Anatomie d'un référentiel OSTree

Table of Contents

Types d'objets principaux et modèle de données

OSTree est profondément inspiré par git; la couche principale est un système de fichiers de gestion des versions adressé au contenu de l'espace utilisateur. Cela vaut la peine de prendre un peu de temps pour vous familiariser avec Git Internals, car cette section supposera une certaine connaissance du fonctionnement de git.

Ses types d'objets sont similaires à git; il a des objets de validation et des objets de contenu. Git a des objets “arborescents”, tandis que OSTree les divise en objets “dirtree” et “dirmeta”. Mais contrairement à git, les sommes de contrôle d'OSTree sont SHA256. Et surtout, ses objets de contenu incluent uid, gid et des attributs étendus (mais toujours pas d'horodatage).

Objets commit

Un objet commit contient des métadonnées telles qu'un horodatage, un message de journal et, surtout, une référence à une paire de sommes de contrôle dirtree / dirmeta qui décrivent le répertoire racine du système de fichiers. Comme git, chaque commit dans OSTree peut avoir un parent. Il est conçu pour stocker un historique de vos builds binaires, tout comme git stocke un historique du contrôle de code source. Cependant, OSTree facilite également la suppression de données, en supposant que On peut les régénérer à partir du code source.

Objets Dirtree

Un dirtree contient un tableau trié de paires (nom de fichier, somme de contrôle) pour les objets de contenu, et un second tableau trié de (nom de fichier, somme de contrôle dirtree, somme de contrôle dirmeta), qui sont des sous-répertoires. Objets Dirmeta

Dans git, les objets arborescents contiennent les métadonnées telles que les autorisations pour leurs enfants. Mais OSTree le divise en un objet séparé pour éviter la duplication des listes d'attributs étendus.

Objets de contenu

Contrairement aux trois premiers types d'objet qui sont des métadonnées, conçus pour être mmap () ed, l'objet de contenu a un en-tête interne séparé et des sections de charge utile. L'en-tête contient uid, gid, mode et cible du lien symbolique (pour les liens symboliques), ainsi que des attributs étendus. Après l'en-tête, pour les fichiers normaux, le contenu suit.

Le format de données OSTree ne contient pas intentionnellement d'horodatages. Le raisonnement est que les fichiers de données peuvent être téléchargés à différents moments et par différents systèmes de construction, et auront donc des horodatages différents mais un contenu physique identique.

Cela pourrait entraîner des problèmes avec les programmes qui vérifient si les fichiers sont obsolètes en comparant les horodatages.

Pour Git, le choix logique est de ne pas jouer avec les horodatages, car une reconstruction inutile est meilleure qu'un arbre cassé.

Cependant, OSTree doit lier des fichiers en dur pour les extraire, et les validations sont supposées être cohérentes en interne sans étapes de génération nécessaires. Pour cette raison, OSTree agit comme si tous les horodatages étaient définis sur **time_t 0**, de sorte que les comparaisons seraient considérées comme à jour. Mais pour quelques versions, OSTree a utilisé **1** pour corriger des avertissements tels que GNU Tar émettant “un horodatage peu plausible” avec **0**; cependant pour la compatibilité, OSTree est rétabli pour utiliser à nouveau **0**.

Types et emplacements de référentiel

Contrairement à git, un référentiel OSTree peut être dans l'un des quatre modes distincts: **bare**, **bare-user**, **bare-user-only** et **archive**.

- Un référentiel **bare** est celui où les fichiers de contenu sont simplement stockés en tant que fichiers normaux; il est conçu pour être la source d'une «ferme de liens durs», où chaque extraction de système d'exploitation est simplement liée à celle-ci. Si vous souhaitez stocker des fichiers appartenant par exemple à root dans ce mode, vous devez exécuter OSTree en tant que root.
- **bare-user** est un ajout ultérieur qui est comme nu dans la mesure où les fichiers sont décompressés, mais il peut (et devrait généralement) être créé en tant que non-root. Dans ce mode, les métadonnées étendues telles que le propriétaire uid, gid et les attributs étendus sont stockées mais ne sont pas réellement appliquées. Le mode utilisateur nu est utile pour les systèmes de génération qui s'exécutent en tant que non root mais qui souhaitent générer du contenu appartenant à root, ainsi que les systèmes de conteneur non root.
- Il existe une variante du mode bare-user appelée **bare-user-only**. Contrairement à bare-user, ni la propriété ni les attributs étendus ne sont stockés. Ces dépôts sont destinés à être extraits en mode utilisateur (avec l'indicateur -U), où ces informations ne sont pas appliquées de toute

façon. Le principal avantage de bare-user-only est que les dépôts peuvent être stockés sur des systèmes de fichiers qui ne prennent pas en charge les attributs étendus, tels que tmpfs.
- En revanche, le mode **archive** est conçu pour être utilisé via HTTP simple. Comme les fichiers tar, il peut être lu/écrit par des utilisateurs non root.

Sur un système déployé par OSTree, le “référentiel système” est `/ostree/repo`. Il peut être lu par n'importe quel uid, mais uniquement écrit par root. La commande ostree fonctionnera par défaut sur le référentiel système; On peut fournir l'argument **–repo** pour remplacer cela ou définir la variable d'environnement **$OSTREE_REPO.**

Réfs

Comme git, OSTree utilise la terminologie “références” (en abrégé “refs”) qui sont des fichiers texte qui nomment (réfèrent) à des commits particuliers. Consultez la documentation de Git pour plus d'informations sur la façon dont git les utilise. Contrairement à git, il n'est généralement pas logique d'avoir une branche “master”. Il existe une convention pour les références dans OSTree qui ressemble à ceci: `exampleos/buildmaster/x86_64-runtime` et `exampleos/buildmaster/x86_64-devel-debug`. Ces deux références pointent vers deux arborescences de systèmes de fichiers générées différentes. Dans cet exemple, l'arborescence “runtime” contient juste assez pour exécuter un système de base, et “devel-debug” contient tous les outils de développement et debuginfo.

L'ostree prend en charge une syntaxe simple utilisant le signe ^ pour faire référence au parent d'un commit donné. Par exemple, `exampleos/buildmaster/x86_64-runtime^` fait référence à la version précédente, et `exampleos/buildmaster/x86_64-runtime^^` fait référence à la précédente.

Le fichier récapitulatif

Un ajout ultérieur à OSTree est le concept d'un fichier “résumé”, créé via la commande **ostree summary -u**. Cela a été introduit pour plusieurs raisons. Un cas d'utilisation principal doit être compatible avec Metalink, qui nécessite un fichier unique avec une somme de contrôle connue comme cible.

Le fichier récapitulatif contient principalement deux mappages:

- Un mappage des références et de leurs sommes de contrôle, équivalent à récupérer le fichier de référence individuellement
- Une liste de tous les deltas statiques, avec leurs sommes de contrôle des métadonnées

Cela signifie actuellement qu'il se développe de façon linéaire avec les deux éléments. En revanche, à l'aide du fichier récapitulatif, un client peut énumérer les branches.

