Livepatch Visibility at Scale

Breno Leitao
leitao@debian.org

Song Liu
song@kernel.org

2023-11-14

Meta

Agenda

- Overview

- Live patch deployment and Visibility
  - How to securely roll out KLPs to millions of machines

- Challenges
  - How to report that a certain machine has a live patch applied

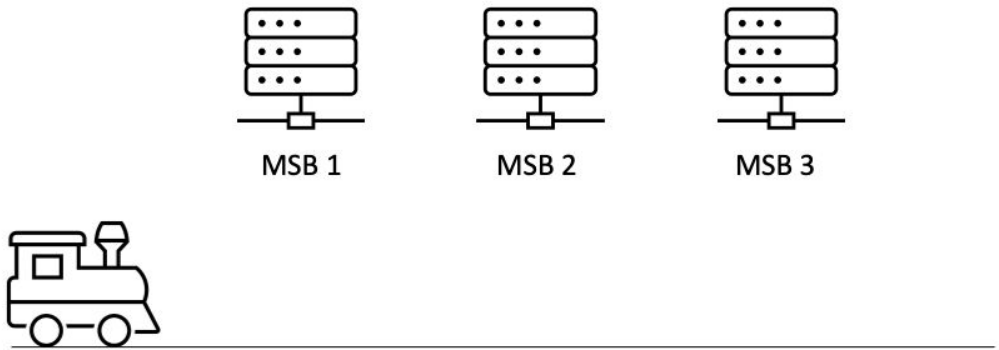
Linux Plumbers Conference

Richmond, Virginia | November 13-15, 2023

Installing a kernel is slow

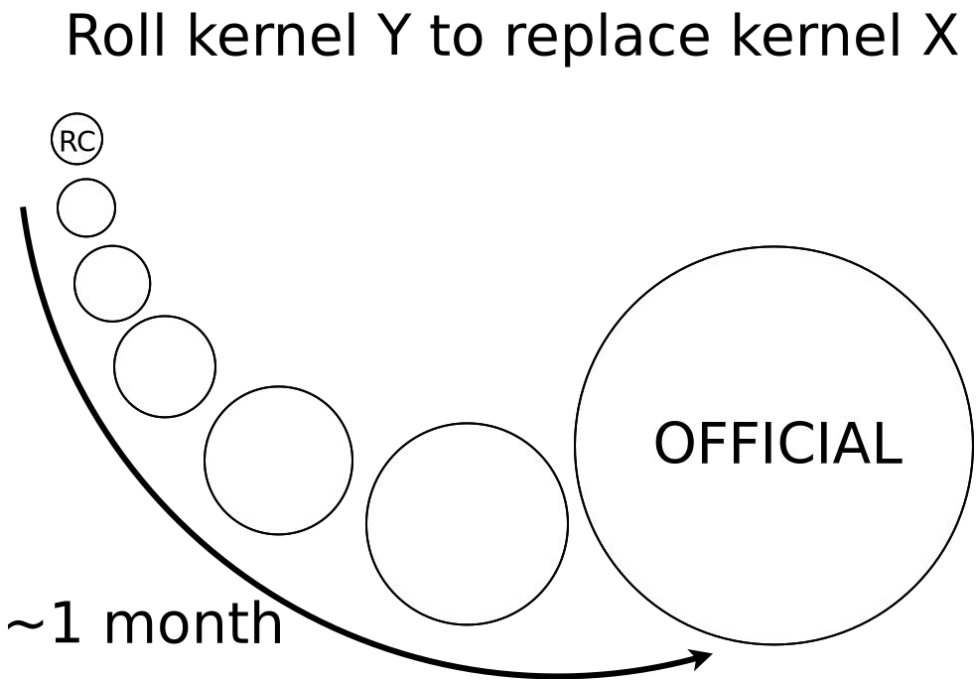
- It takes more than 45 days to roll out a new kernel to all machines
  - Draining and un-draining hosts is hard

