VIRTUALIZATION & CLOUD INFRASTRUCTURE DEVROOM
FOSDEM '24

#snapsafety: de-duplicating state across Virtual Machine clones

Babis Chalios (he/him)

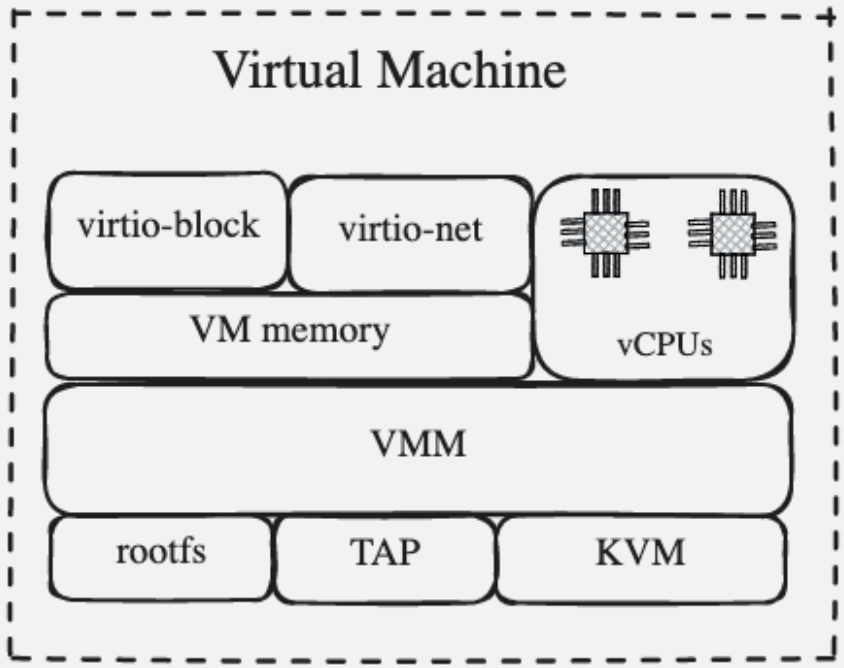
Software Development Engineer
Amazon Web Services

Agenda

1. Virtual Machine snapshots
2. #snapsafety: What’s wrong with VM snapshots?
3. #snapsafety mechanisms
4. System-level #snapsafety
5. Next steps

Virtual Machine snapshots

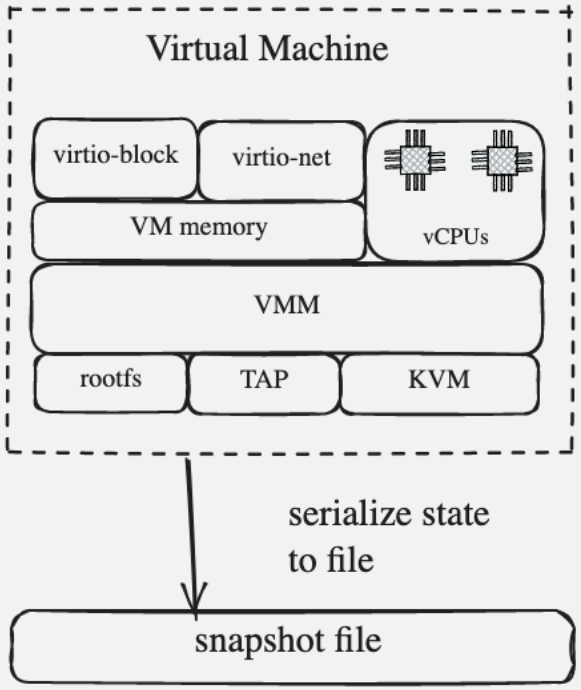
Virtual Machine snapshots



Think of a VM as:

