



Transactional Updates

with Btrfs and RPM

Ignaz Forster
Research Engineer
iforster@suse.com

TOC

- **Concept of transactional updates**
- **Transactional updates with Btrfs and RPM**
- **Live demo (openSUSE Kubic)**
- **A deeper look**
- **Alternatives**
- **What's next?**

Concept of Transactional Updates

What is a Transactional Update?

An update that

- **is atomic**
 - Either fully applied, or not applied at all
 - Update does not influence the running system
- **can be rolled back**
 - A failed or incompatible update can be quickly discarded to restore the previous system condition

Implementations

Common concepts shared between all distributions:

- Read-only root file system
- Transactional / atomic updates
- Often designed for large deployments (Clouds)
- Minimal base system
- Automatic updates / reboots
- Integrity protection

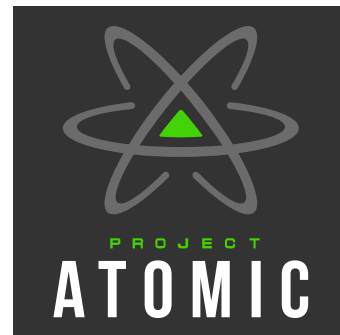
Examples:



Core



**RED HAT®
ENTERPRISE LINUX
ATOMIC HOST**

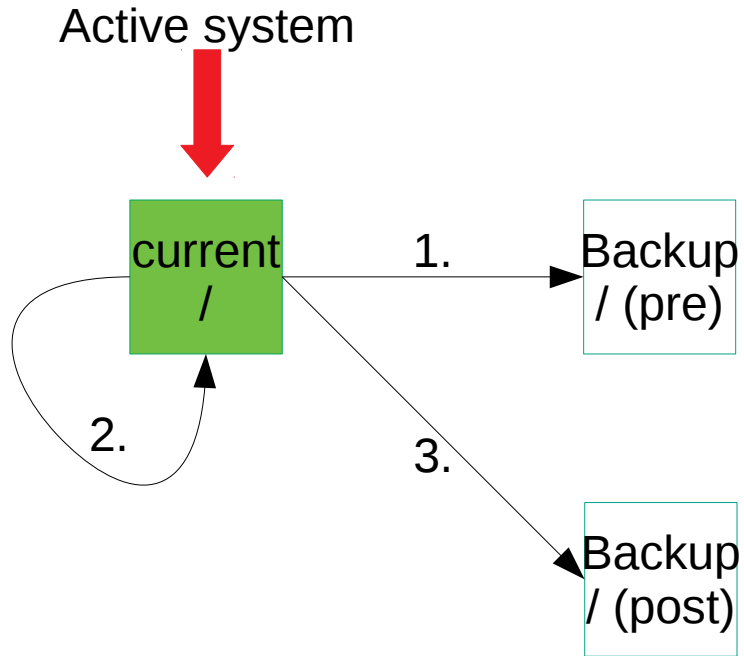


Transactional Updates with Btrfs and RPM

Snapper

- Snapshotting tool
- Called upon invocation of system tools (e.g. zypper or YaST)
- Uses Btrfs snapshot mechanism (but also supports ext4 and LVM)
- Available for a variety of other distributions

Updates with snapper



1. Create “pre” snapshot
2. Update the current system
3. Create “post” snapshot

Update is modifying the currently active file system

Restarts services immediately

Updates with snapper

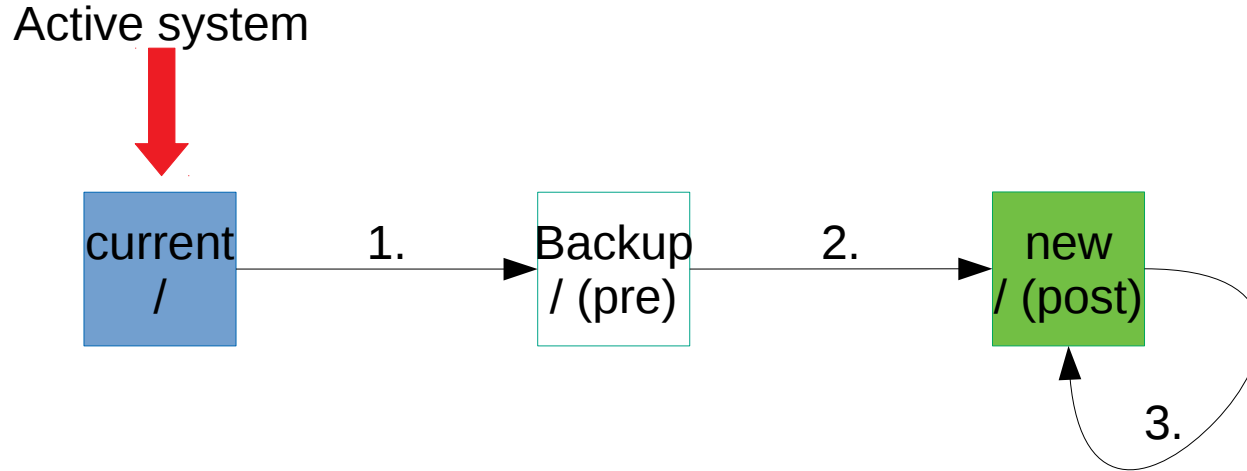
A Transactional Update is an update that

- **is atomic**
 - Either fully applied, or not applied at all
 - Update does not influence the running system
- **can be rolled back**
 - A failed or incompatible update can be quickly discarded to restore the previous system condition

