SERVICE MANAGEMENT AND SYSTEMD MC LPC '22

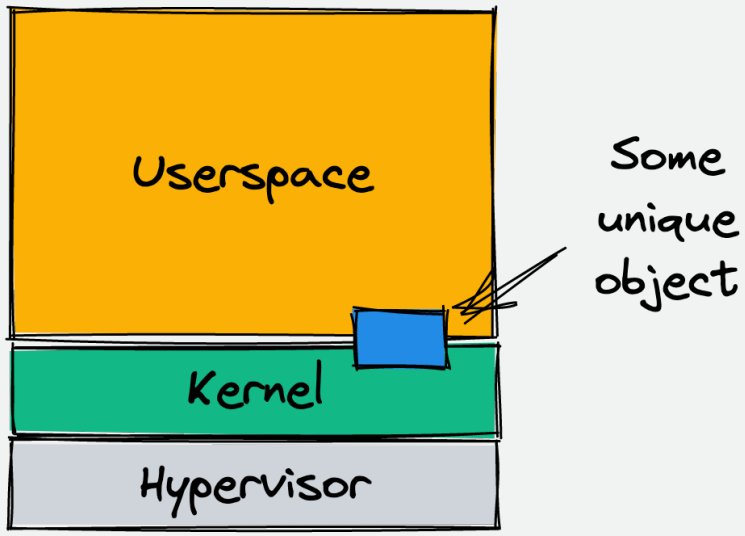
#snapsafe: restoring uniqueness in Virtual Machine clones

Babis Chalios (he/him)

Software Development Engineer
Amazon Web Services

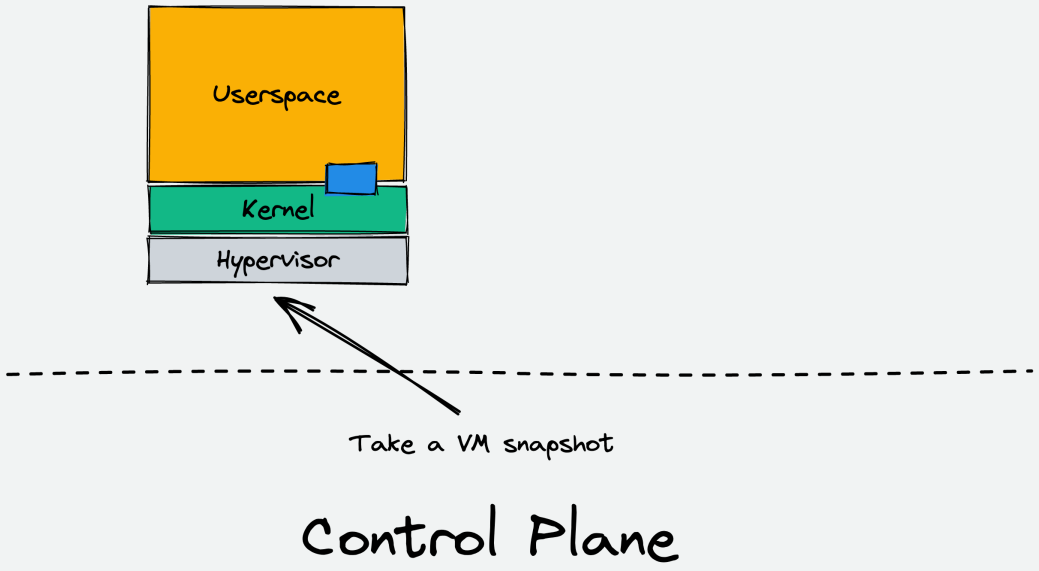
Snapshot safety: What is it about?

Snapshot safety: What is it about?



We start from a VM with some state

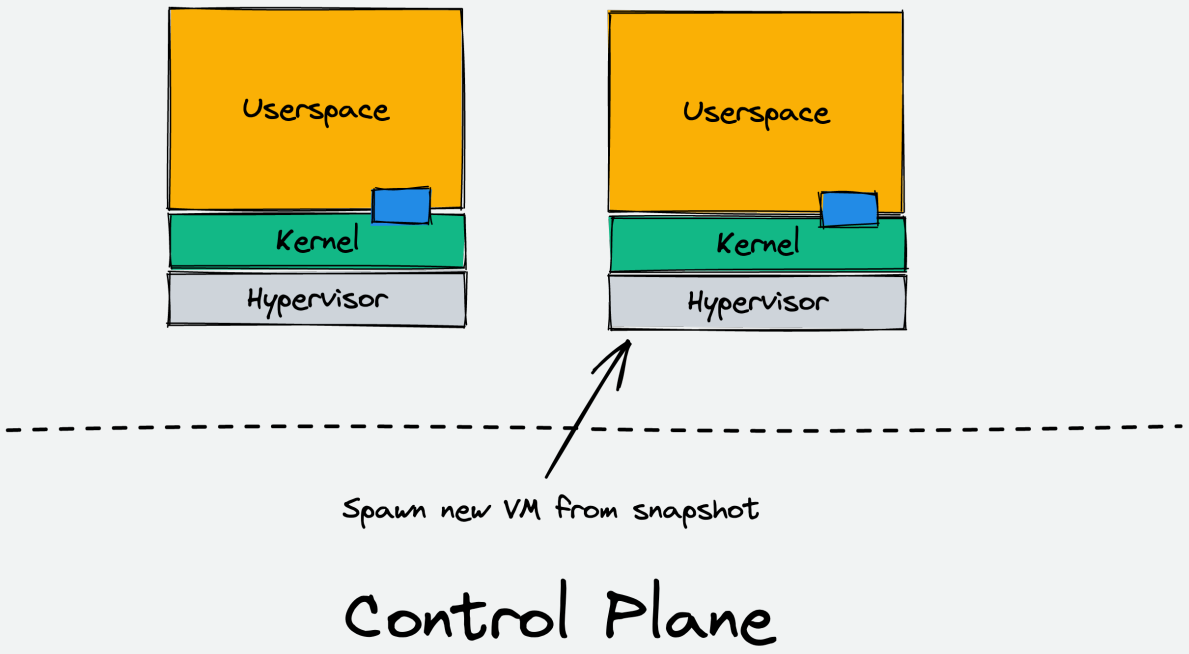
Snapshot safety: What is it about?