- Memory
- VM architectural state
- Device state
- Host resources

Virtual Machine snapshots

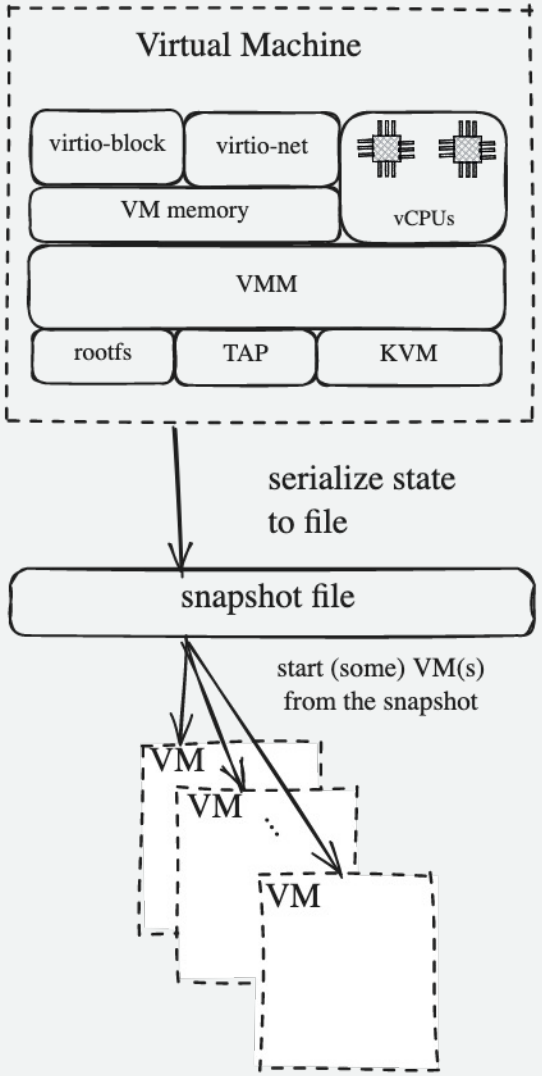


Think of a VM as:

- Memory
- VM architectural state
- Device state
- Host resources

A VM snapshot is the serialization of this state in a file

Virtual Machine snapshots



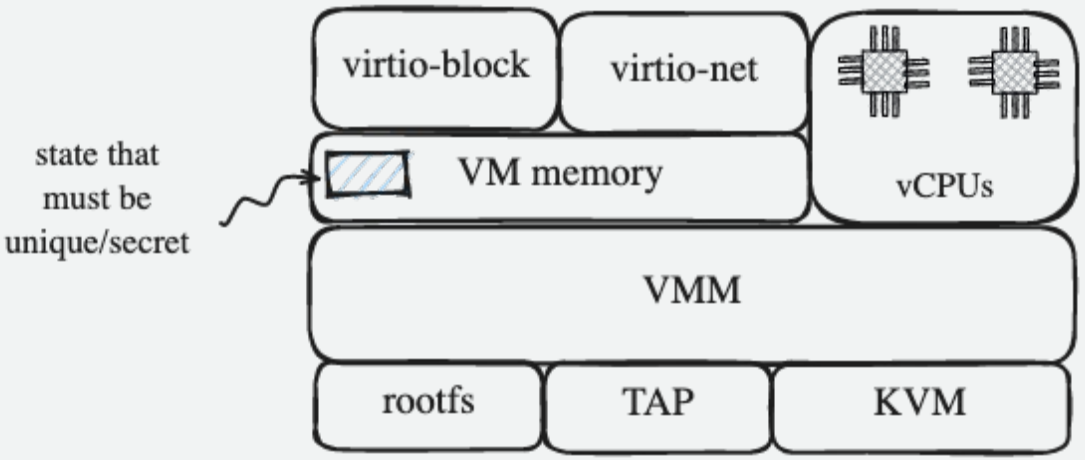
Think of a VM as:

- Memory
- VM architectural state
- Device state
- Host resources

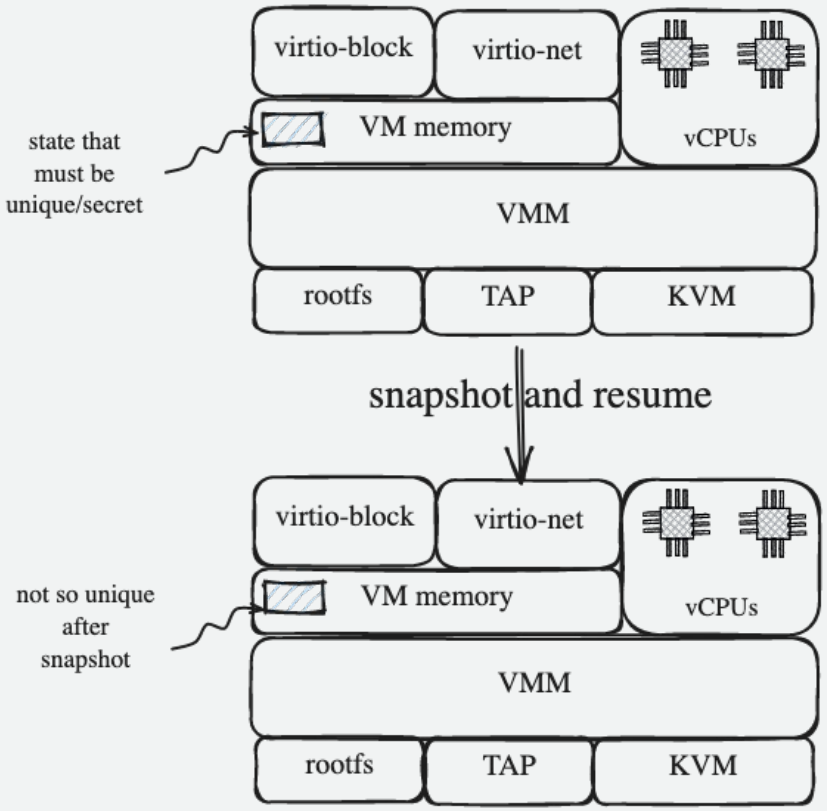
A VM snapshot is the serialization of this state in a file

