VM crash investigation: Unmasking the culprits

Alice Frosi
Principal Software Engineer

Red Hat


libvirt

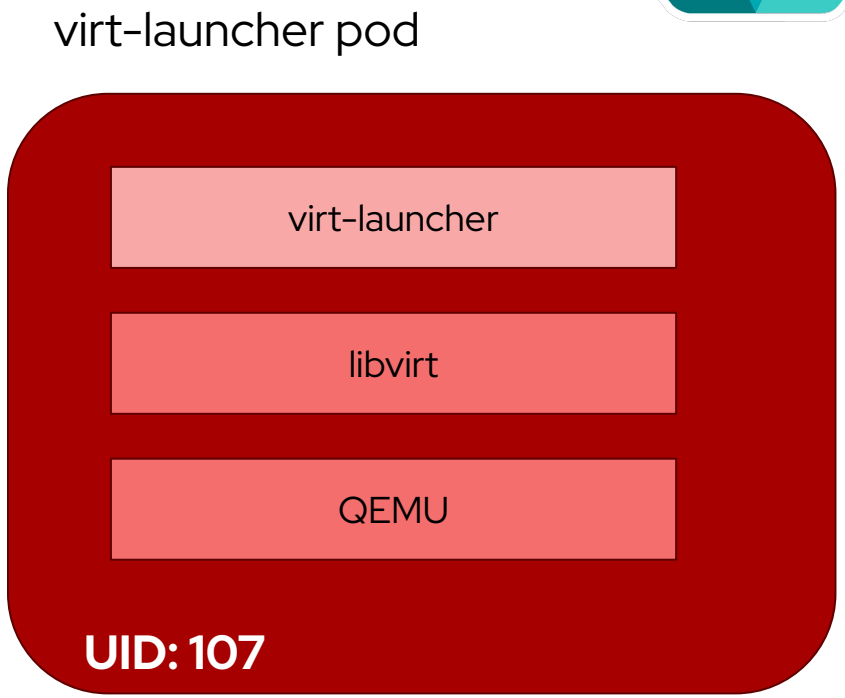
Virt stack deployed in an unprivileged container

- Run isolated in namespaces (e.g mount and pid)
- Run in session mode
- With a minimal set of capabilities

KubeVirt images are distroless images

- Minimal packages in the image for production
- No package manager -> not possible to install any additional packages on the fly

In Kubernetes clusters, regular users have minimal access to the resources and restrictions to the type of pods




KVM


libvirt

`kubectl exec` command execute arbitrary commands in a pod

Virt-launcher container image contains

- virsh
  - Checking XML results or guest status
- virt-admin
  - controlling the logging
- qemu-img
  - inspecting the disk images

If you have **cluster admin privileges**:

- Visibility in all Kubernetes namespaces
- Able to execute privileged containers

Use kubctl debug command

```
$ kubectl debug node/node02 -it --image=quay.io/centos/centos:stream9

[root@node02 /]# ps aux |grep qemu-kvm
107         28702  1.7  3.3 2259844 166508 ?      Sl   09:35   0:08
/usr/libexec/qemu-kvm -name guest=default_vmi-ephemeral,debug-threads=on ..

[root@node02 /]# ls /host/etc/os-release
/host/etc/os-release
[root@node02 /]# cat /host/etc/os-release
NAME="CentOS Stream"
VERSION="9"
ID="centos"
```


https://kubevirt.io/user-guide/debug_virt_stack/privileged-node-debugging/

Ptrace running processes (e.g with strace or gdb)

```
[root@node02 /]# ps aux |grep qemu-kvm
107         28702  1.7  3.3 2259844 166508 ?  Sl   09:35
[root@node02 /]# strace -p 28702
```

Directly execute a command on the host

```
[root@node02 /]# chroot /host
```

Inspecting the containers with crictl

```
$ crictl ps |grep compute
67bc7be3222da Running compute virt-launcher-xg989
$ crictl inspect 67bc7be3222da
"mounts": [
 {
  "containerPath": "/var/run/libvirt",
  "hostPath":
"/var/lib/kubelet/pods/2ccc3e93/volumes/kubernetes.io~empt
y-dir/libvirt-runtime",
 },
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: node01-debug
spec:
  containers:
  - command:
    - /bin/sh
    image: registry:5000/debug-tools:latest
    imagePullPolicy: Always
    name: debug
    securityContext:
      privileged: true
      runAsUser: 0
    volumeMounts:
    - mountPath: /host
      name: host
  hostNetwork: true
  hostPID: true
  nodeName: node01
  volumes:
  - hostPath:
      path: /
      type: Directory
    name: host
```

https://kubevirt.io/user-guide/debug_virt_stack/privileged-node-debugging/
https://kubernetes.io/docs/tasks/debug/debug-cluster/crictl/

