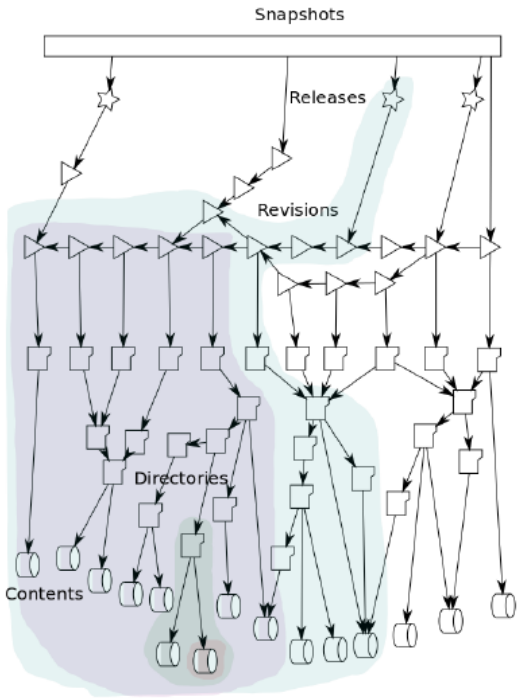
1. Introduction

Le logiciel est un pilier de la plupart des activités de recherche scientifique dans tous les domaines de notre vie et un médiateur pour accéder à toute information numérique. C'est un héritage humain qui doit être préservé des éléments supprimés, endommagés ou égarés. Spéciale ces dernières années, nous avons assisté à la fermeture de codes clés, exposant des centaines de milliers de projets de programmes publics accessibles au public. Nous devons créer un archivage universel du code source des logiciels, afin d'attirer l'attention sur la sécurité, la sécurité, la fiabilité et la traçabilité des logiciels.

Software Héritage est une initiative lancée par Inria, l'institut français de recherche en informatique et en automatisme, et soutenu par l'UNESCO dont le but de remplir cette mission avec trois objectifs principaux: collecter, conserver et partager le code source de tous les logiciels.

1.1. Description de Software Héritage

1.1.1.Data Model



Ils ont identifié les artefacts récurrents suivants fréquemment trouvés sur les lieux d'hébergement du code source: -

Contenu du fichier (Aussi connu comme «blobs»): contenu brut des fichiers (code source) sous forme d'une séquence d'octets, sans nom de fichier ni autre métadonnée. Le contenu des fichiers est souvent récurrent, par exemple, dans différentes versions du même logiciel, différents répertoires du même projet ou différents projets.

Répertoires -Directories -: liste d'entrées de répertoire nommées, chacune d'elles pointant vers d'autres artefacts, généralement le contenu d'un fichier ou des sous-répertoires. Les entrées de répertoire sont également associées à des métadonnées arbitraires, comprenant des bits d'autorisation, des horodatages de modification, etc.

Révisions (Aussi connu comme "commits"): le développement logiciel au sein d'un projet est une série de copies indexées dans le temps d'un seul répertoire "racine" contenant le code source complet du projet.

Le logiciel évolue lorsqu'un développeur modifie le contenu d'un ou de plusieurs fichiers de ce répertoire et enregistre ses modifications, chaque copie enregistrée du répertoire racine étant appelée une «révision». Il pointe

vers un répertoire entièrement déterminé et est équipé de métadonnées arbitraires. Certains sont ajoutés manuellement par le développeur, d'autres sont synthétisés automatiquement.

