

          }

          if ($nomecfr == $nome){

               alternar ($valor) {

                    caso "estado":

                         $status = get\_service\_status($service);

                         if ($status=="") $status = 0;

                         echo $status;

                         pausa;

                    caso "nome":

                         echo $namecfr;

                         pausa;

                    caso "habilitado":

                         if (is\_service\_enabled($service['name']))

                              eco 1;

                         senão

                              eco 0;

                         pausa;

                    case "run\_on\_carp\_slave":

                         if (in\_array($carpcfr,$stopped\_on\_carp\_slave))

                              eco 0;

                         senão

                              eco 1;

                         pausa;

                    predefinição:

                         echo $serviço[$valor];

                         pausa;

               }

          }

    }

}

//Descoberta de gateway

function pfz\_gw\_rawstatus() {

     // Retorna um Raw Gateway Status, útil para scripts de ação (por exemplo, Update Cloudflare DNS config)

     $gws = return\_gateways\_status(true);

     $gw\_string="";

     foreach ($gws as $gw){

          $gw\_string .= ($gw['name'] . '.' . $gw['status'] .",");

     }

     echo rtrim($gw\_string,",");

}

function pfz\_gw\_discovery() {

     $gws = return\_gateways\_status(true);

     $json\_string = '{"data":[';

     foreach ($gws as $gw){

          $json\_string .= '{"{#GATEWAY}":"' . $gw['name'] . '"';

          $json\_string .= '},';

     }

     $json\_string = rtrim($json\_string,",");

     $json\_string .= "]}";

     echo $json\_string;

}

function pfz\_gw\_value($gw, $valuekey) {

     $gws = return\_gateways\_status(true);

     if(array\_key\_exists($gw,$gws)) {

          $valor = $gws[$gw][$valuekey];

          if ($valuekey=="status") {

               //Problema #70: Gateway forçado para baixo

               if ($gws[$gw]["substatus"]<>"nenhum")

                    $valor = $gws[$gw]["substatus"];

               $valor = pfz\_valuemap("gateway.status", $valor);

          }

          echo $valor;

     }

}

// Descoberta IPSEC

function pfz\_ipsec\_discovery\_ph1(){

require\_once("ipsec.inc");

global $configuração;

init\_config\_arr(array('ipsec', 'phase1'));

$a\_phase1 = &$config['ipsec']['phase1'];

$json\_string = '{"data":[';

foreach ($a\_phase1 como $dados) {

$json\_string .= '{"{#IKEID}":"' . $data['ikeid'] . '"';

$json\_string .= ',"{#NAME}":"' . $data['descr'] . '"';

$json\_string .= '},';

}

$json\_string = rtrim($json\_string,",");

    $json\_string .= "]}";

    echo $json\_string;

}

function pfz\_ipsec\_ph1($ikeid,$valuekey){

// Obter valor da configuração da fase 1 do IPsec

// Se Obtendo o valor "disabled" apenas verifique a presença do item no array de configuração

require\_once("ipsec.inc");

global $configuração;

init\_config\_arr(array('ipsec', 'phase1'));

$a\_phase1 = &$config['ipsec']['phase1'];

$valor = "";

switch ($valuekey) {

caso 'status':

$valor = pfz\_ipsec\_status($ikeid);

pausa;

caso 'desabilitado':

$valor = "0";

predefinição:

foreach ($a\_phase1 como $dados) {

if ($data['ikeid'] == $ikeid) {

if(array\_key\_exists($valuekey,$data)) {

if ($valuekey=='disabled')

$valor = "1";

senão

$value = pfz\_valuemap("ipsec." . $valuekey, $data[$valuekey], $data[$valuekey]);

pausa;

}

}

}

}

echo $valor;

}

function pfz\_ipsec\_discovery\_ph2(){

require\_once("ipsec.inc");

global $configuração;

init\_config\_arr(array('ipsec', 'phase2'));

$a\_phase2 = &$config['ipsec']['phase2'];

$json\_string = '{"data":[';

foreach ($a\_phase2 como $dados) {

$json\_string .= '{"{#IKEID}":"' . $data['ikeid'] . '"';

$json\_string .= ',"{#NAME}":"' . $data['descr'] . '"';

$json\_string .= ',"{#UNIQID}":"' . $data['uniqid'] . '"';

$json\_string .= ',"{#REQID}":"' . $data['reqid'] . '"';

$json\_string .= ',"{#EXTID}":"' . $data['ikeid'] . '.' . $data['reqid'] . '"';

$json\_string .= '},';

}

$json\_string = rtrim($json\_string,",");

    $json\_string .= "]}";

    echo $json\_string;

}

function pfz\_ipsec\_ph2($uniqid, $valuekey){

require\_once("ipsec.inc");

global $configuração;

init\_config\_arr(array('ipsec', 'phase2'));

$a\_phase2 = &$config['ipsec']['phase2'];

$valuecfr = explodir(".",$valuekey);

switch ($valuecfr[0]) {

caso 'status':

$idarr = explodir(".", $uniqid);

$statuskey = "estado";

if (isset($valuecfr[1])) $statuskey = $valuecfr[1];

$valor = pfz\_ipsec\_status($idarr[0],$idarr[1],$statuskey);

pausa;

caso 'desabilitado':

$valor = "0";

}

foreach ($a\_phase2 como $dados) {

if ($data['uniqid'] == $uniqid) {

if(array\_key\_exists($valuekey,$data)) {

if ($valuekey=='disabled')

$valor = "1";

senão

$value = pfz\_valuemap("ipsec\_ph2." . $valuekey, $data[$valuekey], $data[$valuekey]);

pausa;

}

}

}

echo $valor;

}

function pfz\_ipsec\_status($ikeid,$reqid=-1,$valuekey='estado'){

require\_once("ipsec.inc");

global $configuração;

init\_config\_arr(array('ipsec', 'phase1'));

$a\_phase1 = &$config['ipsec']['phase1'];

$conmap = array();

foreach ($a\_phase1 como $ph1ent) {

if (get\_ipsecifnum($ph1ent['ikeid'], 0)) {

$cname = "con" . get\_ipsecifnum($ph1ent['ikeid'], 0);

} senão {

$cname = "con{$ph1ent['ikeid']}00000";

}

$conmap[$cname] = $ph1ent['ikeid'];

}

$status = ipsec\_list\_sa();

$ipsecconnected = array();

$carp\_status = pfz\_carp\_status(false);

//Correspondência de fase-status emprestada de status\_ipsec.php

if (is\_array($status)) {

foreach ($status as $l\_ikeid=>$ikesa) {

if (isset($ikesa['con-id'])) {

$con\_id = substr($ikesa['con-id'], 3);

} senão {

$con\_id = filter\_var($ikeid, FILTER\_SANITIZE\_NUMBER\_INT);

}

$con\_name = "con" . $con\_id;

if ($ikesa['versão'] == 1) {

$ph1idx = $conmap[$con\_name];