- It is a trade-off between hosts offline and rollout speed



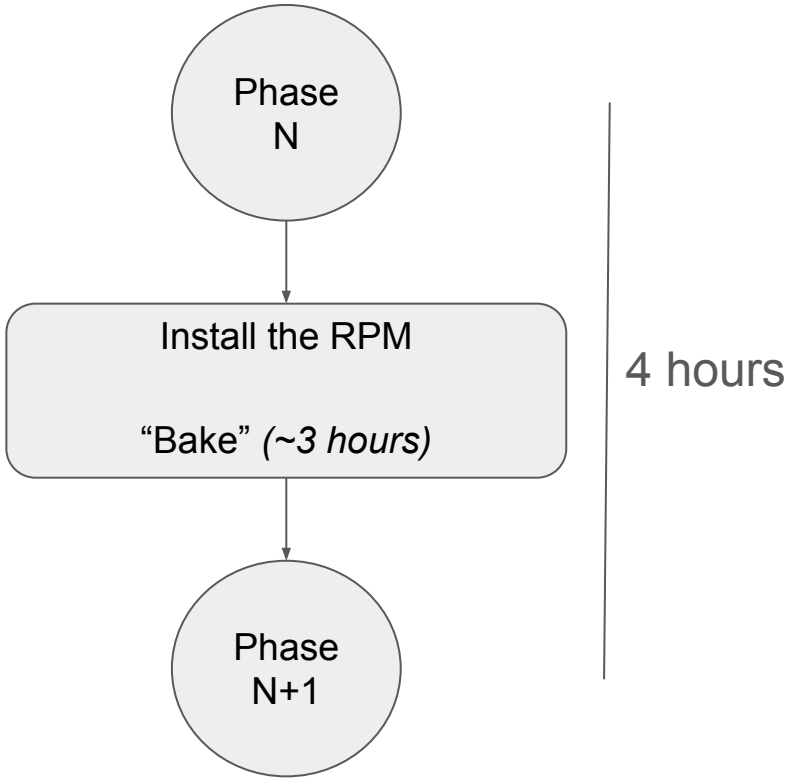
We find bugs during the rollout

- Different phases of the rollout
- Start with an RC tier
- Validate the current tier
  - Number of crashes, errors
- Proceed to the next tier



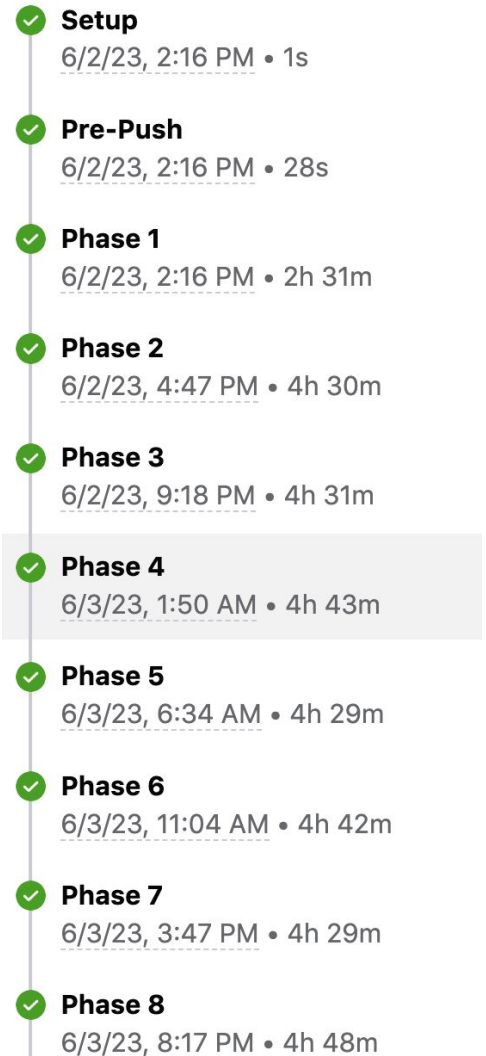
Live patch rollout

- An RPM is generated with the module
- Sharded RPM **automatic** rollout
- One “accumulative” hotfix only
  - `hotfix1` → `hotfix2` → `hotfix3`