Which we can later use to launch one or more identical VMs

#snapsafety: What's wrong with VM snapshots?



There is state in the VM memory that needs to be unique and/or secret



There is state in the VM memory that needs to be unique and/or secret

After taking a snapshot and starting new VMs from it, this property is lost

Many classes of applications rely on the assumption of some state being unique / secret

Many classes of applications rely on the assumption of some state being unique / secret

- getrandom(), or userspace CSPRNGs (OpenSSL, AWS-LC, etc)
  - CSPRNGs in different VMs give the same random bits

Many classes of applications rely on the assumption of some state being unique / secret

- getrandom(), or userspace CSPRNGs (OpenSSL, AWS-LC, etc)
  - CSPRNGs in different VMs give the same random bits
- Network configuration
  - VMs with the same IP / MAC addresses appear in the network

Many classes of applications rely on the assumption of some state being unique / secret

- getrandom(), or userspace CSPRNGs (OpenSSL, AWS-LC, etc)
  - CSPRNGs in different VMs give the same random bits
- Network configuration
  - VMs with the same IP / MAC addresses appear in the network
- Anything that relies on GUIDs
  - Multiple clients modifying database entries using the same GUID as index

Many classes of applications rely on the assumption of some state being unique / secret

- getrandom(), or userspace CSPRNGs (OpenSSL, AWS-LC, etc)
  - CSPRNGs in different VMs give the same random bits
- Network configuration
  - VMs with the same IP / MAC addresses appear in the network
- Anything that relies on GUIDs
  - Multiple clients modifying database entries using the same GUID as index
- Other(?)

#snapsafety mechanisms

Virtual Machine Generation ID (VMGenID)

- Notification mechanism that lets VM know it started from a snapshot
- ACPI virtual device, emulated by VMM, provides generation ID to guest
  - Cyptographically random 128-bits
  - Changes every time the VM resumes from a snapshot
- Linux kernel uses the generation ID to reseed its PRNG
- Relies on handling of ACPI notification
  - Slightly racy

Virtual Machine Generation ID - uevent

- Initial Linux VMGenID implementation did not allow for user space notifications
  - Generation ID used as entropy -> better not expose it
- Extend VMGenID to send a uevent to user space every time generation ID changes
  - Present in Linux >= 6.8
- Asynchronous notification mechanism
  - Perfect fit for applications with event loops

