Identification des checks les plus consommateurs à l'aide de la commande shinken-scheduler-export-data

Sommaire

- Économiser des CPUs en identifiant les checks les plus consommateurs
- Création du tableau récapitulatif sur la consommation totale des temps CPU des royaumes respectifs
  - Consolidation des données: utilisation d'un tableau croisé dynamique
  - Création du tableau croisé dynamique
  - Sélection des commandes en tant que lignes de notre tableau
  - Obtenir la consommation CPU totale par check
    - Passer du nombres de lignes avec "cpu_time" à une vraie somme des temps CPU
    - Rendre le titre de la colonne explicite
  - Obtenir la répartition de la consommation CPU par check
    - Passer la seconde colonne en somme de temps, comme la première
    - Passer du nombre total de temps CPU consommé à la répartition de la consommation CPU par check
    - Rendre le titre de la colonne explicite
  - Obtenir la répartition de la consommation CPU moyenne pour une exécution (en s)
    - Passer du nombre total de temps CPU consommé à la consommation CPU moyenne pour une exécution de check
    - Rendre le titre de la colonne explicite
- Analyse des résultats
  - Analyse du cas présenté: quel check doit être optimisé en priorité pour économiser du CPU?
    - Quel check est le plus consommateur?
    - Exercice d'exemple: quel check optimiser afin de limiter la consommation CPU à un seul CPU?

Économiser des CPUs en identifiant les checks les plus consommateurs

Lorsque l'on souhaite réduire la consommation CPU de sa supervision, optimiser ses commandes de vérification est une étape primordiale.
En effet, les commandes de vérification sont la plupart du temps la première source de consommation CPU de la supervision.
L'objectif de cette page est de permettre l'identification des commandes dont l'optimisation aura le plus d'influence sur la consommation CPU.

Pour cela, nous allons extraire des informations issues de la commande **shinken-scheduler-export-data** afin d'avoir:

- **le temps CPUs utilisés par chaque commande sur l'ensemble des royaumes**

Dans notre exemple. Nous avons un cas simplifié avec **3** checks:

- **check-host-alive**:
  - check par défaut pour la vérification des hôtes
- **dump-check-example-1**:
  - check d'exemple
- **dump-check-exemple-2**:
  - check d'exemple

Sur une installation réelle, il y aura plus de commandes, mais l'analyse restera la même vu que le principe sera de trouver les plus consommateurs.

On a besoin d'un dump de données avec en option une prévision de la charge sur une période, disons par exemple 1 heure. Il suffit alors de lancer la commande comme ceci:

```
shinken-scheduler-export-data --export-type=identify-most-consuming-checks --simulate-scheduling-for-X-seconds=3600
```

L'importation du fichier .csv généré est décrit dans la page suivante : shinken-scheduler-export-data - export des données du Scheduler

Avec ce lancement les noms (*hôte, check, commandes et royaumes*) seront présents dans l'export. Il est tout à fait possible de faire l'analyse sur un export en **--anonymous**, des hash des noms des commandes seront utilisés au lieu des noms finaux.

Le client pourra lancer un export avec les noms en parallèles afin de faire le lien avec les commandes une fois l'analyse effectuée.

Création du tableau récapitulatif sur la consommation totale des temps CPU des royaumes respectifs

Consolidation des données: utilisation d'un tableau croisé dynamique

On va devoir procéder à une consolidation des nombreuses données de checks obtenues afin d'obtenir un résultat exploitable.

- Pour cela on va utiliser la fonctionnalité d'Excel : **le tableau croisé dynamique.**
- Un tableau croisé dynamique dans un tableur est un outil de analyse de données qui vous permet de créer une vue synthétique et facile à lire d'une grande quantité de données (*ici nos exécutions de checks*).

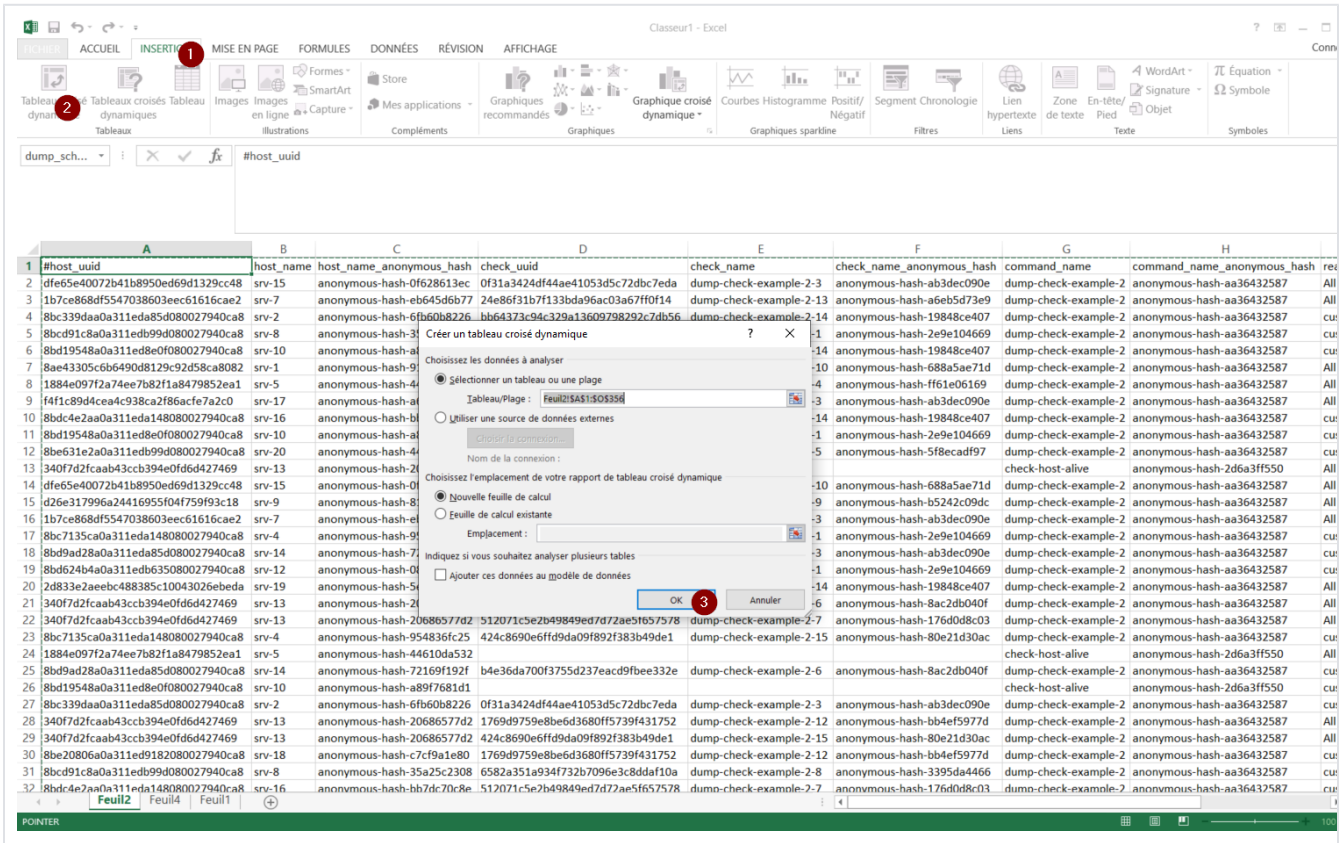
On y choisi les données à inclure, comment les organiser et comment les synthétiser, filtrer, classer et totaliser les données en fonction de nos besoins

