KubeVirt Incubation Due Diligence

September-November 2021

Authors: Fabian Deutsch, David Vossel, Petr Horacek, Chris Callegari, Karena Angell, Diane Mueller, Josh Berkus, Peter Lauterbach

Reviewed by: Ricardo Aravena, Chandler Wilkerson, John Herr, Michael Henriksen, Ryan Hallisey, Roman Mohr

TOC Sponsor: Alena Prokharchyk
TAG Input: TAG Runtime (Ricardo Aravena)
TOC Ticket: https://github.com/cncf/toc/pull/715

Table of Contents

Introduction

Name: KubeVirt
Web site: https://kubevirt.io/
Source control: https://github.com/kubevirt
Code of Conduct: https://github.com/kubevirt/community/blob/main/code-of-conduct.md
License: APL 2.0

CNCF TOC Ticket: https://github.com/cncf/toc/pull/715

Project Description

KubeVirt technology addresses the needs of development teams that have adopted or want to adopt Kubernetes but possess existing Virtual Machine-based workloads that cannot be easily containerized. More specifically, the technology provides a unified development platform where developers can build, modify, and deploy applications residing in both Application Containers as well as Virtual Machines in a common, shared environment.
In addition we have seen an uptick of interest from infrastructure teams, which are aiming to modernise their existing infrastructure - usually a virtualization management stack - with a cloud-native alternative such as Kubernetes. These teams still have a hard requirement to continue running existing as well as new Virtual Machines, i.e. for hard multi-tenancy.

Benefits are broad and significant. Teams with a reliance on existing virtual machine-based workloads are empowered to rapidly containerize applications. With virtualized workloads placed directly in development workflows, teams can decompose them over time while still leveraging remaining virtualized components as is comfortably desired.

KubeVirt entered CNCF Sandbox September 9, 2019.

CNCF Mission & Principles

The KubeVirt project exists to aid users in moving from older frameworks to Cloud Native ones.

Project Overview Presentation

KubeVirt was presented to TAG Runtime:

- Presentation (link)
- Recording (link)

Incubation Criteria Summary

See the Incubation criteria for reference.

CNCF Code of Conduct

- Kubevirt CoC

Adhere to CNCF IP Policy
As an existing Sandbox project this is already in place.

Production usage
Yes.

Project end users are maintained in the Adopters files
Significant production users include:

- Red Hat
- NVIDIA
- SUSE
- Kubermatic
- H3C
- CoreWeave
- Civo
- CloudFlare
- Ateme
- Cloudbase Solutions

Healthy number of committers
Yes - see section on Community

Flow of contributions
While Red Hat remains a core contributor according to the numbers, several larger organisations have picked up KubeVirt, putting it in the best spot to see more external contributions.

Clear versioning scheme & release methodology
Some excerpts from our release procedure:

- Both a new release branch and initial release candidate are cut on the first business day of each month.
- If no blocker issues are discovered the release candidate is promoted to a full release after 5 business days.
- If blockers are detected, a new release candidate is generated and will be promoted after giving the impacted parties enough time to validate the blocker is addressed.
- Releases are cut from release branches must adhere to semantic versioning conventions.

The complete release process is described in detail in release.md and the backporting process in a separate file.

Clearly documented security processes
Yes - https://kubevirt.io/2020/KubeVirt-Security-Fundamentals.html and the KubeVirt Security Policy. The KubeVirt Security Policy will be enhanced with more steps and information.

Technical

Architectural Overview

In essence, in simple words, KubeVirt by itself is a cloud-native application for Kubernetes. It is leveraging the Operator pattern for application (KubeVirt) deployment. And CRs for specifying the user workloads - the VMs.
Controllers and node agents are used to create and manage pods for every VM that is getting created - according to the CRs. Ultimately having the VM running inside a pod leveraging the Kubernetes resource model (incl. CSI and CNI).

An in depth architecture discussion can be found in architecture.md.

Usability tradeoffs

While KubeVirt builds on Kubernetes primitives like pods, KubeVirt is introducing dedicated APIs for the virtualization entities, such as VMs. This was a deliberate decision, to have sane ways to express virtualization functionality like live migrations, and to avoid interference with container workflows and terms. The downside of this approach was that the pod api (and tools building on top of this), could not directly drive VMs.

