Instrumenter les systèmes distribués pour une visibilité opérationnelle

David Yanacek

Instrumenter les systèmes distribués pour une visibilité opérationnelle

Plonger dans les journaux

Lorsque j'ai été embauché chez Amazon après l'université, l'un de mes premiers exercices d'intégration consistait à préparer le serveur Web amazon.com pour le faire fonctionner sur mon bureau de développeur. Je n'ai pas réussi du premier coup, et je ne savais pas vraiment pourquoi. Un collègue aimable m'a conseillé de consulter les journaux pour voir ce qui n'allait pas. Pour cela, il fallait que je fasse un « cat sur le fichier journal ». Je pensais que mes collègues me faisaient une farce ou une blague en rapport avec les chats, que je ne comprenais pas. À l'université, je n'utilisais Linux que pour la compilation, le contrôle de source et l'éditeur de texte. Je ne savais donc pas que « cat » était en fait une commande pour imprimer un fichier sur le terminal que je pourrais intégrer à un autre programme afin de rechercher des modèles.

Mes collègues m'ont redirigé vers des outils comme cat, grep, sed et awk. Équipé de ces nouveaux outils, j'ai plongé dans les journaux du serveur Web amazon.com sur mon bureau de développeur. L'application du serveur Web comportait déjà une instrumentation permettant d'émettre tous types d'informations utiles dans ses journaux. Ainsi, je pouvais voir quelle configuration empêchait le serveur Web de démarrer sans que je ne le sache, où il avait pu planter ou encore pourquoi il ne parvenait pas à communiquer avec un service en aval. Le site Web est composé de nombreuses parties mobiles, et je le voyais comme une boîte noire au début. Cependant, après avoir plongé dans le système, j'ai appris à comprendre comment fonctionnait le serveur et comment interagir avec ses dépendances en regardant simplement la sortie de l'instrumentation.

Pourquoi l'instrumentation

Au fil des ans, alors que je passais d'une équipe à l'autre chez Amazon, j'ai compris que l'instrumentation était un outil inestimable nous permettant d'apprendre comment fonctionnent les systèmes. D'ailleurs, l'instrumentation ne sert pas qu'à comprendre les systèmes. Elle est au cœur du mode de fonctionnement d'Amazon. Une excellente instrumentation nous aide à voir l'expérience que nous offrons à nos clients.

Cet accent sur les performances opérationnelles concerne l'entreprise tout entière. Dans les services associés à amazon.com, une latence plus élevée entraîne une mauvaise expérience d'achat et, par conséquent, des taux de conversion plus bas. Avec les clients qui utilisent AWS, ils dépendent de la disponibilité élevée et de la faible latence des services AWS.

Chez Amazon, nous ne prenons pas en compte que la latence moyenne. Nous nous concentrons davantage sur les rangs de latence élevés, comme le 99,9e ou le 9,99e centile. En effet, l'expérience est mauvaise même lorsque seule une demande sur 1 000 ou sur 10 000 est lente. Nous savons que lorsque nous réduisons les centiles de latence élevés dans un système, nos efforts entraînent une réduction de la latence médiane. En revanche, nous savons que lorsque nous réduisons la latence médiane, les centiles de latence élevés sont moins souvent réduits.

L'autre raison pour laquelle nous nous concentrons sur les centiles de latence élevés est qu'une latence élevée dans un seul service peut avoir un effet multiplicateur sur les autres services. Le site d'Amazon est construit sur une architecture axée sur les services. De nombreux services collaborent entre eux pour mener à bien leurs tâches, comme le rendu d'une page Web sur amazon.com. Ainsi, une augmentation de la latence d'un service en profondeur dans la chaîne d'appel (même si l'augmentation concerne un centile élevé) aura des répercussions importantes sur la latence du côté de l'utilisateur final.

Les grands systèmes d'Amazon sont composés de plusieurs services qui collaborent entre eux. Chaque service est développé et géré par une équipe (les « grands » services sont composés de plusieurs services ou composants en arrière-plan). L'équipe qui détient un service est *propriétaire du service*. Chaque membre de l'équipe réfléchit comme un propriétaire et opérateur du service, qu'il soit développeur logiciel, ingénieur réseau, gestionnaire ou autre. En tant que propriétaires, les équipes définissent des objectifs de performances opérationnelles pour tous les services associés. Nous nous assurons également d'avoir la visibilité nécessaire concernant les opérations de service pour atteindre ces objectifs, faire face à tout problème pouvant survenir et viser des objectifs encore plus élevés l'année suivante. Pour définir leurs objectifs et avoir cette visibilité, les équipes doivent doter les systèmes d'instrumentations.

L'instrumentation nous permet également de détecter les événements opérationnels et d'y répondre tactiquement. L'instrumentation transmet des données aux tableaux de bord opérationnels pour que les opérateurs puissent voir les métriques en temps réel. Elle transmet également des données aux alarmes, qui se déclenchent et impliquent des opérateurs lorsque le système se comporte de façon inattendue. Les opérateurs utilisent la sortie détaillée de l'instrumentation pour diagnostiquer rapidement l'origine du problème. À partir de là, nous pouvons minimiser le problème et y revenir plus tard pour l'empêcher de se produire à nouveau. En l'absence d'une bonne instrumentation dans l'ensemble du code, nous perdons un temps précieux à diagnostiquer les problèmes.

Ce qu'il faut mesurer

Pour exploiter les services conformément à nos normes strictes en matière de disponibilité et de latence, en tant que propriétaires de service, nous devons mesurer le comportement de nos systèmes.