Afin de pouvoir faire une analyse complète, notre objectif sera ici d'obtenir dans notre tableau:

- **la consommation CPU totale par check,**
- **la répartition de la consommation CPU par check par rapport au autres,**
- **et enfin la consommation CPU moyenne pour une exécution pour chaque check**

Création du tableau croisé dynamique

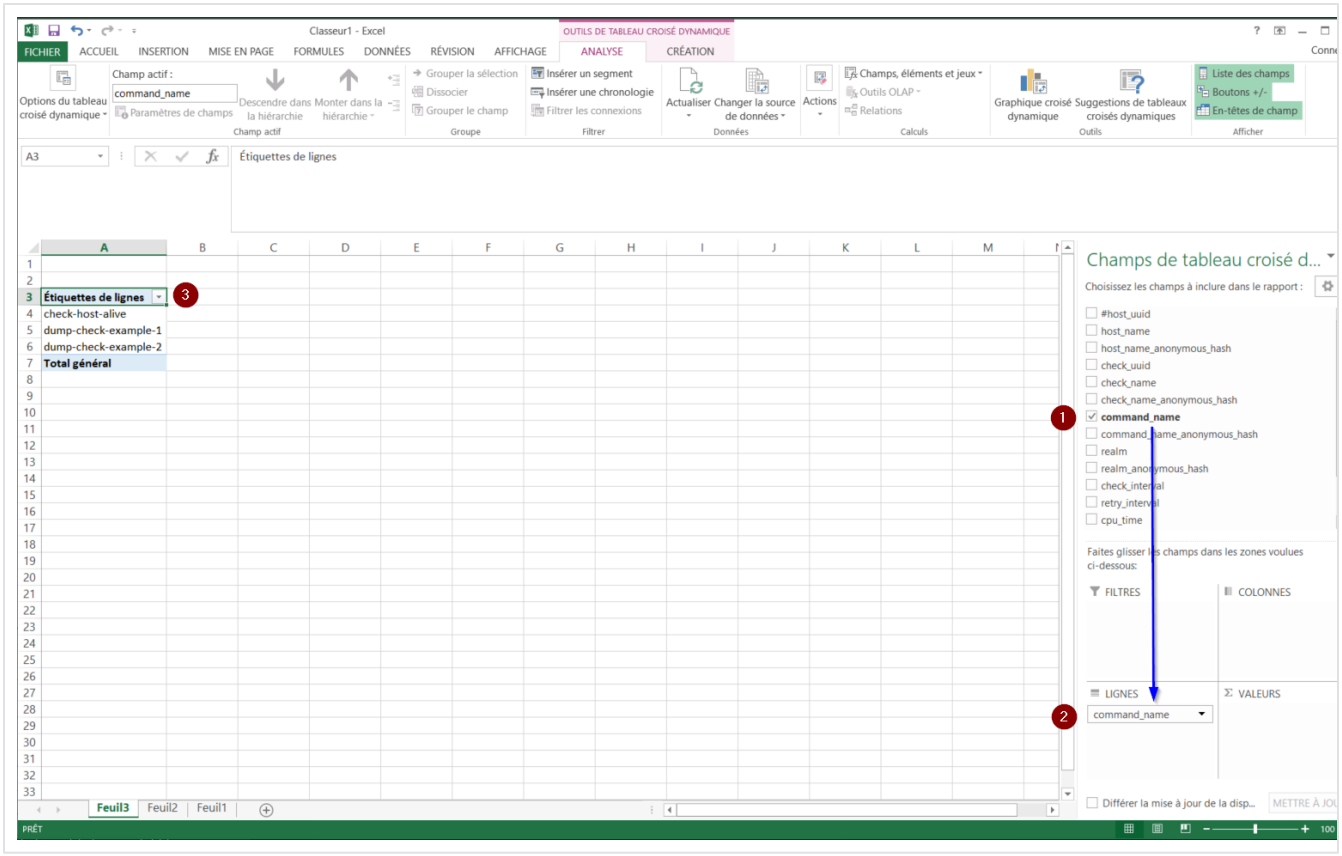
La création du tableau récapitulatif passe par la création d'un **Tableau croisé dynamique**. Depuis votre Feuille d'importation des données, il faut cliquer sur **Insertion Tableau croisé dynamique**, et valider :



Sélection des commandes en tant que lignes de notre tableau

Arrivé sur la nouvelle feuille, Excel demande quelles lignes sélectionner, dans le bloc de droite nommé "**Champs de tableau croisé dynamique**".

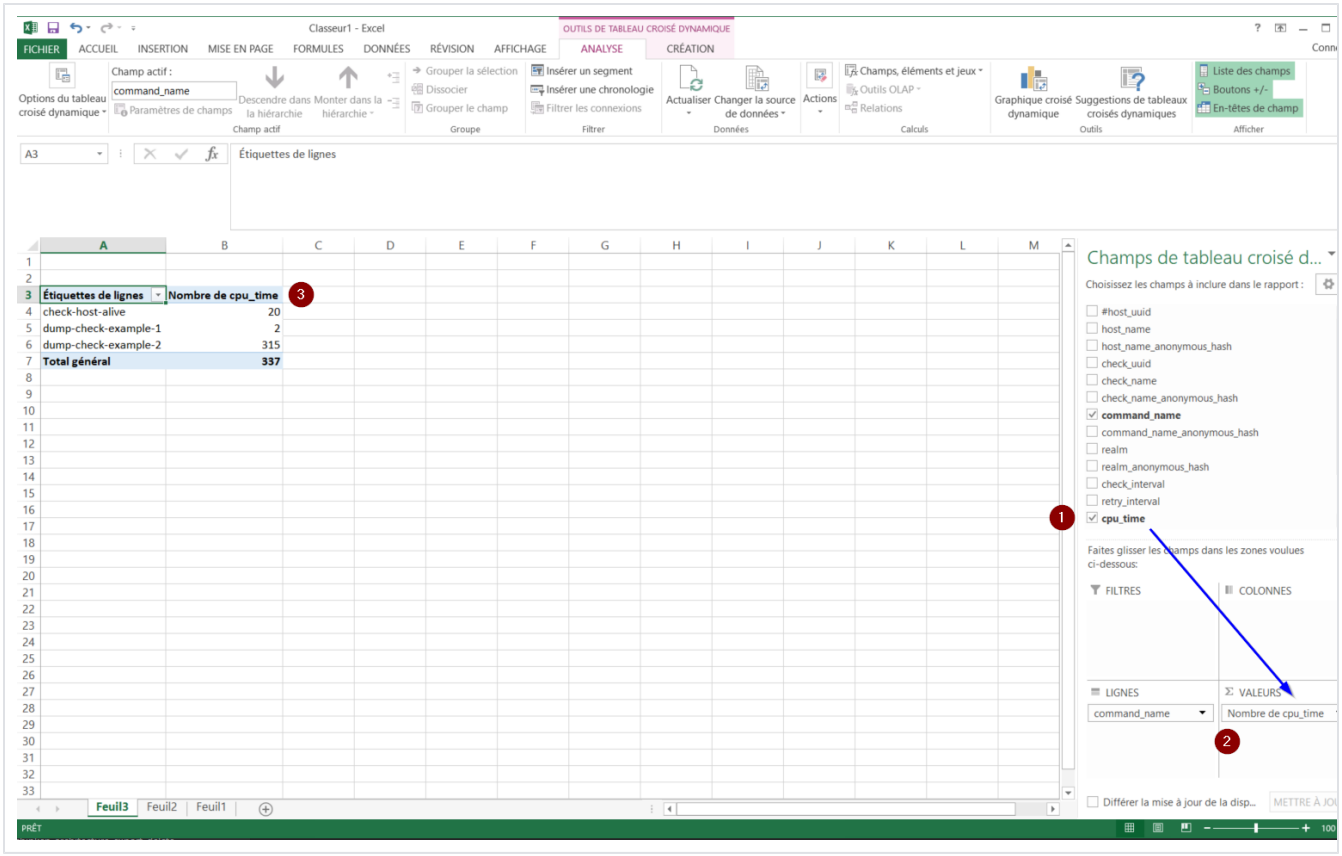
Dans notre cas, il faut faire glisser le champ **command_name** (*ou bien **command_name_anonymous_hash** si on a une version anonyme de l'export*) vers le bloc "**Lignes**" :