Certain tools require the target binary (e.g gdb or systemtap scripts)

Getting the QEMU binary from the target KubeVirt version

```
$ export registry=$(kubectl get kubevirt
kubevirt -n kubevirt  -o
jsonpath='{.status.observedDeploymentConfig}'
|jq '.registry'|tr -d "\"")
$ echo $registry
"registry:5000/kubevirt"

$ export tag=$(kubectl get kubevirt kubevirt
-n kubevirt  -o
jsonpath='{.status.observedDeploymentConfig}'
|jq '.virtLauncherSha'|tr -d "\"")
$ echo $tag
"Sha256:6c8b85eed8e83a4c70779836b246c057d3e882
eb513f3ded0a02e0a4c4bda837"

$ podman build -t debug-tools \
    --build-arg registry=$registry  \
    --build-arg tag=@$tag .
```

```
ARG registry
ARG tag
FROM ${registry/virt-launcher:${tag} AS launcher

FROM quay.io/centos/centos:stream9

RUN yum install -y \
        gdb \
        systemtap-client \
        systemtap-devel \
    && yum clean all
COPY --from=launcher /usr/libexec/qemu-kvm
/usr/libexec/qemu-kvm
```

- Virt-launcher log level →libvirt and QEMU logging level
- VM pod launch with an annotation on the pod

```
annotations:
  kubevirt.io/libvirt-log-filters: "1:libvirt 1:qemu 1:qemu.qemu_monitor 3:*"
```

- Change logging level dynamically at runtime with virt-admin by execing in virt-launcher

```
$ kubectl exec -ti  virt-launcher -- virt-admin  -c virtqemud:///session
daemon-log-filters "1:libvirt 1:qemu 1:qemu.qemu_monitor 3:*"
```

or redirect it to a file

```
$ kubectl  exec -ti virt-launcher -- virt-admin  -c virtqemud:///session
daemon-log-outputs "1:file:/var/run/libvirt/libvirtd.log"
$ kubectl cp virt-launcher:/var/run/libvirt/libvirtd.log libvirt-kubevirt.log
```

KubeVirt sidecars are extra containers able to modify the guest XML

Sidecar shim allow to create a configMap with bash or python script to modify the XML

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-config-map
data:
  my_script.sh: |
    #!/usr/bin/env python3

  ...
```

On the VM definition:

```
annotations:
  hooks.kubevirt.io/hookSidecars: >
    [
        {
            "args": ["--version", "v1alpha2"],
            "configMap": {"name": "my-config-map", "key":
"my_script.sh", "hookPath": "/usr/bin/onDefineDomain"}
        }
    ]
```

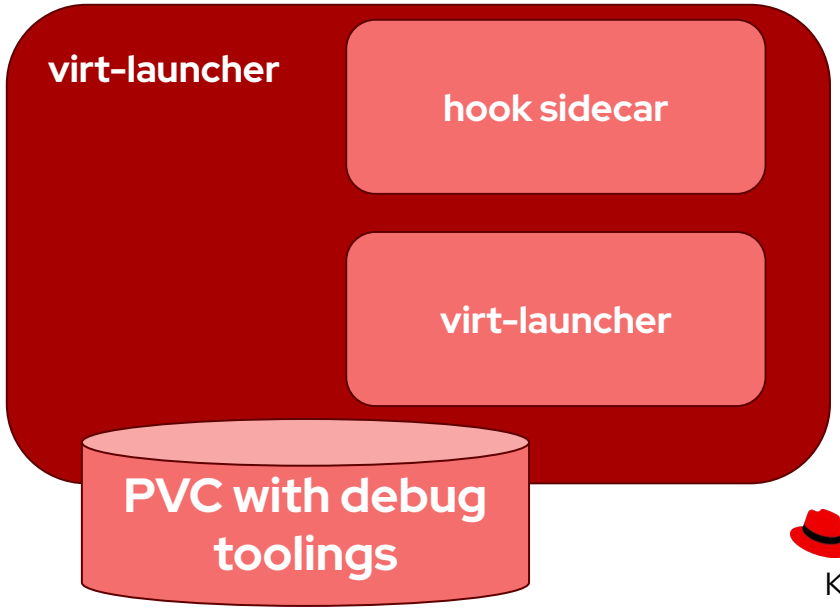
https://kubevirt.io/user-guide/user_workloads/hook-sidecar/
https://gist.github.com/victortoso/4a54ef0e75a2b917bb21cb33ec4da141

Start QEMU with a script and a debugging tool