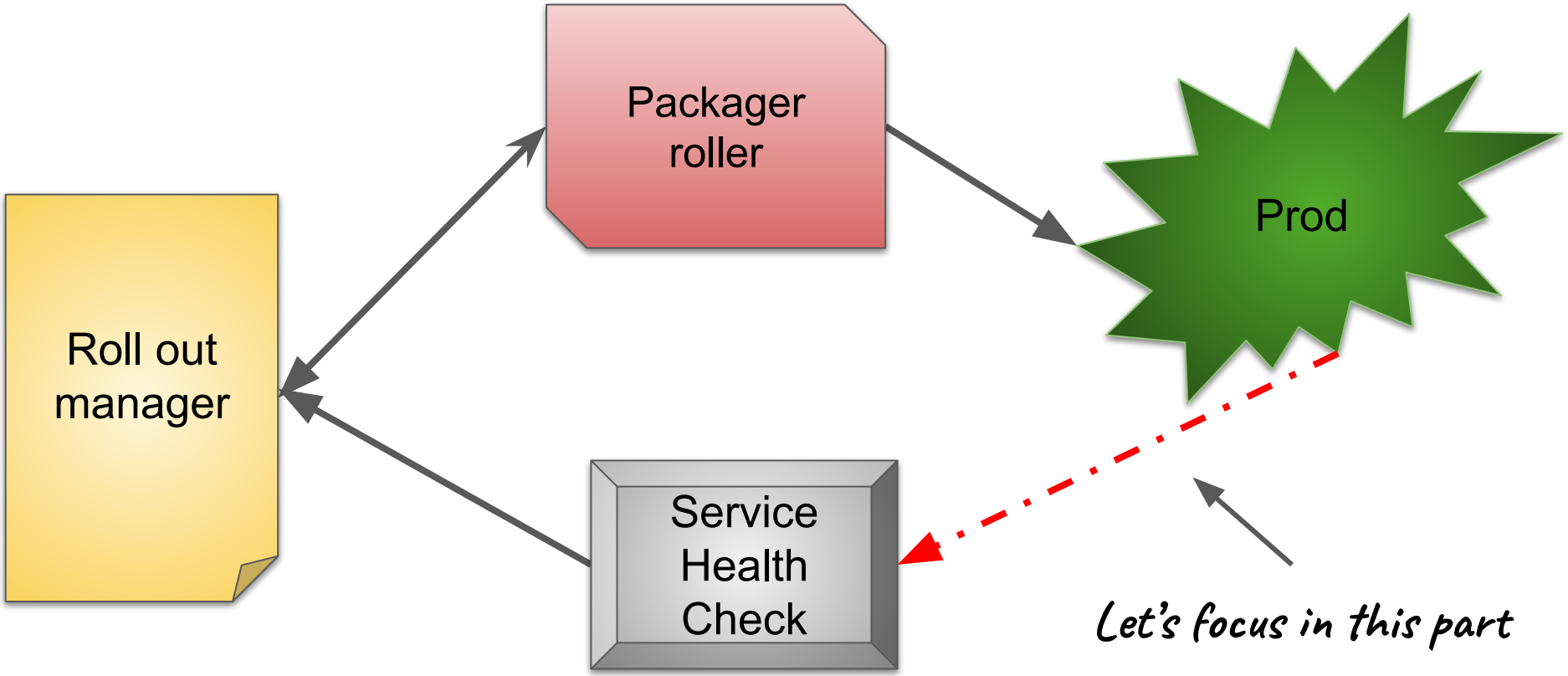


Monitoring

- Roll out stops automatically if errors are above a certain threshold



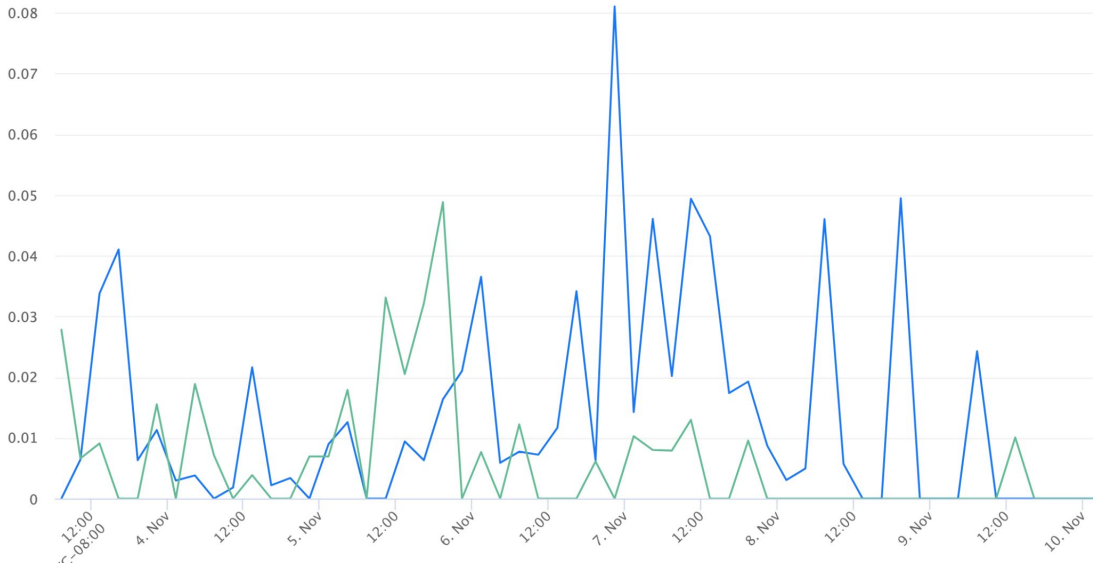
Live patch rollout



Health checks

Compare how health new “kernel is”

1) Number of **crashes**
   a) Stop if > 1 crash per 1K host
   b) Compare to non-hotfix kernel