Ceci permet de définir nos lignes dans la colonne A avec l'ensemble de nos commandes.

Obtenir la consommation CPU totale par check

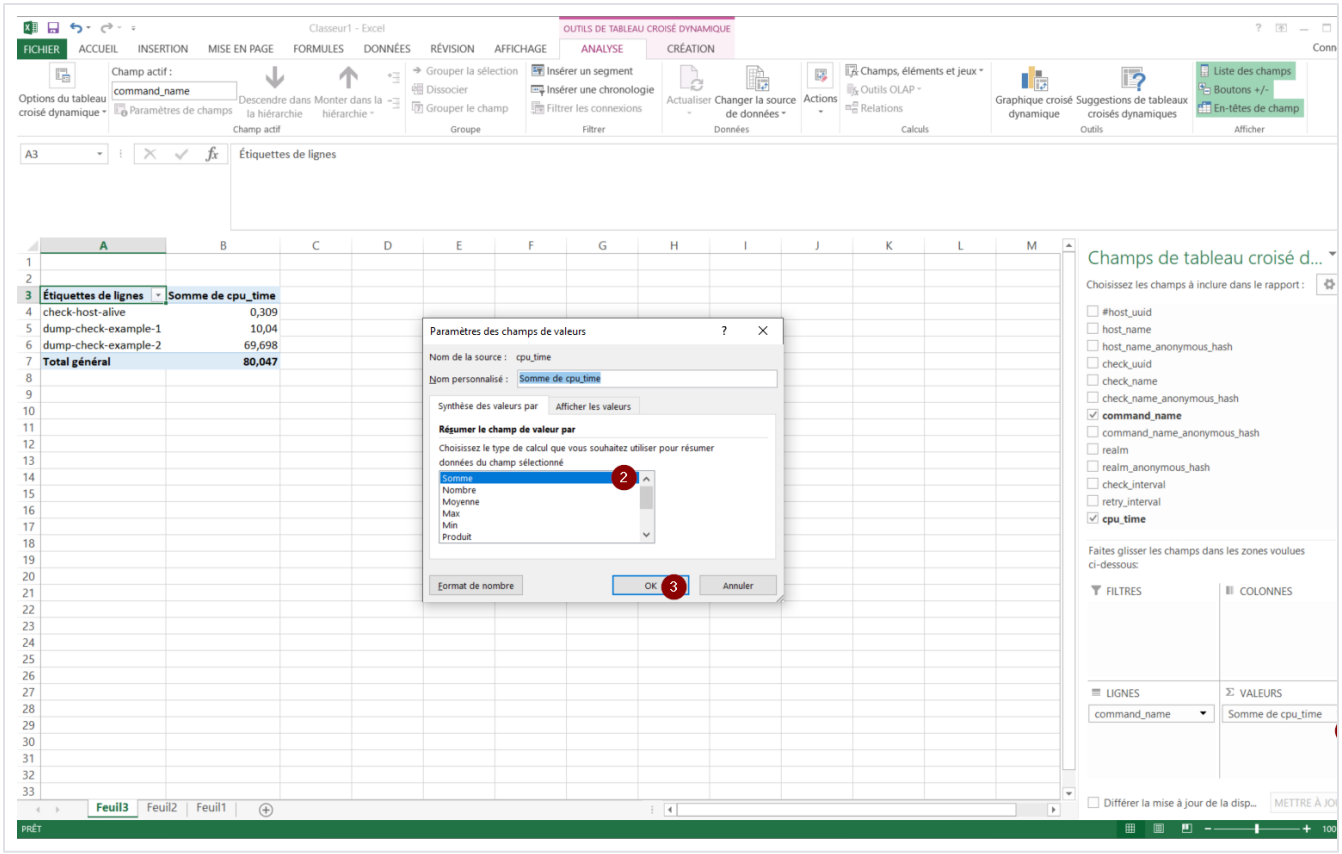
A chaque commande, il faut lui assigner une (*ou plusieurs*) "**Valeurs**". Pour cela, on fait glisser le champ "**cpu_time**" vers le bloc "**Valeurs**" afin d'avoir pour chaque royaume son champs **cpu_time** associé :



A noter que, par défaut, Excel prends le nombre d'occurrences du champ **cpu_time** comme "**Valeur**", ce qui n'est pas ce qui est souhaité (*"Nombre de cpu_time"*).

Passer du nombres de lignes avec "cpu_time" à une vraie somme des temps CPU

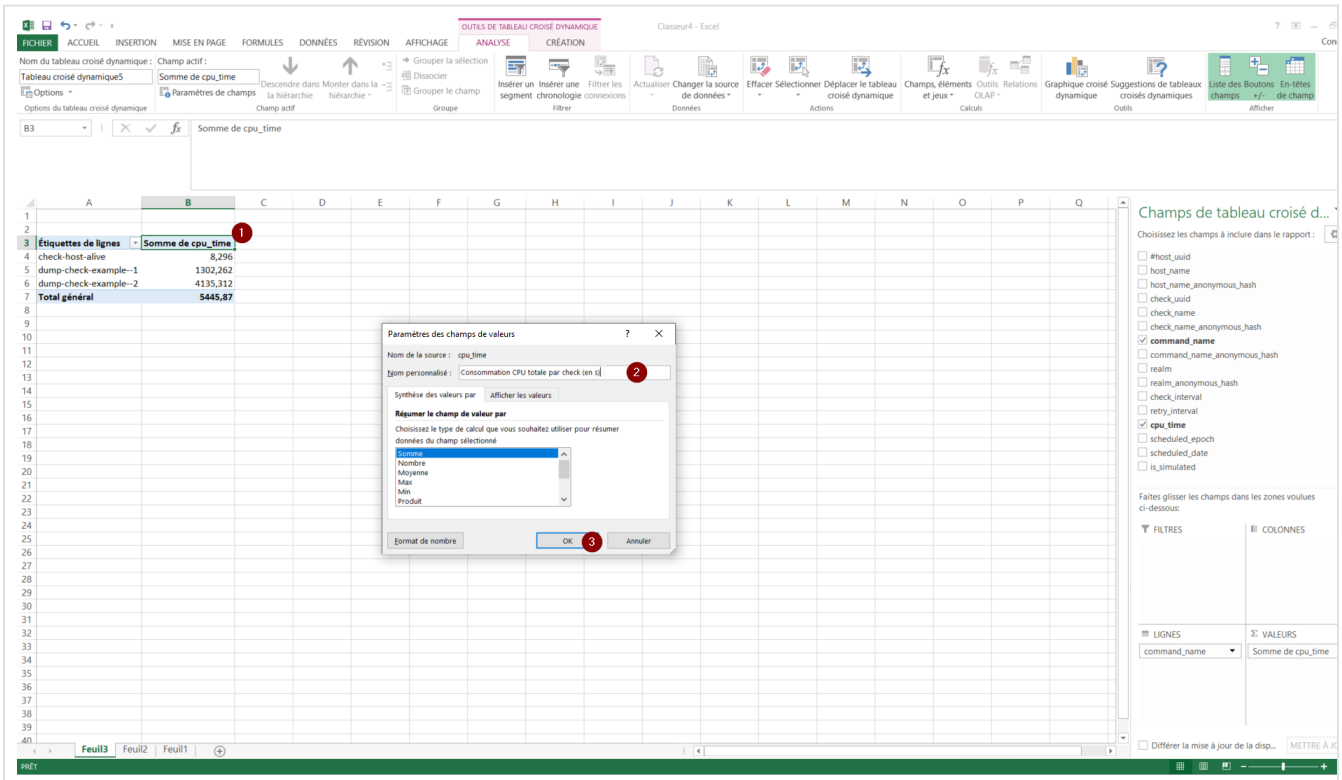
Une modification des "**Paramètres des champs de valeurs**" est nécessaire sur "**Nombre de cpu_time**" qu'il faut changer en Somme pour donner au final "**Somme de cpu_time**" :