Pour obtenir la télémétrie nécessaire, les propriétaires de services mesurent les performances opérationnelles depuis plusieurs endroits afin d'avoir différents points de vue sur les comportements de bout en bout. Cette tâche est compliquée, même dans une architecture simple. Imaginons que des clients appellent un service à travers un équilibreur de charge : le service communique avec un cache et une base de données à distance. Nous voulons que chaque composant émette des métriques sur son comportement. Nous voulons également des métriques sur la perception par chaque composant du comportement des autres composants. Lorsque les métriques pour tous ces points de vue sont rassemblées, un propriétaire de service peut retrouver rapidement la source du problème et l'explorer pour en trouver la cause.

De nombreux services AWS fournissent automatiquement des informations opérationnelles sur vos ressources. Par exemple, Amazon DynamoDB fournit des métriques Amazon CloudWatch sur les taux de réussite et d'échec et sur la latence tell que mesurés par le service. Cependant, lorsque nous construisons des systèmes qui utilisent ces services, nous nécessitons bien plus de visibilité sur le comportement de nos systèmes. L'instrumentation requiert un code explicite qui enregistre le temps nécessaire pour les tâches, la fréquence d'utilisation de certains chemins de code, les métadonnées sur le travail que la tâche effectuait et les parties réussies ou échouées de la tâche. Si une équipe n'ajoutait pas d'instrumentation explicite, elle serait contrainte d'exploiter son propre service comme une boîte noire.

Par exemple, si nous implémentions une opération d'API de service qui extrayait les informations produit par identifiant de produit, le code pourrait ressembler à ce qui suit. Ce code recherche les informations produit dans un cache local, suivi d'un cache à distance, lui-même suivi d'une base de données :

```
public GetProductInfoResponse getProductInfo(GetProductInfoRequest
request) {

  // check our local cache
  ProductInfo info = localCache.get(request.getProductId());

  // check the remote cache if we didn't find it in the local cache
  if (info == null) {
    info = remoteCache.get(request.getProductId());

      localCache.put(info);
  }

  // finally check the database if we didn't have it in either cache
  if (info == null) {
    info = db.query(request.getProductId());

      localCache.put(info);
      remoteCache.put(info);
  }

  return info;
}
```

Si j'utilisais ce service, j'aurais besoin de beaucoup d'instrumentation dans ce code pour pouvoir comprendre son comportement en production. Il me faudrait pouvoir résoudre les demandes échouées ou lentes et suivre les tendances et les signes indiquant que différentes dépendances sont sous-dimensionnées ou ont un mauvais comportement. Voici le même code, annoté avec quelques questions auxquelles il faudrait que je sache répondre pour l'ensemble du système en production ou pour une demande particulière :

```
public GetProductInfoResponse getProductInfo(GetProductInfoRequest
request) {

  // Which product are we looking up?
  // Who called the API? What product category is this in?

  // Did we find the item in the local cache?
  ProductInfo info = localCache.get(request.getProductId());

  if (info == null) {
    // Was the item in the remote cache?
    // How long did it take to read from the remote cache?
    // How long did it take to deserialize the object from the cache?
    info = remoteCache.get(request.getProductId());

    // How full is the local cache?
    localCache.put(info);
  }

  // finally check the database if we didn't have it in either cache
  if (info == null) {
    // How long did the database query take?
    // Did the query succeed?
```

```
    // If it failed, is it because it timed out? Or was it an invalid
query? Did we lose our database connection?
    // If it timed out, was our connection pool full? Did we fail to
connect to the database? Or was it just slow to respond?
    info = db.query(request.getProductId());

    // How long did populating the caches take?
    // Were they full and did they evict other items?
    localCache.put(info);
    remoteCache.put(info);
  }

  // How big was this product info object?
  return info;
}
```

Le code pour répondre à toutes ces questions (et à d'autres) est légèrement plus long que la logique métier elle-même. Certaines bibliothèques peuvent aider à réduire la quantité de code d'instrumentation, mais le développeur doit quand même poser les questions sur la visibilité dont les bibliothèques auront besoin, et le développeur doit se concentrer sur le raccord de l'instrumentation.

Lorsque vous résolvez une demande qui circule dans un système distribué, il peut être difficile de comprendre ce qu'il s'est passé si vous examinez la demande en vous basant sur une seule interaction. Pour rassembler les pièces du puzzle, il nous paraît utile de rassembler dans un seul et même endroit toutes les mesures relatives à tous ces systèmes. Mais pour cela, chaque service doit être instrumenté pour enregistrer un identifiant de trace pour chaque tâche et pour propager cet identifiant de trace à tous les autres services qui collaborent sur cette tâche. La collecte de l'instrumentation parmi les systèmes pour un identifiant de trace donné peut être réalisée soit après coup, soit en temps presque réel à l'aide d'un service comme AWS X-Ray.

Forage

L'instrumentation permet la résolution à plusieurs niveaux, du simple coup d'œil aux métriques pour vérifier la présence d'anomalies trop subtiles pour déclencher les alarmes à la réalisation d'une enquête pour trouver la cause de ces anomalies.

Au niveau le plus élevé, l'instrumentation est agrégée en métriques pouvant déclencher les alarmes et s'afficher sur les tableaux de bord. Ces métriques agrégées permettent aux opérateurs de surveiller le taux de demandes global, la latence des appels de service et les taux d'erreurs. Ces alarmes et métriques nous informent des anomalies ou des changements qui devraient faire l'objet d'une enquête.