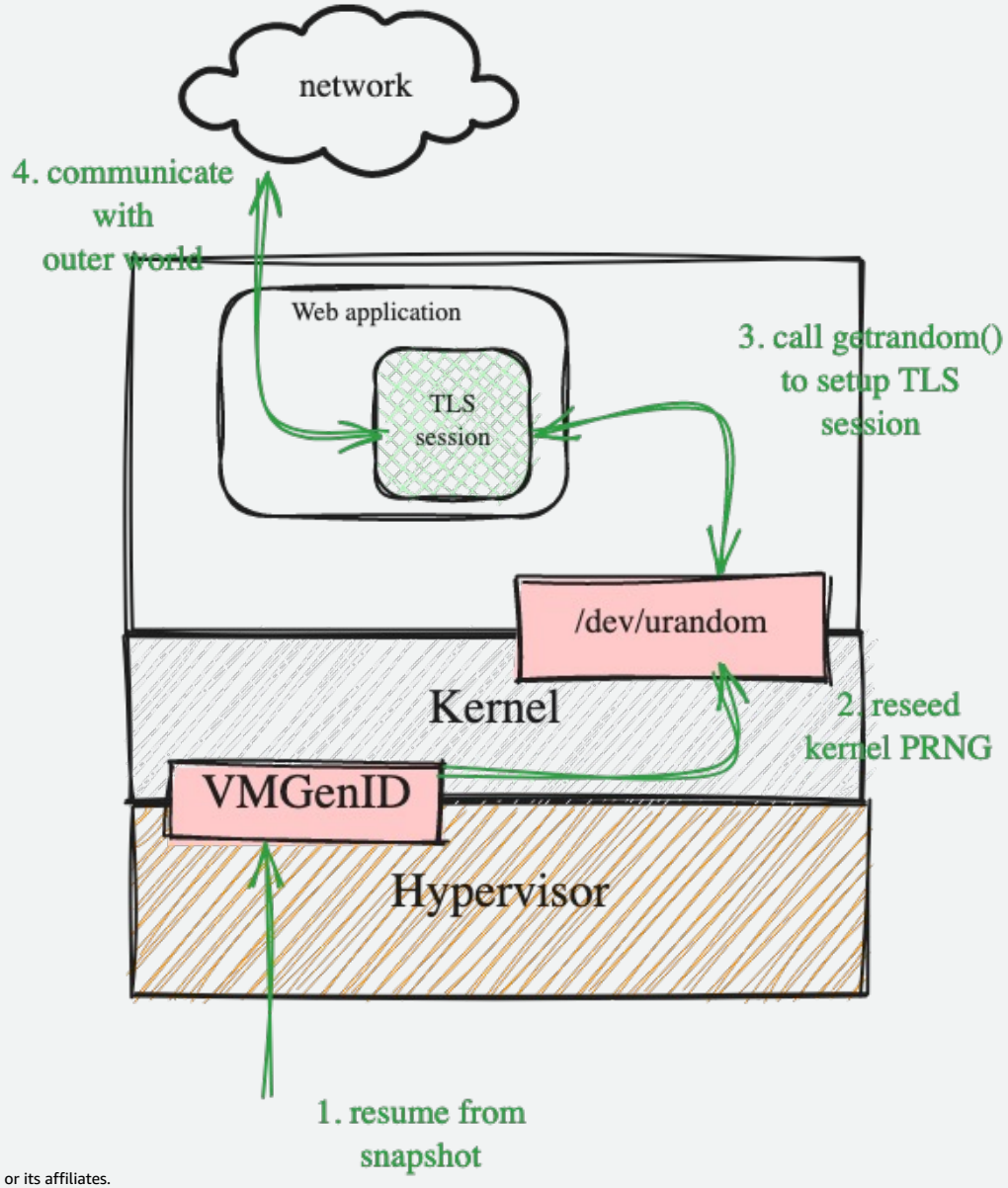
Cryptographically Secure PRNGs

- Often uses for security-sensitive applications, e.g. TLS
- CSPRNGs in runtime systems like JVM
  - Can now use VMGenID uevent mechanism to be notified about snapshots
- CSPRNGs in libraries (OpenSSL, AWS-LC, etc)
  - They don’t have an event loop to check for the event
- We should explore other mechanisms
  - Prediction resistance with HW instructions (RDRAND)
  - Build suitable APIs on top of VMGenID uevent

System-level #snapsafety

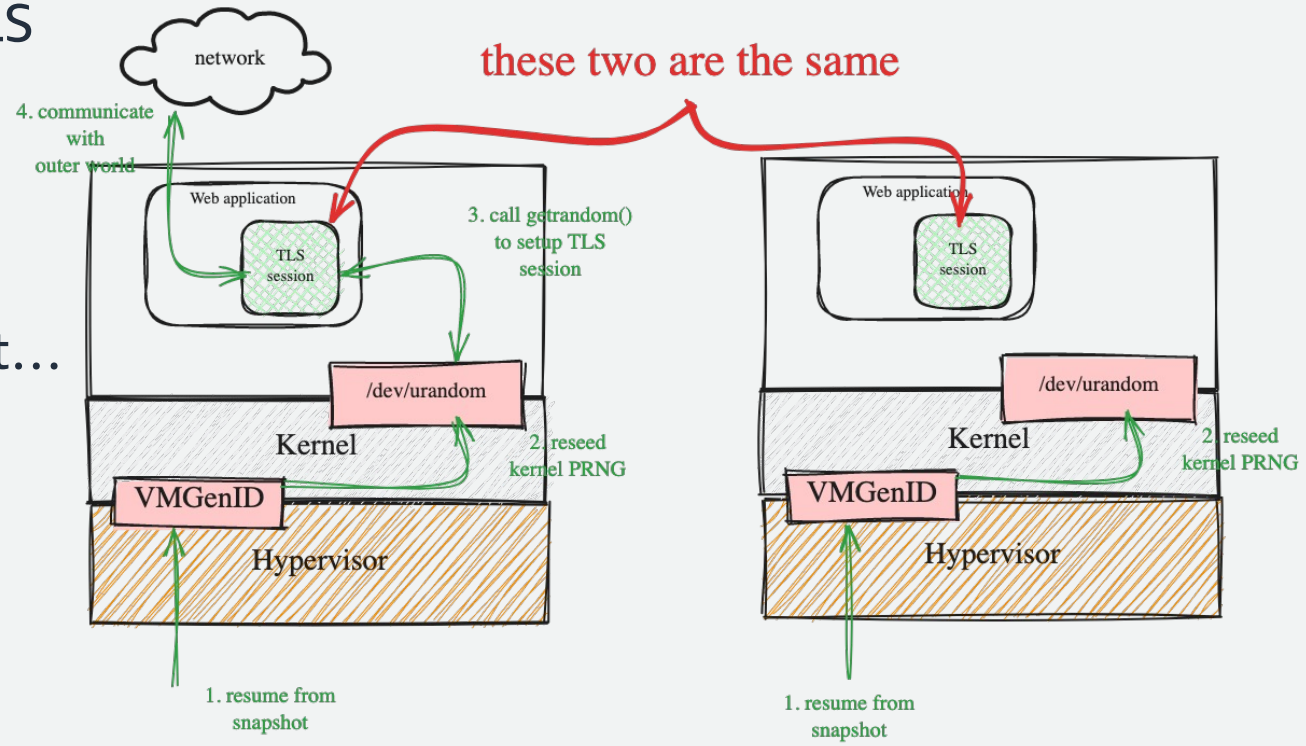
Is VMGenID enough?