Cloud Native Use Cases

- KubeVirt enables a unified development platform for modernization. Developers can build, modify, and deploy applications in both containers and VMs in a collaborative environment.
- Developers can use modern could-native tooling like Tekton, Istio, and ArgoCD, whether the functionality is in a VM or a container.
- Application modernization - Legacy 3-tier architecture apps, which consist of a presentation tier, application tier and data tier, can run directly on Kubernetes. Some of all of the application functionality can be refactored as cloud native as the pace of the business demands.
- For applications that will not be modernised, you can run legacy VMs (Linux and Microsoft Windows) on the same container platform, with the same operations team.
- KubeVirt enables infrastructure teams to modernise their stack using cloud-native technologies like Kubernetes - but still require VMs to meet existing and new (hard-multi tenancy for Kubernetes) use-cases.

Progress since CNCF Sandbox Status

There were almost 30 releases between entering the Sandbox and today. The list below is incomplete and is just mentioning (in a lossy way) highlights:

- API
    - Almost all APIs are stable (v1) by now
- Performance & Scale
    - Several performance improvements (pinning of emulator threads, iothreads, vNUMA, …) and tunables
    - Several scale improvements and tunables (controller threads, …)
    - Scale tested (by community members) up to 10k VMs
    - Creation of SIG Scale
- Storage
    - CSI based snapshot support
    - Disk hot-plug with “sidecar pods”
- Compute
    - ARM64 support
    - UEFI support
    - Support for PCI, GPU, and vGPU configuration and passthrough
    - Cgroups v2
    -
- Network
    - SR-IOV support
- Operations
    - SSH endpoint for easier SSH access to VMs
    - Guest Agent support for more guest informations

    - Requests, limits, and priority classes for control plane components for resource constrained environments
    - Libguetsfs integration for easy disk manipulation
- Observability
    - Standardisation of alert and metric names
    - Many more valuable alerts around control plane readiness
    - Runbooks (https://github.com/kubevirt/monitoring/tree/main/docs/runbooks)
    - Grafana Dashboards (https://github.com/kubevirt/monitoring/tree/main/dashboards)
- Security
    - Dropped requirement on privileged pods for VMs
    - Dropped requirement on multiple capabilities for VM pods
- CI/CD
    - Full move to tide and prow
    - Significantly lower number of flaky tests
    - Continuously increased number of functional tests
    - Significantly improved test runner infrastructure (kubevirtci)
    - Merge gating is active
- Tooling
    - Extraction of goclient
- Ecosystem
    - Fixes in Istio for enabling KubeVirt
    - Tekton Tasks
    - ArgoCD helpers
    - Konveyor for import from existing VM mgmt platforms to KubeVirt
    - Healthy relationship to the libvirt and qemu projects
- Vendors
    - Red Hat OpenShift includes KubeVirt
    - SUSE Rancher Harvester includes KubeVirt
    - Platform9 provides managed KubeVirt
    - Google Anthos includes KubeVirt
    - …
- Community
    - Adoption of conscious and inclusive language
    - Continuous stream of new contributors

Performance and Scalability

Performance Goals and Achievements

When speaking of performance, we are usually differentiating between data plane (VM workload performance - "How fast is the VM?") and control plane (VM creation latency - "How fast is the VM scheduled and entering the running state?").
Scheduling related performance is often related to scale, and therefore covered in the scale section below.
This section will cover workload performance.

KubeVirt's goal for workload performance is to be as fast as plain KVM. Thus a VM running on nothing but a clean host OS. This is usually equivalent to being as fast as existing VM management platforms like oVirt or OpenStack.

Considering that KubeVirt is running on top of Kubernetes, including cri-o and all it entails, it might be surprising that the actual stack of technologies between KubeVirt at runtime is very very similar to the stack when running VMs with libvirt on a clean OS.
This is due to the fact that on a plain OS, systemd and libvirtd are leveraging similar mechanisms (mainly namespaces and cgroups, but also SELinux on certain OSses) as the container stack is using.

Given the fact that VMs are essentially run as on a plain OS, KubeVirt is benefitting from usually all improvements done around KVM, qemu, and libvirt. And also avoiding adding any penalty due to additional layers - which KubeVirt does not introduce on the node level.

Compute performance

Significant progress has been made since joining the Sandbox in i.e. supporting CPU pinning, host topology representation in the guest (vNUMA), support for CPU manager, and dedicated IO and emulator threads. PCI and mdev passthrough is indirectly relevant, as it enables the use of accelerators such as (v)GPUs and FPGAs inside the VM.

Recent evaluations performed by SUSE have confirmed that good performance can be achieved.