De plus, la récupération du fichier récapitulatif par exemple TLS épinglé crée une vérification de bout en bout solide du commit ou du delta statique.

Le fichier récapitulatif peut également être signé GPG (détaché). C'est actuellement le seul moyen de fournir (de manière transitoire) des signatures GPG sur les deltas.

Si un administrateur de référentiel crée un fichier récapitulatif, il doit ensuite exécuter ostree summary -u pour le mettre à jour chaque fois qu'une référence est mise à jour ou qu'un delta statique est généré.

Déploiements

Aperçu

Construit au-dessus du noyau du système de fichiers versioning OSTree est une couche qui sait comment déployer, installer en parallèle et gérer les systèmes d'exploitation de type Unix (accessible via ostree admin). Le contenu principal de ces systèmes d'exploitation est traité en lecture seule, mais ils partagent de manière transparente le stockage.

Un déploiement est physiquement situé à un chemin de la forme `/ostree/deploy/$stateroot/deploy/$checksum`. OSTree est conçu pour démarrer directement dans exactement un déploiement à la fois; chaque déploiement est destiné à être une cible pour chroot () ou équivalent.

"stateroot" ("osname"): groupe de déploiements qui partagent /var

Chaque déploiement est regroupé dans exactement un “stateroot” (également appelé “osname”); l'ancien terme est préféré.

D'en haut, On peut voir qu'un stateroot est physiquement représenté dans le répertoire `/ostree/deploy/$stateroot`. Par exemple, OSTree peut autoriser l'installation parallèle de Debian dans `/ostree/deploy/debian` et Red Hat Enterprise Linux dans `/ostree/deploy/rhel` (sous réserve de la prise en charge du système d'exploitation, les versions publiées actuelles de ces systèmes d'exploitation peuvent ne pas la prendre en charge).

Chaque stateroot possède exactement une copie du traditionnel Unix `/var`, stocké physiquement dans `/ostree/deploy/$stateroot/var`. OSTree fournit des outils de prise en charge pour systemd pour créer un montage de liaison Linux qui garantit que le déploiement démarré voit la copie partagée de `/var`.

OSTree ne touche pas le contenu de `/var`. Les composants du système d'exploitation tels que les services démon sont nécessaires pour créer tous les répertoires dont ils ont besoin au moment de l'exécution (par exemple `/var/cache/$daemonname`), et pour gérer la mise à niveau des formats de données à l'intérieur de ces répertoires.

Contenu d'un déploiement

Un déploiement commence par une validation spécifique (représentée comme un hachage SHA256)

dans le référentiel OSTree dans `/ostree/repo`. Cette validation fait référence à une arborescence de systèmes de fichiers qui représente la base sous-jacente d'un déploiement. Pour faire court, nous l'appellerons “l'arbre”, pour le distinguer du concept de déploiement.

Tout d'abord, l'arborescence doit inclure un noyau (et éventuellement un initramfs). Les emplacements standard actuels sont:

- `/usr/lib/modules/$kver/vmlinuz`
- `/usr/lib/modules/$kver/initramfs.img`

La **somme de contrôle de démarrage** sera calculée automatiquement. Ceci suit la disposition actuelle du noyau Fedora et est le chemin actuellement recommandé. Cependant, les anciennes versions de libostree ne le prennent pas en charge; il faudra peut-être également placer les noyaux dans les chemins précédents (hérités), qui sont vmlinuz `vmlinuz(-.*)?-$checksum` dans `/boot` ou `/usr/lib/ostree-boot`.

La somme de contrôle doit être un hachage SHA256 du contenu du noyau; il doit être pré-calculé avant de stocker le noyau dans le référentiel. Facultativement, le répertoire peut également contenir un **initramfs** `initramfs(-.*)?-$checksum` et/ou une arborescence de périphériques, stocké sous `devicetree(-.*)?-$checksum`.

S'il existe un **initramfs** ou un **devicetree**, la somme de contrôle doit inclure tout le contenu du noyau, **initramfs** et **devicetree**. OSTree l'utilisera pour déterminer quels noyaux sont partagés. La raison en est d'éviter le calcul des sommes de contrôle sur le client par défaut.

Le déploiement ne doit pas avoir un UNIX `/etc` traditionnel; à la place, il doit inclure `/usr/etc`. Il s'agit de la “configuration par défaut”. Lorsque OSTree crée un déploiement, il effectue une fusion à trois en utilisant l'ancienne configuration par défaut, le système actif `/etc` et la nouvelle configuration par défaut. Dans l'arborescence finale du système de fichiers pour un déploiement, `/etc` est un répertoire normal accessible en écriture.

Outre les exceptions de `/var` et `/etc`, le reste du contenu de l'arborescence est extrait sous forme de liens physiques dans le référentiel. Il est fortement recommandé que les systèmes d'exploitation expédient tout leur contenu dans `/usr`, mais ce n'est pas une exigence difficile.

Enfin, un déploiement peut avoir un fichier .origin, stocké à côté de son répertoire. Ce fichier indique à ostree admin upgrade comment le mettre à niveau. Pour le moment, OSTree ne prend en charge que la mise à niveau d'une seule spécification de référence. Cependant, à l'avenir, OSTree pourrait prendre en charge une syntaxe pour la composition de couches d'arbres, par exemple.

Le système /boot

Alors que OSTree Parallel installe proprement les déploiements dans le répertoire `/ostree`, il doit finalement contrôler le répertoire /boot du système. Cela fonctionne via la spécification du chargeur de démarrage, qui est une norme pour les fichiers de configuration de dépôt indépendants du chargeur de démarrage.

Lorsqu'une arborescence est déployée, un fichier de configuration est généré sous la forme `/boot/loader/entries/ostree-$stateroot-$checksum.$serial.conf`. Ce fichier de configuration comprendra un argument spécial **ostree=kernel** qui permet aux initramfs de trouver (et chroot() dans) le déploiement spécifié.

À l'heure actuelle, tous les chargeurs de démarrage n'implémentent pas BootLoaderSpec, donc OSTree contient du code pour certains d'entre eux pour régénérer les fichiers de configuration natifs (tels que `/boot/syslinux/syslinux.conf`) en fonction des entrées.

Adapter les distributions traditionnelles existantes

Disposition du système

D'abord, OSTree encourage les systèmes à implémenter UsrMove C'est simplement pour éviter d'avoir besoin de plus de montages de liaison. Par défaut, le crochet dracut d'OSTree crée un montage de liaison en lecture seule sur `/usr`; On peut bien sûr générer des montages de liaison individuels pour `/bin`, toutes les variantes `/lib`, etc. Ce n'est donc pas une exigence difficile.