Take a snapshot of the VM

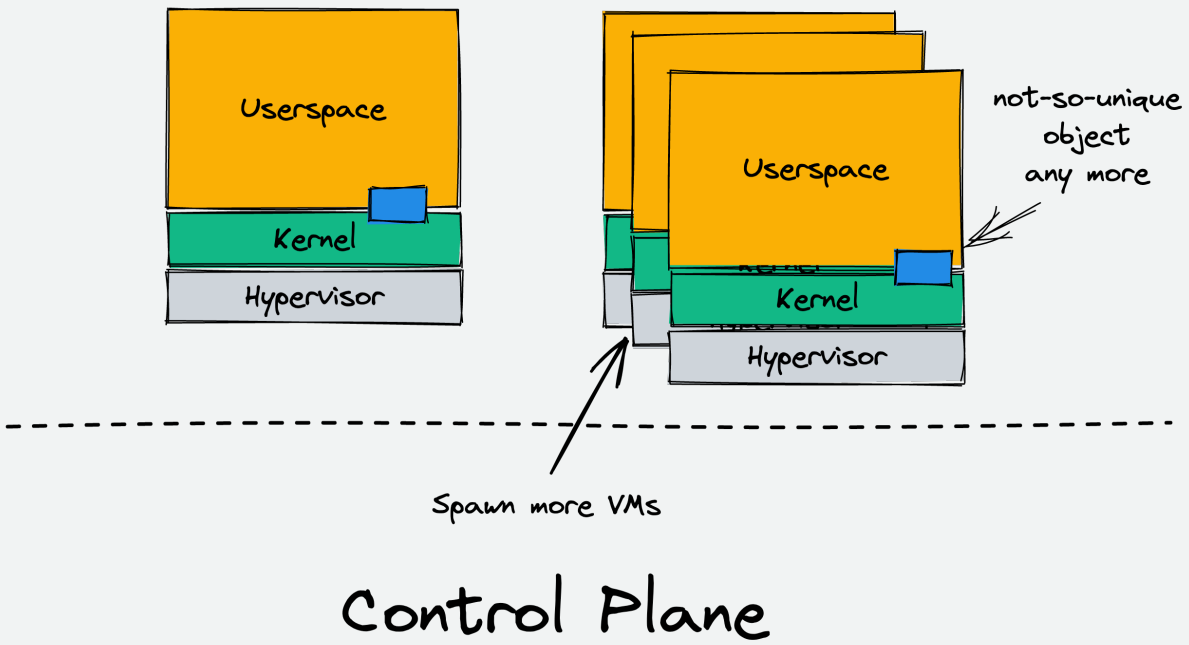
- Back-up VM
- Scale-out a service
- Use snapshot for fast cold-boots

Snapshot safety: What is it about?



Spawn one new VM from snapshot

Snapshot safety: What is it about?



… or more of those

Snapshot safety: Affected applications

- Applications that use PRNGs
  - Kernel-space
  - User-space libraries, e.g. OpenSSL
  - Language runtimes, e.g. Java
- Applications that use “unique” data

Snapshot safety

- Real world problem
- Increasingly important
  - ➢(micro)VMs used more and more to isolate workloads
- Both Kernel and User-space affected
- Need for end-to-end generic solution

Agenda

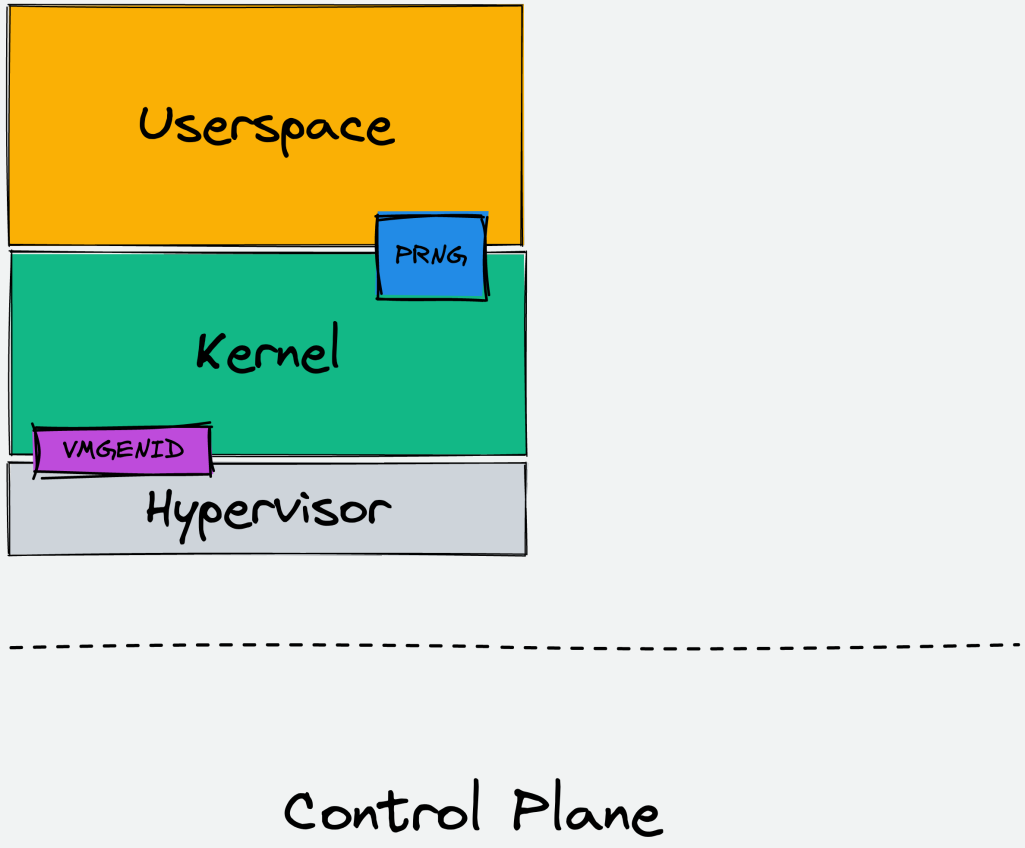
1. Current Linux Landscape
2. User-space considerations
3. System-wide snapshot safety
4. Summary & Next steps

Current Linux Landscape

Virtual Machine Generation ID

- Emulated device providing a generation ID to guest
  - Cryptographically random 128-bits integer
- Changes every time the VM *“executes from a different configuration file“*
- Notification mechanism for VM-lifecycle events
- Can be used as source of entropy
- Defined as an ACPI device