On obtient alors la somme en temps (*secondes*) consommée par chaque commande.

Rendre le titre de la colonne explicite

Le titre de la colonne n'étant pas des plus lisible pour notre analyse, on le change en double cliquant dessus, et le renomme alors "**Consommation CPU totale par check (en s)**" :



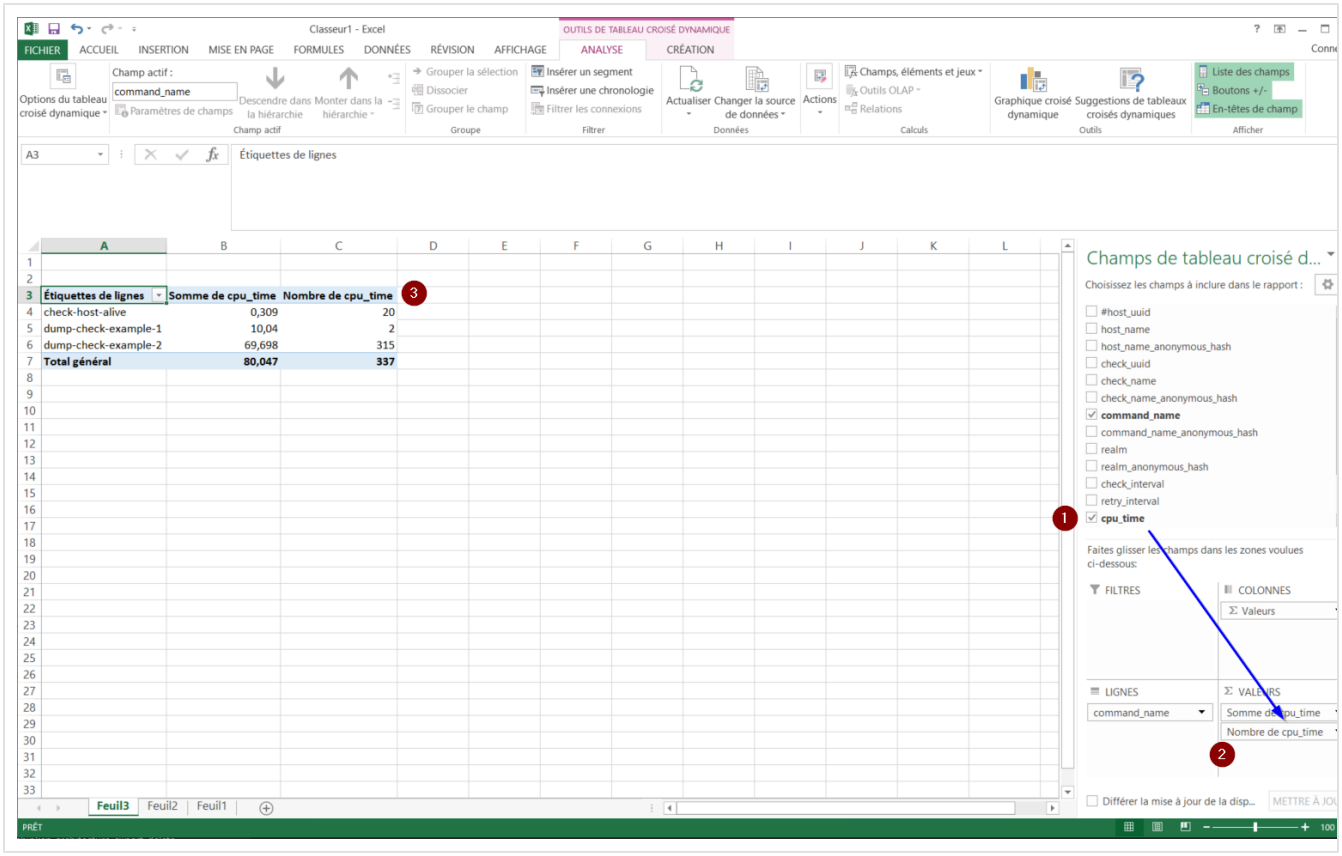
Notre première valeur est obtenue: **la consommation CPU totale par check**.

Obtenir la répartition de la consommation CPU par check

S'il est possible de trier par rapport au temps absolu consommé par un check et donc avoir les plus consommateurs, il peut être également très intéressant d'avoir leur pourcentage par rapport à la consommation totale.