Rappelez-vous, parce que par défaut le système est démarré dans un équivalent chroot, il doit y avoir un moyen de reportez-vous au système de fichiers racine physique réel. Par conséquent, l'arborescence de votre système d'exploitation doit contenir un répertoire `/sysroot` vide; au démarrage, OSTree en fera un montage de liaison vers le répertoire physique `/root`. Il existe un précédent pour ce nom dans le contexte initramfs. De plus, vous devez créer un lien symbolique de haut niveau `/ostree` qui pointe vers `/sysroot/ostree`, afin que l'outil OSTree au moment de l'exécution puisse trouver de manière cohérente les données système, qu'il fonctionne sur une racine physique ou à l'intérieur d'un déploiement.

Comme OSTree ne conserve `/var` que sur les mises à niveau (le répertoire chroot de chaque déploiement sera éventuellement récupéré), vous devrez choisir comment gérer les autres répertoires inscriptibles de niveau supérieur spécifiés par la norme de hiérarchie du système de fichiers. Votre système d'exploitation peut bien sûr choisir de ne pas prendre en charge certains d'entre eux tels que `/usr/local`, mais voici l'ensemble recommandé:

```
/home      → /var/home
/opt       → /var/opt
/srv       → /var/srv
/root      → /var/roothome
/usr/local → /var/usrlocal
/mnt       → /var/mnt
/tmp       → /sysroot/tmp
```

De plus, comme `/var` est vide par défaut, votre système d'exploitation devra créer dynamiquement les cibles de ceux-ci au démarrage. Une bonne façon de procéder consiste à utiliser systemd-tmpfiles, si votre système d'exploitation utilise systemd. Par exemple:

```
d /var/log/journal 0755 root root -
L /var/home - - - - ../sysroot/home
d /var/opt 0755 root root -
d /var/srv 0755 root root -
d /var/roothome 0700 root root -
d /var/usrlocal 0755 root root -
d /var/usrlocal/bin 0755 root root -
d /var/usrlocal`/etc` 0755 root root -
d /var/usrlocal/games 0755 root root -
d /var/usrlocal/include 0755 root root -
d /var/usrlocal/lib 0755 root root -
d /var/usrlocal/man 0755 root root -
d /var/usrlocal/sbin 0755 root root -
d /var/usrlocal/share 0755 root root -
d /var/usrlocal/src 0755 root root -
d /var/mnt 0755 root root -
d /run/media 0755 root root - -
```

Notons ici en particulier la double indirection de `/home`. Par défaut, chaque déploiement partagera le répertoire global de haut niveau `/home` sur le système de fichiers racine physique. Il appartient ensuite à des niveaux supérieurs d'outils de gestion de maintenir `/etc/passwd` ou équivalent synchronisé entre les systèmes d'exploitation. Chaque déploiement peut facilement être reconfiguré pour avoir son propre répertoire personnel défini simplement en faisant de `/var/home` un vrai répertoire.

Amorçage et technologie initramfs

OSTree est livré avec le code d'intégration facultatif dracut + systemd qui suit cette logique:

- Analyser l'argument de ligne de commande **ostree=kernel** dans les initramfs
- Configurer un montage de liaison en lecture seule sur /usr
- Lier le `/sysroot` du déploiement au physique `/`
- Utiliser mount (MS_MOVE) pour faire apparaître la racine de déploiement comme le système de fichiers root

Après ces étapes, systemd bascule root.

Si vous n'utilisez pas dracut ou systemd, utiliser OSTree devrait toujours être possible, mais vous devrez écrire le code d'intégration. Voir les sources existantes dans `src/switchroot` comme référence, ainsi que `src/switchroot/switchroot.sh`.

Les correctifs pour prendre en charge d'autres technologies et systèmes initramfs, s'ils sont suffisamment propres, seront probablement acceptés en amont.

Une autre note spécifique concernant sysvinit: OSTree utilisé pour prendre en charge les fichiers de périphérique d'enregistrement tels que `/dev/initctl FIFO`, mais ne le fait plus. Il est recommandé de simplement patcher vos initramfs pour le créer au démarrage.

/usr/lib/passwd

Contrairement aux systèmes de packages traditionnels, les arbres OSTree contiennent des uid et des gids numériques. De plus, il n'a pas de mécanisme de type% post où useradd pourrait être invoqué. Pour expédier un système d'exploitation qui contient à la fois des utilisateurs système et des utilisateurs créés dynamiquement sur des machines clientes, vous devrez choisir une solution pour `/etc/passwd`. Le problème principal est que si vous ajoutez un utilisateur au système pour un démon, le processus de mise à niveau OSTree pour `/etc` remarquera simplement que parce que `/etc/passwd` diffère de la valeur par défaut précédente, il conservera le fichier de configuration modifié et votre nouveau L'utilisateur du système d'exploitation ne sera pas visible. La solution choisie pour le système d'exploitation Gnome Continuous est de créer `/usr/lib/passwd`, et d'inclure un module NSS nss-altfiles qui demande à la glibc de lire à partir de celui-ci. Ensuite, le système de construction place tous les utilisateurs du système là-bas, libérant `/etc/passwd` pour être purement une base de données d'utilisateurs locaux. Voir également un effort plus récent de Systemd Stateless

Adapter les gestionnaires de packages existants

Le plus gros effort est probablement de repenser le gestionnaire de paquets de votre distribution pour qu'il soit au-dessus d'OSTree, en particulier si vous souhaitez conserver la compatibilité avec «l'ancienne» méthode d'installation dans le /. Cette section utilisera des exemples à la fois de dpkg et de rpm car l'auteur est familier avec les deux; mais les concepts abstraits devraient s'appliquer à la plupart des gestionnaires de paquets traditionnels.

Il existe de nombreux niveaux d'intégration possibles; dans un premier temps, nous décrirons l'implémentation la plus naïve qui est la plus simple mais aussi la moins efficace. Nous supposerons ici que l'administrateur est démarré dans un système compatible OSTree et souhaite ajouter un ensemble de packages.

De nombreux gestionnaires de packages stockent leur état dans /var; mais puisque dans le modèle OSTree ce répertoire est sharEntre les versions indépendantes, la base de données du package doit d'abord être trouvée dans le répertoire per-deployment /usr. Il devient en lecture seule; rappelez-vous, toutes les mises à niveau impliquent la construction d'une nouvelle arborescence de système de fichiers, donc votre gestionnaire de packages devra également créer une copie de sa base de données. Très probablement, si vous souhaitez continuer à prendre en charge les déploiements non OSTree, demandez simplement à votre gestionnaire de packages de revenir à l'emplacement hérité `/var` si celui de `/usr` n'est pas trouvé.

Pour installer un ensemble de nouveaux packages (sans supprimer les packages existants), énumérez l'ensemble des packages dans le déploiement actuellement démarré et effectuez une résolution des dépendances pour calculer l'ensemble complet des nouveaux packages. Téléchargez et décompressez ces nouveaux packages dans un répertoire temporaire.