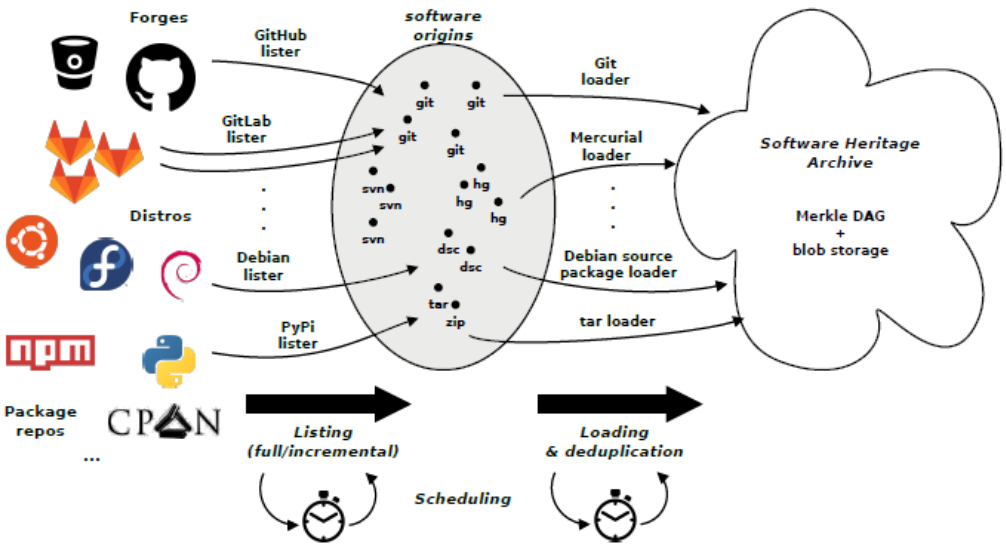
Releases (Aussi connu comme "tags"): les révisions qui sont plus égales que d'autres sont sélectionnées par les développeurs comme désignant des jalons appelés «versions». Chaque version pointe vers la dernière validation dans l'historique du projet correspondant à la version et peut contenir des métadonnées arbitraires.

En suite, les informations suivantes relatives à l'analyse sont stockées en tant qu'informations de provenance dans les archives Software Héritage:-

- Les **Origines** du logiciel sont des références précises sur le lieu de récupération des artefacts de code source archivés par Software Héritage. Ils prennent la forme de paires (type, url), où url est une URL canonique (par exemple, l'adresse à laquelle on peut cloner un git de référentiel ou wget une archive source) et taper le type d'origine du type de logiciel (par exemple, git, svn, ou pour les paquets sources Debian).
- Les **Projets** sont des entités abstraites, qui peuvent être imbriquées de manière arbitraire dans une hiérarchie de projet / sous-projet versionné, et qui peuvent être associées à des métadonnées arbitraires ainsi qu'à des origines où leur code source peut être trouvé.
- **Snapshots** n'importe quel type d'origine logicielle offrent de multiples pointeurs sur l'état «actuel» d'un projet de développement. Un "snapshot" d'une origine logicielle donnée enregistre tous les points d'entrée trouvés là-bas et où chacun d'eux pointait à l'époque. Par exemple, un objet instantané peut suivre la validation vers laquelle la branche principale pointait à un moment donné.
- Les **Visites** relient les origines logicielles avec des instantanés. Chaque fois qu'une origine est consultée, un nouvel objet de visite est créé, enregistrant quand (selon l'horloge de Software Héritage) la visite a eu lieu et l'instantané complet de l'état de l'origine du logiciel à ce moment-là.

1.1.2.Architecture et flux des données

Le modèle de données décrit et une architecture logicielle adaptée à l'intégration d'artefacts de code source ont été mis en œuvre dans le cadre de Software Héritage.



Ingestion data flow in Software Heritage

Ingestion agit comme la plupart des moteurs de recherche, explorant périodiquement un ensemble de "pistes" pour le contenu à archiver et les pistes suivantes. Pour faciliter l'ingestion, il a été divisé en deux phases conceptuelles: listing et loading.

- Listing

L'entrée de Listing est un lieu d'hébergement unique (par exemple, GitHub, PyPi, Listing ou Debian) et est chargée d'énumérer toutes les origines logicielles trouvées à ce moment-là. L'implémentation de listing varie selon les plates-formes d'hébergement, et des composants logiciels de lister dédiés doivent être implémentés pour chaque type de plate-forme (par exemple, des listers existent pour GitHub ou Bitbucket).
La Listing peut être entièrement réalisée: - en collectant en une fois la liste complète des origines disponibles sur un lieu d'hébergement donné, elle est utilisée au besoin pour s'assurer qu'aucune origine n'est négligée, mais elle risque d'être lourde si elle est effectuée trop souvent sur de grandes plates-formes .ou incrémental: - listing seulement les nouvelles origines depuis la dernière liste. Mettre à jour rapidement la liste des origines disponibles à ces endroits. En outre, la Listing peut être exécuté dans le style pull ou push. Mais ils envisagent de pousser une optimisation à ajouter en plus de la traction, afin de réduire les retards le cas échéant.