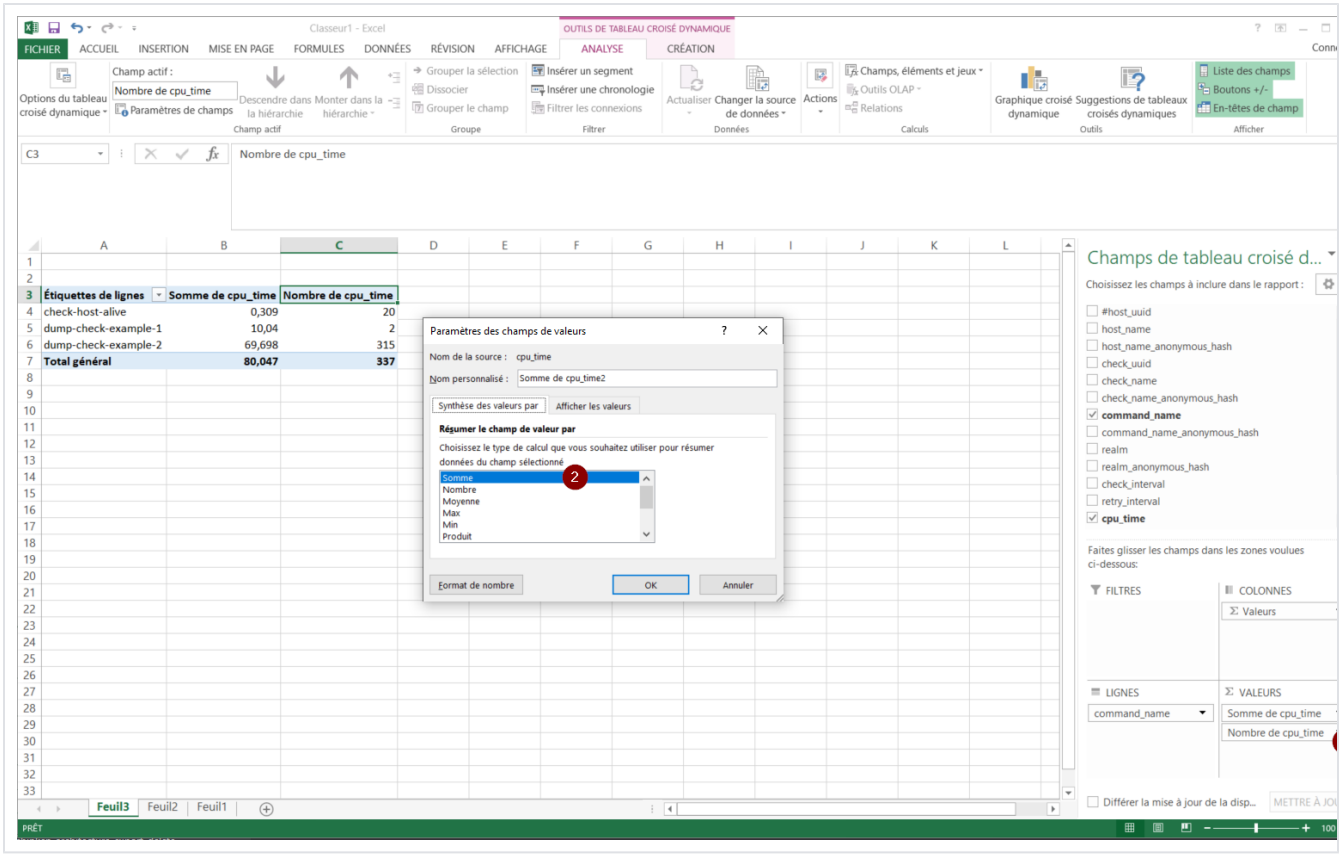
Ceci en facilitera grandement la lecture, et il sera ainsi possible de savoir de quel ordre de grandeur une optimisation sera bénéfique pour la consommation.

Il faut rajouter une seconde fois le champ "**cpu_time**" dans les valeurs, ce qui rajoute une nouvelle colonne :



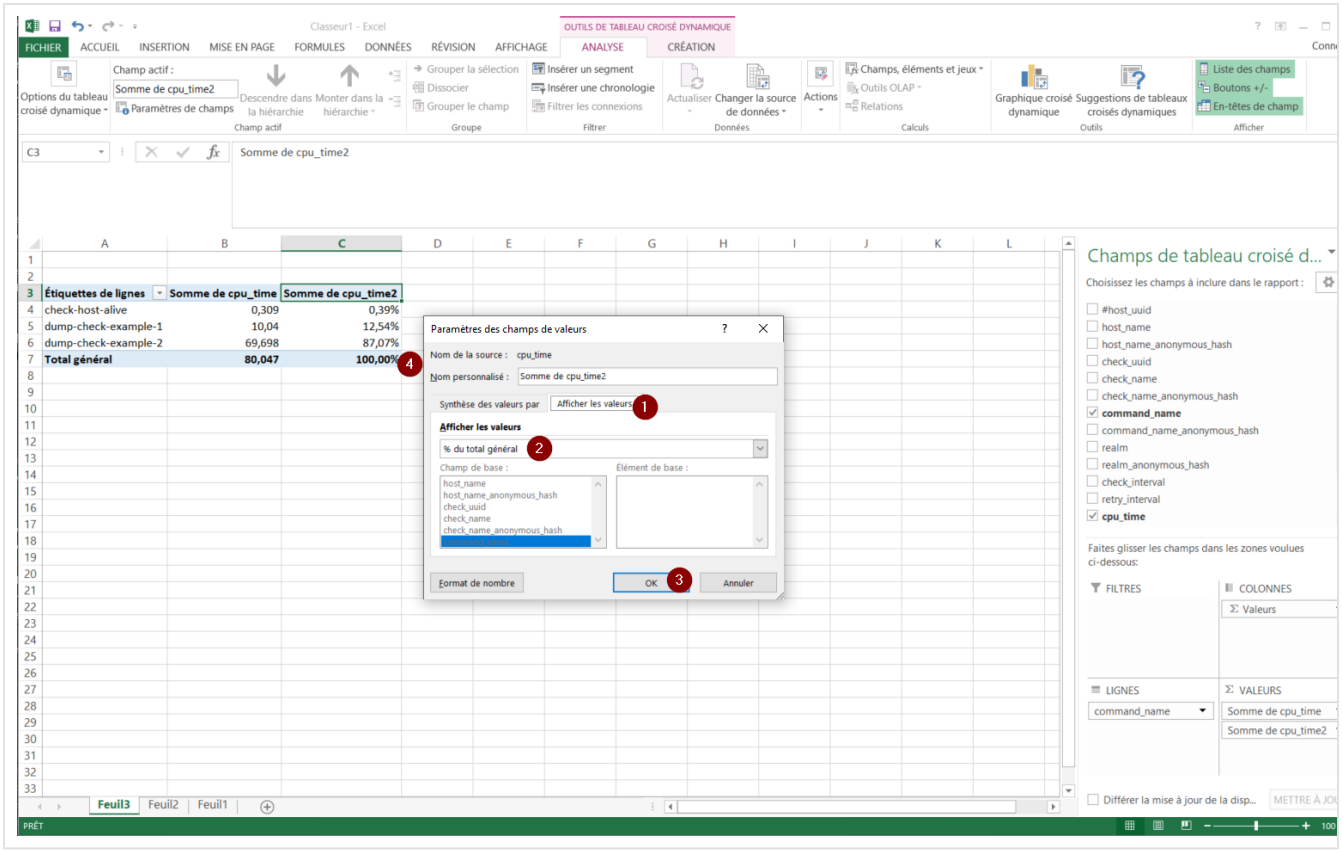
Passer la seconde colonne en somme de temps, comme la première

Ensuite il faut passer, comme pour la colonne précédente, d'un calcul en **Nombre** à un calcul en "**Somme de cpu_time**" :