Maintenant, parce que nous installons simplement de nouveaux packages et ne supprimons rien, nous pouvons faire l'optimisation majeure de la réutilisation de notre arborescence de systèmes de fichiers existante, et simplement superposer l'arborescence du système de fichiers composée de ces nouveaux packages sur le dessus. Une commande comme celle-ci:

```
ostree commit -b osname/releasename/description
--tree=ref=$osname/$releasename/$description
--tree=dir=/var/tmp/newpackages.13A8D0/
```

va créer un nouveau commit dans la branche `$osname/$releasename/$description`. La somme de contrôle OSTree SHA256 de tous les fichiers dans /var/tmp/newpackages.13A8D0/ sera calculée, mais nous ne revérifierons pas l'arborescence existante. Dans ce modèle de superposition, les répertoires antérieurs auront priorité, mais les fichiers des calques ultérieurs remplaceront silencieusement les calques antérieurs.

Ensuite, pour déployer réellement cette arborescence pour le prochain démarrage: ostree admin deploy $osname/$releasename/$description

C'est essentiellement ce que fait rpm-ostree pour prendre en charge son modèle de superposition de packages.

Formats de données OSTree

Sur le thème des "serveurs intelligents"

Une différence vraiment cruciale entre OSTree et git est que git a un “serveur intelligent”. Même lors de la récupération sur https://, ce n'est pas seulement un serveur Web statique, mais celui qui, par exemple calcule et compresse dynamiquement les fichiers pack pour chaque client.

En revanche, l'auteur d'OSTree estime que pour les mises à jour du système d'exploitation, de nombreux déploiements voudront utiliser de simples serveurs Web statiques, la même cible que la plupart des systèmes de packages ont été conçus pour utiliser. Les principaux avantages sont la sécurité et l'efficacité du calcul. Des services comme Amazon S3 et CDN sont une cible canonique, ainsi qu'un serveur nginx statique stock.

Le format d'archive

Dans la section repo, le concept d'objets a été introduit, où les objets fichier/contenu sont additionnés et gérés individuellement. (Contrairement à un système de package, qui fonctionne sur des agrégats compressés).

Le format d'archive compresse simplement gzip chaque objet de contenu. Les objets de métadonnées sont stockés non compressés. Cela signifie qu'il est facile de servir via HTTP statique ce format était appelé archive-z2 pour des raisons historiques.

Lorsqu'on valide un nouveau contenu, on verra de nouveaux fichiers .filez apparaître dans les objets /.

efficacité des archives

Les avantages de l'archive:

- C'est facile à comprendre et à mettre en œuvre
- Peut être servi directement sur HTTP simple par un serveur Web statique
- Les clients peuvent télécharger/décompresser les mises à jour de manière incrémentielle
- Gain d'espace sur le serveur

Le plus gros inconvénient de ce format est que pour qu'un client effectue une mise à jour, une requête HTTP par fichier modifié est requise. Dans certains scénarios, cela n'est en fait pas mal du tout, en particulier avec des techniques pour réduire la surcharge HTTP, telles que HTTP/2.

Pour que ce format fonctionne correctement, vous devez concevoir votre contenu de telle sorte que les grandes données qui changent rarement (par exemple les images graphiques) soient stockées séparément des petites données changeant fréquemment (code d'application).

Autres inconvénients de l'archive:

- C'est assez mauvais lorsque les clients effectuent un pull initial (sans HTTP/2),
- On ne connaît pas la taille totale (compressée ou non compressée) du contenu avant de tout télécharger

Mis à part: les formats bare et bare-user

L'opération la plus courante consiste à extraire un référentiel d'archives dans un référentiel formaté nu ou utilisateur nu. Ces deux derniers ne sont pas compressés sur le disque. En d'autres termes, les tirer est similaire à décompresser (mais pas à installer) un paquet RPM / deb.

Le format utilisateur nu est un peu spécial en ce que l'uid / gid et les xattrs du contenu sont ignorés. Ceci est principalement utile si vous souhaitez avoir le même contenu géré par OSTree qui peut être exécuté sur un système hôte ou un conteneur non privilégié. Deltas statiques

OSTree lui-même était initialement axé sur un modèle de livraison continue, dans lequel les systèmes clients sont censés être mis à jour régulièrement. Cependant, de nombreux fournisseurs de systèmes d'exploitation souhaitent fournir un contenu mis à jour, par exemple une fois par mois ou moins souvent.

Pour ce modèle, nous pouvons faire beaucoup mieux pour prendre en charge les mises à jour par lots qu'un dépôt d'archives de base. Cependant, nous voulons toujours conserver le modèle de “serveur Web statique uniquement”. De ce fait, OSTree a acquis le concept de “delta statique”.

Ces deltas sont censés être un delta entre deux objets de validation spécifiques, y compris les deltas “bsdiff” et “rsync-style” dans un objet de contenu. Les deltas statiques sont également pris en charge à partir de NULL, où le client peut télécharger plus efficacement un objet de validation à partir de zéro - cela est surtout utile lors de l'utilisation d'OSTree pour les conteneurs, plutôt que des images du système d'exploitation. Pour les images du système d'exploitation, on a tendance à télécharger une image ISO ou qcow2 du programme d'installation qui est un fichier unique qui contient déjà les données de l'arborescence.

En effet, nous dépensons du stockage côté serveur (et des coûts de calcul uniques) et gagnons en efficacité dans la bande passante du réseau client.

Disposition du référentiel delta statique

Étant donné que les deltas statiques peuvent ne pas exister, le client doit d'abord essayer d'en localiser un. Supposons qu'un client veuille récupérer la validation $ {new} alors qu'il exécute actuellement $ {current}.

La première chose à comprendre est que pour économiser de l'espace, ces deux validations sont “base64 modifiée” - le caractère / est remplacé par _.

Comme les objets de validation, un “répertoire de préfixes” est utilisé pour faciliter la gestion des outils du système de fichiers

Un delta est nommé `$(mbase64 $from)-$(mbase64 $to)`, par exemple

```
GpTyZaVut2jXFPWnO4LJiKEdRTvOw_mFUCtIKW1NIX0-
L8f+VVDkEBKNc1Ncd+mDUrSVR4EyybQGCkuKtkDnTwk
```

, qui est au format SHA256

```
1a94f265a56eb768d714f5a73b82c988a11d453bcec3f985502b48296d4d217d-2fc7fe5550e
410128d73535c77e98352b495478132c9b4060a4b8ab640e74f09`.
```

Enfin, le contenu réel peut être trouvé dans `deltas/$fromprefix/$fromsuffix-$to`.

Structure interne delta statique

Un delta est lui-même un répertoire. À l'intérieur, il y a un fichier appelé superbloc qui contient des métadonnées. Les autres fichiers seront des entiers portant des packs de contenu.

