Supervision répartie sur plusieurs sites avec Icinga

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 16 mai 2020

https://www.bortzmeyer.org/icinga-distributed.html

De nos jours, quand on est un ou une ingénieur système responsable d'un service Internet un peu sérieux, on a une supervision automatique du service, pour être prévenu des pannes avant de se faire engueuler sur Twitter. C'est classique et cela va se soi, d'autant plus qu'il existe de très nombreux logiciels libres pour cette tâche (j'utilise Icinga). Mais, souvent, les tests sont faits depuis un seul point, en général dans les locaux de l'organisation qui gère le service. C'est insuffisant car, l'Internet étant ce qu'il est, les problèmes peuvent dépendre du point de mesure. Il existe plusieurs solutions pour faire de la supervision répartie sur plusieurs points, je vais parler ici du système maîtres/satellites/agents d'Icinga.

(Si vous voulez des idées d'autres solutions, j'en ai parlé aux JRES <https://www.jres.org/>, voici la vidéo <https://replay.jres.org/videos/watch/4c826f2c-3ff6-430f-aa3b-b2a8c795c18a> et l'article <https://conf-ng.jres.org/2019/document_revision_4815.html?download>.)

Pour illustrer cette histoire de supervision depuis plusieurs points, je prendrai comme exemple le système de répartition d'Icinga. Ce système est souvent présenté uniquement comme un moyen de répartir la charge entre plusieurs machines. Mais il a une bien plus grande utilité : permettre de voir la connectivité ou les données depuis plusieurs points de l'Internet. En effet, bien des choses dépendent d'où on est dans l'Internet. Un problème de routage ne va frapper que certains AS. Un « trou noir » annoncé à tort ne va capter qu'une partie du trafic. L'"*anycast*" va vous diriger vers un serveur différent (et potentiellement à problèmes) selon votre point de départ. Pour toutes ces raisons, il est crucial aujourd'hui d'avoir une supervision largement répartie. (Une autre utilité est la possibilité de superviser des choses purement locales à la machine, comme l'occupation de l'espace disque, comme on faisait autrefois avec NRPE.)

Commençons par la documentation d'Icinga <https://www.icinga.com/docs/icinga2/latest/doc/06-distributed-monitoring/>, très détaillée. Elle n'est pas d'un abord facile, car il existe plusieurs façons de configurer sa supervision répartie. Personnellement, j'ai choisi un mode simple, avec une seule machine maîtresse ("*Master*"), plusieurs machines agentes ("*Agent*"), et pas de satellite. J'ai configuré de façon à ce que ce soit toujours le maître qui contacte les agents (ce qui simplifie les configurations des pare-feux), mais Icinga permet également la connexion en sens opposé (ce qui peut être

utile si c'est le maître qui est un serveur public, et l'agent qui est coincé derrière du NAT), voire dans les deux sens. Et j'ai choisi de centraliser la configuration sur le maître, les agents n'ont aucune initiative, ils ne font qu'exécuter ce qu'a décidé le maitre. (Mais rappelez-vous qu'Icinga offre de nombreuses autres possibilités, et c'est une des raisons pour lesquelles la documentation est assez touffue.)

Il faut installer Icinga sur chaque machine, maître ou agent. Il n'y a pas de logiciels séparés pour les deux rôles (contrairement avec ce qui se fait chez, par exemple, Zabbix), c'est le même Icinga.

La configuration initiale peut se faire très simplement avec le "*Icinga 2 Setup Wizard*" (je donne un exemple plus loin) mais je vais d'abord montrer comment le faire à la main. Un petit mot sur la sécurité, d'abord : la communication entre les machines Icinga est protégée par TLS et il nous faut donc un certificat valable sur chaque machine. Comme certaines ne sont pas visibles depuis l'Internet, utiliser Let's Encrypt n'aurait pas forcément été pratique (oui, il y a des solutions, je sais, mais que je trouvais compliquées, dans mon cas). J'ai donc choisi de faire gérer tous les certificats par CAcert <https://www.bortzmeyer.org/cacert.html>. Comme il ne s'agit que de machines que je contrôle, l'obligation d'ajouter le certificat de CAcert au magasin d'AC n'est pas un problème.

