Silence Cop internals

Démo équipe Monitoring 17/10/2023

Expression de besoin (1/2)

Les gens posent des silences sur certaines alertes. Parfois, ces silences peuvent rester alors qu’ils n’ont plus lieu d’être. Cela peut cacher des problèmes.

L’équipe Supervision vérifie chaque semaine les silences posés.

L’opération était manuelle. Elle a été semi automatisée.

Au 05/10/2023 il y a 353 silences répartis sur 59 pages

Expression du besoin (2/2)

L’équipe Supervision voudrait automatiser entièrement cette tâche de vérification des silences et avoir une approche réactive et non proactive.

La vérification des silences devrait se faire en tâche de fond et alerter

- soit par un dashboard dédié (site web ? dashboard Grafana ? Autre ?)
- soit par un système d’alerte (dans Karma ? sur RocketChat ? Autre ?)

Silence_cop

https://git.corp.caascad.com/caascad/applications/silence_cop

L’outil Silence_cop

- se connecte à un Alertmanager pour obtenir la liste des silences
- identifie certains motifs dans le commentaire de chaque silence
- lorsqu’un motif mentionne un ticket Jira, il effectue une requête dans Jira pour obtenir l’état du ticket
- en fonction du motif (qui peut mentionner une date ou un ticket Jira), il détermine si le silence est valide ou non
- il renvoie un rapport au format JSON

Le futur (et le présent)

Aujourd’hui, l’outil est limité à l’identification des silences et leur validité sur un seul Alertmanager ainsi qu’à la production du rapport au format JSON sur la sortie standard du terminal.

Un autre outil, en shell, permet de lancer l’outil sur tous les Alertmanagers de toutes les zones Caascad et de filtrer les silences pour n’afficher que les silences non valides.

Cet outil est en bash et il s’agit d’un bricolage en attendant mieux.

Démo d’utilisation

Voir dans le fichier README des exemples d’utilisation.

La démo dépendra des résultats obtenus sur les diverses zones testées

Compilation

Pré-requis : go-1.18 minimum.

La compilation est décrite dans le fichier README.md

Démo : `go build` (il n'y a rien à voir : le fichier `silence_analyser` est juste généré)

Étude du code source : les packages

- `main` : le package servant à générer silence_analyzer
- `alertmanager` : la connexion à Alertmanager et l'obtention des silences
- `jira` : la connexion à Jira et l'obtention d'infos sur les tickets Jira
- `logger` : pour la journalisation
- `parser` : pour lire la ligne de commande et les variables d'environnement
- `vault` : pour se connecter à Vault et lire des secrets
- `zones` : pour obtenir des informations sur les zones

Étude du code source : main.go et args_logger.go

Ces fichiers contiennent principalement

- l’initialisation (avec la lecture de la ligne de commande et des variables d’environnement, et l’initialisation du système de logging)
- la connexion à Alertmanager et l’obtention des silences
- la vérification des silences

Étude du code source : logger/logger.go

Ce répertoire contient le nécessaire pour

- initaliser le système de logging
- émettre des messages d’information, warning, erreur… dans le système de logging

Étude du code source : parser/parser.go

Ce répertoire contient le nécessaire pour

- lire et analyser la ligne de commande
- lire les variables d’environnement nécessaires
- calculer des variables nécessaires pour joindre Alertmanager, Vault et Jira

Étude du code source : alertmanager/xxx.go

Ce répertoire contient le nécessaire pour

- se connecter à Alertmanager
- obtenir la liste des silences
- pré-calculer quelques informations nécessaires pour déterminer ultérieurement (main.go) pour chaque silence s’il est valide

Étude du code source : jira/jira.go

Ce répertoire contient le nécessaire pour

- se connecter à Jira
- obtenir la description d’un ticket Jira
- définir si l’état d’un ticket est acceptable ou non (la liste est codée en dur dans ce fichier)

Étude du code source : vault/vault.go

Ce répertoire contient le nécessaire pour

- se connecter à Vault (avec le tokenHelper)
- lire un secret dans Vault
- lire une liste de secrets dans Vault

Étude du code source : zones/zones.go

Ce répertoire contient le nécessaire pour

- obtenir la liste des zones (depuis envs-ng/gen/zones_static/zones.json via une requête HTTPS)
- calculer l'URL Rancher d'une zone
- calculer l'URL Karma (utilisée uniquement dans le rapport à titre informatif pour faciliter la tâche de l'opérateur)

Le script supervision_check_silences.sh

Le `silence_analyzer` ne se connecte qu'à un seul Alertmanager. En attendant de reprendre le développement du projet, il fallait :

- obtenir la liste des zones
- boucler sur tous les alertmanagers
- générer un rapport en filtrant les silences légitimes

C'est ce que fait ce script. Le rapport est sur la sortie standard, en JSON, mais il est lisible.

Le script supervision_check_silences.sh : démo

Lancer le script.

Le résultat dépendra des silences posés sur le moment.

Le futur : PF-1600

PF-1600 : Compatibilité de Silence_cop avec NGOT

Il faut adapter l'outil `silence_analyzer` pour qu'il puisse se connecter aux deux Alertmanagers NGOT.

Il faut ajouter ces deux Alertmanagers dans le script `supervision_check_silences.sh`

Le futur : PF-1601

PF-1601 : Automatiser le lancement de l'outil (itération 2)

Il faut créer un nouvel outillage autour de `silence_analyzer` afin

- qu'il soit lancé régulièrement
- que son rapport soit facile d'accès et facile à interpréter

Cela implique de choisir un scheduler :

- Cronjobs dans Kubernetes. Quel cluster ?
- Concourse ? Mais est-ce encore un bon choix fin 2023 ?
- Gitlab Runner ? Est-il l'outil ultime ou une fausse bonne idée ?
- Autre ?

Le futur : PF-1601 (suite)

PF-1601 : Automatiser le lancement de l’outil (itération 2)

L’équipe Supervision, au lieu de devoir lancer `supervision_check_silences.sh` et attendre le résultat, pourra alors directement aller voir le résultat quand elle en a besoin.

Le futur : PF-1602

PF-1602 : Alerter lorsqu'un silence est mal posé (itération 3)

Il faut que le rapport créé lors de l'itération 2 (PF-1601) soit utilisé par un système d'alerting (Karma, Rocketchat…) .

Le ticket PF-1602 impose Prometheus, donc Karma. Mais il n'a pas été affiné. Cependant, lorsque ce ticket sera pris en compte, le besoin et l'outillage de PF aura peut-être évolué. Il faudra en tenir compte et confirmer qu'on veut bien des alertes Prometheus.

Le futur : l’équipe Supervision et les autres

Conclusion : lorsque ces fonctionnalités auront été implémentées, la vérification des silences s’effectuera autrement

- l’équipe Supervison sera déchargée de cette tâche
- ceux qui posent des silences seront alertés lors de la pose d’un silence non justifié
- l’équipe Supervision veillera sur les silences par l’intermédiaire des alertes qui lui remonteront
- les silences non légitimes seront identifiés plus rapidement.