1. Resume a VM from a snapshot
2. VMM & Linux supports VMGenID
   - ➢ The kernel will reseed its PRNG using the generation ID
3. Some application reads random bytes from /dev/urandom to create a TLS session key
   - ➢ /dev/urandom is safe!!
4. Application starts communicating over network
5. Take a snapshot of the VM



Is VMGenID enough?

- Snapshot at this point will duplicate the TLS state
- Even though the kernel PRNG is safe, the system is not
- Application *could* use VMGenID uevent, but…
  - uevent available in clone, not initial VM
  - Race condition in reacting
- TLS state should not be serialized in the snapshot in the first place



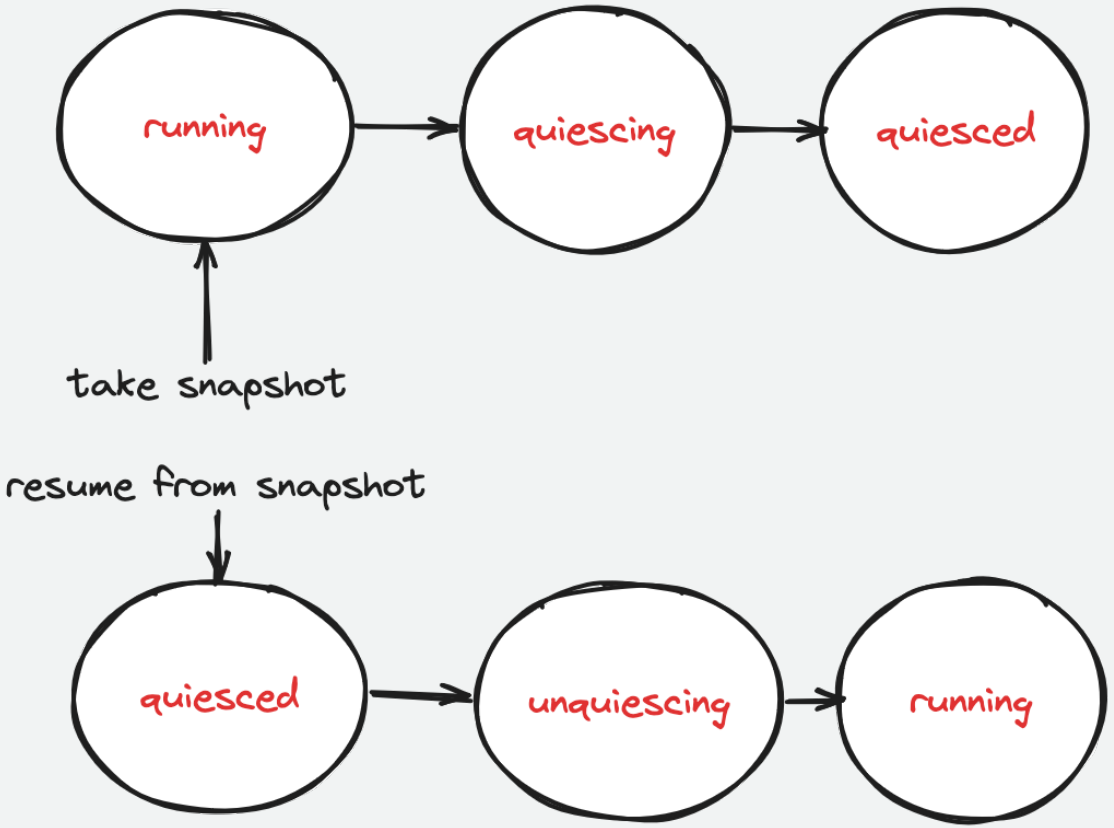
VMGenID is a “post-mortem” mechanism

- Notification that VM has resumed from a snapshot
- Sensitive info might already be in-flight