Après avoir détecté une anomalie, nous devons trouver pourquoi cette anomalie se produit. Pour répondre à cette question, nous nous reposons sur les métriques rendues possibles par encore davantage d'instrumentation. En instrumentant le temps nécessaire pour réaliser différentes étapes de la réponse d'une demande, nous pouvoir voir quelle partie du processus est plus lente ou déclenche le plus souvent des erreurs.

Alors que les minuteurs et les métriques agrégés peuvent nous aider à écarter des pistes de recherche et à insister sur d'autres, ils ne fournissent pas toujours d'explications complètes. Par exemple, nous savons grâce aux métriques que des erreurs proviennent d'une opération d'API

donnée, mais les métriques ne donnent pas assez de détails sur la raison de l'échec de l'opération. À ce stade, nous regardons les données de journalisation brutes et détaillées émises par le service pour le créneau en question. Les données brutes montrent la source du problème : soit l'erreur précise qui est en train de se produire, soit les aspects particuliers de la demande qui déclenchent une erreur en périphérie.

Comment nous instrumentons

L'instrumentation requiert un codage. Cela signifie que lorsque nous implémentons de nouvelles fonctionnalités, nous devons prendre le temps d'ajouter du code pour indiquer ce qu'il s'est passé, si cela a réussi ou échoué et le temps qu'il a fallu. Étant donné le caractère courant de la tâche de codage qu'est l'instrumentation, des pratiques sont apparues chez Amazon au fil des années pour gérer les scénarios fréquents : la normalisation des bibliothèques d'instrumentation courantes et la normalisation du compte rendu des métriques structurées et axées sur la journalisation.

La normalisation des bibliothèques d'instrumentation courantes aide les auteurs de bibliothèques à donner de la visibilité à leurs clients sur le fonctionnement des bibliothèques. Par exemple, les clients HTTP fréquemment utilisés intègrent ces bibliothèques courantes, de sorte que si une équipe de service implémente un appel à distance vers un autre service, elle obtient automatiquement l'instrumentation pour ces appels.

Lorsqu'une application d'instrumentation exécute et effectue le travail, les données de télémétrie qui en résultent sont écrites sur un fichier de journalisation structuré. En général, elles sont émises sous la forme d'une entrée de journalisation par « unité de travail », qu'il s'agisse d'une demande à un service HTTP ou d'un message extrait d'une file d'attente.

Chez Amazon, les mesures dans l'application ne sont pas regroupées et sont vidées de temps en temps vers un système d'agrégation des métriques. Tous les minuteurs et compteurs de chaque unité de travail sont écrits dans un fichier journal. Ensuite, les journaux sont traités et les métriques agrégées sont calculées après coup par un autre système. Ainsi, nous avons toutes les informations nécessaires, des métriques opérationnelles agrégées de haut niveau aux données de résolution détaillées au niveau de la demande, toutes avec une approche unique du code d'instrumentation. Chez Amazon, nous journalisons d'abord, et nous produisons des métriques agrégées ensuite.

Instrumenter via la journalisation

En général, nous instrumentons nos services de manière à ce qu'ils émettent deux types de données de journalisation : les données de demandes et les données de débogage. Les données de journalisation des demandes sont généralement représentées sous la forme d'une entrée de journalisation structurée unique pour chaque unité de travail. Ces données contiennent des propriétés sur la demande et sur la personne qui a effectué la demande, sur la finalité de la demande, sur les compteurs de fréquence des différentes actions et sur les minuteurs de durée des actions. La journalisation des demandes sert de journal d'audit et de trace de tout ce qu'il s'est passé dans le service. Les données de débogage incluent des données pas ou peu structurées sur les lignes de débogage émises par l'application. Ces données sont généralement des données de journal non structurées comme l'erreur Log4j ou des lignes de journal d'avertissement. Chez Amazon, ces deux types de données sont généralement émis dans deux fichiers journaux distincts, en partie pour des raisons historiques, mais également parce qu'il peut être pratique d'analyser les journaux sur un format d'entrée de journal homogène.

Les agents comme CloudWatch Logs traitent les deux types de données de journal en temps réel et acheminent les journaux vers CloudWatch Logs. CloudWatch Logs produit à son tour des métriques agrégées sur le service en temps presque réel. Amazon CloudWatch Alarms lit ces métriques agrégées et déclenche des alarmes.

Bien que la journalisation de tous ces détails sur chaque demande peut être coûteuse, chez Amazon, nous pensons qu'elle est indispensable. Après tout, nous devons résoudre les légères anomalies de disponibilité, les pics de latence et les problèmes signalés par les clients. Sans journaux détaillés, nous ne pouvons pas donner de réponses aux clients ni améliorer leurs services.

Entrer dans les détails

Le sujet de la surveillance et des alarmes est vaste. Dans cet article, nous n'allons pas aborder les sujets tels que la configuration et le réglage des seuils d'alarmes, l'organisation des tableaux de bord opérationnels, la mesure des performances à la fois côté serveur et côté client, l'exécution continue d'applications « canary » ou encore la sélection du système approprié à utiliser pour agréger les métriques et analyser les journaux.

En revanche, cet article porte essentiellement sur la nécessité d'instrumenter nos applications pour produire les données de mesure brutes appropriées. Nous allons décrire ce que les équipes d'Amazon s'efforcent d'inclure (ou d'éviter) lorsqu'elles instrumentent leurs applications.