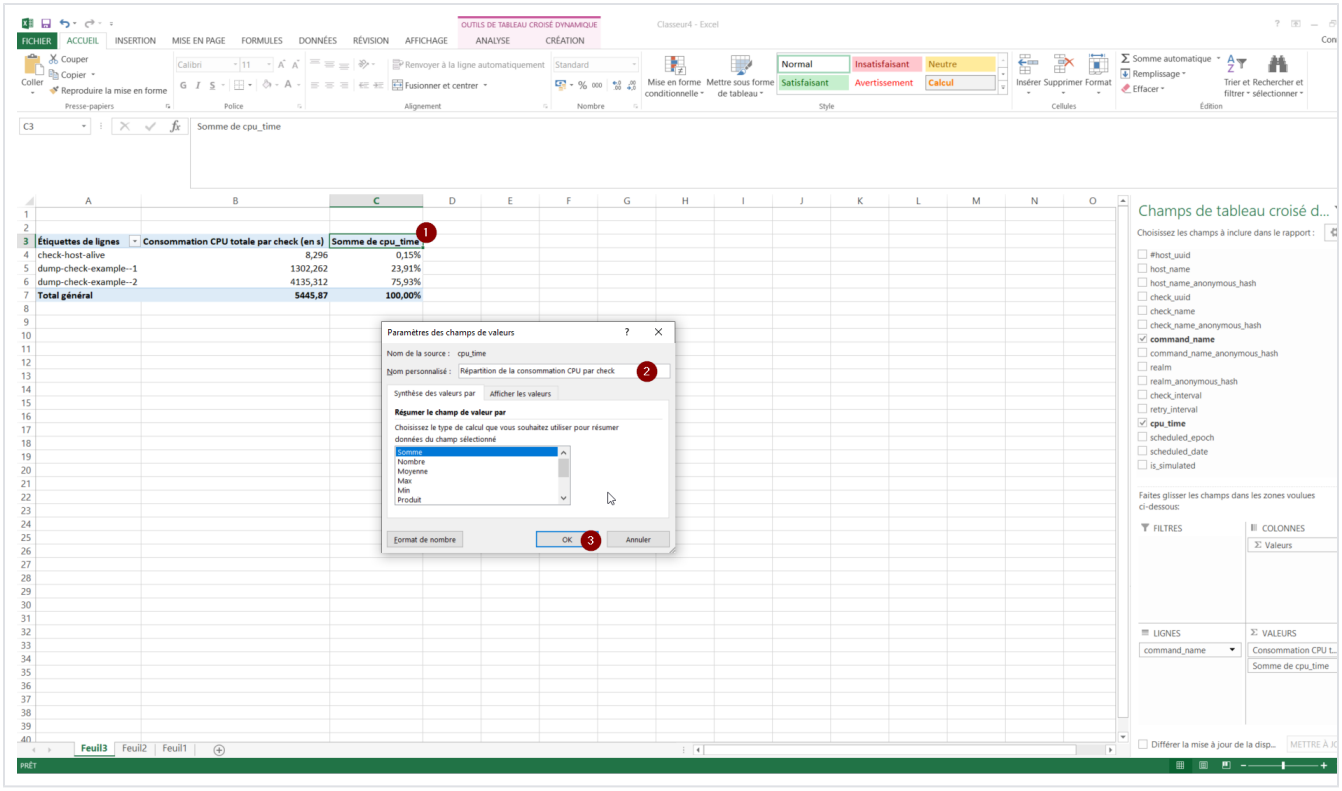
Passer du nombre total de temps CPU consommé à la répartition de la consommation CPU par check

Le changement dans cette étape consiste à changer l'affichage dans le menu "**Afficher les valeurs**" en choisissant "**% du total général**" pour obtenir la **Répartition de la consommation CPU par check** :



Rendre le titre de la colonne explicite

Comme précédemment, on change le titre par défaut en double cliquant dessus. On le renomme alors "**Répartition de la consommation CPU par check**" :



Ces deux premières colonnes nous permettent d'identifier rapidement les checks les plus consommateurs.

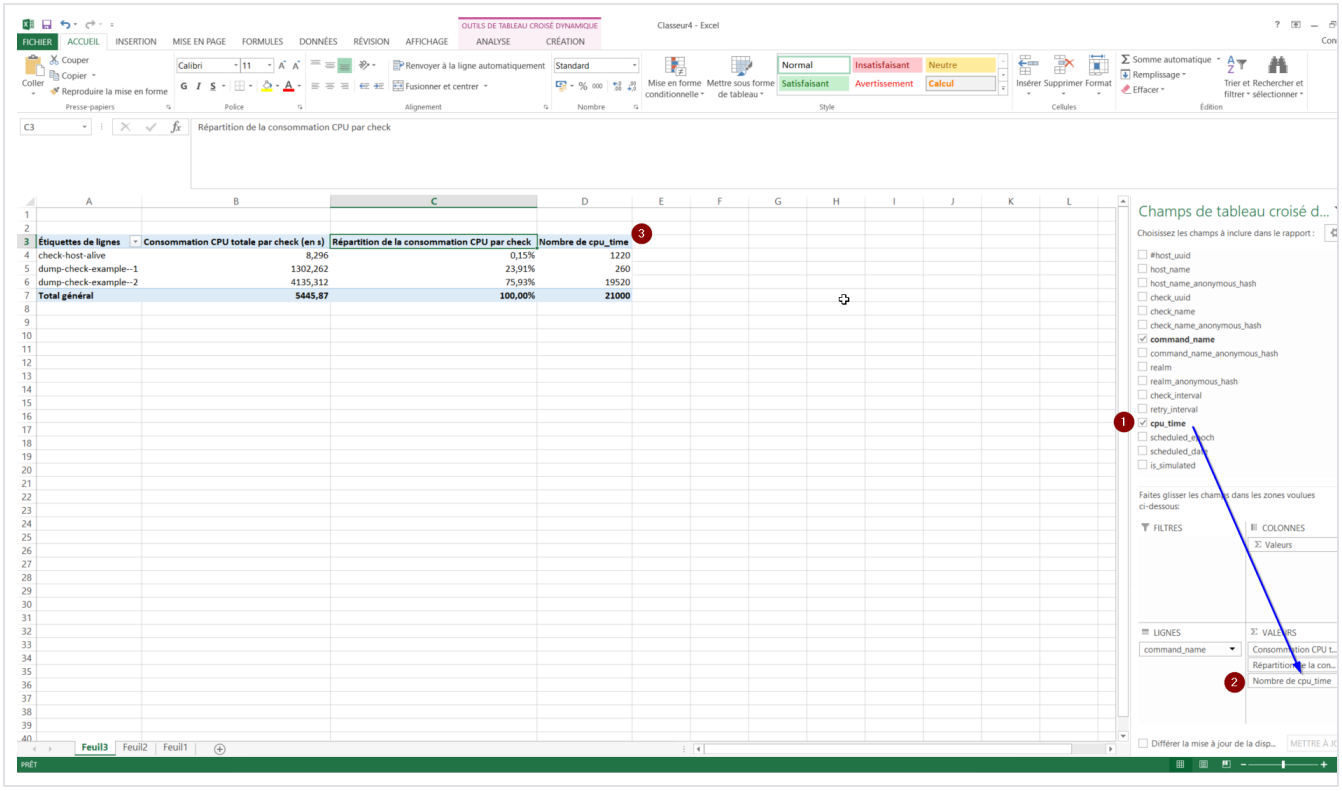
On a alors obtenu la seconde donnée intéressante: **la répartition de la consommation CPU par check par rapport au autres.**

La suivante, et dernière, va nous permettre d'affiner l'analyse en calculant le temps moyen de CPU par exécution de check.