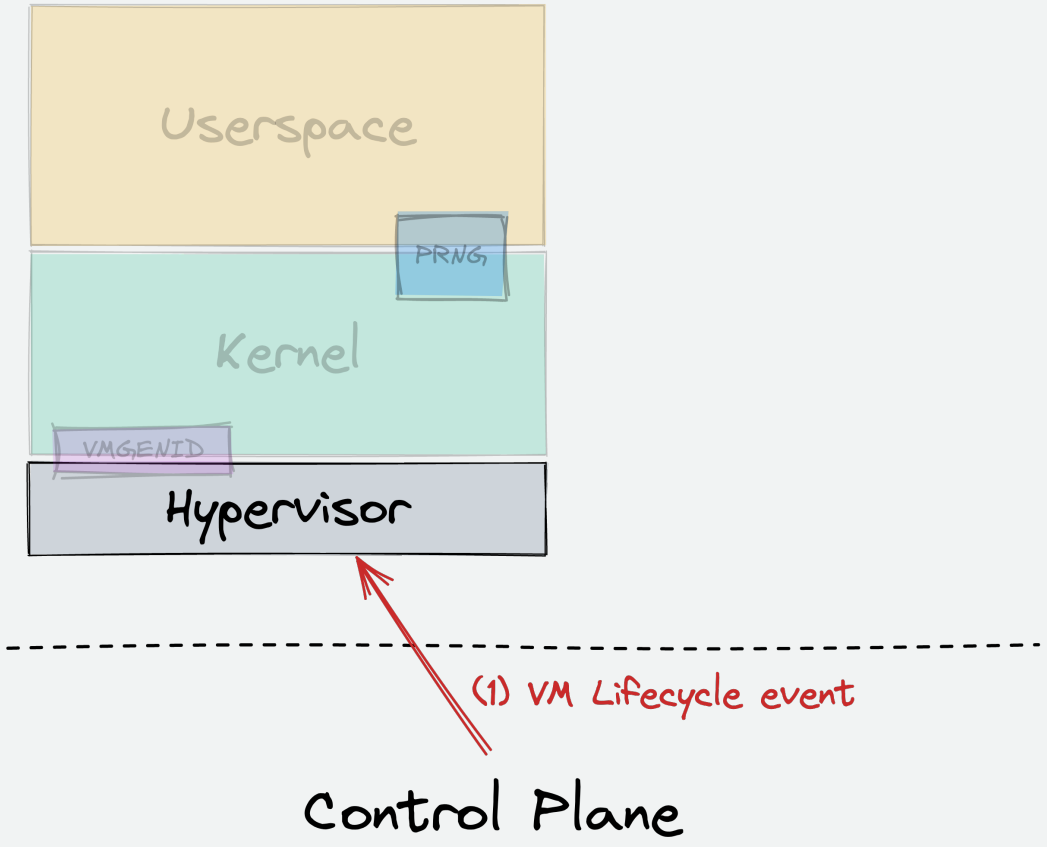
Virtual Machine Generation ID



Linux Implementation

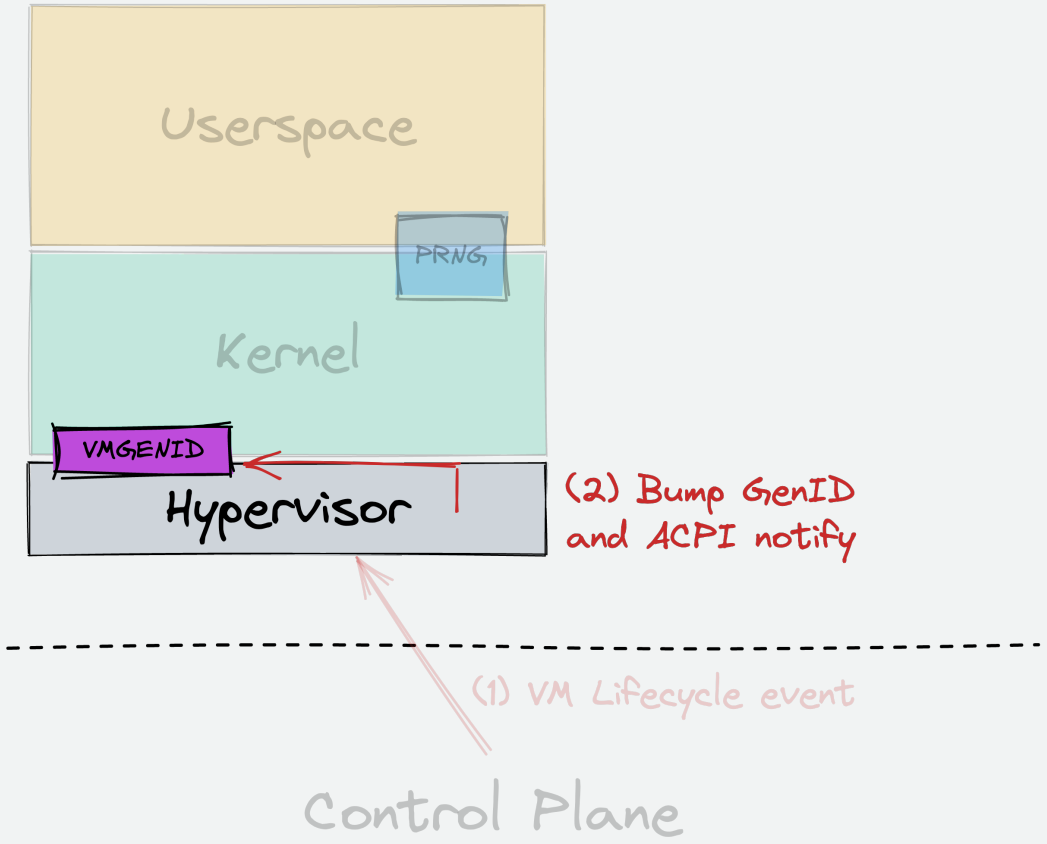
- ACPI driver mapping the Generation ID
- Handles ACPI notifications
- Uses Generation ID to re-seed PRNG