Bonnes pratiques en matière de journaux de demandes

Dans cette section, nous allons décrire les bonnes habitudes acquises au fil du temps chez Amazon en matière de journalisation des données « par unité de travail » structurée. Un journal qui répond à ces critères contient des compteurs représentant la fréquence des différentes actions, des minuteurs indiquant la durée des actions et des propriétés telles que les données sur chaque unité de travail.

Comment nous journalisons

- **Nous émettons une entrée de journal de demande pour chaque unité de travail.** Une unité de travail est généralement une demande reçue par notre service ou un message que notre service extrait d'une file d'attente. Nous écrivons une entrée de journal de service pour chaque demande reçue par notre service. Nous ne mélangeons pas plusieurs unités de travail. Ainsi, lorsque nous résolvons une demande échouée, nous n'avons qu'une entrée de journal à consulter. Cette entrée contient les paramètres d'entrée pertinents sur la demande permettant de voir ce qui était tenté, qui était le mandataire et les informations sur le minuteur et le compteur dans un seul endroit.
- **Nous n'émettons pas plus d'une entrée de journal de demande par demande.** Dans une implémentation de service non bloquant, il peut convenir d'émettre une entrée de journal distincte pour chaque étape d'un pipeline de traitement. En réalité, nous parvenons davantage à résoudre ces systèmes en sondant un handle dans un seul « objet de métriques » autour entre les étapes du pipeline, puis en sérialisant les métriques comme une unité une fois toutes les étapes complétées. La présence de plusieurs entrées de

journal par unité de travail complique l'analyse du journal et multiple les frais de journalisation, déjà onéreux. Si nous écrivons un service non bloquant, nous tentons de planifier le cycle de vie de la journalisation des métriques à l'avance, car il est très difficile de le remanier ou de le régler plus tard.

- **Nous divisons les tâches à exécution longue en plusieurs entrées de journal.** Contrairement à ce qui est recommandé plus haut, si nous avons une tâche à exécution longue, de plusieurs minutes ou plusieurs heures, semblable à un flux de travail, nous pouvons décider d'émettre ponctuellement une entrée de journal distincte afin de déterminer si son exécution avance ou pourquoi elle est lente.
- **Nous enregistrons les détails sur la demande avant de passer à des actions telles que la validation.** Nous estimons qu'il est important, lors de résolutions et de journalisations d'audit, de journaliser assez d'informations sur les demandes pour savoir ce qu'elles ont tenté d'accomplir. De même, il convient de journaliser ces informations aussi tôt que possible, avant que la demande ne puisse être rejetée par la logique de validation, d'authentification ou de limitations. Si nous journalisons les informations d'une demande entrante, nous nous assurons de nettoyer l'entrée (encodage, échappement et tronquage) avant de la journaliser. Par exemple, nous ne voulons pas inclure des chaînes longues de 1 Mo dans notre entrée de journal de service si le mandataire en a passées. Sinon, le disque risquerait de se remplir et le stockage de journal risquerait de nous coûter plus que prévu. Un autre exemple de nettoyage est le filtrage des caractères de contrôle ASCII ou des séquences d'échappement pertinents dans le format de journal. Si le mandataire passait lui-même une entrée de journal de service et pouvait l'intégrer à nos journaux, cela serait déroutant ! Voir aussi https://xkcd.com/327/
- **Nous trouvons un moyen de journaliser avec plus de verbosité.** Pour la résolution de certains types de problèmes, le journal n'aura pas assez de détails sur les demandes qui posent problème pour trouver l'origine du problème. Ces informations peuvent se trouver dans le service, mais leur volume peut être trop grand pour justifier une journalisation permanente. Il peut être utile d'avoir un outil de configuration que vous pouvez appeler pour augmenter temporairement la verbosité du journal pendant que vous enquêtez sur le problème. Vous pouvez activer l'outil sur des hôtes, pour des clients individuels ou à un taux d'échantillonnage à travers la flotte. Il ne faut pas oublier de désactiver l'outil lorsque vous avez terminé.
- **Nous donnons des noms courts aux métriques (mais pas trop courts).** Amazon utilise la même sérialisation de journaux de service depuis plus de 15 ans. Avec cette méthode, chaque nom de compteur et minuteur est reproduit en texte brut dans chaque entrée de journal de service. Pour essayer de minimiser les frais de journalisation, nous utilisons des noms de minuteurs courts, mais explicites. Amazon commence à adopter de nouveaux formats de sérialisation basés sur un protocole de sérialisation binaire appelé Amazon Ion. Enfin, il est important de choisir un format que les outils d'analyse de journaux peuvent comprendre et qui est aussi facile que possible à sérialiser, à désérialiser et à stocker.
- **Nous nous assurons que les volumes de journalisation sont assez importants pour gérer la journalisation à un débit maximal.** Nous testons la charge de nos services à charge constante maximale (voire en surcharge) pendant des heures. Nous devons nous assurer que lorsque notre service gère un trafic en excès, il dispose encore de ressources pour envoyer les journaux externes au taux où ils produisent de nouvelles entrées de journal. Sinon, les disques finissent par se remplir. Vous pouvez également configurer la journalisation pour qu'elle s'effectue dans une autre partition de système de fichiers que la partition racine. Ainsi, le système ne plantera pas en cas de surcharge de journalisation. Nous verrons d'autres méthodes d'atténuation plus tard (par exemple, l'échantillonnage

dynamique proportionnel au débit), mais quelle que soit la stratégie utilisée, il est primordial de tester la charge.