Le format de fichier des deltas statiques doit actuellement être considéré comme un détail d'implémentation OSTree. De toute évidence, rien n'empêche d'écrire du code compatible avec OSTree aujourd'hui. Cependant, nous aimerions avoir la souplesse nécessaire pour étendre et changer les choses, et havL'utilisation de plusieurs bases de code rend cela plus problématique. Veuillez contacter les auteurs pour toute demande.

Cela dit, une chose essentielle à comprendre à propos de la conception est que les charges utiles delta ressemblent un peu plus à des “programmes restreints” qu'à des données brutes. Il y a une phase de “compilation” qui génère une sortie que le client exécute.

Ce modèle «met à jour en tant que code» permet plusieurs stratégies de génération de contenu. La conception de celui-ci a été inspirée de celle de Chromium: ChromiumOS Autoupdate.

Le superbloc delta

Le superbloc contient:

- métadonnées arbitraires
- horodatage de génération delta
- le nouvel objet commit
- Un tableau de deltas récursifs à appliquer
- Un tableau de métadonnées par partie, y compris la taille totale des objets (compressés et non compressés),
- Un tableau d'objets de secours

Définissons une partie delta, puis revenons pour discuter des détails:

Une partie delta

Une partie delta est une combinaison d'un blob brut de données, plus un bytecode très restreint qui fonctionne sur elle. Disons par exemple que deux fichiers se trouvent partager une section commune. Il est possible que la compilation delta inclue cette section une fois dans le blob de données delta, puis génère des instructions pour écrire ce blob deux fois lors de la génération des deux objets.

En réalité, cependant, il est très courant que la majeure partie d'un delta soit simplement un “flux de nouveaux objets” - si l'on y réfléchit, cela n'a pas de sens d'avoir trop de duplication dans le contenu du système d'exploitation à ce niveau.

Alors, ce qui est plus intéressant, c'est que les deltas statiques OSTree prennent en charge un algorithme delta par fichier appelé bsdiff qui fonctionne particulièrement bien sur le code exécutable.

Le compilateur delta actuel recherche les fichiers avec des noms de base correspondants dans chaque commit qui ont une taille similaire, et essaie un bsdiff entre eux. (Il serait logique plus tard qu'un système de génération fournisse un indice pour cela - par exemple, des fichiers dans un même package).

Un bsdiff généré est inclus dans le blob de charge utile, et son application est une instruction.

Objets de secours

Il est possible qu'il y ait des fichiers de grande taille qui pourraient être résistants à bsdiff. Un bon exemple est qu'il est courant pour les systèmes d'exploitation d'utiliser un “initramfs”, qui est lui-même un système de fichiers compressé. Cette “compression interne” bat l'analyse bsdiff.

Pour ces types d'objets, le superbloc delta contient un tableau “d'objets de secours”. Ces objets ne sont pas inclus dans les parties delta - le client les récupère simplement de l'objet .filez sous-jacent.

Rédaction d'un buildsystem et gestion des référentiels

OSTree n'est pas un système de package. Il ne prend pas directement en charge la création de code source. Il s'agit plutôt d'un outil de transport et de gestion de contenu, ainsi que d'aspects indépendants du système de package comme la gestion du chargeur de démarrage pour les mises à jour.

Nous supposerons ici que nous prévoyons de générer des validations sur un serveur de génération, puis que les systèmes clients le répliquent. Faire un assemblage côté client est également possible bien sûr, mais cette discussion se concentrera principalement sur les problèmes côté serveur.

Initialisation

Pour cette discussion initiale, nous supposons que vous avez un seul référentiel d'archives:

```
mkdir repo
ostree --repo=repo init --mode=archiv
```

On peut l'exporter via un serveur Web statique et configurer les clients pour qu'ils en tirent.

Écrire son propre système de construction OSTree

Il existe de nombreux systèmes qui suivent essentiellement ce modèle:

```
$pkg --installroot=/path/to/tmpdir install foo bar baz
$imagesystem commit --root=/path/to/tmpdir
```

Pour diverses valeurs de $pkg telles que yum, apt-get, etc., et les valeurs de $imagesystem peuvent être de simples tarballs, Amazon Machine Images, ISO, etc.

Évidemment, dans ce document, nous allons parler de la situation où $imagesystem est OSTree. L'idée générale avec OSTree est que, où que vous puissiez stocker une série de tarballs pour des applications ou des images de système d'exploitation, OSTree va probablement être meilleur. Par exemple, il prend en charge les signatures GPG, les deltas binaires, l'écriture de la configuration du chargeur de démarrage, etc.

OSTree n'inclut pas de système de construction de packages/composants simplement parce qu'il en existe déjà de nombreux - il est plutôt destiné à fournir une couche d'infrastructure.

L'arbre de composition rpm-ostree mentionné ci-dessus choisit RPM comme valeur de $pkg - donc les binaires sont construits en tant que RPM, puis validés dans leur ensemble dans une validation OSTree.

Mais discutons de la construction de la nôtre. Si vous ne faites qu'expérimenter, il est assez facile de commencer avec la ligne de commande. Nous supposerons à cet effet que vous avez un processus de construction qui génère une arborescence de répertoires - nous appellerons cet outil $pkginstallroot (qui pourrait être yum –installroot ou debootstrap, etc.).

Votre prototype initial va ressembler à:

```
$pkginstallroot /path/to/tmpdir
ostree --repo=repo commit -s 'build' -b exampleos/x86_64/standard --
tree=dir=/path/to/tmpdir
```

Alternativement, si votre système de génération peut générer une archive tar, On peut valider cette archive dans OSTree. Par exemple, OpenEmbedded peut sortir un tarball, et on peut le valider via:

```
ostree commit -s 'build' -b exampleos/x86_64/standard --tree=tar=myos.tar
```

Construire des arbres à partir des unions

Ce qui précède est un modèle très simpliste, et vous remarquerez rapidement qu'il est lent. En effet, OSTree doit recalculer et recompresser le contenu chaque fois qu'il est validé. (La majeure partie du temps CPU est consacrée à la compression qui est jetée si le contenu s'avère déjà stocké).

Une approche plus avancée consiste à stocker les composants dans OSTree lui-même, puis à les associer et à les réengager. À ce stade, nous vous recommandons de jeter un œil à l'API OSTree et de choisir un langage de programmation pris en charge par GObject Introspection pour écrire vos scripts de buildsystem. Python peut être un bon choix, ou On peut choisir un code C personnalisé, etc.

Pour les besoins de ce tutoriel, nous utiliserons un script shell, mais il est fortement recommandé de choisir un véritable langage de programmation pour votre système de build.

Supposons que votre système de génération génère des artefacts distincts (qu'il s'agisse de RPM, de fichiers zip ou autre). Ces artefacts doivent être le résultat de make install DESTDIR= ou similaire. Essentiellement équivalent aux RPM / debs.