$ipsecconnected[$ph1idx] = $ph1idx;

} senão {

if (!ipsec\_ikeid\_used($con\_id)) {

// provavelmente um v2 com conexão dividida então

$ph1idx = $conmap[$con\_name];

$ipsecconnected[$ph1idx] = $ph1idx;

} senão {

$ipsecconnected[$con\_id] = $ph1idx = $con\_id;

}

}

if ($ph1idx == $ikeid){

if ($reqid!=-1) {

// Solicitando o valor do status da fase 2

foreach ($ikesa['child-sas'] as $childsas) {

if ($childsas['reqid']==$reqid) {

if (strtolower($childsas['state']) == 'rekeyed') {

//se o estado for redigitado continue

$tmp\_value = $childsas[$valuekey];

} senão {

$tmp\_value = $childsas[$valuekey];

pausa;

}

}

}

} senão {

$tmp\_value = $ikesa[$valuekey];

}

pausa;

}

}

}

switch($valuekey) {

caso 'estado':

$valor = pfz\_valuemap('ipsec.state', strtolower($tmp\_value));

if ($carp\_status!=0) $valor = $valor + (10 \* ($carp\_status-1));

pausa;

predefinição:

$valor = $tmp\_value;

pausa;

}

retorna $valor;

}

function pfz\_carp\_status($echo = true){

     //Detectar Status de CARP

     global $configuração;

     $status\_return = 0;

     $status = get\_carp\_status();

     $carp\_detected\_problems = get\_single\_sysctl("net.inet.carp.demotion");

//CARP está desabilitado

$ret = 0;

     if ($status != 0) { //CARP está habilitado

          if ($carp\_detected\_problems != 0) {

  //Há alguns grandes problemas com CARP

               $ret = 4;

               if ($echo == true) echo $ret;

               retorno $ret;

          }

          $status\_changed = false;

          $prev\_status = "";

          foreach ($config['virtualip']['vip'] as $carp) {

    if ($carpa['modo'] != "carpa") {

    Prosseguir;

    }

               $if\_status = get\_carp\_interface\_status("\_vip{$carp['uniqid']}");

               if ( ($prev\_status != $if\_status) && (empty($if\_status)==false) ) { //Algumas falhas na GUI

                    if ($prev\_status!="") $status\_changed = true;

                    $prev\_status = $if\_status;

               }

          }

          if ($status\_changed) {

               //O status CARP é inconsistente nas interfaces

               $ret=3;

               eco 3;

          } senão {

               if ($prev\_status=="MASTER")

                    $ret = 1;

               senão

$ret = 2;

          }

     }

     if ($echo == true) echo $ret;

     retorno $ret;

}

// Verificações DHCP (cópia de status\_dhcp\_leases.php, aguardando pfsense 2.5)

function pfz\_remove\_duplicate($array, $campo) {

foreach ($array como $sub) {

$cmp[] = $sub[$campo];

}

$único = array\_unique(array\_reverse($cmp, true));

foreach ($único como $k => $rien) {

$novo[] = $array[$k];

}

retorna $novo;

}

// Obter matrizes DHCP (copiadas de status\_dhcp\_leases.php, aguardando pfsense 2.5, para usar system\_get\_dhcpleases();)

function pfz\_dhcp\_get($valuekey) {

require\_once("config.inc");

$leasesfile = "{$g['dhcpd\_chroot\_path']}/var/db/dhcpd.leases";

$awk = "/usr/bin/awk";

/\* este padrão coloca comentários em um único item de array \*/

$cleanpattern = "'{ gsub(\"#.\*\", \"\");} { gsub(\";\", \"\"); print;}'";

/\* Em seguida, dividimos o arquivo de concessões por } \*/

$splitpattern = "'BEGIN { RS=\"}\";} {for (i=1; i<=NF; i++) printf \"%s \", \$i; printf \"}\\n\ ";}'";

/\* enche o arquivo de concessões em um formato adequado em um array por linha \*/

@exec("/bin/cat {$leasesfile} 2>/dev/null| {$awk} {$cleanpattern} | {$awk} {$splitpattern}", $leases\_content);

$leases\_count = count($leases\_content);

@exec("/usr/sbin/arp -an", $rawdata);

foreach ($leases\_content as $lease) {

/\* divide a linha por espaço \*/

$data = explodir(" ", $lease);

/\* anda pelos campos \*/

$f = 0;

$fconta = contagem($dados);

/\* com menos de 20 campos não há nada útil \*/

if ($fconta < 20) {

$i++;

Prosseguir;

}

while ($f < $fconta) {

switch ($dados[$f]) {

caso "failover":

$pools[$p]['name'] = trim($data[$f+2], '"');

$pools[$p]['name'] = "{$pools[$p]['name']} (" . convert\_friendly\_interface\_to\_friendly\_descr(substr($pools[$p]['name'], 5)) . " )";

$pools[$p]['mystate'] = $data[$f+7];

$pools[$p]['peerstate'] = $data[$f+14];

$pools[$p]['mydate'] = $data[$f+10];

$pools[$p]['mydate'] .= " " . $dados[$f+11];

$pools[$p]['peerdate'] = $data[$f+17];

$pools[$p]['peerdate'] .= " " . $dados[$f+18];

$p++;

$i++;

continuar 3;

caso "arrendamento":

$leases[$l]['ip'] = $data[$f+1];

$leases[$l]['type'] = $dynamic\_string;

$f = $f+2;

pausa;

caso "inicia":

$leases[$l]['start'] = $data[$f+2];

$leases[$l]['start'] .= " " . $dados[$f+3];

$f = $f+3;

pausa;

caso "termina":

if ($dados[$f+1] == "nunca") {

// Citação da página man dhcpd.leases(5):

// Se uma concessão nunca expirar, a data nunca será em vez de uma data real.