- **Nous examinons le comportement du système lorsque le disque se remplit.** Lorsque le disque d'un système se remplit, le serveur ne peut pas journaliser dans le disque. Dans ce cas de figure, le service devrait-il arrêter d'accepter des demandes ou abandonner les journaux et continuer de fonctionner sans surveillance ? Fonctionner sans journalisation est risqué, c'est pourquoi nous testons les systèmes pour nous assurer que les serveurs avec des disques presque remplis sont détectés.
- **Nous synchronisons les horloges.** Nous savons que la notion d'« heure » dans les systèmes distribués est compliquée. Nous ne comptons pas sur la synchronisation des horloges des algorithmes distribués, mais celle-ci est nécessaire pour comprendre les journaux. Nous exécutons des démons comme Chrony ou ntpd pour la synchronisation des horloges, et nous surveillons les décalages d'horloges des serveurs. Pour mieux comprendre cette tâche, consultez le service de synchronisation temporelle Amazon.
- **Nous émettons le nombre zéro pour les métriques de disponibilité.** Les nombres d'erreurs sont utiles, mais les pourcentages d'erreurs peuvent l'être aussi. Pour ajouter un instrument pour une métrique de « pourcentage de disponibilité », nous avons découvert une stratégie utile qui consiste à émettre le nombre 1 lorsque la demande réussit et le nombre 0 lorsque la demande échoue. La statistique « moyenne » de la métrique alors obtenue correspond au taux de disponibilité. Émettre volontairement un point de données 0 peut être utile dans d'autres situations également. Par exemple, si une application réalise une sélection de leader, il peut être utile d'émettre un 1 périodiquement lorsqu'un processus est le leader et un 0 lorsque le processus n'est pas le leader pour surveiller la santé des suiveurs. Ainsi, si un processus arrête d'émettre un 0, il est plus facile de savoir que quelque chose a planté dans celui-ci et il ne pourra pas reprendre son travail si quelque chose arrive au leader.

Ce que nous journalisons

- **Nous journalisons la disponibilité et la latence de toutes les dépendances.** Nous trouvons cela particulièrement utile pour répondre aux questions comme « pourquoi la demande était-elle lente ? » ou « pourquoi la demande a-t-elle échoué ? ». Sans cette journalisation, nous ne pouvons que comparer des graphiques d'indépendances avec des graphiques de services et deviner si un pic de latence d'un service dépendant a entraîné l'échec d'une demande que nous examinons. De nombreux cadres de services et de clients sondent automatiquement les métriques, mais d'autres cadres (comme le kit SDK AWS, par exemple) nécessitent une instrumentation manuelle.
- **Nous divisons les métriques de dépendance par appel, ressource, code d'état et plus.** Si nous interagissons avec la même dépendance plusieurs fois dans la même unité de travail, nous incluons des métriques sur chaque appel séparément, ce qui permet de savoir avec quelle ressource chaque demande interagit. Par exemple, lorsqu'elles appellent Amazon DynamoDB, certaines équipes jugent utile d'inclure des métriques de minuteur et de latence par table et par code d'erreur, voire même par nombre de tentatives. Cela leur facilite la résolution dans les cas où le service est ralenti par des tentatives dues à des échecs de vérification conditionnelle. Ces métriques révèlent également les situations où les augmentations de la latence perçue par le client sont en fait dues aux tentatives de limitations ou à la pagination à travers un ensemble de résultats, et non à une perte de paquets ou à la latence réseau.

- **Nous enregistrons les profondeurs de files d'attente en mémoire lorsque nous y accédons.** Si une demande interagit avec une file d'attente et que nous sommes en train d'extraire un objet de la file ou d'en insérer un, nous enregistrons la profondeur de file d'attente actuelle dans l'objet de métriques pendant que nous y sommes. Pour les files d'attente en mémoire, obtenir ces informations n'est pas onéreux. Pour les files d'attente distribuées, ces métadonnées peuvent être disponibles gratuitement dans les réponses aux appels d'API. Cette journalisation aide à trouver les backlogs et les sources de latence à l'avenir. En outre, lorsque nous extrayons des éléments d'une file d'attente, nous mesurons la durée pendant laquelle ils étaient dans la file. Cela signifie que nous devons d'abord ajouter notre propre métrique de « durée en file d'attente » au message avant de remettre l'élément dans la file d'attente.
- **Nous ajoutons un compteur supplémentaire pour chaque raison d'erreur.** Pensez à ajouter du code qui compte la raison spécifique de l'erreur pour chaque demande échouée. Le journal d'application inclura les informations qui sont à l'origine de l'échec, ainsi qu'un message d'exception détaillé. Cependant, nous trouvons également cela utile de voir des tendances dans les raisons d'erreurs des métriques au fil du temps sans avoir à extraire ces informations dans le journal d'application. Commencer avec une métrique distincte pour chaque classe d'exception d'échec est utile.
- **Nous organisons les erreurs par catégorie de cause.** Si toutes les erreurs sont regroupées dans la même métrique, cette dernière devient bruyante et inutile. Nous pensons qu'il est important de séparer au moins les erreurs qui sont « dues au client » de celles qui sont « dues au serveur ». Cependant, une séparation encore plus poussée peut être utile. Par exemple, dans DynamoDB, les clients effectuent des demandes d'écriture conditionnelle qui renvoient une erreur si l'élément qu'ils modifient ne remplit pas les conditions préalables de la demande. Ces erreurs sont voulues et sont censées se produire de temps en temps. En revanche, les « demandes invalides » de la part des clients sont le plus souvent des bogues que nous devons réparer.
- **Nous journalisons les métadonnées importantes relatives à l'unité de travail.** Dans les journaux de métriques structurés, nous incluons également assez de métadonnées sur la demande afin de pouvoir déterminer plus tard qui avait effectué la demande et ce que la demande essayait d'accomplir. Cela inclut les métadonnées que les clients s'attendent à voir dans notre journal lorsqu'ils cherchent à résoudre des problèmes. Par exemple, DynamoDB journalise le nom de la table avec laquelle une demande interagit, ainsi que les métadonnées comme celles indiquant si l'opération de lecture était une lecture à cohérence ou non. Cependant, le service ne journalise pas les données stockées sur la base de données ou récupérées depuis celle-ci.
- **Nous protégeons les journaux avec des contrôles d'accès et des chiffrements.** Étant donné que les journaux contiennent des informations sensibles, nous prenons des mesures pour protéger et sécuriser ces données. Ces mesures incluent le chiffrement des journaux, la restriction de l'accès aux opérateurs qui résolvent les problèmes et le référencement régulier de cet accès.
- **Nous évitons d'ajouter des informations trop sensibles dans les journaux.** Les journaux doivent contenir des informations sensibles pour être utiles. Chez Amazon, il nous semble important que les journaux contiennent assez d'informations sur la provenance des demandes, mais nous excluons les informations trop sensibles, comme les paramètres des demandes qui n'affectent pas le routage ou le comportement du traitement des demandes. Par exemple, si le code convertit le message d'un client et que la conversion échoue, il est important de ne pas journaliser la charge utile afin de protéger la vie privée du client, même si cela peut compliquer les résolutions à l'avenir. Nous utilisons des outils