Virtual Machine Generation ID



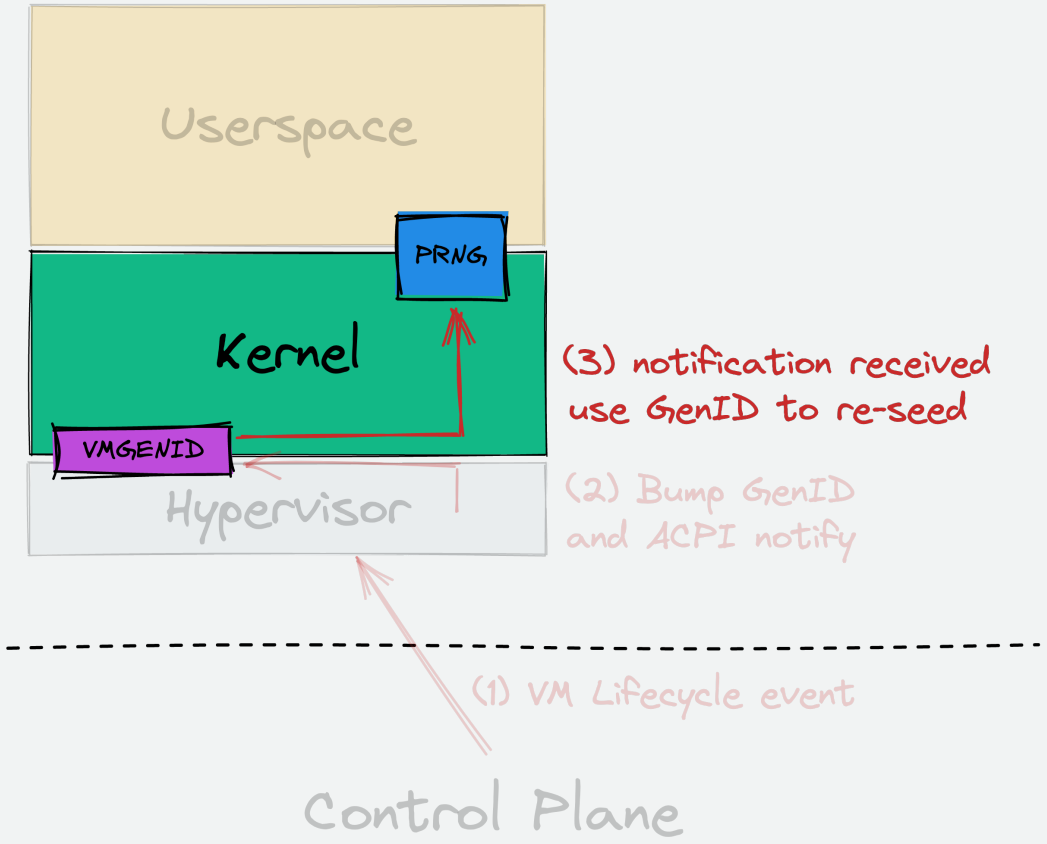
1. VM lifecycle event arrives

Virtual Machine Generation ID



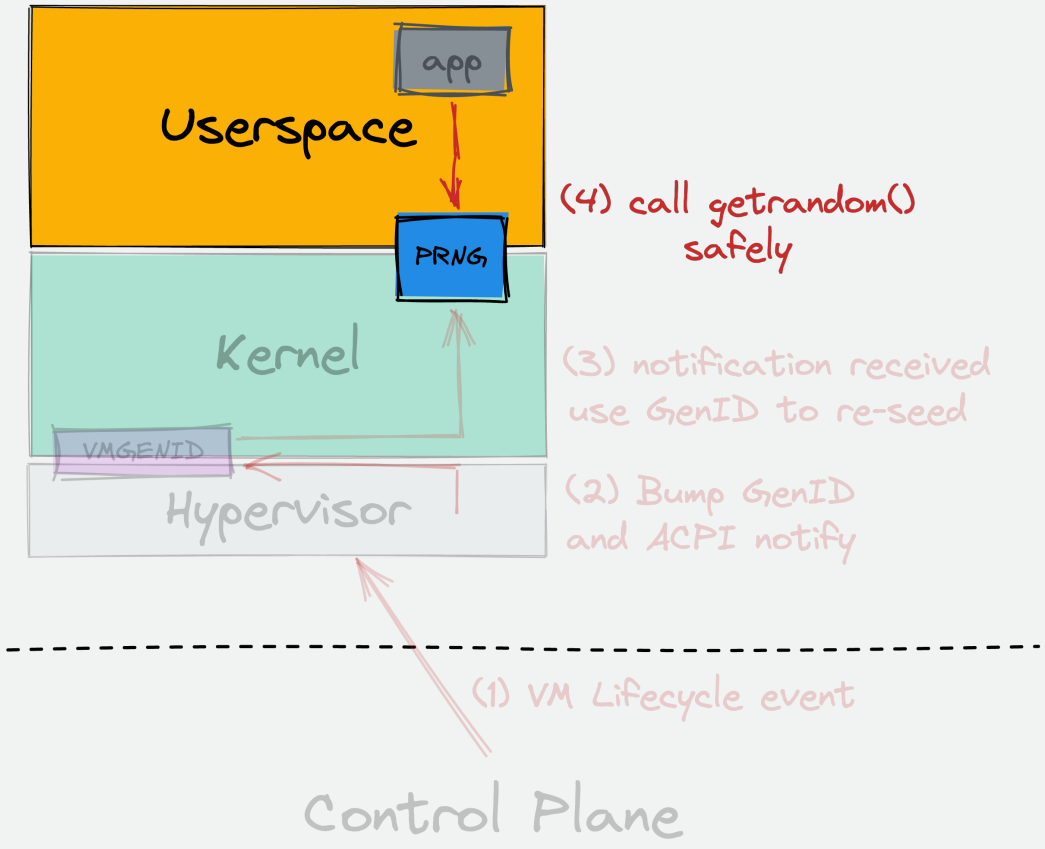
1. VM lifecycle event arrives
2. Hypervisor updates Generation ID and send ACPI notification and resumes vCPUs

Virtual Machine Generation ID



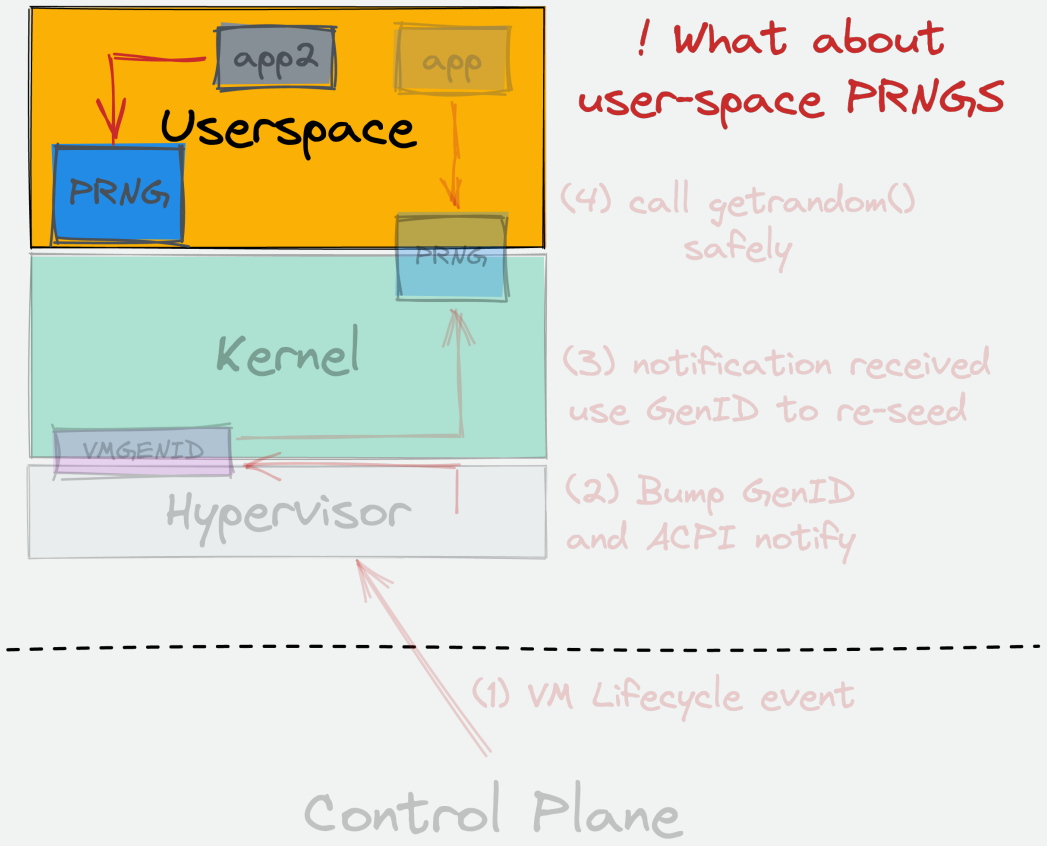
1. VM lifecycle event arrives
2. Hypervisor updates Generation ID and send ACPI notification and resumes vCPUs
3. Kernel driver handles notification. If generation ID changed use it as entropy

Virtual Machine Generation ID



1. VM lifecycle event arrives
2. Hypervisor updates Generation ID and send ACPI notification and resumes vCPUs
3. Kernel driver handles notification. If generation ID changed use it as entropy
4. User-space getting random bits from PRNG "safely":
   - Small race-window with ACPI notification handling

Virtual Machine Generation ID



But we have user-space PRNGs!

No mechanism to "let the user-space know"