Une note sur le nom des fichiers de configuration. Icinga permet d'inclure depuis le fichier de configuration principal d'autres fichiers, et cela peut même être récursif (`include_recursive "conf.d"`). Vous pouvez donc choisir l'organisation que vous voulez pour vos fichiers de configuration. Ici, j'ai choisi celle qu'a par défaut le paquetage Debian. Quant aux noms de domaine, j'ai tout mis sous `foobar.example`, la zone du maître étant `icinga-master.foobar.example`.

Bon, d'abord la configuration du maître. Il faut un fichier de configuration `zones.conf` où on déclare les agents à contacter (rappelez-vous que j'ai choisi une configuration où c'est toujours le maître qui contacte l'agent) :

```
const NodeName = "icinga-master.foobar.example"

const ZoneName = "icinga-master.foobar.example"

// Dans le cas le plus simple, il y a une zone par machine.
object Zone ZoneName {
endpoints = [ NodeName ]
}

// La machine maîtresse
object Endpoint NodeName {
}

// Un des agents
object Endpoint "bellovese.foobar.example" {
  host = "bellovese.foobar.example" // Nécessaire pour pouvoir le
  // contacter, ça pourrait être une adresse IP.
}

// L'agent fait confiance à la machine maîtresse (la directive "parent")
object Zone "bellovese.foobar.example" {
  endpoints = [ "bellovese.foobar.example" ]
  parent = ZoneName
}

// La configuration qui sera poussée vers les agents
object Zone "global-templates" {
  global = true
}
```

On crée ensuite un répertoire `zones.d` où on va placer ce qui concerne la configuration des agents. Il y aura les gabarits communs à tous les agents. Ici, la partie qui permettra de faire des ping à distance :

```
% cat zones.d/global-templates/templates.conf
template Host "remote-host" {
     vars.ping_wrta = 300
     vars.ping_crta = 600
     ...
}
...
```

Avec cette configuration, qui sera poussée vers les agents, tous testeront avec les mêmes paramètres. Autre cas dans ce fichier des gabarits communs, la supervision du démon OpenDNSSEC, qui n'est pas accessible de l'extérieur :

```
% cat zones.d/global-templates/templates.conf
...
object CheckCommand "opendnssec" {
  command = [ PluginContribDir + "/check_opendnssec" ]

  arguments = {
      "-e" = "$opendnssec_enforcer_expect$",
      "-s" = "$opendnssec_signer_expect$"
  }
}

apply Service "remote-opendnssec" {
  import "generic-service"

  check_command = "opendnssec"
  assign where host.vars.opendnssec
}
```

Avec cela, toute machine où la variable `opendnssec` a été définie sera testée (on va vérifier que le démon tourne bien).

Ça, c'était la configuration globale, commune à tous les agents. Ensuite, pour chaque agent, on va mettre une configuration spécifique, sous, par exemple, `zones.d/nom-de-l-agent.conf`, avec des choses comme :

```
object Host "test-bellovese" {
  import "remote-host"
  address = "bellovese.foobar.example"
  address6 = "bellovese.foobar.example"
}
```

qui fera que l'agent testera la machine `bellovese.foobar.example` avec le gabarit `remote-host` commun.

Il reste à créer un certificat :

```
[Choisir éventuellement des options OpenSSL]
%     openssl req  -new -nodes
[Répondre aux questions d'OpenSSL, notamment :]
Server name []:icinga-master.foobar.example
```

et à le faire signer par l'AC. Puis on concatène les deux certificats de l'AC :

```
#    cat /usr/share/ca-certificates/CAcert/root.crt /usr/share/ca-certificates/CAcert/class3.crt > /var/lib/
```

Et on met les certificats en `/var/lib/icinga2/certs`, aussi bien celui de la machine maître que celui de l'AC (ici, CAcert) ;

```
# ls -l /var/lib/icinga2/certs
total 16
-rw-rw---- 1 nagios nagios 5179 Jun  7  2018 ca.crt
-rw-rw---- 1 nagios nagios 1875 Jan  7 09:44 icinga-master.foobar.example.crt
-rw-rw---- 1 nagios nagios 1704 Jun  7  2018 icinga-master.foobar.example.key
```