$leases[$l]['end'] = gettext("Nunca");

$f = $f+1;

} senão {

$leases[$l]['end'] = $data[$f+2];

$leases[$l]['end'] .= " " . $dados[$f+3];

$f = $f+3;

}

pausa;

caso "tstp":

$f = $f+3;

pausa;

caso "tsfp":

$f = $f+3;

pausa;

caso "atsfp":

$f = $f+3;

pausa;

caso "clt":

$f = $f+3;

pausa;

caso "ligação":

alternar ($dados[$f+2]) {

caso "ativo":

$leases[$l]['act'] = $active\_string;

pausa;

caso "livre":

$leases[$l]['act'] = $expired\_string;

$leases[$l]['online'] = $offline\_string;

pausa;

caso "backup":

$leases[$l]['act'] = $reserved\_string;

$leases[$l]['online'] = $offline\_string;

pausa;

}

$f = $f+1;

pausa;

caso "próximo":

/\* pula a próxima instrução de ligação \*/

$f = $f+3;

pausa;

caso "rebobinar":

/\* pula a declaração de vinculação de retrocesso \*/

$f = $f+3;

pausa;

caso "hardware":

$leases[$l]['mac'] = $data[$f+2];

/\* verifica se está online e o lease está ativo \*/

if (in\_array($leases[$l]['ip'], $arpdata\_ip)) {

$leases[$l]['online'] = $online\_string;

} senão {

$leases[$l]['online'] = $offline\_string;

}

$f = $f+2;

pausa;

case "client-hostname":

if ($dados[$f+1] <> "") {

$leases[$l]['hostname'] = preg\_replace('/"/', '', $data[$f+1]);

} senão {

$hostname = gethostbyaddr($leases[$l]['ip']);

if ($hostname <> "") {

$leases[$l]['hostname'] = $hostname;

}

}

$f = $f+1;

pausa;

caso "uid":

$f = $f+1;

pausa;

}

$f++;

}

$l++;

$i++;

/\* lentamente esculpindo o array de origem \*/

array\_shift($leases\_content);

}

/\* remove itens duplicados pelo endereço mac \*/

if (count($leases) > 0) {

$locações = pfz\_remove\_duplicate($locações, "ip");

}

if (count($pools) > 0) {

$pools = pfz\_remove\_duplicate($pools, "nome");

sort($pools);

}

switch ($valuekey){

caso "piscinas":

retornar $ pools;

pausa;

caso "failover":

return $failover;

pausa;

caso "aluguéis":

predefinição:

retornar $ arrendamentos;

}

}

function pfz\_dhcpfailover\_discovery(){

//As funções do sistema relacionadas a concessões de DHCP estarão disponíveis na próxima versão do pfSense, então vamos esperar

require\_once("system.inc");

$leases = system\_get\_dhcpleases();

$json\_string = '{"data":[';

if (count($leases['failover']) > 0){

foreach ($leases['failover'] as $data){

  $json\_string .= '{"{#FAILOVER\_GROUP}":"' . str\_replace(" ", "\_\_", $data['name']) . '"';

}

}

$json\_string = rtrim($json\_string,",");

    $json\_string .= "]}";

    echo $json\_string;

}

function pfz\_dhcp\_check\_failover(){

// Verifica o status de failover do DHCP

// Retorna o número de pools de failover cujo estado não é normal ou

// diferente do estado do par

$failover = pfz\_dhcp\_get("failover");

$ret = 0;

foreach ($failover como $f){

if ( ($f["mystate"]!="normal") || ($f["mystate"]!=$f["peerstate"])) {

$ret++;

}

}

retorno $ret;

}

function pfz\_dhcp($seção, $valuekey=""){

switch ($seção){

caso "failover":

echo pfz\_dhcp\_check\_failover();

pausa;

predefinição:

}

}

//Pacotes

function pfz\_packages\_uptodate(){

require\_once("pkg-utils.inc");

$installed\_packages = get\_pkg\_info('all', false, true);

$ret = 0;

foreach ($installed\_packages as $package){

if ($package['version']!=$package['installed\_version']){

$ret++;

}

}

retorno $ret;

}

//Informação do sistema

function pfz\_get\_system\_value($seção){

     switch ($seção){

          caso "versão":

               echo( get\_system\_pkg\_version()['versão']);

               pausa;

          case "versão\_instalada":

               echo( get\_system\_pkg\_version()['installed\_version']);

               pausa;

          case "new\_version\_available":

               $pkgver = get\_system\_pkg\_version();

               if ($pkgver['version']==$pkgver['installed\_version'])

                    echo "0";

               senão

                    echo "1";

               pausa;

          case "packages\_update":

           echo pfz\_packages\_uptodate();

           pausa;

     }

}

//Estado SMART

// Retirado de /usr/local/www/widgets/widgets/smart\_status.widget.php

function pfz\_get\_smart\_status(){

$devs = get\_smart\_drive\_list();

$status = 0;

foreach ($devs as $dev) { ## para cada unidade encontrada do

                $smartdrive\_is\_displayed = true;

                $dev\_ident = exec("diskinfo -v /dev/$dev | grep ident | awk '{print $1}'"); ## obtém identificador da unidade

                $dev\_state = trim(exec("smartctl -H /dev/$dev | awk -F: '/^Resultado do teste de autoavaliação de saúde geral SMART/ {print $2;exit}

/^SMART Health Status/ {print $2;exit}'")); ## obtém o estado SMART da unidade

                switch ($dev\_state) {

                        caso "APROVADO":

                        caso "OK":

                                //OK

                                $status=0;

                                pausa;

                        caso "":

                                //Desconhecido

                                $status=2;

                                retornar $status;

                                pausa;

                        predefinição:

                         //Erro

                                $status=1;

                                retornar $status;

                                pausa;

                }

}

echo $status;

}

// Data de validade dos certificados

function pfz\_get\_cert\_date($valuekey){

    global $configuração;

    $valor = 0;

foreach (array("cert", "ca") as $cert\_type) {

switch ($valuekey){

case "validFrom.max":

foreach ($config[$cert\_type] as $cert) {

$certinfo = openssl\_x509\_parse(base64\_decode($cert[crt]));

if ($value == 0 ou $value < $certinfo['validFrom\_time\_t']) $value = $certinfo['validFrom\_time\_t'];

            }

pausa;

case "validTo.min":

foreach ($config[$cert\_type] as $cert) {

$certinfo = openssl\_x509\_parse(base64\_decode($cert[crt]));

if ($value == 0 ou $value > $certinfo['validTo\_time\_t']) $value = $certinfo['validTo\_time\_t'];

}

pausa;

}

}

echo $valor;

}

// O arquivo está presente

function pfz\_file\_exists($filename) {

if (file\_exists($filename))

echo "1";

senão

echo "0";

}

// Mapeamentos de valor

// Cada mapa de valores é representado por um array associativo

function pfz\_valuemap($valuename, $value, $default="0"){

     switch ($valuename){

          case "openvpn.server.status":

                    $ valor mapa = array(

                         "baixo" => "0",

                         "para cima" => "1",

                         "nenhum" => "2",

                         "reconectando; ping-restart" => "3",

                         "esperando" => "4",

                         "server\_user\_listening" => "5");

          pausa;

          case "openvpn.client.status":

                    $ valor mapa = array(

                         "para cima" => "1",

                         "baixo" => "0",

                         "nenhum" => "0",

                         "reconectando; ping-restart" => "2");

          pausa;

          case "openvpn.server.mode":

                    $ valor mapa = array(

                         "p2p\_tls" => "1",

                         "p2p\_shared\_key" => "2",

                         "server\_tls" => "3",

                         "server\_user" => "4",

                         "server\_tls\_user" => "5");

          pausa;

          case "gateway.status":