User-space considerations

Virtual Machine Generation ID – User-space Concerns

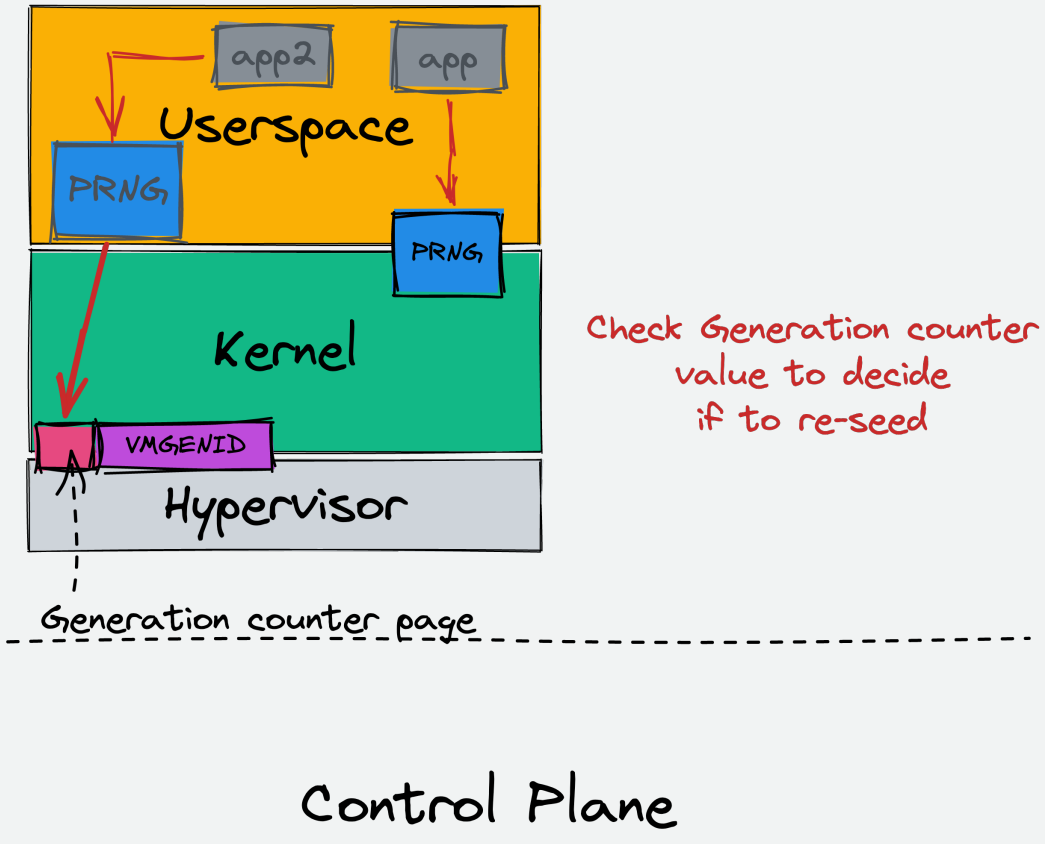
- No user-space facing mechanism at the moment
- Generation ID consumed in kernel as entropy for PRNG
  - Not safe to expose to user-space (?)
- Race-condition on ACPI notification
  - For example, RNG will produce identical results until ACPI notification is handled

Virtual Machine Generation Counter

Extend the device with word-size counter which increases every time the Generation ID changes

- Not a source of entropy
  - ➢No leaking of potentially sensitive data
- mmap() interface for user-space applications
  - ➢Directly observe changes in value, no need to wait for ACPI notification
- Can add as well poll() interface for applications with event loops
- Word-size means we can read it with a single instruction

Virtual Machine Generation Counter



User-space PRNG can now monitor Generation counter before returning random values

Virtual Machine Generation Counter – Alternatives

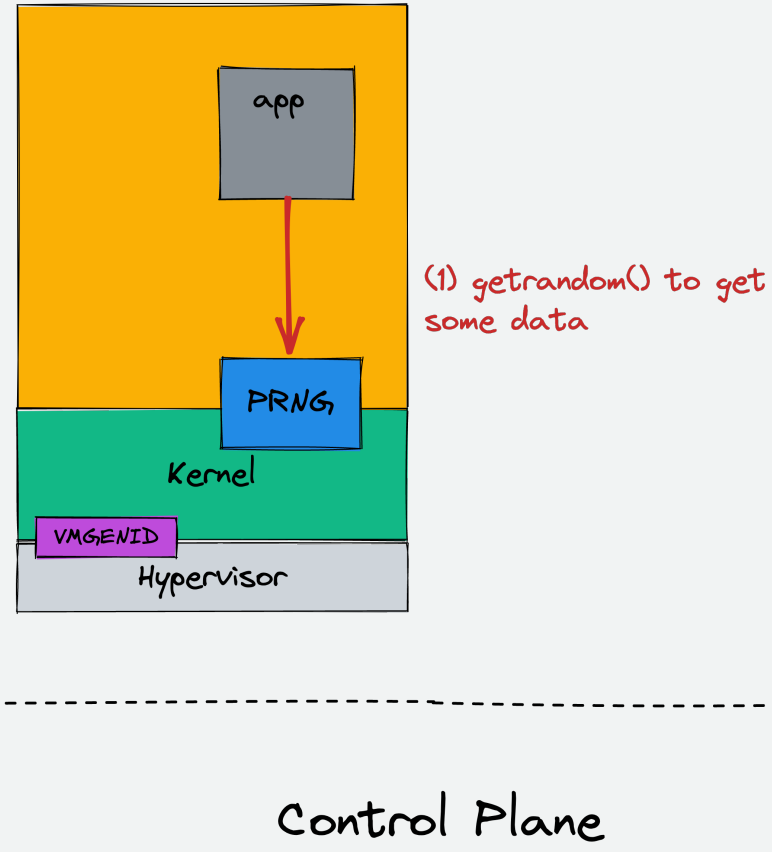
- Generation ID is used as entropy but it isn’t clear whether revealing it to user-space is actually harmful
- We could actually read Generation ID directly without waiting for notification
- Word-size reads are faster but maybe not that much (?)

Idea: Just expose Virtual Machine Generation ID

- Need to clarify security concerns with exposing Generation ID

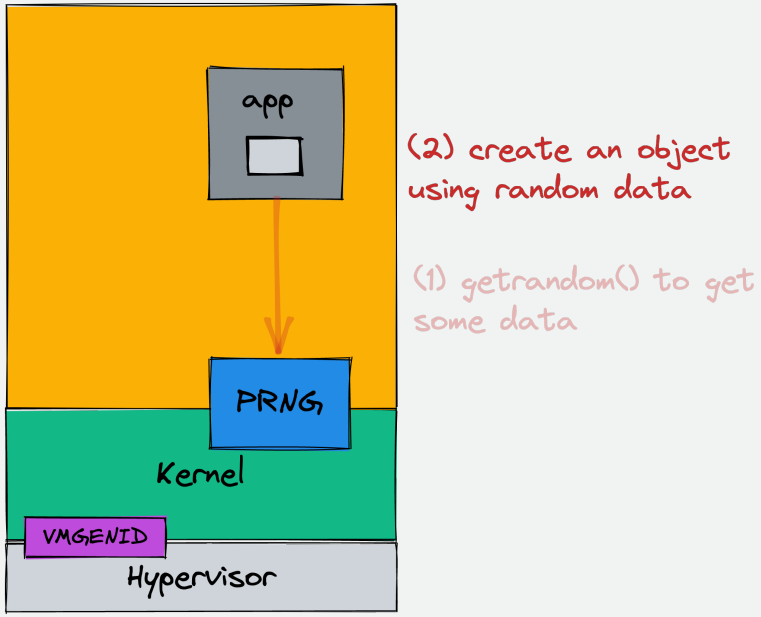
System-wide snapshot safety

Is VM Generation ID enough?