pour décider de ce qui peut être journalisé par choix ou non, afin d'empêcher la journalisation d'un nouveau paramètre sensible par la suite. Les services comme Amazon API Gateway permettent de configurer les données à inclure dans son journal d'accès, qui sert de mécanisme de choix efficace.

- **Nous journalisons un identifiant de trace et le propageons dans des appels backend.** Une demande de client donnée implique généralement plusieurs services travaillant en collaboration, allant de seulement deux ou trois services pour de nombreuses demandes AWS à bien plus de services pour les demandes amazon.com. Pour comprendre ce qu'il s'est passé lors de la résolution d'un système distribué, nous propageons le même identifiant de trace entre ces systèmes afin de pouvoir aligner les journaux de différents systèmes et voir où se sont produits les échecs. Un identifiant de trace est une sorte de méta-identifiant de demande qui est indiqué sur une unité de travail distribuée par le service de « porte d'entrée » qui était le point de départ de l'unité de travail. AWS X-Ray est un service d'aide qui fournit une partie de cette propagation. Nous pensons qu'il est important de transmettre la trace à notre dépendance. Dans un environnement multithread, faire en sorte que le cadre effectue cette propagation à notre place est très difficile et sujet aux erreurs, c'est pourquoi nous avons pris l'habitude de transmettre les identifiants de trace et d'autres contenus de demandes (comme un objet de métrique !) dans nos signatures de méthode. Nous jugeons également pratique de transmettre un objet de contexte dans nos signatures de méthode, pour ne pas avoir à effectuer de remaniement lorsque nous trouverons un modèle similaire à transmettre à l'avenir. Pour les équipes AWS, il ne s'agit pas simplement de résoudre les problèmes de leurs systèmes, mais également de permettre aux clients de résoudre leurs propres problèmes. Les clients comptent sur les traces AWS X-Ray transmises entre les services AWS lorsqu'ils interagissent entre eux au nom du client. Nous devons donc propager les identifiants de trace AWS X-Ray des clients entre les services afin qu'ils aient des données de trace complètes.
- **Nous journalisons différentes métriques de latence en fonction du code d'état et de la taille.** Les erreurs sont souvent rapides, comme les réponses aux accès refusés, aux limitations et aux erreurs de validation. Si des clients sont de plus en plus limités à une fréquence élevée, la latence peut sembler étonnamment faible. Pour éviter cette pollution de métrique, nous journalisons un minuteur distinct pour les réponses réussies, et nous nous concentrons sur cette métrique dans nos bureaux de bords et alarmes au lieu d'utiliser une métrique de temps générique. De même, si une opération peut être plus lente selon la taille de l'entrée ou la taille de la réponse, nous étudions la possibilité d'émettre une métrique de latence catégorisée, comme SmallRequestLatency et LargeRequestLatency. En outre, nous nous assurons que notre demande et nos réponses sont correctement limitées pour éviter des modes de baisse de tension et d'échec complexes, mais même dans un service soigneusement conçu, cette technique de compartimentation des métriques peut isoler le comportement du client et garder le bruit gênant hors des tableaux de bord.

Bonnes pratiques en matière de journaux d'applications

Cette section décrit les bonnes habitudes que nous avons acquises chez Amazon en matière de journalisation de données de journal de débogage non structurées.