De plus, afin d'accélérer les choses, nous aurons besoin d'un référentiel séparé pour les utilisateurs nus afin d'effectuer rapidement les extractions via des liens physiques. Nous exporterons ensuite le contenu dans le référentiel d'archives pour une utilisation par les systèmes clients.

```
mkdir build-repo
ostree --repo=build-repo init --mode=bare-user
```

On peut commencer à les valider en tant que branches individuelles:

```
ostree --repo=build-repo commit -b exampleos/x86_64/bash --
tree=tar=bash-4.2-bin.tar.gz
ostree --repo=build-repo commit -b exampleos/x86_64/systemd --
```

```
tree=tar=systemd-224-bin.tar.gz
```

Configurez les choses de sorte que chaque fois qu'un package change, vous refaites la validation avec la nouvelle version du package - conceptuellement, la branche suit les versions individuelles du package au fil du temps, et par défaut “dernière”. Ce n'est pas obligatoire - on pourrait également inclure la version dans le nom de la branche et avoir des métadonnées à l'extérieur pour déterminer la “dernière” (ou la version souhaitée).

Maintenant, pour construire l'arbre final:

```
rm -rf exampleos-build
for package in bash systemd; do
  ostree --repo=build-repo checkout -U --union exampleos/x86_64/${package}
exampleos-build
done
# Set up a "rofiles-fuse" mount point; this ensures that any processes
# we run for post-processing of the tree don't corrupt the hardlinks.
mkdir -p mnt
rofiles-fuse exampleos-build mnt
# Now run global "triggers", generate cache files:
ldconfig -r mnt
  (Insert other programs here)
fusermount -u mnt
ostree --repo=build-repo commit -b exampleos/x86_64/standard --link-
checkout-speedup exampleos-build
```

Il y a un certain nombre de choses intéressantes qui se passent ici. Le principal changement architectural est qu'on utilise –link-checkout-speedup. Ceci est un moyen de dire à OSTree que notre paiement est effectué via des liens physiques et de scanner le référentiel afin de créer un reverse (device,inode) → checksum mapping.

Pour que ce mappage soit précis, nous avions besoin de rofiles-fuse pour nous assurer que tous les fichiers modifiés avaient de nouveaux inodes (et donc une nouvelle somme de contrôle).

Migration de contenu entre référentiels

Maintenant que nous avons du contenu dans notre référentiel build-repo (en mode utilisateur nu), nous devons déplacer le contenu de la branche exampleos/x86_64/standard dans le référentiel nommé repo (en mode archive) pour l'exportation, ce qui impliquera la compression zlib de nouveaux objets. Nous souhaitons également générer des deltas statiques après cela.

Copions le contenu:

```
ostree --repo=repo pull-local build-repo exampleos/x86_64/standard
```

Les clients peuvent désormais télécharger de nouveaux objets de manière incrémentielle - cependant, ce serait également le bon moment pour générer un delta à partir de la validation précédente.

```
ostree --repo=repo static-delta generate exampleos/x86_64/standard
```

Gestion de référentiel plus sophistiquée

Ensuite, consultez Gestion du référentiel pour les étapes suivantes de gestion du contenu dans les référentiels OSTree.

Mises à niveau atomiques

On peut couper l'alimentation à tout moment ...

OSTree est conçu pour implémenter des mises à niveau entièrement atomiques et sûres; plus généralement, les transitions atomiques entre les listes de déploiements amorçables. Si le système tombe en panne ou que vous coupez le courant, vous aurez soit l'ancien système, soit le nouveau.

Mises à niveau simples via HTTP

Tout d'abord, le modèle le plus élémentaire pris en charge par OSTree est celui où il réplique les arborescences de système de fichiers pré-générées à partir d'un serveur via HTTP, en suivant exactement une référence, qui est stockée dans le fichier .origin pour le déploiement. La commande ostree admin upgrade implémente ceci.

Pour commencer une mise à niveau simple, OSTree récupère le contenu de la référence depuis le serveur distant. Supposons que nous suivions une référence nommée exampleos/buildmaster/x86_64-runtime. OSTree récupère l'URL http://example.com/repo/refs/exampleos/buildmaster/x86_64-runtime, qui contient une somme de contrôle SHA256. Cela détermine l'arborescence à déployer et `/etc` sera fusionné à partir de l'arborescence actuellement démarrée.

Si nous n'avons pas cette validation, alors, nous effectuons un processus d'extraction. À l'heure actuelle (sans deltas statiques), cela implique tout simplement de récupérer chaque objet individuel que nous n'avons pas, de manière asynchrone. Autrement dit, nous téléchargeons uniquement les fichiers modifiés (compressés zlib). Chaque objet a sa somme de contrôle validée et est stocké dans `/ostree/repo/objects/`.

Une fois l'extraction terminée, nous avons tous les objets localement nécessaires pour effectuer un déploiement.

Mises à niveau via des outils externes (par exemple, les gestionnaires de packages)

Comme mentionné dans l'introduction, OSTree est également conçu pour permettre un modèle où les

arborescences de système de fichiers sont calculées sur le client. Il est complètement indépendant de la façon dont ces arbres sont générés; ils peuvent être calculés avec des packages traditionnels, des packages avec des scripts post-déploiement au-dessus, ou construits par des développeurs directement à partir du contrôle de révision localement, etc.

Sur le plan pratique, la plupart des gestionnaires de packages d'aujourd'hui (dpkg et rpm) fonctionnent “en direct” sur le système de fichiers actuellement démarré. La façon dont ils pourraient travailler avec OSTree consiste à prendre la liste des packages installés dans l'arborescence actuellement démarrée et à calculer un nouveau système de fichiers à partir de cela. Un chapitre ultérieur décrit plus en détail comment cela pourrait fonctionner: Adapter les systèmes existants.

Aux fins de cette section, supposons que nous ayons une arborescence de système de fichiers nouvellement générée stockée dans le référentiel (qui partage le stockage avec l'arborescence amorcée existante). Nous pouvons ensuite passer à la vérification du repo dans un déploiement.

Assemblage d'un nouveau répertoire de déploiement

Étant donné un commit à déployer, OSTree lui alloue d'abord un répertoire. Il s'agit de la forme /boot/loader/entries/ostree-$stateroot-$checksum.$serial.conf. Le $ serial est normalement 0, mais si un commit donné est déployé plus d'une fois, il sera incrémenté. Ceci est pris en charge car le déploiement précédent peut avoir une configuration dans / etc que nous ne voulons pas utiliser ou remplacer.

Maintenant que nous avons un répertoire de déploiement, une fusion à 3 voies est effectuée entre le `/etc` (par défaut) du déploiement actuellement démarré, sa configuration par défaut et le nouveau déploiement (basé sur son /usr/etc).

Échange atomique de la configuration de démarrage

À ce stade, un nouveau répertoire de déploiement a été créé en tant que batterie de liens durs; le système en cours d'exécution est intact et la configuration du chargeur de démarrage est intacte. Nous voulons ajouter ce déploiement à la “liste des déploiements”.