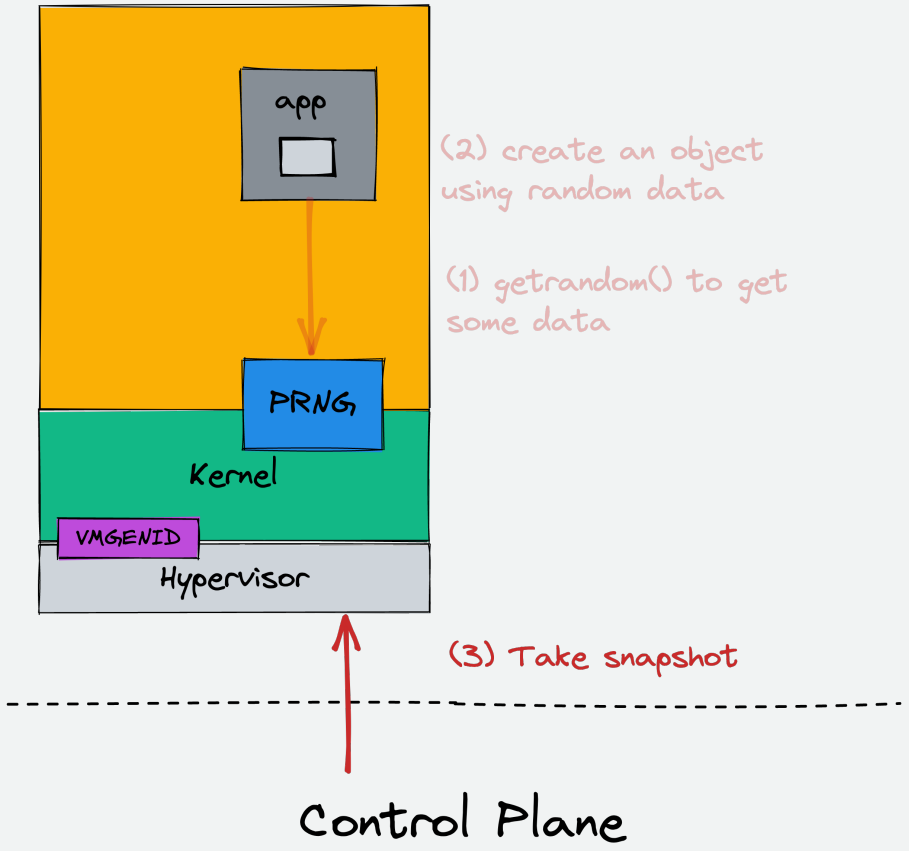
Application uses safe getrandom()

Is VM Generation ID enough?



And creates some state

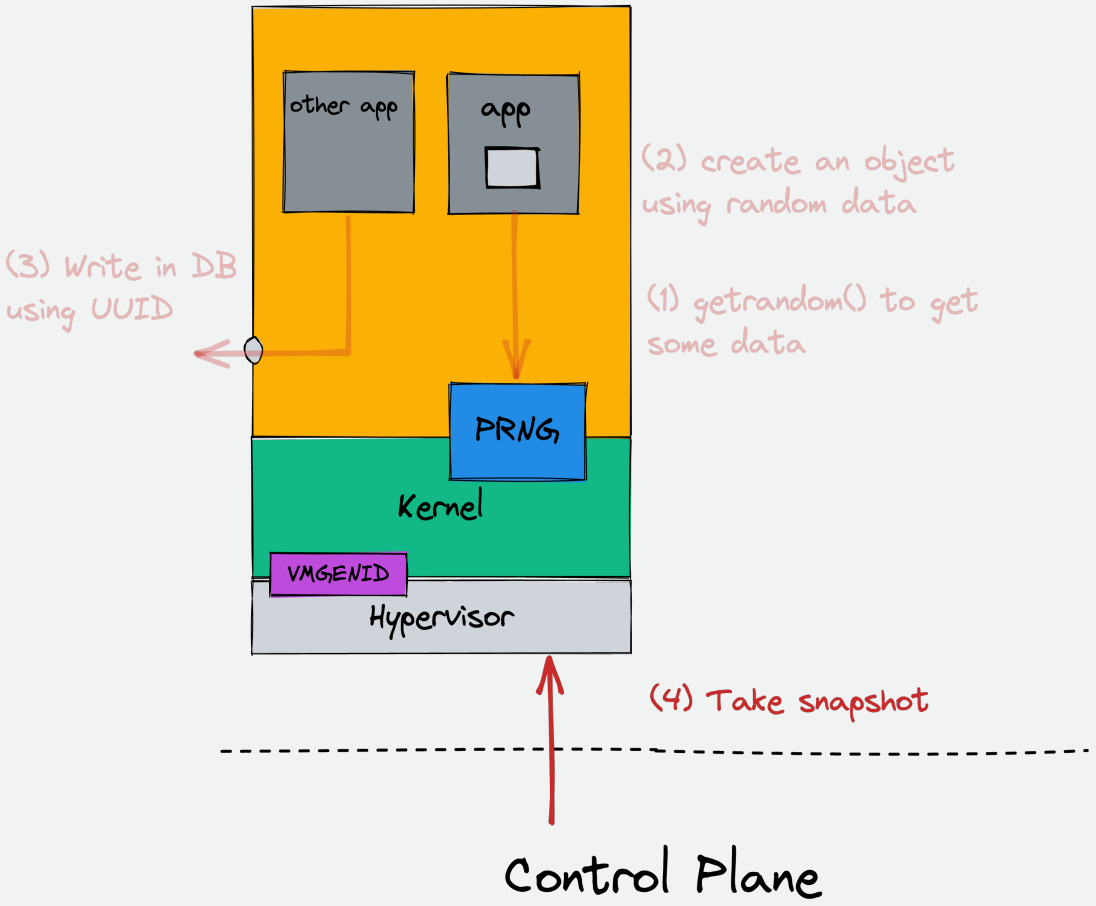
Is VM Generation ID enough?



And then VM snapshot event arrives:

- Data received from getrandom() are ok
- Application's state is now duplicated

Is VM Generation ID enough?



Not an RNG-specific issue

System-wide snapshot safety

VMGenID mechanism allows us to detect world changes, but it is not enough:

- To ensure uniqueness/secrecy across all layers we would need to validate (some) VMGenID along every step of the way
- Nothing we can do for operations that are already in-flight

System-wide snapshot safety – Observations

1. VMGenID-like mechanisms are post-mortem
   - React after snapshot & restore
   - We should probably do “something” before the VM event

System-wide snapshot safety – Observations

1. VMGenID-like mechanisms are post-mortem
   - React after snapshot & restore
   - We should probably do “something” before the VM event
2. Problem arises by allowing VM events arriving at arbitrary points in time

System-wide snapshot safety – Observations

1. VMGenID-like mechanisms are post-mortem
   - React after snapshot & restore
   - We should probably do “something” before the VM event
2. Problem arises by allowing take-a-snapshot events arriving at arbitrary points in time
3. Problem is important only when we communicate with the outter world

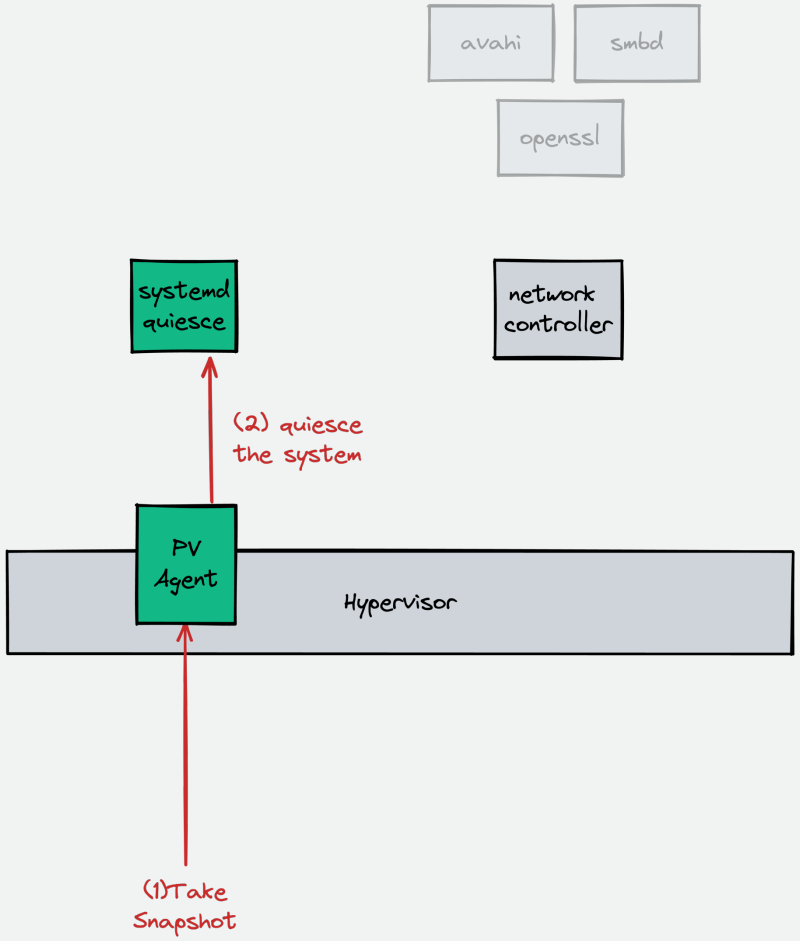
System-wide snapshot safety – Solution

1. Control the timing of snapshot events
2. Only perform the event when it is safe
   - ➢ No communication with outer world (e.g. cut the network)
3. During snapshot-restore allow applications to re-adjust before marking the system as *safe*
   - ➢ e.g. restart the network