Updates with transactional-update

- Using zypper & snapper in the background
- Also creates two snapshots
 - Pre: Backup of the current system
 - Post: Working snapshot
- Will not touch the currently running system
- Sets “Post” snapshot as new default btrfs root file system
- Changes applied on reboot
- If something goes wrong during the update nothing will be changed at all

Updates with transactional-update



1. Snapshot of current system
 2. Create new target snapshot
 3. Update system and set as default for next boot
- Current root file system is not modified

Live demo

Live Demo

Cheat Sheet

Transactional Updates

List repositories

`zypper lr -d`

Refresh repositories

`zypper ref`

Update installed packages

`transactional-update up`

Perform a distribution update

`transactional-update dup`

Install package(s)

`transactional-update pkg in <name>`

Update package(s)

`transactional-update pkg up <name>`

Remove package(s)

`transactional-update pkg rm <name>`

List snapshots

`snapper list`

Mark snapshots for removal by snapper

`transactional-update cleanup`

View default subvolume

`btrfs subvolume get-default /`

Open shell

`transactional-update shell`

Request reboot

`transactional-update reboot`

System rollback

`transactional-update rollback [number]`

Pitfalls

- Snapshots will be branched from the *current* system
 - snapshots will not contains the previous snapshot's contents if the system hasn't been rebooted!
- When using transactional-update on a read-write system
 - don't forget to reboot your system before making any changes to the root file system!



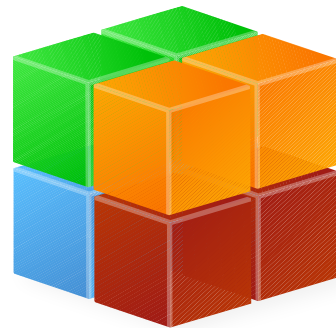
A deeper look

Handling of special directories

Writable directories on an otherwise read-only system:

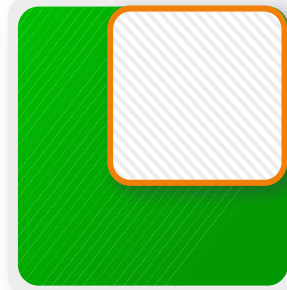
- `/var`
- `/etc`

/var handling



- **/var is a special directory as it contains variable data**
 - has to have read-write permissions
- **Cannot be rolled back**
 - A rollback would usually delete production data (e.g. your new orders in your database or your Docker images)
- **Typically stored on a separate subvolume or partition**
- **/var will not be mounted into the update snapshot, i.e. packages can not modify it (but we have some special handling for plain files and directories)**

/etc handling



- **On read-only systems /etc has to be writable**
 - Mounted as an overlay file system
 - Overlay stored in /var
- **On snapshot creation /etc contents will be synced into root file system**
 - Configuration is part of the snapshot
- **On reboot into new snapshot delete overlay contents**
- **Only files modified after snapshot creation will remain**

Other subvolumes

- **/opt, /var/log and /boot/grub2 will be bind mounted into the update snapshot**
 - **Everything else, including /srv, won't!**
- **Packages have to follow the FHS and packaging guidelines**

Helper applications: health-checker

- Add your own checker scripts to check for system consistency
- Automatic rollback if checks fail

Helper applications: rebootmgr

- `transactional-update.timer` triggers daily update including reboot
- `rebootmgr` manages reboot (e.g. in maintenance windows or synchronized via `etcd`)

What else is worth noting?

- Works with any standards-compliant RPM package
- General purpose tool: Especially useful for servers and clusters
- Fast snapshot switching
- Sane /etc and /var handling
- **Only works with BTRFS root file systems**
- **Configuration file:** /etc/transactional-update.conf (template in /usr/etc/transactional-update.conf)
- **Snapper will clean up old snapshots**
- **transactional-update is the only way to update a read-only system**

Alternatives

What's next?

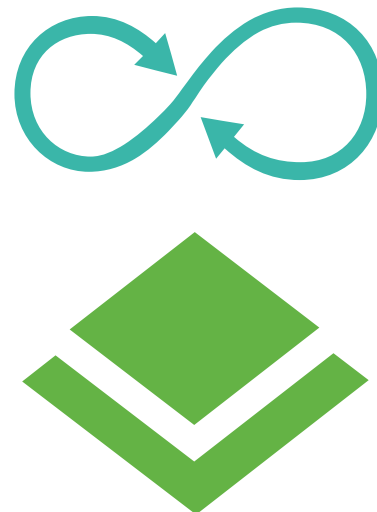
Availability



SUSE CaaS Platform



openSUSE Kubic



openSUSE Tumbleweed
openSUSE Leap 15
("Transactional Server" role)

Future development

- Integration into SLES 15
- Integrate transactional-update as zypper plugin
- IMA / EVM support for system verification / integrity
- Fix RPM packages with scripts modifying /var and /srv





Unpublished Work of SUSE LLC. All Rights Reserved.

This work is an unpublished work and contains confidential, proprietary and trade secret information of SUSE LLC. Access to this work is restricted to SUSE employees who have a need to know to perform tasks within the scope of their assignments. No part of this work may be practiced, performed, copied, distributed, revised, modified, translated, abridged, condensed, expanded, collected, or adapted without the prior written consent of SUSE. Any use or exploitation of this work without authorization could subject the perpetrator to criminal and civil liability.

General Disclaimer

This document is not to be construed as a promise by any participating company to develop, deliver, or market a product. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. SUSE makes no representations or warranties with respect to the contents of this document, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. The development, release, and timing of features or functionality described for SUSE products remains at the sole discretion of SUSE. Further, SUSE reserves the right to revise this document and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes. All SUSE marks referenced in this presentation are trademarks or registered trademarks of Novell, Inc. in the United States and other countries. All third-party trademarks are the property of their respective owners.