                    $ valor mapa = array(

                         "online" => "0",

                         "nenhum" => "0",

                         "perda" => "1",

                         "atraso alto" => "2",

                         "alta perda" => "3",

                         "force\_down" => "4",

                         "baixo" => "5");

          pausa;

          case "ipsec.iketype":

           $ valor mapa = array (

           "auto" => 0,

           "ikev1" => 1,

           "ikev2" => 2);

          pausa;

          case "ipsec.mode":

           $ valor mapa = array (

           "principal" => 0,

           "agressivo" => 1);

          pausa;

          case "ipsec.protocol":

           $ valor mapa = array (

           "ambos" => 0,

           "inet" => 1,

           "inet6" => 2);

          pausa;

          case "ipsec\_ph2.mode":

           $ valor mapa = array (

           "transporte" => 0,

           "túnel" => 1,

           "túnel6" => 2);

          pausa;

          case "ipsec\_ph2.protocol":

           $ valor mapa = array (

           "esp" => 1,

           "ah" => 2);

          pausa;

  case "ipsec.state":

           $ valor mapa = array (

           "estabelecido" => 1,

           "conectando" => 2,

           "instalado" => 1,

           "redigitado" => 2);

          pausa;

     }

     if (is\_array($valuemap)) {

      $valor = strtolower($valor);

      if (array\_key\_exists($value, $valuemap))

           return $valuemap[$value];

     }

     retorna $ padrão;

}

// Analisadores de argumentos para Discovery

function pfz\_discovery($seção){

     switch (strtolower($seção)){

          caso "gw":

               pfz\_gw\_discovery();

               pausa;

          caso "wan":

             pfz\_interface\_discovery(true);

               pausa;

          case "openvpn\_server":

               pfz\_openvpn\_serverdiscovery();

               pausa;

          case "openvpn\_server\_user":

               pfz\_openvpn\_server\_userdiscovery();

               pausa;

          case "openvpn\_client":

               pfz\_openvpn\_clientdiscovery();

               pausa;

          caso "serviços":

               pfz\_services\_discovery();

               pausa;

          caso "interfaces":

               pfz\_interface\_discovery();

               pausa;

          case "ipsec\_ph1":

             pfz\_ipsec\_discovery\_ph1();

               pausa;

          case "ipsec\_ph2":

             pfz\_ipsec\_discovery\_ph2();

               pausa;

          case "dhcpfailover":

             pfz\_dhcpfailover\_discovery();

               pausa;

     }

}

//Código principal

switch (strtolower($argv[1])){

     caso "descoberta":

          pfz\_discovery($argv[2]);

          pausa;

     case "gw\_value":

          pfz\_gw\_value($argv[2],$argv[3]);

          pausa;

     case "gw\_status":

          pfz\_gw\_rawstatus();

          pausa;

case "if\_speedtest\_value":

      pfz\_speedtest\_cron\_install();

  pfz\_interface\_speedtest\_value($argv[2],$argv[3]);

  pausa;

     case "openvpn\_servervalue":

          pfz\_openvpn\_servervalue($argv[2],$argv[3]);

          pausa;

     case "openvpn\_server\_uservalue":

          pfz\_openvpn\_server\_uservalue($argv[2],$argv[3]);

          pausa;

     case "openvpn\_server\_uservalue\_numeric":

          pfz\_openvpn\_server\_uservalue($argv[2],$argv[3],,"0");

          pausa;

     case "openvpn\_clientvalue":

          pfz\_openvpn\_clientvalue($argv[2],$argv[3]);

          pausa;

     case "valor\_serviço":

          pfz\_service\_value($argv[2],$argv[3]);

          pausa;

     case "carp\_status":

          pfz\_carp\_status();

          pausa;

     case "if\_name":

          pfz\_get\_if\_name($argv[2]);

          pausa;

     caso "sistema":

          pfz\_get\_system\_value($argv[2]);

          pausa;

     case "ipsec\_ph1":

          pfz\_ipsec\_ph1($argv[2],$argv[3]);

          pausa;

     case "ipsec\_ph2":

          pfz\_ipsec\_ph2($argv[2],$argv[3]);

          pausa;

     caso "dhcp":

        pfz\_dhcp($argv[2],$argv[3]);

          pausa;

     case "arquivo\_existe":

        pfz\_file\_exists($argv[2]);

        pausa;

     case "speedtest\_cron":

        pfz\_speedtest\_cron\_install();

        pfz\_speedtest\_cron();

        pausa;

     case "cron\_cleanup":

        pfz\_speedtest\_cron\_install(false);

        pausa;

     case "smart\_status":

          pfz\_get\_smart\_status();

          pausa;

     case "cert\_date":

          pfz\_get\_cert\_date($argv[2]);

          pausa;

     predefinição:

          pfz\_test();

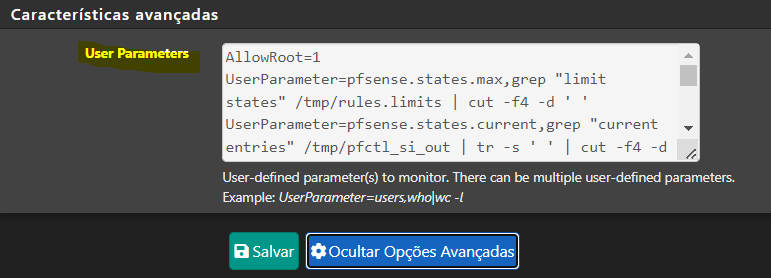
}

**Salvar**

**6-**

****

**7-**

****

**Colar Parameters**

AllowRoot=1

UserParameter=pfsense.states.max,grep "limit states" /tmp/rules.limits | cut -f4 -d ' '

UserParameter=pfsense.states.current,grep "current entries" /tmp/pfctl\_si\_out | tr -s ' ' | cut -f4 -d ' '

UserParameter=pfsense.mbuf.current,netstat -m | grep "mbuf clusters" | cut -f1 -d ' ' | cut -d '/' -f1

UserParameter=pfsense.mbuf.cache,netstat -m | grep "mbuf clusters" | cut -f1 -d ' ' | cut -d '/' -f2

UserParameter=pfsense.mbuf.max,netstat -m | grep "mbuf clusters" | cut -f1 -d ' ' | cut -d '/' -f4

UserParameter=pfsense.discovery[\*],/usr/local/bin/php /root/scripts/pfsense\_zbx.php discovery $1