On peut maintenant tester la configuration, avant de redémarrer Icinga :

```
#  icinga2 daemon -C
...
[2020-05-15 12:49:18 +0000] information/cli: Finished validating the configuration file(s).
```

C'est parfait, on redémarre le maître avec sa nouvelle configuration :

```
# systemctl restart icinga2
```

Mais les agents, eux, ne sont pas encore configurés et ne vont pas répondre aux demandes du maître. Configurons donc le premier agent. D'abord, activer l'API (on ne le fait pas sur le maître puisqu'on a choisi une configuration où c'est toujours le maître qui se connecte aux agents, jamais le contraire) :

```
#  icinga2 feature enable api
```

Et n'oublions pas d'ajouter dans `features-available/api.conf` :

```
object ApiListener "api" {
...
  accept_config = true
  accept_commands = true
```

Et dans le `zones.conf` de l'agent, on va indiquer qui est le maître et donc à qui faire confiance (l'authentification sera faite via les certificats, dans la session TLS) :

```
// Moi (l'agent)
object Endpoint NodeName {
  host = NodeName
}
object Zone ZoneName {
  endpoints = [ NodeName ]
  parent = "icinga-master.foobar.example"
}

// Mon maître
object Endpoint "icinga-master.foobar.example" {
}
object Zone "icinga-master.foobar.example" {
  endpoints = [ "icinga-master.foobar.example" ]
}

object Zone "global-templates" {
  global = true
}
```

Ensuite, on crée des certificats pour l'agent, on les fait signer par l'AC CAcert et on les installe.

Comme avec le maître, on teste la configuration avec `icinga2 daemon -C` et on redémarre Icinga. Comme l'API a été activée, on doit voir Icinga écouter sur le réseau :

```
# lsof -i:5665
COMMAND   PID   USER   FD   TYPE    DEVICE SIZE/OFF NODE NAME
icinga2 21041 nagios   14u  IPv4 261751305      0t0  TCP *:5665 (LISTEN)
```

Pour tester qu'Icinga écoute bien, et en TLS, essayons depuis le maître :

```
% gnutls-cli bellovese.foobar.example:5665
...
- Server has requested a certificate.
...
- Certificate[0] info:
 - subject 'CN=bellovese.foobar.example', issuer 'CN=CAcert Class 3 Root,OU=http://www.CAcert.org,O=CAcert Inc.'
...
- Status: The certificate is trusted.
- Description: (TLS1.3)-(ECDHE-SECP256R1)-(RSA-PSS-RSAE-SHA256)-(AES-256-GCM)
```

Si cette connexion TCP plus TLS ne fonctionne pas, c'est peut-être un problème de réseau et/ou de pare-feu. À vous de déboguer !

Maintenant que l'agent fonctionne, on dit au maitre de s'y connecter. Dans le `zones.conf` du maître :

```
object Zone "bellovese.foobar.example" {
  endpoints = [ "bellovese.foobar.example" ]
  parent = ZoneName
}
```

Et, toujours sur le maître, dans le fichier qui sert à configurer ce qu'on supervise sur cette machine :

```
object Host "bellovese" {
   ...
    vars.client_endpoint = "bellovese.foobar.example"
    zone = "icinga-master.foobar.example"
```

Et on crée sur le maitre un fichier `zones.d/bellovese.conf` pour indiquer ce qu'on veut faire faire à l'agent. Par exemple :

```
object Host "test-segovese" {
  vars.outside = true
  import "remote-host"
  address = "segovese.foobar.example"
  address6 = "segovese.foobar.example"

}

object Host "bellovese-opendnssec" {
  import "generic-host"
  check_command = "remote_always_true"
  vars.opendnssec = true
  vars.opendnssec_signer_expect="There are 7 zones configured"
  vars.opendnssec_enforcer_expect="bortzmeyer\\.org"
}
```

La première directive `object Host` dit à l'agent de tester la machine `segovese.foobar.example` (ce qui donnera un point de vue différent du maître, si l'agent est situé dans un autre réseau). La deuxième directive dit que `bellovese` est un site OpenDNSSEC et qu'on veut donc surveiller que les deux démons OpenDNSSEC tournent bien (ils ne sont pas accessibles de l'extérieur, il faut donc un agent Icinga sur cette machine).

