KubeVirt: Virtuelle Maschinen neu in Kubernetes denken

Category: Online-Marketing
geschrieben von Tobias Hager | 6. Februar 2026

KubeVirt: Virtuelle Maschinen neu in Kubernetes denken

Du dachtest, Kubernetes sei nur für Container? Denk nochmal nach. Mit *KubeVirt* wird deine heißgeliebte VM plötzlich zum First-Class-Citizen im Kubernetes-Cluster – und das ist nicht nur ein nettes Gimmick, sondern ein massiver Paradigmenwechsel. Willkommen in der Welt, in der klassische Infrastruktur und Cloud-native Welten endlich zusammenwachsen – in einer Art,

die nicht nur DevOps-Nerds, sondern auch alte Sysadmins aus der Rente holt.

- Was KubeVirt ist und warum es Kubernetes fundamental erweitert
- Wie KubeVirt virtuelle Maschinen in Container-Workflows integriert
- Warum das für DevOps, SREs und Plattform-Teams ein Gamechanger ist
- Welche Architektur hinter KubeVirt steckt – von libvirt bis QEMU
- Wie du KubeVirt installierst, konfigurierst und produktiv einsetzt
- Was KubeVirt für Hybrid-Cloud, Legacy-Workloads und Edge bedeutet
- Welche Herausforderungen bleiben – Netzwerk, Storage, Performance
- Warum KubeVirt die nächste Evolutionsstufe für Kubernetes ist

Virtuelle Maschinen und Kubernetes – das klingt im ersten Moment wie ein Rückschritt. Schließlich war der ganze Container-Hype doch genau dafür da, VMs endlich abzulösen, oder? Falsch gedacht. KubeVirt bringt nicht nur VMs in Kubernetes, sondern verbindet das Beste aus beiden Welten. Legacy-Workloads, die du nicht einfach containerisieren kannst? Kein Problem. Infrastruktur, die du lieber als VM managst? Auch kein Problem. Mit KubeVirt orchestrierst du VMs genauso elegant wie Pods – mit denselben Tools, denselben APIs, derselben Automatisierung. Und genau deshalb ist KubeVirt nicht einfach ein weiteres Tool, sondern ein strategischer Gamechanger.

KubeVirt erklärt: Kubernetes trifft virtuelle Maschinen

KubeVirt ist ein Kubernetes-Addon, das virtuelle Maschinen als native Kubernetes-Ressourcen behandelt. Ja, du hast richtig gehört – deine gute alte VM wird zum Kubernetes-Objekt, das du mit kubectl starten, stoppen, skalieren und überwachen kannst. Es ist die Brücke zwischen dem klassischen VM-Paradigma und der containerisierten, cloud-nativen Welt. Der Hauptkeyword KubeVirt ist dabei keine Spielerei, sondern ein ernstzunehmender Layer in deiner Infrastrukturstrategie – vor allem, wenn du nicht bei null anfängst, sondern jahrzehntelange Legacy mitschleppst.

Die Grundidee hinter KubeVirt: Statt Container und VMs getrennt zu behandeln, integrierst du sie in eine einheitliche Plattform. Entwickler und Ops müssen sich nicht mehr entscheiden, ob ein Workload in eine VM oder einen Container gehört – sie definieren einfach, was gebraucht wird, und Kubernetes kümmert sich um den Rest. Das reduziert Komplexität, verbessert Konsistenz und erhöht die Automatisierungsmöglichkeiten massiv.

Und ja, das funktioniert wirklich – KubeVirt basiert auf bewährten Virtualisierungstechnologien wie QEMU und libvirt, die innerhalb eines Kubernetes-Pods betrieben werden. Das bedeutet: Deine VM läuft als Pod, nutzt Kubernetes-Netzwerk und -Storage, lässt sich über kubectl managen und in CI/CD-Pipelines integrieren. Willkommen in der Zukunft der Virtualisierung.