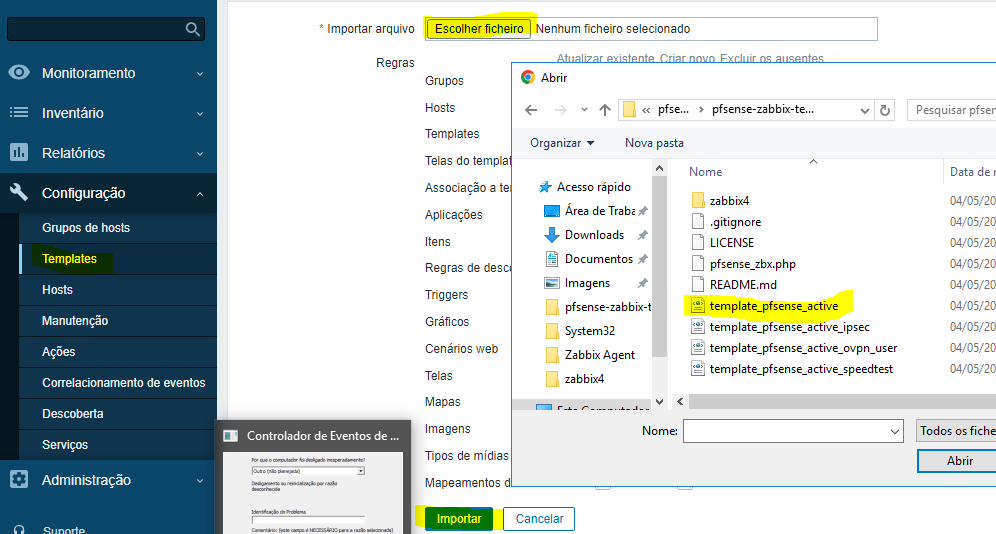
UserParameter=pfsense.value[\*],/usr/local/bin/php /root/scripts/pfsense\_zbx.php $1 $2 $3

**Salvar**

**8-**

**Baixar Arquivo**

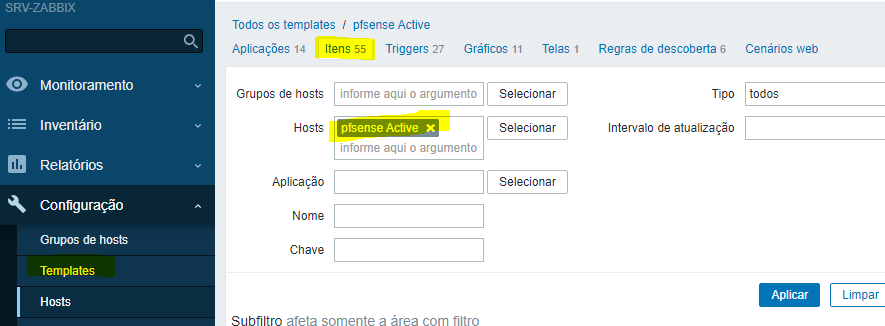
<https://github.com/rbicelli/pfsense-zabbix-template/archive/refs/heads/master.zip>

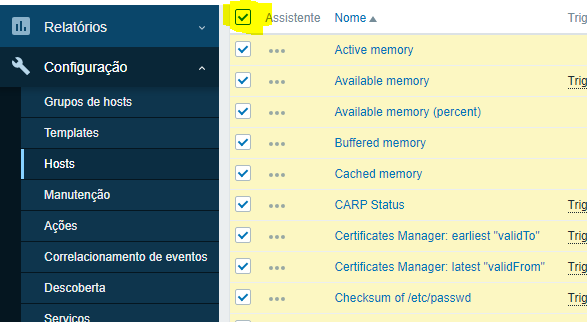


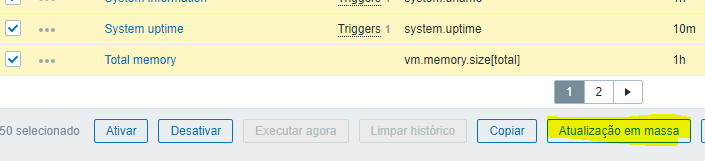
**Extrair arquivo e importar template**

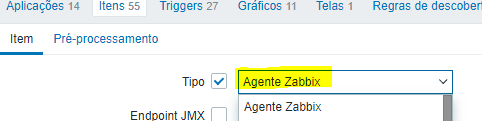
**9-**

**Ir ao Template importado e alterar o tipo para agente zabbix**

****

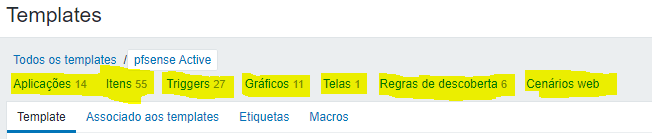
****

****

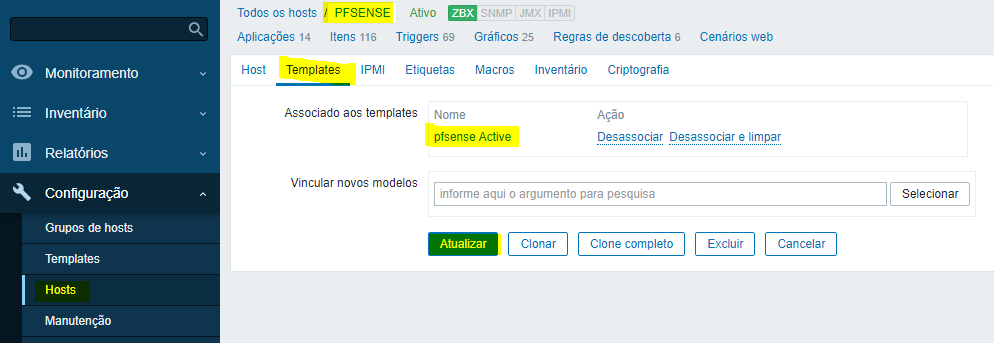
****

**Salvar**

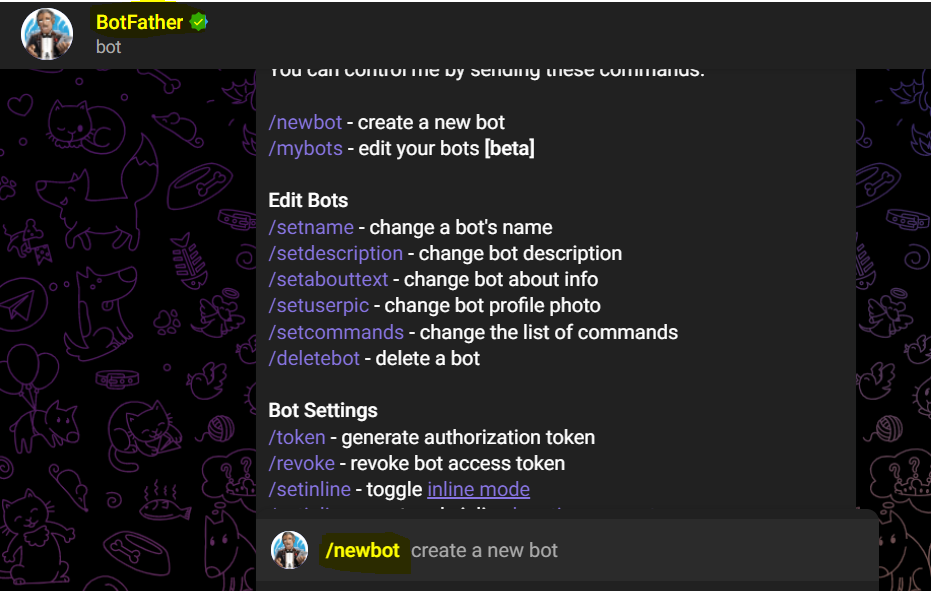
**Fazer o mesmo procedimento com os outros**