Control the timing of snapshot events

- We should do “something” before we even take the snapshot
- Only take snapshot when it is safe to do, e.g. no in-flight TLS connections
- Only resume normal system operation after all components have been notified about resume event

System-level #snapsafety: systemd prototype



Model when it is safe for a system to run:

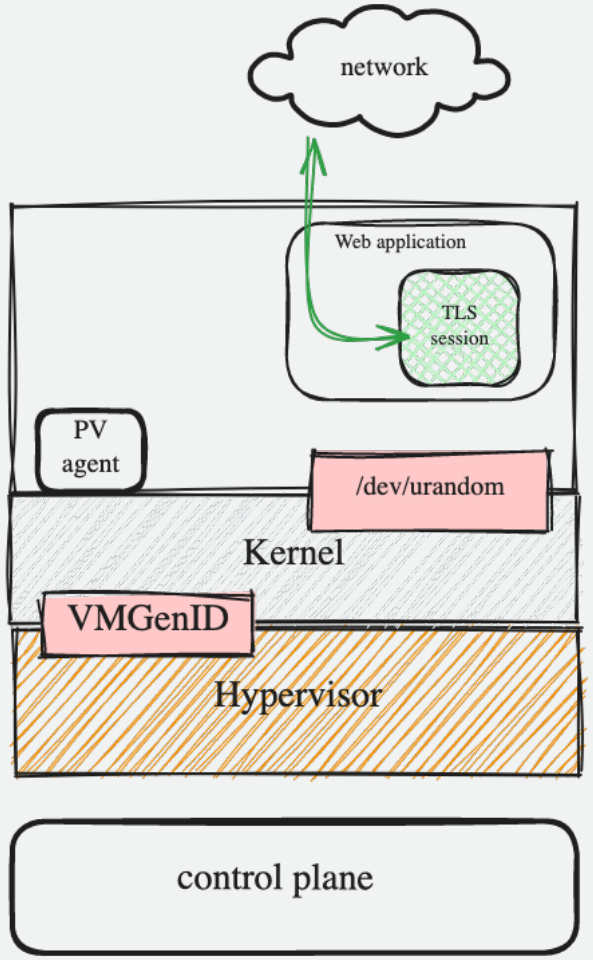
- quiesce before getting a snapshot
- After resuming from snapshot, unquiesce only after it is safe

Inhibitors for quiescing/unquiescing transitions

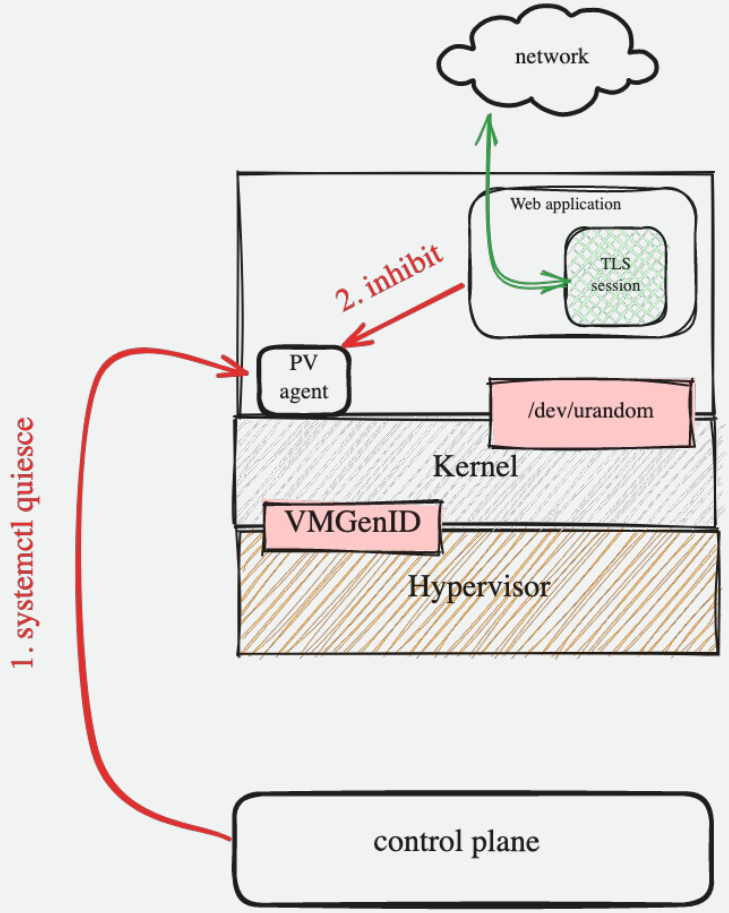
- systemctl suspend?