Obtenir la répartition de la consommation CPU moyenne pour une exécution (en s)

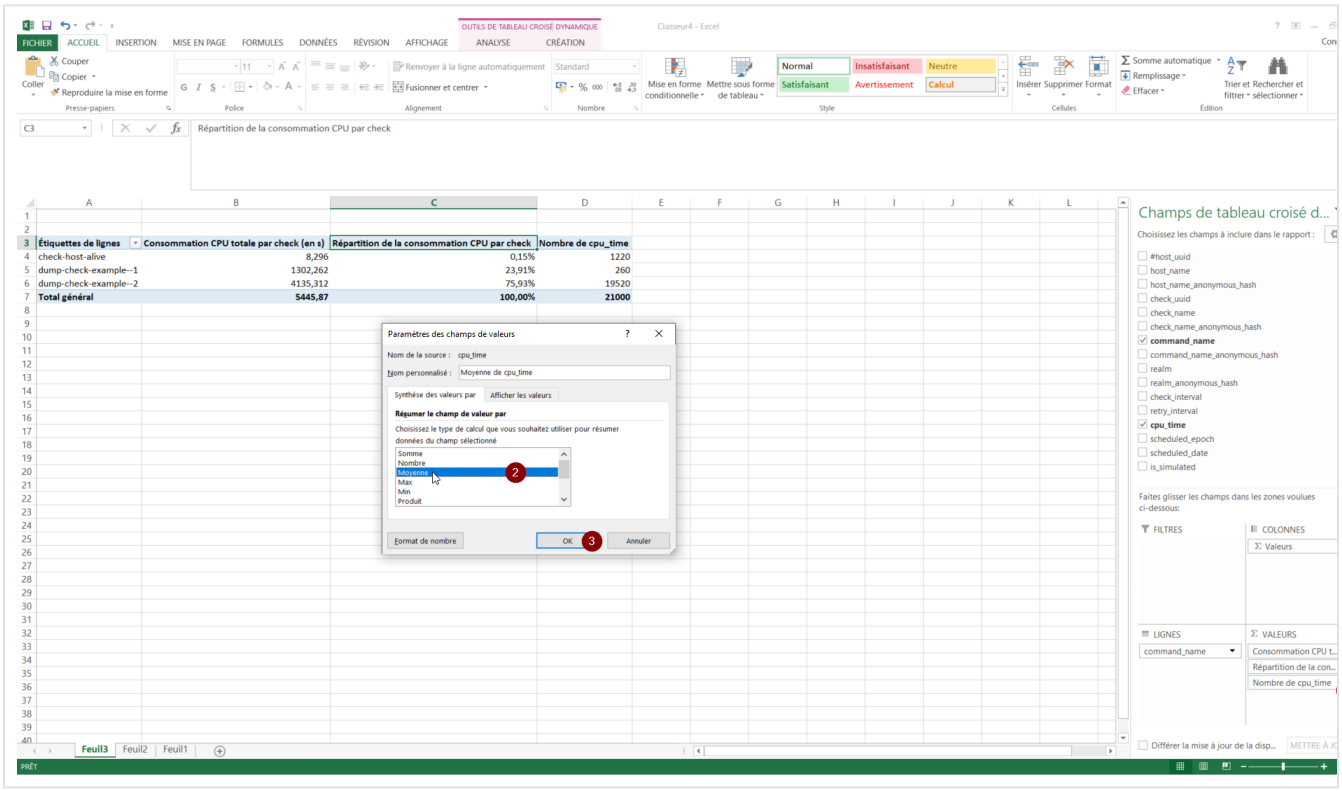
La dernière statistique qui va nous intéresser pour prioriser nos actions d'optimisation va être d'obtenir la consommation CPU moyenne pour une exécution de check.

Pour cela, il faut rajouter une dernière fois le champ "**cpu_time**" dans les valeurs, exactement comme pour les deux précédentes colonnes :



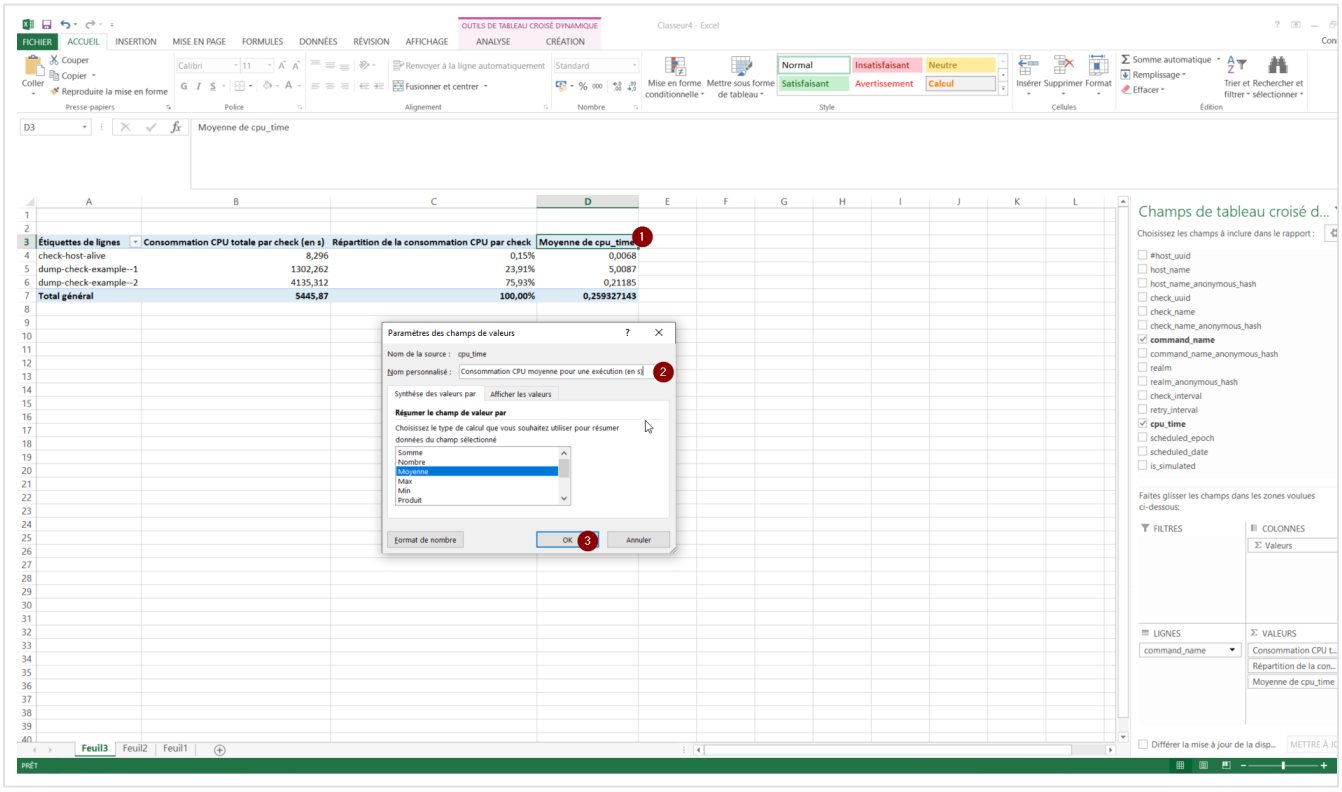
Passer du nombre total de temps CPU consommé à la consommation CPU moyenne pour une exécution de check

Pour cette colonne, il faut changer le calcul de **Nombre** à un calcul en "**Moyenne de cpu_time**" :



Rendre le titre de la colonne explicite

Comme pour les deux colonnes précédentes, on change le titre de la colonne en change en double cliquant dessus. On le renomme alors "**Consommation CPU moyenne pour une exécution (en s)**" :