```
FROM quay.io/centos/centos:stream9 as build

ENV DIR /debug-tools
RUN mkdir -p ${DIR}/logs

RUN  yum install  --installroot=${DIR} -y strace && yum
clean all

COPY ./wrap_qemu_strace.sh $DIR/wrap_qemu_strace.sh
RUN chmod 0755 ${DIR}/wrap_qemu_strace.sh
RUN chown 107:107 ${DIR}/wrap_qemu_strace.sh
RUN chown 107:107 ${DIR}/logs
```

```
#!/bin/bash

LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/var/run/debug/usr/lib64
/var/run/debug/usr/bin/strace \
        -o /var/run/debug/logs/strace.out \
        /usr/libexec/qemu-kvm $@
```

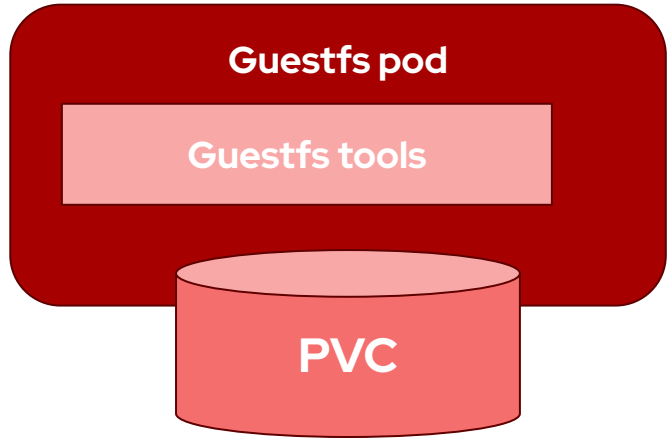
```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-config-map
data:
  my_script.sh: |
    #!/bin/sh
    tempFile=`mktemp --dry-run`
    echo $4 > $tempFile
    sed -i
"s|<emulator>/usr/libexec/qemu-kvm</emulator>|
<emulator>/var/run/debug/wrap_qemu_strace.sh</
emulator>|" $tempFile
    cat $tempFile
```



https://kubevirt.io/user-guide/debug_virt_stack/launch-qemu-strace/

Guestfs tools can be used to inspect and repair broken disk images

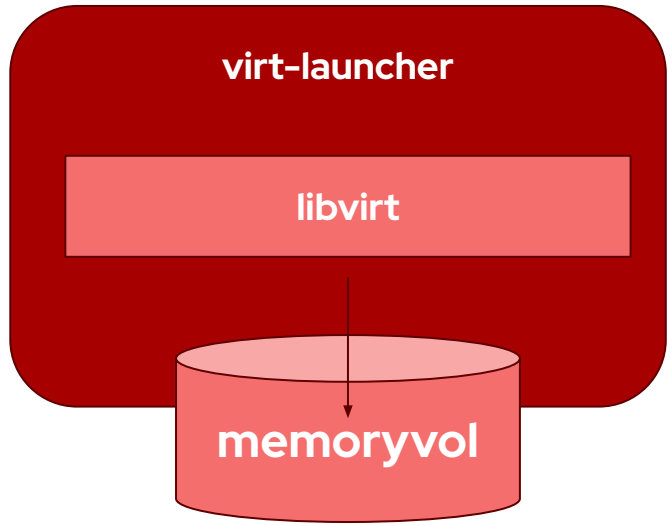
```
$ virtctl guestfs pvc
bash-5.0# virt-rescue -a disk.img
[...]
><rescue> fdisk -l
Device       Start      End  Sectors Size Type
/dev/sda1     2048     4095     2048   1M BIOS boot
/dev/sda2     4096  2101247  2097152   1G Linux filesystem
/dev/sda3  2101248 12580863 10479616   5G Linux filesystem