System-wide snapshot safety - systemd

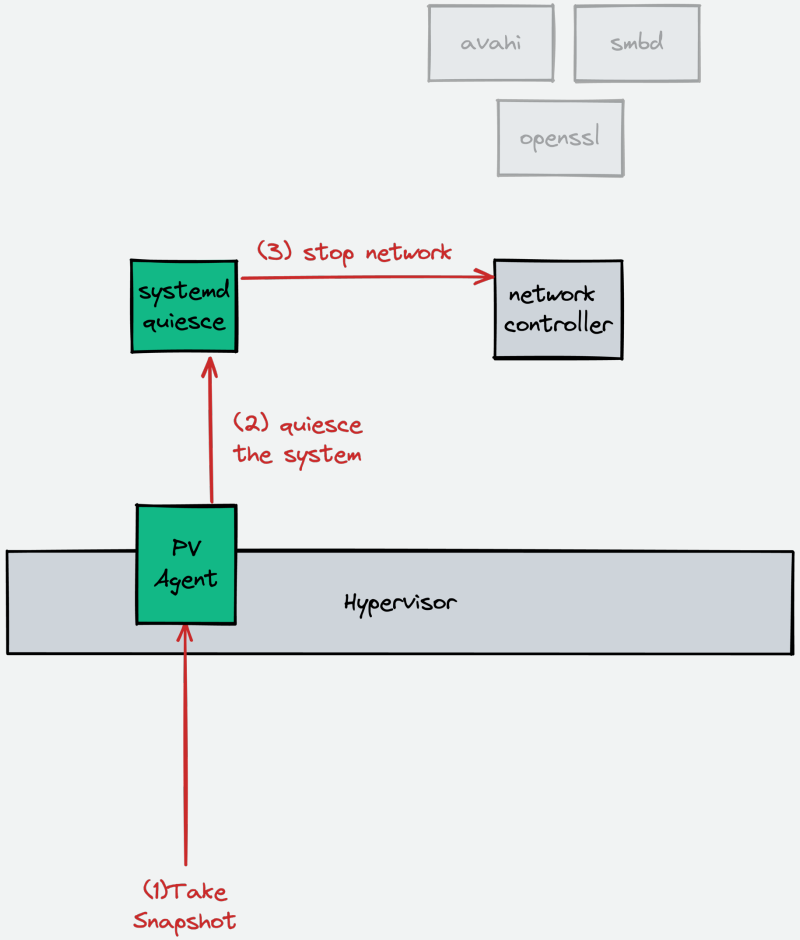
- Model this in systemd by defining four states:
  running → quiescing → quiesced → unquiescing → running
- Define inhibitors for quiescing/unquiescing transitions
  - Similar to inhibitors for *systemctl suspend*
  - Network Manager, networkd, etc. would get inhibitor locks for the former
  - Avahi, openssh, language runtimes etc, would get inhibitor locks for the latter
- Paravirtual interface and system service to orchestrate everything

System-wide snapshot safety – Snapshot path



Paravirtual agent receives the snapshot-request and initiates system quiescing

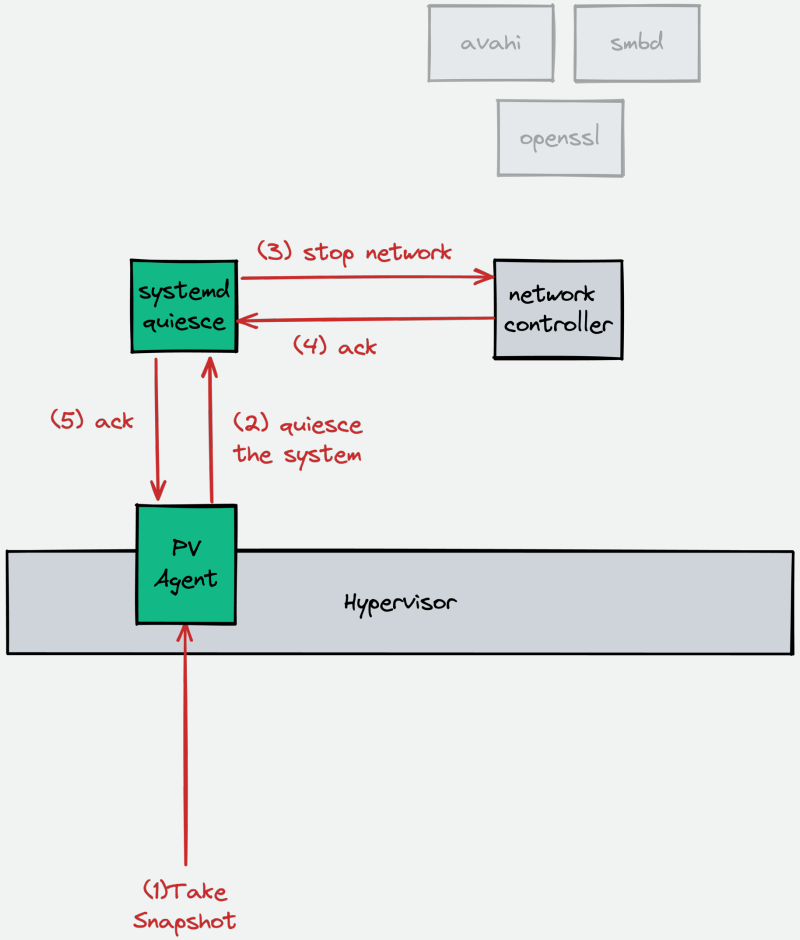
System-wide snapshot safety – Snapshot path