****

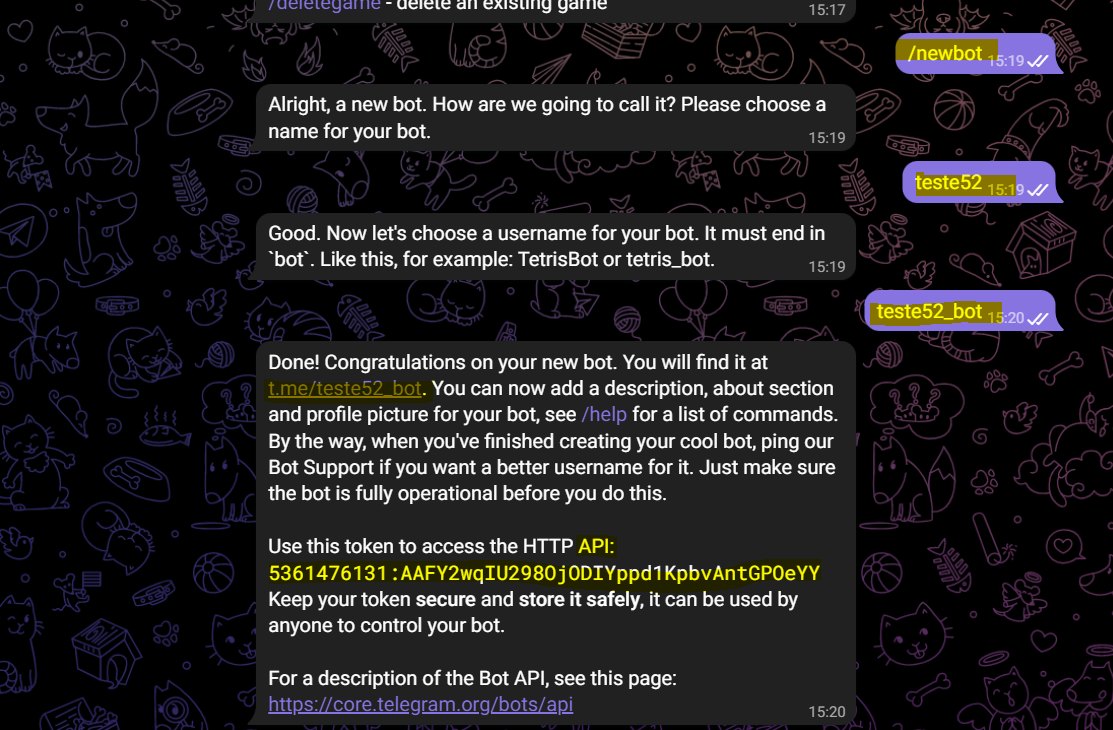
**10- Criar Host e associar ao template**