Paravirtual agent to orchestrate everything

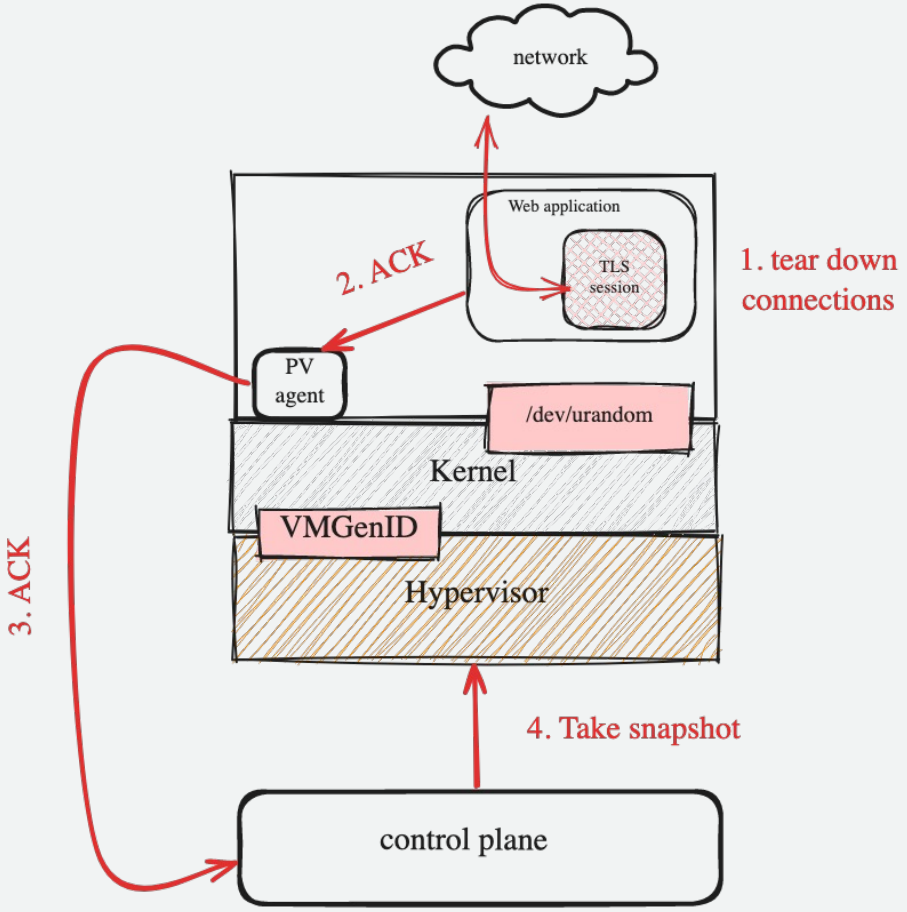
State: running (pre-snapshot)