Das macht KubeVirt besonders interessant für Teams, die sowohl moderne als auch traditionelle Workloads betreiben müssen – also im Prinzip für alle, die nicht erst gestern ihren Tech-Stack gestartet haben. Es ist der Missing Link zwischen Cloud-Native und Legacy, zwischen DevOps und klassischer IT,

zwischen Container und VM.

Die Architektur von KubeVirt: libvirt, QEMU und Kubernetes im Dreiklang

Damit du verstehst, warum KubeVirt mehr ist als ein Hack, musst du dir die Architektur anschauen. Im Kern besteht KubeVirt aus einer Reihe von Kubernetes Custom Resource Definitions (CRDs), die es ermöglichen, VMs als Ressourcen zu definieren. Aber das ist nur die Oberfläche. Im Hintergrund werkeln libvirt und QEMU – die bewährten Arbeitspferde der Linux-Virtualisierung.

Jede virtuelle Maschine in KubeVirt wird als VirtualMachine oder VirtualMachineInstance (VMI) definiert. Diese Ressourcen werden von einem Operator verwaltet, der sicherstellt, dass die VMs entsprechend gestartet, gestoppt oder migriert werden. Der Clou: Die eigentliche VM läuft als Pod mit einem sogenannten virt-launcher, der QEMU enthält und die VM-Instanz startet.

libvirt kümmert sich um die Verwaltung der VM – inklusive Start, Stop, Snapshot und mehr. QEMU emuliert die Hardware. Und Kubernetes orchestriert das Ganze. Netzwerkanschlüsse werden über Multus oder andere CNI-Plugins integriert. Storage wird über PVCs und CSI-Treiber zugewiesen. Die Steuerung erfolgt über die Kubernetes-API – das heißt, du kannst alles, wirklich alles in deine bestehenden CI/CD-Prozesse integrieren.

Die Architektur sieht also so aus:

- Kubernetes CRDs: Definieren VirtualMachine und VirtualMachineInstance
- virt-controller: Kontrolliert Lebenszyklus der VMs
- virt-handler: Läuft auf jedem Node und kommuniziert mit libvirt
- virt-launcher: Pod, der QEMU startet und die VM hostet

Das Besondere: KubeVirt arbeitet vollständig im Kubernetes-Paradigma. Keine externen Management-Systeme, keine API-Brüche. Alles ist Kubernetes. Und das ist kein Zufall – sondern die Zukunft.

KubeVirt installieren und produktiv nutzen – Schritt für Schritt

Die Installation von KubeVirt ist für Kubernetes-Admins mit Erfahrung kein Hexenwerk. Aber: Du brauchst ein funktionierendes Kubernetes-Cluster mit mindestens einem Node, der Hardware-Virtualisierung unterstützt (VT-x oder

AMD-V). Ohne das wird’s nichts.

Hier die Schnellstartanleitung:

1. Kubernetes-Cluster vorbereiten: Stelle sicher, dass deine Nodes Nested Virtualization unterstützen und KVM installiert ist.
2. KubeVirt Operator installieren: Nutze die offiziellen YAML-Manifeste von kubevirt.io oder deploye via OperatorHub (z.B. in OpenShift).
3. KubeVirt CRDs anwenden: Damit deine API neue Ressourcen wie VirtualMachine kennt.
4. Test-VM deployen: Erstelle eine VirtualMachine-Definition mit einem passenden Disk-Image (z.B. CirrOS oder Fedora).
5. Netzwerk und Storage konfigurieren: Nutze Multus für zusätzliche Netzwerkschnittstellen und CSI-Treiber für persistente Volumes.

Sobald die VM läuft, kannst du sie wie jeden anderen Pod behandeln:

- Mit kubectl get vmi prüfen, ob sie läuft
- Mit virtctl console dich auf die Konsole verbinden
- Mit kubectl delete vm wieder löschen

Der Einsatz von KubeVirt in der Produktion erfordert allerdings mehr: Monitoring, Logging, Backup, Live-Migration, evtl. Integration mit Prometheus/Grafana oder ELK. Aber das Prinzip bleibt: Alles ist Kubernetes. Keine Extra-Konsole. Kein zusätzlicher Verwaltungsaufwand. Kein Shadow-IT.