Maintenant que maître et agent sont prêts, la connexion devrait se faire, ce qu'on peut voir dans le journal, ici celui du maître :

```
May 15 13:52:48 ambigat icinga2[31436]: [2020-05-15 13:52:48 +0000] information/ApiListener: Finished syncin
```

On aurait aussi pu faire la configuration avec le "*wizard*". Voici un exemple où on configure le maître :

```
# icinga2 node wizard
Welcome to the Icinga 2 Setup Wizard!
...
Please specify if this is a satellite setup ('n' installs a master setup) [Y/n]: n
Starting the Master setup routine...
Please specify the common name (CN) [ambigat]: icinga-master.foobar.example
...
information/cli: Dumping config items to file '/etc/icinga2/zones.conf'.
...
information/cli: Updating constants.conf.
...
Done.

Now restart your Icinga 2 daemon to finish the installation!
```

Et pour configurer un agent :

```
# icinga2 node wizard
Welcome to the Icinga 2 Setup Wizard!
...
Please specify if this is a satellite setup ('n' installs a master setup) [Y/n]: y
Starting the Node setup routine...
Please specify the common name (CN) [sumarios]: sumarios.foobar.example
Please specify the master endpoint(s) this node should connect to:
Master Common Name (CN from your master setup): icinga-master.foobar.example
Do you want to establish a connection to the master from this node? [Y/n]: n
...
Connection setup skipped. Please configure your master to connect to this node.
...
```

Bon, si tout va bien, les tests sont maintenant exécutés par les agents et vous pouvez voir les résultats dans l'interface Web ou en ligne de commande via l'API. Mais si ça ne marche pas ? On a vu que la configuration d'Icinga était complexe, et la documentation pas toujours d'un abord facile. Alors, voyons maintenant le déboguage des problèmes. D'abord, et avant tout, si ça ne marche pas, il faut évidemment regarder le journal (`journalctl -t icinga2 -f -n` avec systemd). Par exemple, ici, on voit que le maître ne peut pas se connecter à l'agent, car le certificat de ce dernier a expiré :

```
Feb 12 14:00:41 ambigat icinga2[31333]: [2020-02-12 14:00:41 +0000] warning/ApiListener: Certificate validation
```

De la même façon, si vous voyez des tests qui restent désespérément affichés dans l'état "*pending*" sur le maître, le journal de l'agent pourra vous renseigner sur ce qui se passe. Il peut, par exemple, manquer une commande externe (`execvpe(/usr/local/lib/nagios/plugins/check_whichasn) failed: No such file or directory`). La panne peut être aussi due à une mauvaise configuration dans le `zones.conf` de l'agent, par exemple l'oubli de la directive `parent`. Autre cas, si vous trouvez des messages comme :

```
warning/ApiListener: Ignoring config update. 'api' does not accept config.
```

c'est que vous avez oublié `accept_config = true` dans `features-available/api.conf`.

Si le journal par défaut n'est pas suffisant, vous pouvez aussi rendre Icinga bien plus bavard. Vérifiez que `debuglog` est bien activé :

```
# icinga2 feature list
Disabled features: compatlog elasticsearch gelf graphite icingastatus influxdb livestatus opentsdb perfdata
Enabled features: api checker command debuglog ido-pgsql mainlog notification
```

Ou, s'il ne l'est pas, activez-le avec `icinga2 feature enable debuglog` et configurez dans `features-available/debuglog.conf`. Par exemple, cette configuration :

```
object FileLogger "debug-file" {
  severity = "debug"
  path = LogDir + "/debug.log"
}
```

Va vous produire un `debug.log` très bavard.