****

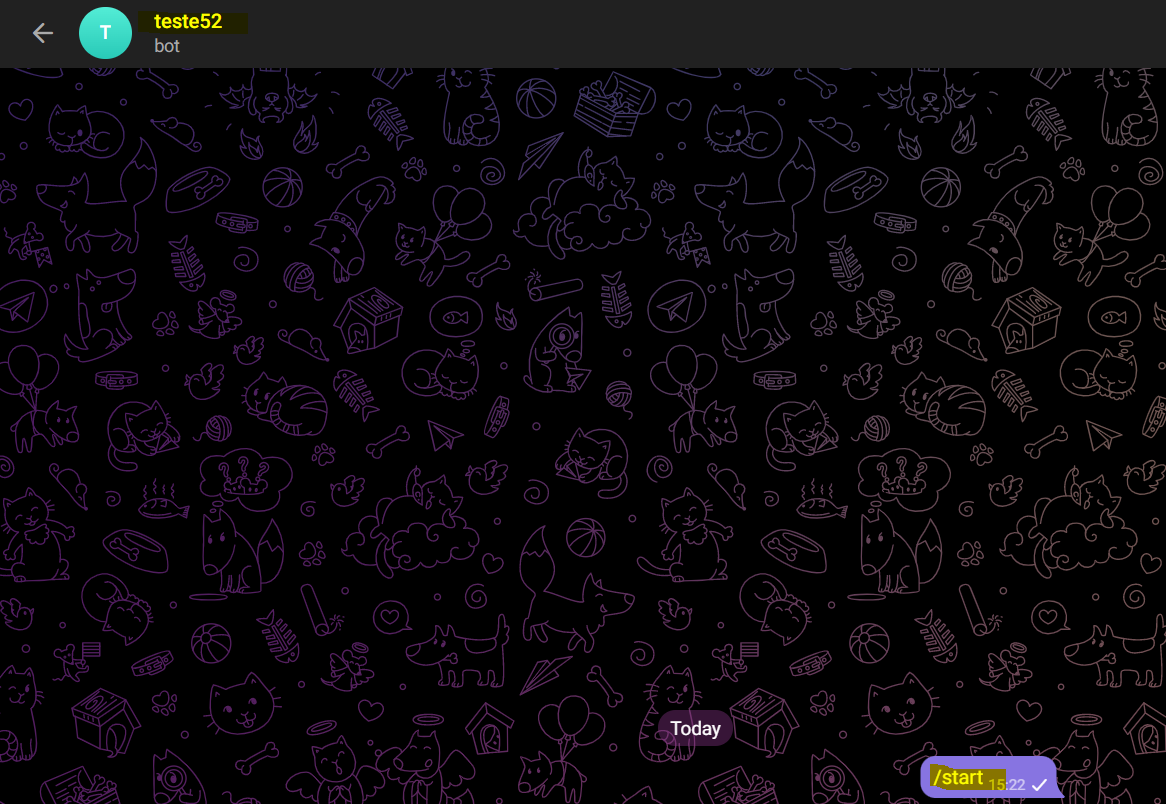
**11-  Criar bot no telegram**

****

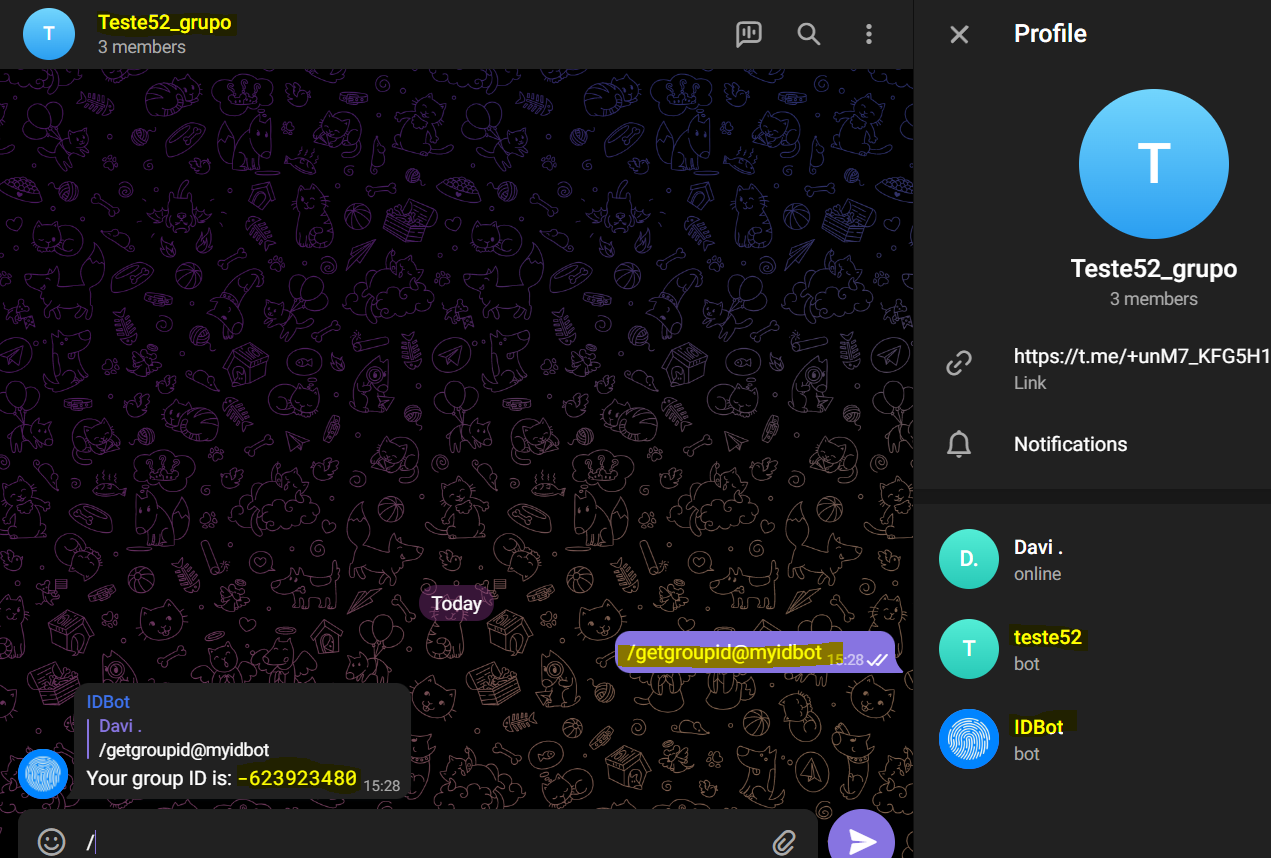
**12-  Colocar nome no bot e copiar API**

****

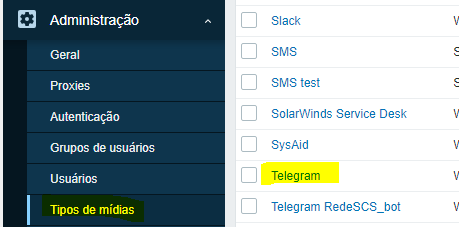
**13-  Acessar o bot**

****

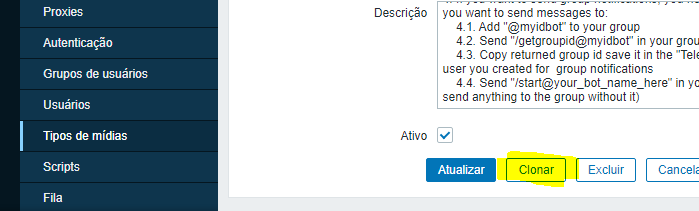
**14-  Criar um grupo no telegram e adicionar o bot criado e o IDbot, classificar ambos como administrador do grupo.**