quiesce service shuts down the network

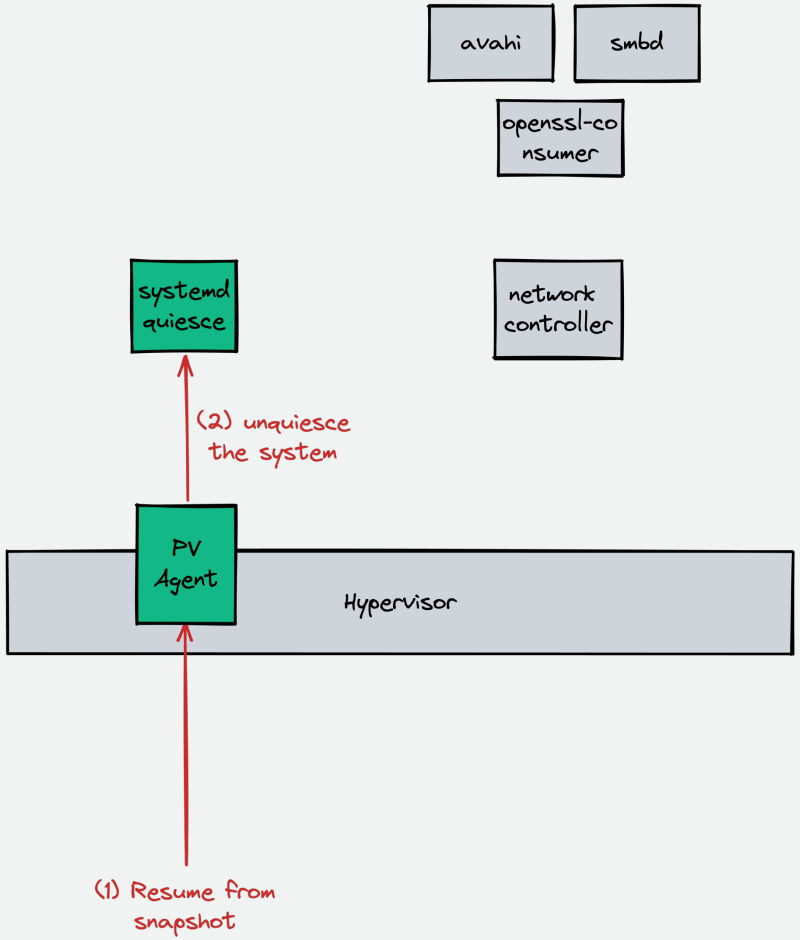
- Any other service that needs to do something pre-snapshot could participate

System-wide snapshot safety – Snapshot path



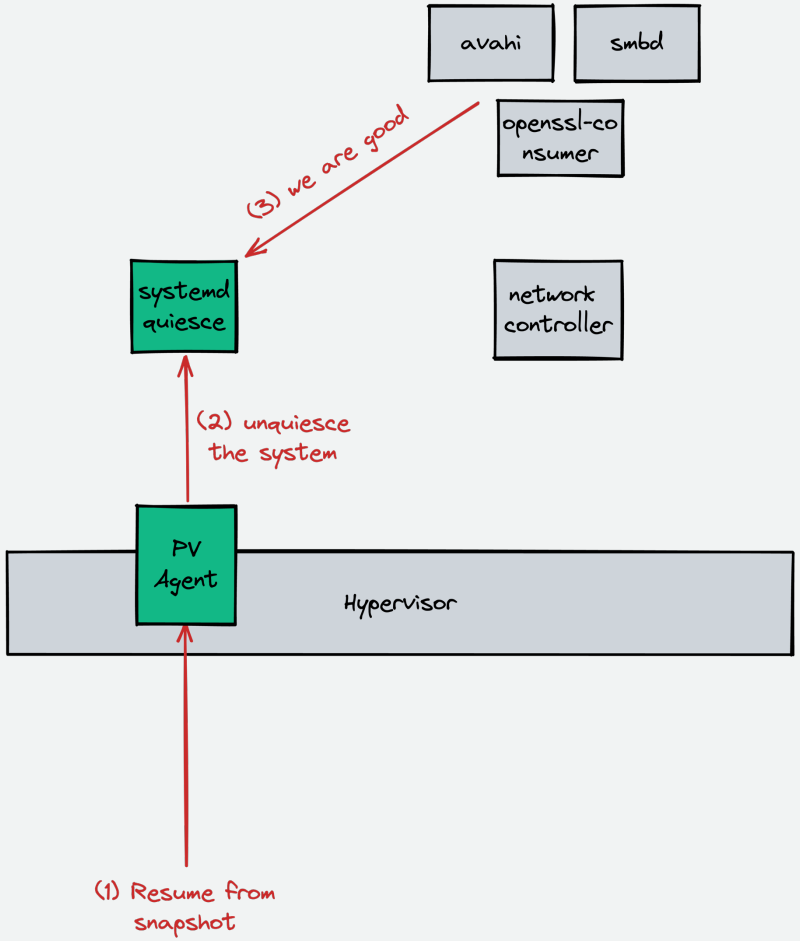
It is safe to snapshot now!

System-wide snapshot safety – Restore path



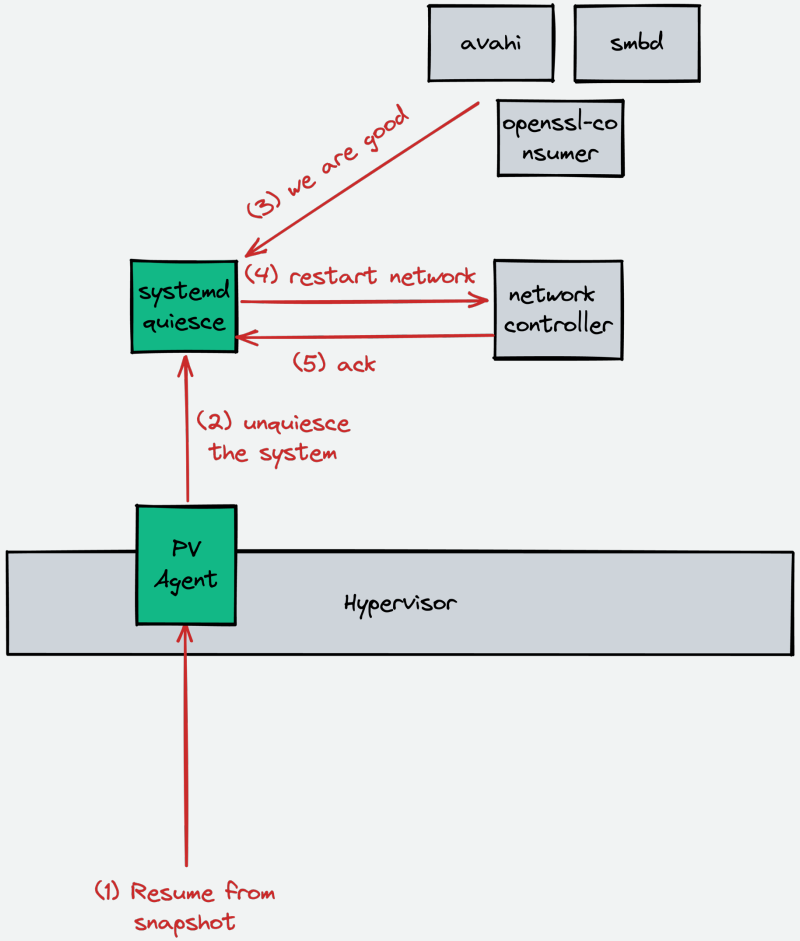
Paravirtual agent receives the restore-request and initiates system unquiescing

System-wide snapshot safety – Restore path



Applications with unique/secret state adapt to new world (through a VMGenID-like mechanism) and acknowledge they are ready

System-wide snapshot safety – Restore path



Start-up network, we ’re up, running and safe!

Summary & Next steps

Summary

1. Snaphot safety is a real problem
2. No user-space mechanism to address the issue
3. Need for system-wide solutions

Next Steps

1. Work with the community to define suitable user-space notification mechanisms and APIs
   - ➢ Provide the basic components of an end-to-end solution
2. Design and implement system-wide solution
   - ➢ Make systemd #snapsafe
   - ➢ Hopefully, more service management systems will follow

Q&A

Thank you!

Babis Chalios
bchalios@amazon.es