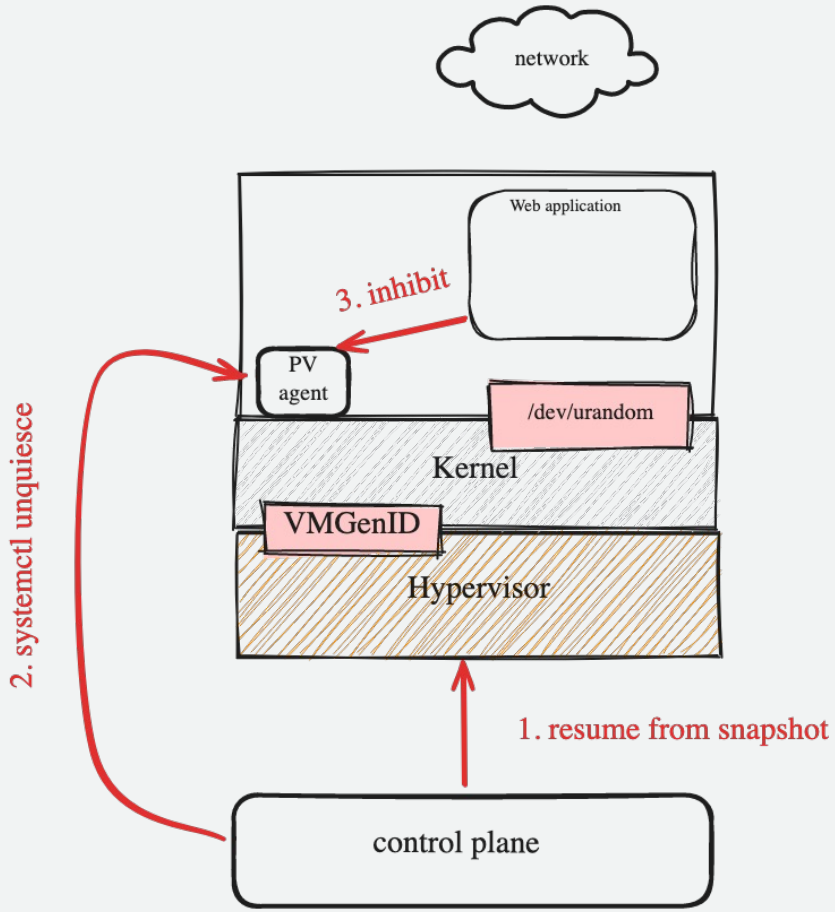
State: quiescing



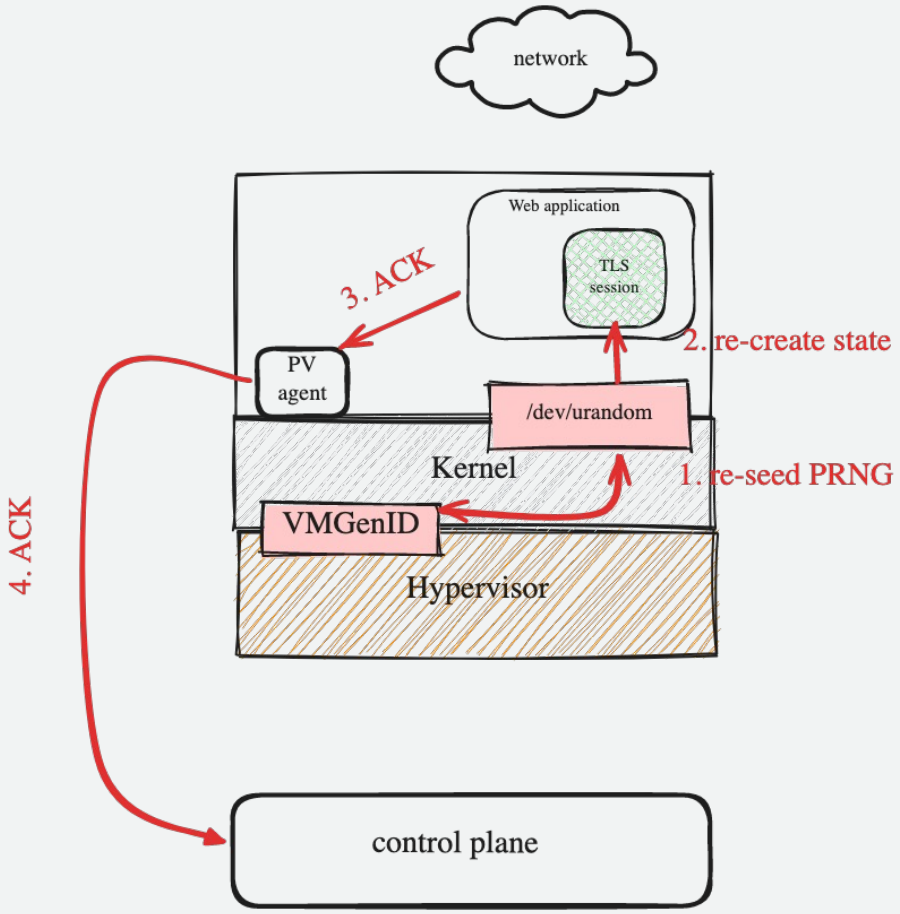
State: quiesced



State: unquiescing



State: running (post-resume)



Summary & Next steps

Next Steps

1. Support for VMGenID in Firecracker
2. Work with PRNG owners to find proper ways to make libraries #snapsafe
3. Make systemd #snapsafe
   - Ground work already done in https://github.com/systemd/systemd/issues/19269
   - Hopefully, more service management systems will follow

Q&A

Thank you!

Babis Chalios
bchalios@amazon.es