Pour prendre en charge un cas plus général, OSTree prend en charge la transition atomique entre des ensembles de déploiements arbitraires, avec la restriction que le déploiement actuellement démarré doit toujours être dans le nouvel ensemble. Dans le cas normal, nous avons exactement un déploiement, qui est celui démarré, et nous voulons ajouter le nouveau déploiement à la liste. Une commande plus complexe pourrait permettre de créer 100 déploiements dans le cadre d'une transaction atomique, de sorte que l'on puisse configurer un système automatisé pour les couper en deux.

La version de démarrage

OSTree permet l'échange entre les configurations de démarrage en implémentant le “swapped directory pattern” dans /boot. Cela signifie qu'il s'agit d'un lien symbolique vers l'un des deux répertoires /ostree/boot.[0|1]. Pour permuter le contenu atomiquement, si la version actuelle est 0,

nous créons /ostree/boot.1, le remplissons avec le nouveau contenu, puis permutons atomiquement le lien symbolique. Enfin, l'ancien contenu peut être récupéré à tout moment.

Le répertoire /ostree/boot

Cependant, nous voulons optimiser pour le cas où l'ensemble des ensembles noyau / initramfs / devicetree est le même entre les anciennes et les nouvelles listes de déploiement. Cela se produit lorsque vous effectuez une mise à niveau qui n'inclut pas le noyau; pensez à une simple mise à jour de la traduction. OSTree est optimisé dans ce cas, car sur certains systèmes / le démarrage peut se faire sur un support distinct tel que le stockage flash non optimisé pour des quantités importantes de trafic d'écriture. En relation avec cela, OSTree moderne prend en charge le fait que /boot soit un montage en lecture seule par défaut - il remontera automatiquement en lecture-écriture juste pour la partie du temps nécessaire pour mettre à jour la configuration du chargeur de démarrage.

Pour l'implémenter, OSTree gère également le répertoire /ostree/boot.$bootversion, qui est un ensemble de liens symboliques aux répertoires de déploiement. La $bootversion ici doit correspondre à la version de /boot. Cependant, afin de permettre les transitions atomiques de ce répertoire, il s'agit également d'un répertoire échangé, donc tout comme /boot, il a une version de 0 ou 1 ajoutée.

Chaque entrée du chargeur de démarrage a un argument ostree = spécial qui fait référence à l'un de ces liens symboliques. Ceci est analysé lors de l'exécution dans les initramfs.

Gestion du contenu dans les référentiels OSTree

Une fois que vous avez un système de construction en cours, si vous voulez réellement que les systèmes clients récupèrent le contenu, vous ressentirez rapidement le besoin de “gestion de référentiel”.

L'outil de ligne de commande ostree couvre certaines fonctionnalités de base, mais n'inclut pas de workflows de très haut niveau. L'une des raisons est que la manière dont le contenu est fourni et géré présente des préoccupations très spécifiques à l'organisation. Par exemple, certains fournisseurs de contenu de système d'exploitation peuvent souhaiter l'intégration avec un système de notification d'errata spécifique lors de la génération des validations.

Dans cette section, nous décrirons quelques idées et méthodes de haut niveau pour gérer le contenu dans les référentiels OSTree, principalement indépendantes de tout modèle ou outil particulier. Cela dit, il existe un projet ostree-releng-scripts en amont associé qui a certains scripts qui sont destinés à implémenter des parties de ce document.

Un autre exemple de logiciel qui peut aider à gérer les référentiels OSTree aujourd'hui est le Pulp Project, qui a un plugin Pulp OSTree.

Mise en miroir des référentiels

Il est très courant de vouloir effectuer un miroir complet ou partiel, en particulier au-delà des frontières organisationnelles (par exemple, un fournisseur de système d'exploitation en amont et un utilisateur qui souhaite un accès hors ligne et plus rapide au contenu). OSTree prend en charge la mise en miroir complète et partielle du contenu de l'archive de base, bien qu'il ne s'agisse pas encore de deltas statiques.

Pour créer un miroir, créer d'abord un référentiel d'archives (il n'est pas besoin de l'exécuter en tant que root), puis ajoutez l'amont en tant que distant, puis utiliser `pull --mirror`.

```
ostree --repo=repo init --mode=archive
ostree --repo=repo remote add exampleos https://exampleos.com/ostree/repo
ostree --repo=repo pull --mirror exampleos:exampleos/x86_64/standard
```

On peut utiliser l'option `--depth=-1` pour récupérer tout l'historique, ou un entier positif comme 3 pour récupérer uniquement les 3 derniers commits.

Des référentiels de développement et de publication distincts

Par défaut, OSTree accumule l'historique côté serveur. Ceci est en fait facultatif dans la mesure où votre système de build peut (en utilisant l'API) écrire un commit sans parent. Mais d'abord, nous étudierons les ramifications de l'historique côté serveur.

De nombreux fournisseurs de contenu voudront séparer leur développement interne de ce qui est rendu public dans le monde. Par conséquent, vous voudrez (au moins) deux référentiels OSTree, nous les appellerons “dev” et “prod”.

Pour exprimer cela d'une autre manière, disons que vous avez un système de livraison continu qui se construit à partir de git et s'engage dans votre référentiel OSTree “dev”. Cela peut arriver des dizaines à des centaines de fois par jour. Il s'agit d'un historique important au fil du temps, et il est peu probable que la plupart de vos consommateurs de contenu (c'est-à-dire pas les développeurs / testeurs) soient intéressés par tout cela.

La vision originale d'OSTree était de remplir ce rôle de “dev”, et en particulier le format “archive” a été conçu pour lui.

Ensuite, ce que vous voudrez faire, c'est promouvoir le contenu de “dev” à “prod”. Nous en discuterons plus tard, mais d'abord, parlons de la promotion dans notre référentiel “dev”.

Promouvoir le contenu le long des branches OSTree - "buildmaster", "smoketested"

Outre plusieurs référentiels, OSTree prend également en charge plusieurs branches dans un même

référentiel, équivalentes aux branches de git. Nous avons vu dans une section précédente un exemple de nom de branche comme exampleos/x86_64/standard. Le choix du nom de la branche pour votre référentiel “prod” est absolument critique car les systèmes clients le référenceront. Cela devient une partie importante de votre visage pour le monde, de la même manière que la branche “master” dans un référentiel git.

Mais avec votre référentiel “dev” en interne, il peut être très utile d'utiliser les concepts de branchement d'OSTree pour représenter différentes étapes d'un pipeline de livraison de logiciels.

Dérivant de `exampleos/x86_64/standard`, disons que notre référentiel “dev” contient `exampleos/x86_64/buildmaster/standard`. Nous choisissons le terme “buildmaster” pour représenter quelque chose qui vient directement de git master. Elle peut ne pas être testée beaucoup.