><rescue> mount /dev/sda3 sysroot/
><rescue> mount /dev/sda2 sysroot/boot
><rescue> chroot sysroot/
><rescue> ls boot/
System.map-5.11.12-300.fc34.x86_64
config-5.11.12-300.fc34.x86_64
efi
grub2
initramfs-0-rescue-8afb5b540fab48728e48e4196a3a48ee.img
initramfs-5.11.12-300.fc34.x86_64.img
loader
vmlinuz-0-rescue-8afb5b540fab48728e48e4196a3a48ee
><rescue> dracut -f boot/initramfs-5.11.12-300.fc34.x86_64.img 5.11.12-300.fc34.x86_64
```



https://kubevirt.io/user-guide/storage/guestfs/

Troubleshooting the guest memory

```
$ virtctl memory-dump get myvm --claim-name=memoryvol
--create-claim --output=memoryDump.dump.gz
```



Monitoring the memory dump progress from the VM status

```
memoryDumpRequest:
  claimName: memory-dump
  phase: Completed
  startTimestamp: "2022-03-29T11:00:04Z"
  endTimestamp: "2022-03-29T11:00:09Z"
  fileName: my-vm-my-pvc-20220329-110004
```

Uses the hotplug and export functionalities from KubeVirt, hence the dump volume can be dynamically attached and detached, and locally downloaded.

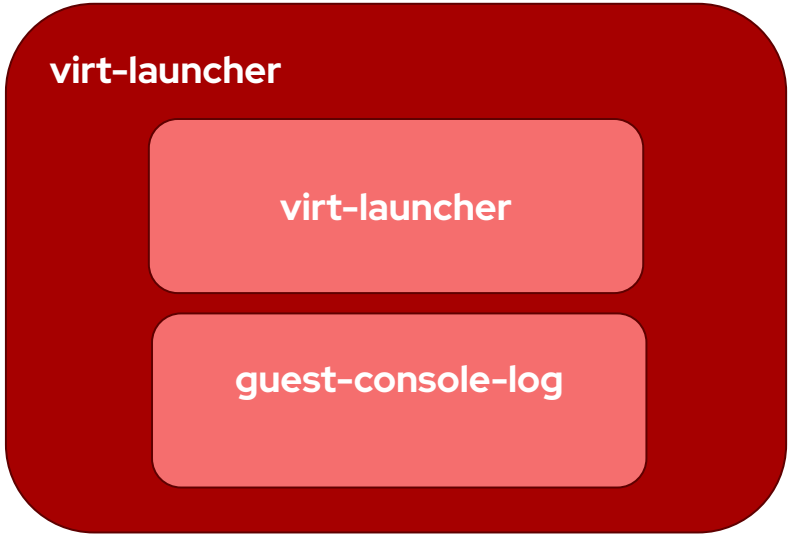
https://kubevirt.io/user-guide/compute/memory_dump/

Interactively attaching to the guest console

```
$ virtctl console vm
```

or through VNC

```
$ virtctl vnc vm
```



Guest console is redirected to a container output

```
$ kubectl logs virt-launcher -c guest-console-log
```

- `kubectl logs` and `kubectl describe`
  - retrieve the container logs or information
- `kubectl port-forward`
  - Forward a container port locally
- `kubectl debug`
  - Ephemeral containers, start an additional container inside a running pod
- `kubectl get events`
  - Events generated by the controllers

https://kubernetes.io/docs/tasks/access-application-cluster/port-forward-access-application-cluster/
https://kubernetes.io/docs/concepts/workloads/pods/ephemeral-containers/

- KubeVirt supports some extra annotations for debugging or functional tests
  - `kubevirt.io/keep-launcher-alive-after-failure`→keep virt-launcher running, useful for inspecting the container
  - kubevirt.io/func-test-block-migration-target-preparation →doesn't finish the migration virt-launcher running, useful for inspecting the source and destination containers during the migration
- Control and increase the verbosity of kubevirt components using
  - `virtctl adm log-verbosity --virt-handler=5`
- Port forwarding
  - `virtctl port-forward`
- Retrieve guest information from the guest agent
  - `virtctl guestosinfo vm`

- Different Kubernetes users can have different privileges and different debugging capabilities
- From the KubeVirt released version, you know exactly the LibVirt and QEMU binary version
- Distroless images lack of debugging tools, but possible with KubeVirt sidecars and PVCs to add them
- virtctl and kubectl are your debugging companions

Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.


linkedin.com/company/red-hat


youtube.com/user/RedHatVideos


facebook.com/redhatinc


twitter.com/RedHat

Red Hat