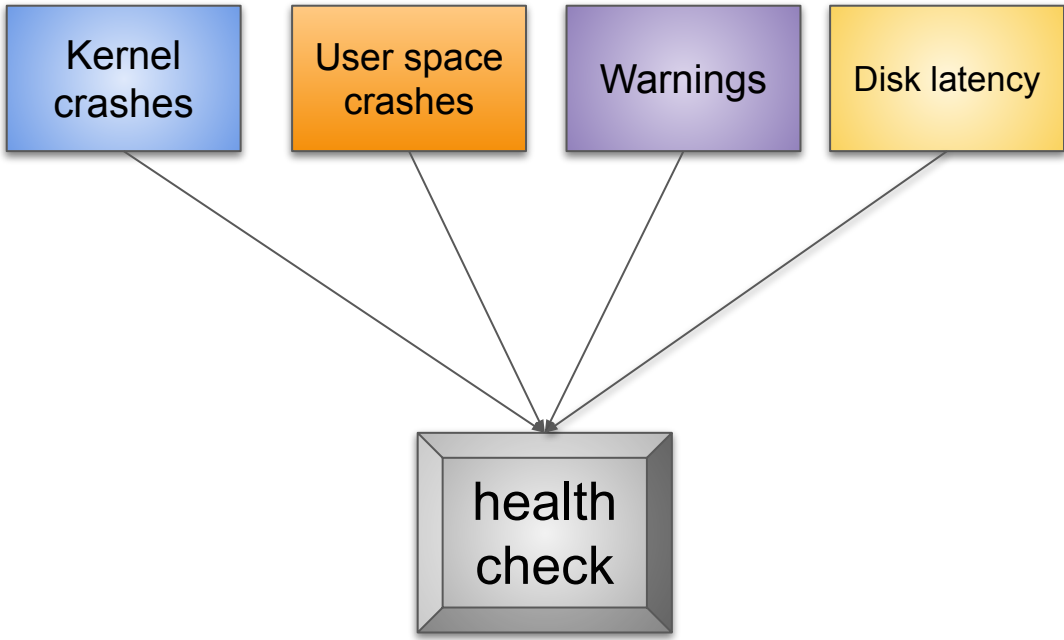


2) Number of **major alarms**
   a) Bugs, oops, Warnings, etc

3) **Service** metrics
   a) Application problems, performance

Data sources

- Metrics coming from different sources
- Kernel team:
  - Kernel crash
    - Coming from **crash**
  - Warnings/Bugs/Oops/OOMs
    - Coming from **netconsole**
- Platform
  - Apps Crashdump
    - Coming from random **coredumper**
- Workloads
  - Workload Performance
    - **Random** team metrics



Challenges

1) Pass the information that a KLP was applied to all health check
    a) Aka **Uname**

2) Visibility of performance impact

3) Transition failures

Uname challenges

1) All the teams need to read and expose “hotfixes” and export them
    a) So we can compare “hotfix” metrics with standard metrics

2) Not easy to get the current KLP that is loaded
    a) Get in different surfaces
        i) Kernel that crashed

3) Hard to find a KLP is enabled
    a) Check if a KLP was loaded and **active** when the kernel **warning** happened

Hack#1: Netconsole

- Append `-hotfix` to kernel version to `init_utsname()->release` in several places
  - Netconsole
  - Crash dump
  - Workload metrics

- Every hotfix has a macro
  - `#define HF_VERSION "hotfixX”`
  - Appending it to a printk dictionary
    - `msg_add_dict_text(buf + len, size - len, "UNAME", uname_value);`

Hack#2: Crashdump

- Kernel crashed
  - Use **drgn** to parse the list of modules

```
def get_kernel_hotfix_version(prog: Program) -> Optional[str]:
    try:
        modules = prog["modules"].address_of_()
        for module in list_for_each_entry("struct module", modules, "list"):
            # This can return a "wrong" version if we crash while transitioning
            # from one hotfix module to the next.
            modname = module.name.string_().decode("ascii")
            match = re.match(r"klp.*_(hotfix\d+)", modname)
            if match:
                return "-" + match.group(1)
```

Hack#3: Running system

- For a running system
  - C++ function that read `/sys/kernel/livepatch`
    - Check for enabled/disabled
    - Checks the directory name and report

- Repeat the same procedure for any other language

Next steps

- How do we solve this problem upstream?

- Simplify the read from the KLPs applied
  - No need to play `whack-a-mole` to report the KLP applied

Upstream possibilities

- Change `utsname` to append KLP loaded (?)
  - `uname` would return the KLP loaded
  - Probably not feasible

- Append them to netconsole outputs (?)
  - Create a netconsole option that appends livepatch applied together with the kernel version
  - Done in userspace using Dynamic configuration
    - `echo XXXX > /sys/kernel/config/netconsole/cmdline0/dictionary`

- Easy to read loaded and **active** KLPs (?)
  - Create a `/sys/kernel/livepatch/active_modules` that is easy to load and parse

Visibility of performance impact

- The performance overhead of livepatch is small, but there is always concern when a relatively hot function is patched

- Measure/estimate the performance impact: tracing
- Built-in solutions with lower overhead

```
struct klp_func {
    ...
    u64 __percpu
counter;
    ...
};
```

Visibility of transition failures

- kpatch prints /proc/<pid>/stack for <pid> that cannot finish transition
- pr_debug in klp_try_switch_task()

- Summary of a transition failure
  - `Failed KLP transition: X tasks are always running; Y tasks are sleeping on being patched function, ...`


Meta