- **Nous excluons les indésirables des journaux d'applications.** Même si nous avons des événements de journaux de niveaux INFO et DEBUG sur le chemin de demande pour aider au développement et au débogage dans les environnements de test, nous préférons désactiver ces niveaux de journaux en production. Au lieu de compter sur le journal d'application pour les informations de suivi des demandes, nous considérons le journal de service comme un emplacement des informations où nous pouvons produire des métriques et voir les tendances d'agrégation facilement au fil du temps. Cependant, cela ne présente pas que des avantages. Notre approche consiste à réviser en permanence nos journaux pour voir s'ils sont trop bruyants (ou trop peu bruyants) et à ajuster les niveaux des journaux progressivement. Par exemple, lorsque nous plongeons dans les journaux, nous trouvons souvent des événements de journaux qui sont trop bruyants ou des métriques que nous aimerions avoir. Heureusement, ces améliorations sont souvent faciles à apporter, c'est pourquoi nous avons pris l'habitude de remplir nos éléments de backlog de suivi rapide afin de garder nos journaux propres.
- **Nous incluons l'identifiant de demande correspondant.** Lorsque nous résolvons une erreur dans le journal d'application, nous souhaitons généralement voir les informations sur la demande ou le mandataire qui a déclenché l'erreur. Si les deux journaux contiennent le même identifiant de demande, nous pouvons facilement passer d'un journal à l'autre. Les bibliothèques de journaux d'applications écrivent l'identifiant de demande correspondant s'il est correctement configuré, et l'identifiant de demande est défini en tant que ThreadLocal. Si une application est multithread, il faut veiller particulièrement à définir l'identifiant de demande correct lorsqu'un thread commence à travailler sur une nouvelle demande.
- **Nous définissons une limite de taux pour une erreur indésirable dans le journal d'application.** Généralement, les services n'émettent pas grand chose dans le journal d'application, mais s'ils commencent tout à coup à présenter des volumes d'erreurs importants, ils peuvent se mettre à écrire un taux élevé d'entrées de journal de très grande taille avec des traces de pile. Nous avons réussi à contourner ce problème en définissant une limite de fréquence de journalisation d'un enregistreur donné.
- **Nous privilégions les chaînes de format à String#format ou à la concaténation de chaîne.** Les opérations d'API de journal d'application plus anciennes acceptent un seul message de chaîne au lieu de l'API de chaîne de format varargs de log4j2. Si le code est instrumenté avec des événements DEBUG, mais que la configuration est configurée au niveau ERROR, il est possible de perdre du travail en formatant des chaînes de message DEBUG qui sont ignorées. Certaines opérations d'API de journalisation prennent en charge la transmission d'objets arbitraires dont les méthodes toString() seront appelées uniquement si l'entrée de journal sera écrite.
- **Nous journalisons des identifiants de demande à partir d'appels de service échoués.** Si un service et appelé puis renvoie une erreur, le serveur renvoie généralement un identifiant de demande. Nous trouvons utile d'inclure l'identifiant de demande dans notre journal : ainsi, si nous devons contacter le propriétaire de ce service, nous pouvons lui permettre de trouver facilement ses propres entrées de journal de service correspondantes. Les erreurs de délai peuvent compliquer cette tâche si le service n'a pas encore renvoyé d'identifiant de demande ou si une bibliothèque client ne l'a pas encore analysé. Néanmoins, si nous recevons un identifiant de demande de la part du service, nous le journalisons.

Bonnes pratiques en matière de services à débit élevé

Pour la plupart des services chez Amazon, la journalisation de chaque demande ne génère pas de coûts excessifs. Les services à débit plus élevé entrent dans une zone plus grise, mais nous journalisons quand même souvent chaque demande. Par exemple, il est normal de penser que DynamoDB, avec ses pics de prise en charge de plus de 20 millions de demandes par seconde en comptant uniquement le trafic interne d'Amazon, ne journalise pas beaucoup, mais en réalité, il journalise chaque demande pour la résolution et à des fins d'audit et de conformité. Voici quelques conseils de haut niveau que nous utilisons chez Amazon pour rendre la journalisation plus efficace à un débit par hôte plus élevé :

- **Journaliser des échantillons.** Envisagez d'écrire toutes les N entrées, plutôt que chaque entrée. Chaque entrée inclut également le nombre d'entrées ignorées pour que les systèmes d'agrégation des métriques puissent estimer le volume de journaux réel dans les métriques qu'ils calculent. D'autres algorithmes d'échantillonnage comme l'échantillonnage de réservoir fournissent davantage d'échantillons représentatifs. D'autres algorithmes privilégient les erreurs de journalisation ou les demandes lentes par rapport aux demandes rapides et réussies. Cependant, avec l'échantillonnage, la capacité à aider les clients et à résoudre des échecs spécifiques est perdue. Certaines exigences en matière de conformité interdisent tout cela.
- **Décharger la sérialisation et journaliser les évacuations dans un autre thread.** Cette modification est couramment utilisée et facile à appliquer.
- **Renouveler fréquemment les journaux.** Le renouvèlement des journalisations des fichiers journaux toutes les heures peut sembler utile pour avoir moins de fichiers à traiter, mais en les renouvelant toutes les minutes, vous constaterez davantage d'améliorations. Par exemple, l'agent qui lit et compresse le fichier journal lira le fichier à partir du cache de page au lieu du disque, le CPU et les E/S des journaux de compression et d'envoi seront répartis sur l'heure au lieu de toujours se déclencher à la fin de l'heure.
- **Écrire des journaux pré-compressés.** Si un agent d'envoi des journaux compresse les journaux avant de les envoyer à un service d'archive, le CPU et le disque du système présenteront des pics réguliers. Il est possible d'amortir ce coût et de réduire les E/S du disque de moitié en diffusant des journaux compressés vers le disque. Cela présente toutefois des risques. Nous trouvons utile d'utiliser un algorithme de compression qui peut prendre en charge les fichiers tronqués si une application plante.
- **Écrire sur un disque virtuel/tmpfs.** Il peut être plus facile pour un service d'écrire les journaux sur la mémoire jusqu'à ce qu'ils soient envoyés hors du serveur, plutôt que de les écrire sur le disque. D'après notre expérience, cette méthode fonctionne mieux avec le renouvèlement de journaux toutes les minutes plutôt que toutes les heures.
- **Agrégations en mémoire.** Si vous devez prendre en charge des centaines de milliers de transactions par seconde dans une seule machine, il peut être onéreux d'écrire une seule entrée de journal par demande. Cependant, vous perdez beaucoup en observabilité en ignorant cette étape, c'est pourquoi nous trouvons utile de ne pas optimiser trop tôt.
- **Surveiller l'utilisation des ressources.** Nous faisons attention à ne pas atteindre certaines limites de mise à l'échelle. Nous mesurons nos E/S et CPU par serveur et la consommation de ces ressources par les agents de journalisation. Lorsque nous effectuons des tests de charge, nous les exécutons pendant assez longtemps pour pouvoir montrer que nos agents d'envoi des journaux peuvent suivre notre débit.