****

**15-  Executar o comando no grupo /getgroupid e copiar o id do grupo**

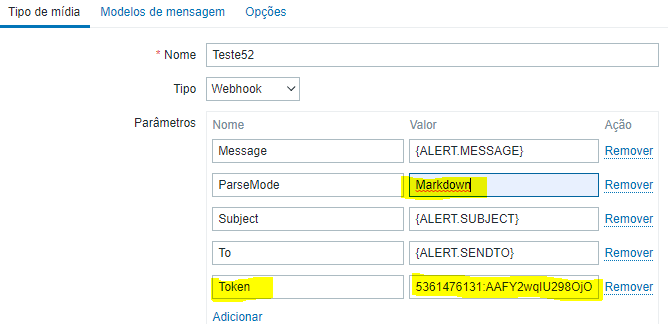
****

**16-**

****

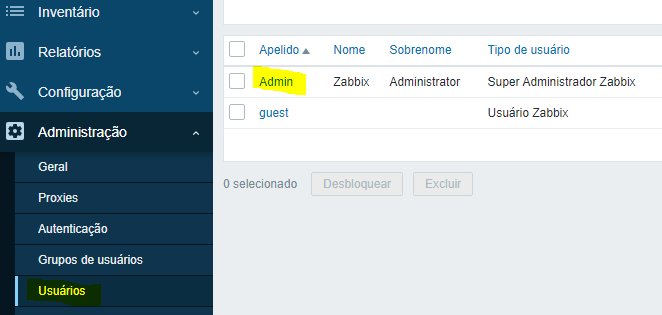
**Clonar**

**17- Colar Token do bot criado**

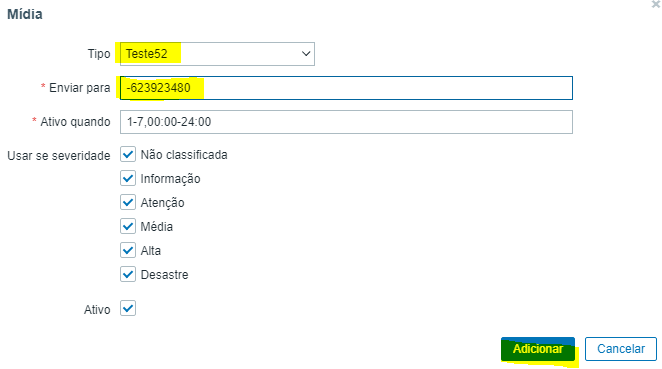
****

**Salvar**

**18-**

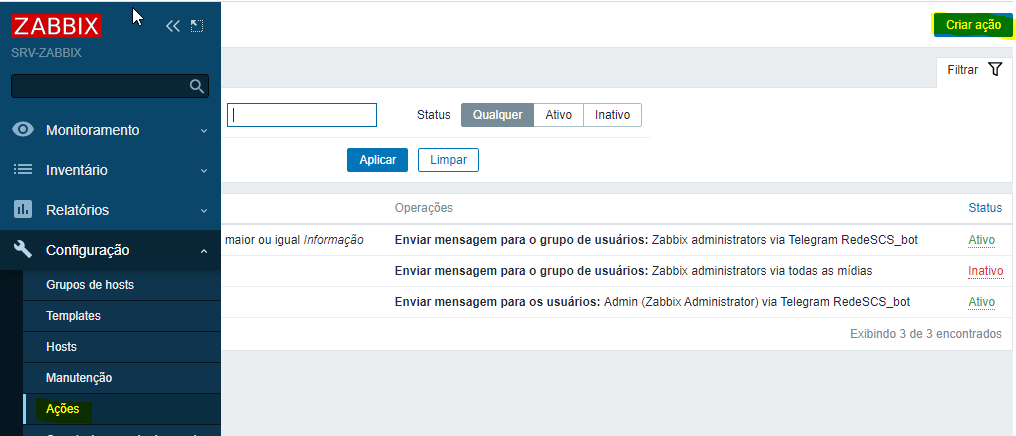
****

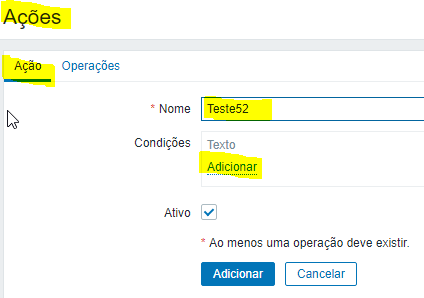
****

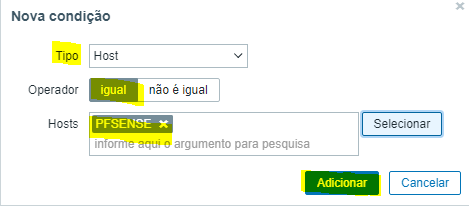
****

**Selecionar Midia criada, colar id do grupo do telegram e salvar**

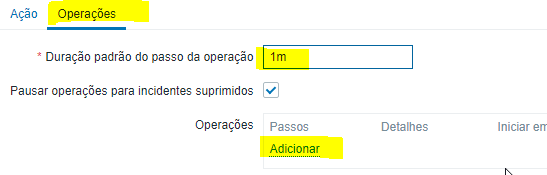
**19-**

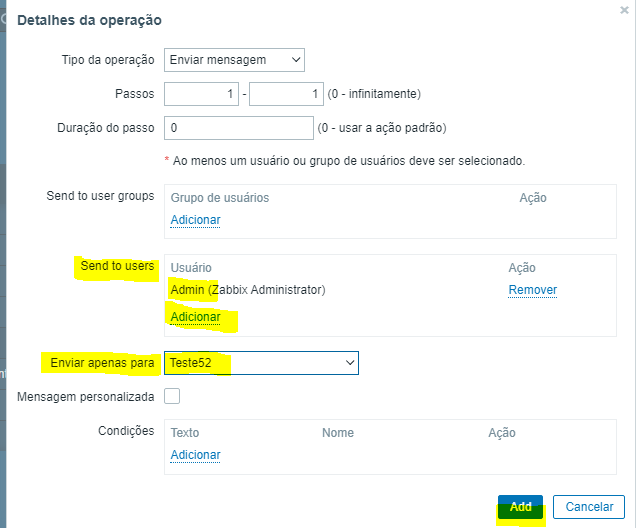
****

****

****

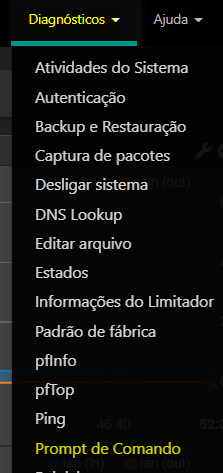
**Selecionar o host monitorado**

****

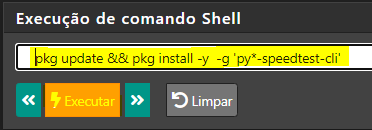
****

**Salvar**

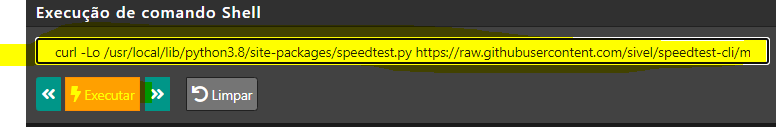
**20- Voltando ao Pfsense**

****

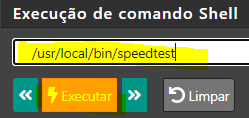
**Executar sequência de comandos**

****

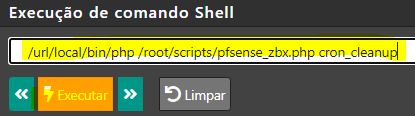
**pkg update && pkg install -y  -g 'py\*-speedtest-cli'**

****

**curl -Lo /usr/local/lib/python3.8/site-packages/speedtest.py** [**https://raw.githubusercontent.com/sivel/speedtest-cli/master/speedtest.py**](https://raw.githubusercontent.com/sivel/speedtest-cli/master/speedtest.py)

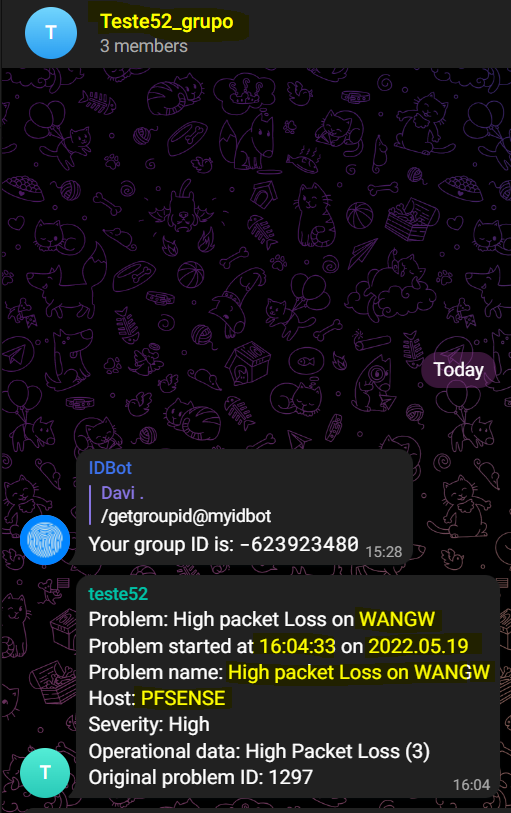
****

**/usr/local/bin/speedtest**

****

**/url/local/bin/php /root/scripts/pfsense\_zbx.php cron\_cleanup**

**Fazer o teste**

****