KubeVirt Use Cases: Legacy, Hybrid-Cloud, Edge

Warum das Ganze? Warum VMs in Kubernetes? Die Antwort ist einfach: Weil du’s brauchst. Niemand startet heute Infrastruktur auf der grünen Wiese. Es gibt Altlasten, die du nicht einfach containerisieren kannst: Monolithische Applikationen, Datenbankserver, Windows-basierte Workloads, Spezial-Software mit Lizenzbindung an MAC-Adressen. Du willst sie nicht pflegen, aber du musst.

Und genau hier kommt KubeVirt ins Spiel. Du kannst diese Legacy-VMs weiterhin betreiben – aber orchestriert, automatisiert und skalierbar. Du bringst sie in denselben Deployment-Workflow wie deine Container. Du nutzt dieselben Tools, dieselben Pipelines, dieselben Security Policies. Kein Silo mehr zwischen Containern und VMs.

Ein besonders spannender Use Case: Edge Computing. Dort, wo du auf Bare Metal keine Container-Runtimes verwenden kannst – oder willst – aber trotzdem orchestrieren musst. Mit KubeVirt kannst du VMs auf Edge-Nodes deployen, sie zentral managen und über GitOps aktualisieren. Clean, effizient, skalierbar.

Auch für Hybrid-Cloud-Szenarien bietet KubeVirt Vorteile: Du kannst VMs sowohl on-prem als auch in der Cloud betreiben – unter einer einheitlichen Steuerung. Kein Wechsel der Umgebung, keine Brüche in der CI/CD-Pipeline.

Einfach Kubernetes, egal wo.

Grenzen und Herausforderungen von KubeVirt

Natürlich ist KubeVirt kein Wundermittel. Es gibt technische Herausforderungen, die du kennen musst. Besonders im Bereich Storage und Netzwerk musst du aufpassen. VMs sind keine Container – sie brauchen persistente, performante Storage-Systeme. Ein NFS-Share mit Latenzproblemen killt deine VM-Performance schneller, als du “PersistentVolumeClaim” sagen kannst.

Im Netzwerkbereich musst du mit Multus oder SR-IOV arbeiten, um dedizierte Interfaces bereitzustellen. Das kann komplex werden – besonders in Multi-Tenant-Umgebungen. Auch Live-Migration funktioniert, ist aber abhängig von Storage-Backend und Netzwerk-Setup. Ohne Shared Storage kein Move.

Security ist ein weiteres Thema. VMs innerhalb von Pods zu betreiben, bringt neue Angriffsflächen. Du musst sicherstellen, dass QEMU-Prozesse sauber isoliert sind, dass virt-launcher nicht kompromittiert werden kann und dass deine VM-Images vertrauenswürdig sind. Das ist machbar – aber nicht trivial.

Und: KubeVirt ist noch nicht in Version 1.0. Der Code ist stabil, aber einige Features (z. B. Windows VM-Support, Lifecycle-Automatisierung) sind noch in Entwicklung oder Beta. Für produktive Umgebungen brauchst du gute Tests, Monitoring und ein solides Verständnis der Architektur.

Fazit: Warum KubeVirt ein Gamechanger ist

KubeVirt ist nicht die Rückkehr der VMs – es ist ihre Evolution. Es ist der Moment, in dem klassische und moderne IT-Welten kollidieren und du als Admin, DevOps oder SRE endlich eine einheitliche Plattform bekommst. Keine Inseln mehr. Keine Sonderlocken. Kein “Das ist halt Legacy”.

Wenn du ernsthaft Infrastruktur betreibst – mit echten Workloads, echten Altlasten und echten Anforderungen – dann ist KubeVirt dein Upgrade-Pfad. Es bringt dir Kontrolle, Konsistenz und Automatisierung. Es macht aus alten VMs Kubernetes-native Ressourcen. Und es zeigt: Kubernetes ist nicht nur für Container. Es ist für alles, was du orchestrieren willst. Willkommen im neuen Paradigma.