Avoir les bons outils d'analyse des journaux

Chez Amazon, nous exploitons les services que nous écrivons, nous devons donc tous apprendre à les résoudre aisément. Nous devons entre autres pouvoir analyser les journaux sans effort. Nous avons de nombreux outils à notre disposition, des analyses de journaux locales pour consulter des volumes relativement faibles de journaux à l'analyse de journaux distribués pour trier et agréger les résultats d'un très grand volume de journaux.

Nous jugeons important d'investir dans des outils et des runbooks d'analyse de journaux pour l'équipe. Si les journaux sont petits pour le moment, mais qu'un service est censé grandir petit à petit, nous surveillons le moment où nos outils actuels arrêteront d'arrêter, afin de pouvoir investir dans une solution d'analyse de journaux distribuée.

Analyse de journaux locale

Le processus d'analyse des journaux peut nécessiter une connaissance de différents utilitaires de ligne de commande Linux. Par exemple, la commande courante « trouver les premières adresses IP de communication dans le journal » est tout simplement :

```
cat log | grep -P "^RemoteIp=" | cut -d= -f2 | sort | uniq -c | sort -nr |
head -n20
```

Cependant, de nombreux autres outils sont utiles pour répondre à des questions plus complexes avec nos journaux, par exemple :

- jq : https://stedolan.github.io/jq/
- RecordStream : https://github.com/benbernard/RecordStream

Analyse de journaux distribuée

Tout service d'analyse de big data peut être utilisé pour effectuer des analyses de journaux distribuées (par exemple, Amazon EMR, Amazon Athena, Amazon Aurora et Amazon Redshift). Toutefois, certains services sont équipés de systèmes de journalisation, comme Amazon CloudWatch Logs.

- CloudWatch Logs Insights
- AWS X-Ray : https://aws.amazon.com/xray/
- Amazon Athena : https://aws.amazon.com/athena/

Conclusion

En tant que propriétaire de service et développeur logiciel, je passe énormément de temps à examiner les résultats de l'instrumentation (les graphiques sur les tableaux de bord, les fichiers journaux individuels) et à utiliser des outils d'analyse de journaux distribuée comme CloudWatch

Logs Insights. Voici quelques-unes de mes activités préférées. Lorsque j'ai besoin d'une pause après avoir terminé une tâche difficile, je recharge les batteries et me récompense en plongeant dans les journaux. Je démarre avec des questions comme « pourquoi cette métrique présente-t-elle un pic ici ? » ou « la latence de cette opération peut-elle être réduite ? ». Lorsque mes questions sont sans réponse, je trouve souvent des mesures qui pourraient être utiles dans le code. J'ajoute donc l'instrumentation, je la teste et je demande une vérification du code à mes coéquipiers.

Même si de nombreuses métriques sont fournies avec les services gérés que nous utilisons, nous devons beaucoup réfléchir à l'instrumentation de nos propres services afin d'avoir la visibilité nécessaire pour les utiliser correctement. Pendant les événements opérationnels, nous devoir rapidement déterminer l'origine des problèmes et les solutions pour les minimiser. Il est indispensable d'avoir les bonnes métriques sur nos tableaux de bord afin d'effectuer les diagnostics rapidement. En outre, étant donné que nous changeons nos services, ajoutons des fonctionnalités et modifions leur interaction avec nos dépendances en continu, mettre à jour et ajouter la bonne instrumentation est un exercice sans fin.

Liens

- « Look at your data » (Consultez vos données) par John Rauser, ancien collègue Amazon : https://www.youtube.com/watch?v=coNDCIMH8bk (à 13:22, il imprime littéralement les journaux pour mieux les examiner)
- « Investigating anomalies » (Rechercher les anomalies) par John Rauser, ancien collègue Amazon : https://www.youtube.com/watch?v=-3dw09N5_Aw
- « How humans see data » (Comment les hommes voient les données) par John Rauser, ancien collègue Amazon : https://www.youtube.com/watch?v=fSgEeI2Xpdc
- https://www.akamai.com/uk/en/about/news/press/2017-press/akamai-releases-spring-2017-state-of-online-retail-performance-report.jsp