Notre prochaine étape devrait être de brancher un système de test (Jenkins, Buildbot, etc.) à cela. Lorsqu'un build (commit) réussit certains tests, nous voulons “promouvoir” ce commit. Créons une nouvelle branche appelée smoketested pour dire que certains contrôles d'intégrité de base passent sur le système complet. C'est peut-être là que les testeurs humains interviennent, par exemple.

Un moyen basique de “promouvoir” le commit buildmaster qui a réussi les tests comme celui-ci:

```
ostree commit -b exampleos/x86_64/smoketested/standard -s 'Passed tests' --
tree=ref=aec070645fe53...
```

Ici, nous générons un nouvel objet de validation (peut-être inclure dans les liens du journal de validation pour créer des journaux, etc.), mais nous réutilisons le contenu du buildmaster commit aec070645fe53 qui a réussi les tests de fumée.

Pour une implémentation plus sophistiquée de ce modèle, voir le script do-release-tags, qui inclut la prise en charge de choses comme la propagation des numéros de version à travers la promotion de validation.

Nous pouvons facilement généraliser ce modèle pour avoir un nombre arbitraire d'étapes comme `exampleos/x86_64/stage-1-pass/standard`, `exampleos/x86_64/stage-2-pass/standard`, etc., selon on exigences et logique métier.

Dans ce modèle proposé, les «étapes» sont de plus en plus chères. La logique est que nous ne voulons pas consacrer beaucoup de temps, par exemple, à tests de performances du réseau si quelque chose de basique comme un fichier d'unité systemd échoue au démarrage.

Promotion du contenu entre les référentiels OSTree

Maintenant, nous avons notre flux de livraison continu interne qui coule, il est testé et fonctionne. Nous voulons prendre périodiquement le dernier commit sur exampleos/x86*64/stage-3-pass/standard et l'exposer dans notre référentiel “prod” comme exampleos/x86*64/standard, avec un historique beaucoup plus petit.

Nous aurons d'autres exigences commerciales telles que la rédaction de notes de publication (et éventuellement leur mise dans le message de validation OSTree), etc.

Dans Build Systems, nous avons vu comment la commande pull-local peut être utilisée pour migrer le contenu du référentiel “build” (en mode bare-user) vers un référentiel d'archives pour servir aux systèmes clients.

Après cette section, nous avons maintenant trois référentiels, appelons-les repo-build, repo-dev et repo-prod. Nous avons extrait du contenu de la génération de référentiel vers le développement de référentiel (ce qui implique entre autres la compression gzip car il s'agit d'un changement de format).

Lors de l'utilisation de pull-local pour migrer du contenu entre deux référentiels d'archives, le contenu binaire n'est pas modifié. Allons de l'avant et générons un nouveau commit dans notre référentiel de prod:

```
checksum=$(ostree --repo=repo-dev rev-parse exampleos/x86_64/stage-3-
pass/standard`)
ostree --repo=repo-prod pull-local repo-dev ${checksum}
ostree --repo=repo-prod commit -b exampleos/x86_64/standard \
       -s 'Release 1.2.3' --add-metadata-string=version=1.2.3 \
       --tree=ref=${checksum}
```

Il se passe quelques choses ici. Tout d'abord, nous avons trouvé la dernière somme de contrôle de validation pour le “dev-stage-3” et demandé à pull-local de la copier, sans utiliser le nom de la branche. Nous le faisons parce que nous ne voulons pas exposer le nom de la branche exampleos/x86_64/stage-3-pass/standard dans notre référentiel “prod”.

Ensuite, nous générons un nouveau commit dans prod qui fait référence au contenu binaire exact dans dev. Si les référentiels “dev” et “prod” se trouvent sur le même système de fichiers Unix, (comme git) OSTree utilisera des liens durs pour éviter de copier du contenu - ce qui rend le processus très rapide.

Une autre chose intéressante à noter ici est que nous ajoutons une chaîne de métadonnées de version au commit. Il s'agit d'un élément facultatif de métadonnées, mais nous encourageons son utilisation dans l'écosystème d'outils OSTree. Les commandes telles que le statut d'administrateur ostree le montrent par défaut.

Données dérivées - deltas statiques et le fichier récapitulatif

Comme indiqué dans Formats, le référentiel d'archives que nous utilisons pour “prod” nécessite une extraction HTTP par demande client par défaut. Si nous effectuons uniquement une version, par exemple une fois par semaine, il convient d'utiliser des “deltas statiques” pour accélérer les mises à jour des clients.

Donc, une fois que nous avons utilisé la commande ci-dessus pour extraire le contenu de repo-dev vers repo-prod, générons un delta par rapport au commit précédent:

```
ostree --repo=repo-prod static-delta generate exampleos/x86_64/standard
```

Lorsqu'on veut également prendre en charge la mise à niveau des systèmes clients à partir de deux

validations précédentes.

```
ostree --repo=repo-prod static-delta generate --
from=exampleos/x86_64/standard^^ --to=exampleos/x86_64/standard
```

La génération d'une permutation complète des deltas dans toutes les versions antérieures peut coûter cher, et il existe une certaine prise en charge dans le noyau OSTree pour les deltas statiques qui “récurrentes” à un parent. Cela peut aider à créer un modèle dans lequel les clients téléchargent une chaîne de deltas. Cependant, la prise en charge n'est pas encore entièrement mise en œuvre.

Que vous choisissiez ou non de générer des deltas statiques, vous devez mettre à jour le fichier récapitulatif:

```
ostree --repo=repo-prod summary -u
```

(N'oubliez pas que la commande summary ne peut pas être exécutée simultanément, elle doit donc être déclenchée en série par d'autres travaux).

Il y a plus d'informations sur la conception du fichier récapitulatif dans Repo.

Élagage des référentiels de compilation et de développement

Premièrement, il est conseillé de ne pas utiliser OSTree comme “magasin de contenu principal”. Les binaires d'un référentiel OSTree doivent être dérivés d'un référentiel git. Le système de génération doit enregistrer les métadonnées appropriées telles que les options de configuration utilisées pour générer la génération, et on doit pouvoir la reconstruire si nécessaire. Les ressources artistiques doivent être stockées dans un système conçu pour cela (par exemple, Git LFS).

Une autre façon de le dire est que cinq ans plus tard, il est peu probable que l'on se soucie de conserver les binaires exacts d'une build du système d'exploitation mercredi après-midi il y a trois ans.

Pour économiser de l'espace et tailler le référentiel “dev”.

```
ostree --repo=repo-dev prune --refs-only --keep-younger-than="6 months ago"
```

Cela tronquera l'histoire de plus de 6 mois. Les validations supprimées auront des “marqueurs tombstone” ajoutés afin que vous sachiez qu'ils ont été explicitement supprimés, mais tout leur contenu (qui n'est pas référencé par une validation encore conservée) sera récupéré.