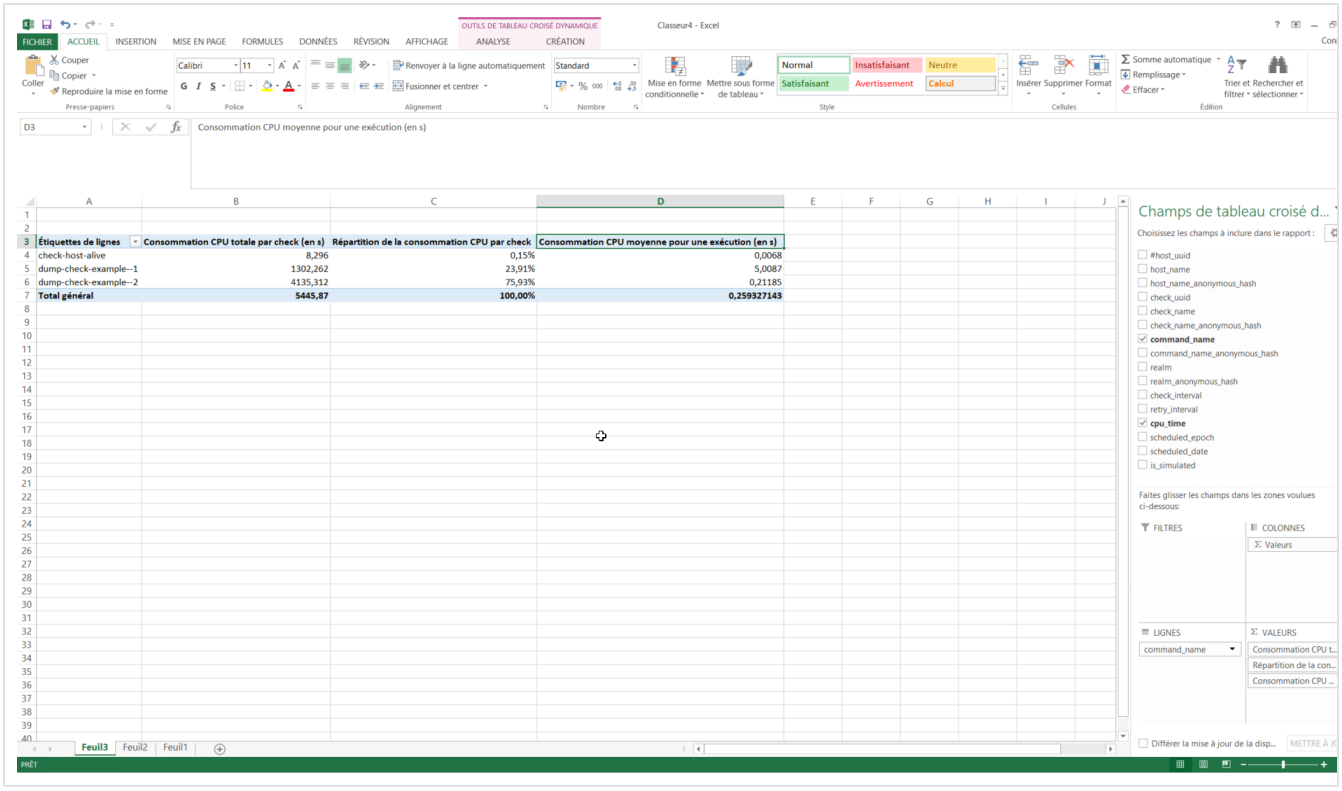


On obtient alors la dernière de nos trois données d'analyse: **la consommation CPU moyenne pour une exécution pour chaque check.**

Analyse des résultats

Le tableau final obtenu, on peut analyser afin d'identifier de la manière la plus efficace possible le ou les checks les plus consommateurs.

Il est ainsi possible de se concentrer sur l'optimisation des checks qui auront un véritable impact sur les performances de la plateforme.



Étiquettes de lignes	Consommation CPU totale par check (en s)	Répartition de la consommation CPU par check	Consommation CPU moyenne pour une exécution (en s)
check-host-alive	8,296	0,15%	0,0068
dump-check-example--1	1302,262	23,91%	5,0087
dump-check-example--2	4135,312	75,93%	0,21185
Total général	**5445,87**	**100,00%**	**0,259327143**

Analyse du cas présenté: quel check doit être optimisé en priorité pour économiser du CPU?

Quel check est le plus consommateur?

Faisons l'exercice avec notre exemple. Nous avons **3** checks (*leur temps d'exécution est extrait de la colonne "Consommation CPU moyenne pour une exécution (en s)"*) :

- **check-host-alive**:
  - check par défaut pour la vérification des hôtes, seulement **0.006s** (*6ms*) de temps moyen d'exécution, très peu couteux car il lance principalement un simple ping
- **dump-check-example-1**:
  - check d'exemple qui consomme en moyenne **5s** de temps CPU par exécution, ce qui est **très élevé** pour un check
- **dump-check-exemple-2**:
  - second check d'exemple qui consomme **0.2s** (*200ms*) de temps CPU par exécution, ce qui est **légèrement élevé** pour un check

En ne regardant que ces valeurs, et en ne se basant que sur notre première impression, on serait tenté de se dire que le check le plus consommateur est **dump-check-example-1** avec ses **5s** de temps d'exécution par check. C'est en effet un temps anormal pour un check de supervision.

Mais cette première impression est trompeuse. En effet, si on regarde mieux notre tableau on s'aperçoit que ce check ne représente qu'environ **25%** du temps CPU total consommé (***23,91%**, colonne "**Répartition de la consommation CPU par check**"*). Il n'est ainsi pas le plus consommateur /prioritaire.

Le check le plus consommateur est en fait **dump-check-example-2**, avec ses plus de **75%** de temps CPU total consommé (***75,93%***).
Il est donc intéressant de voir en priorité les gains que l'on peut faire sur ce dernier, surtout qu'avec ses **200ms** de temps d'exécution, il n'est pas léger, et donc des optimisations doivent être possibles.

Et seulement dans un second temps voir l'optimisation de **dump-check-example-1** qui avec ses **5s** de temps d'exécution doit largement avoir de quoi être optimisé.

Exercice d'exemple: quel check optimiser afin de limiter la consommation CPU à un seul CPU?

On utilise nos données pour faire un exercice pratique et concret: on ne souhaite utiliser qu'un seul CPU pour l'ensemble de nos checks.

Dans notre cas, on avait pris un temps de simulation de **3600s**=**1h** pour notre extraction d'ordonnancement. La somme totale des temps d'exécutions de toutes les checks est de **5445s**, soit moins de **2h** de temps de calcul (*7200s*).

L'ensemble des checks consomment ainsi actuellement moins de 2 CPUs (*5445/3600=1.5 CPUs*).

Si l'on souhaite ne consommer qu'un seul CPU sur cette plateforme, il faut ainsi gagner **5445-3600=1845s** de temps de traitement. Deux possibilités s'offrent alors:

- si on se limite à optimiser le check **dump-check-example-1**:
  - l'objectif de se limiter à 1CPU est **impossible**, car il ne consomme au total que **1300s** (*1302*), ce qui est moins que le gain nécessaire de **1845s**,
- si on se concentre sur le check **dump-check-example-2**:
  - il est le plus consommateur, ceci demande une optimisation de 1845/4135=**45%** de son temps d'exécution, et ainsi passer sous les **110ms** de temps d'exécution moyen, ce qui est raisonnablement atteignable comme optimisation.

Dans notre cas, il faut donc un gain de **45%** sur le check **dump-check-example-2** afin de limiter la consommation CPU de la plateforme à 1 seul CPU.