- Loading

Loading est responsable de l'ingestion réelle dans l'archive du code source Loading présent dans les origines logicielles connues. Les Loaders sont les composants logiciels chargés d'extraire les artefacts de code source des origines logicielles et de les ajouter à l'archive.
Les Loaders sont spécifiques à la technologie utilisée pour distribuer le code source: il y aura un Loader pour chaque type de système de contrôle de version (Git, Subversion, Mercurial, etc.) ainsi qu'un pour chaque format de paquet source (paquets source Debian, RPM source), tarballs, etc). Les Loaders dédupliquent nativement l'archive entière, ce qui signifie que tout artefact (contenu du fichier, révision, etc.) rencontré à une origine quelconque sera ajouté à l'archive uniquement si un nœud correspondant ne peut pas être trouvé dans l'archive dans son ensemble. Par exemple, lors de sa toute première rencontre, le chargeur Git chargera essentiellement tout son contenu de fichier, ses révisions, etc., dans les archives Software Héritage. Lors de la prochaine rencontre d'un référentiel identique, rien ne sera ajouté. Lors de la rencontre d'une copie légèrement différente, par exemple un dépôt contenant une douzaine d'engagements supplémentaires non encore intégrés à la version officielle de Linux, seuls les noeuds de révision correspondants, ainsi que le nouveau contenu du fichier et les nouveaux répertoires désignés par ceux-ci, seront chargés. Dans les archives.

- Scheduling

Listing et Loading se produisent périodiquement selon Schedule. Le composant Scheduling de Software Héritage est chargé de garder une trace du scheduler pour savoir quand les prochaines actions de Listing/Loading doivent avoir lieu. Software Héritage adopte une stratégie de retrait exponentiel, dans laquelle la période de visite est divisée par deux lorsque l'activité est signalée et doublée lorsqu'aucune activité n'a été observée. Actuellement, le site le plus rapide qui sera consulté est deux fois par jour et le plus lent tous les 64 jours.

- Archive

L'archive Software Héritage correspond à la structure de données Merkle DAG. L'archive est stockée à l'aide de technologies différentes en raison des différences de taille requise pour le stockage.
Les nœuds de contenu de fichier nécessitent le plus d'espace de stockage car ils contiennent tout le contenu de tous les fichiers de code source archivés. Ils sont donc stockés dans un stockage d'objet clé-valeur qui utilise comme clés les identifiants de nœud intrinsèques du Merkle DAG. Cela permet une distribution triviale du stockage d'objets sur plusieurs machines à des fins de performances et de redondance. Le reste du graphique est stocké dans une base de données relationnelle (SGBDR), avec

environ une table par type de nœud. Chaque table utilise comme clé primaire l'identifiant de nœud intrinsèque et peut facilement être partagée sur plusieurs serveurs. La réplication maître / esclave et la récupération à un point dans le temps peuvent être utilisées pour améliorer les performances et garantir la récupération. Point de faiblesse en deduplication qu'il est vulnérable à une collision hash: Si deux objets ont le même identifiant, il existe un risque différent de ne stocker qu'un seul d'entre eux avec la conviction qu'ils ont tous les deux stockés. Pour cette raison, ils utilisent plusieurs kits de test, avec des restrictions individuelles, pour détecter toutes les collisions avant d'ajouter de nouveaux artefacts à l'archive logicielle existante.
En plus du stockage d'objets, un composant logiciel d'archivage est chargé à la fois de l'application des règles de rétention et de la réparation automatique de la corruption des objets, le cas échéant, en raison, par exemple, de la dégradation des supports de stockage.

- L'archiveur vérifie également chaque copie de tous les objets connus - de manière aléatoire - et vérifie son intégrité en recalculant son identifiant intrinsèque et en le comparant à celui connu. En cas de non concordance, toutes les copies connues de l'objet sont à nouveau vérifiées à la volée, à condition qu'au moins une copie vierge soit trouvée, elle sera utilisée pour écraser les copies corrompues et les «soigner» automatiquement.

1.2. Contexte du TER

Le but de ce TER est de réaliser un listé pour un dépôt qui n'est pas encore traité par Software Héritage, nous avons réalisé un lister pour launchpad et le tester après que nous avons étudié les documentations, le tutoriel et les autres artefacts mis à disposition par Software Héritage, et étudier autre listers (Docker, Bitbucket, Github,..).

1.3. Plan du rapport

2. Résultats