Network performance

High performance networking is enabled by supporting SR-IOV via PCI passthrough.

Complexity/reliability tradeoffs

When at all possible, KubeVirt leverages existing Kubernetes constructs rather than inventing something net-new. This is why VMs are launched within Pods and why VM persistent state is represented as PVCs. This tradeoff limits KubeVirt to the feature set provided to Pods, which do not always align with how VMs are used in practice. One of the most glaring examples of this is that pods are immutable, while VMs traditionally are not. It's a common use case to add memory/cpu/disks to a running VM, but those concepts don't exist for pods. However, in return KubeVirt leverages the existing Kubernetes scheduler and existing CRI implementations to schedule/launch VM environments.

Integration/Performance tradeoffs

KubeVirt has always favoured a good integration over performance. Luckily this tradeoff had not to be made too often, but does shine through i.e. in the SLIRP network binding.

Scalability Goals and Achievements

KubeVirt is an add-on to Kubernetes, thus the general guideline has been to follow Kubernetes' when we speak about scale.
However, by now Kubernetes has continued to improve its scale, and can be scaled to very large footprints. KubeVirt will discuss if it can and/or should - or can - scale this well as well. There is a dedicated SIG (SIG Scale) part of the community to look at scale - looking at several scale related aspects.

Architecture wise Kubernetes patterns for controllers and node agents have been adopted in order to scale as well as Kubernetes does.

Recently the introduction of tunables around controller memory and CPU requests, as well as controller threads, rate limiting etc has helped to scale KubeVirt up to 10k (and beyond) VMs.

Performance/scalability tradeoffs

In order to simplify the architecture and ensure that VMI workloads remain independent from one another, the decision was made to have a 1 to 1 relationship between `libvirt` instances and VMI workloads, where the `libvirt` instance lives in the same pod as the VMI workloads. This tradeoff favoured reliability and security over performance and scalability. It requires a slightly (almost neglectable) larger memory and cpu footprint for each VMI workload to have a `libvirt` instance running per VMI, but it avoided the complication of attempting to have a node level centralised `libvirt` manage `qemu` processes with containers launched by the container run time.

High Availability

Control Plane

KubeVirt mirrors the Kubernetes architecture. It consist of three main components:

- **virt-api** - stateless and therefore scalable component which deals with API-level tasks
    - Webhooks
    - Proxying (vnc, port-forward, …)
    - The virt-api **Deployment** has a service as frontend which is registered in the apiserver which leads to traffic distribution and automatic failover
- **virt-controller** - contains all kubevirt related controllers. It is very similar to the kubernetes-controller-manager. It can not be scaled, but hot-standby replicas are running which take over if the current leading virt-controller can't refresh the leader lock. It uses a **Deployment**.
- **virt-handler** - Node daemon, comparable with the kubelet. It is distributed to the nodes via a **DaemonSet**. If virt-handler on a node goes down, it will be restarted.

Restarts have no impact on running VM workloads. If virt-handler can't be started again, the VM workloads are not endangered, but no new VM can be scheduled to the node and some high-level functions do not work anymore for these VMs.

KubeVirt has a zero-downtime update flow. The sole possible visible interruptions are:

- Proxy connections through a virt-api instance may be terminated, but can be re-established immediately
- On nodes, during the update of virt-handler, some high-level functions may not work for a few seconds. Only a limited set of nodes is affected at the same time.

Workloads

Workload high availability i.e. restarting VMs in the case of a node failure is indirectly provided.
KubeVirt aligns to Kubernetes behaviours when it comes to node unavailability: A VM will stay in an unknown state unless the node becomes ready again, or the node is getting deleted (incl. All of it's workloads). Thus in effect the remediation of an unhealthy node is required in order to provide workload high availability. In the past such a mechanism was called fencing.
The Kubernetes ecosystem has a couple of tools to do automatic node remediation. One of them - tested with KubeVirt - is medik8s.
It was clear that KubeVirt would need to find a solution for HA, but at the same time it was clear that this was not a KubeVirt only problem. Therefore the focus was on adopting ecosystem solutions for node remediation.

Coding Standards and Quality

Code Quality

A list of architectural decisions and coding standards which have to be considered and are not fully enforced through automation is available:
https://github.com/kubevirt/kubevirt/blob/main/docs/reviewer-guide.md

For defining the lowest acceptable standards the project relies on automation. People have to pass the automated check and they have to add unit tests and end-to-end tests for their features and fixes. All tests are run and required to pass on each PR. Maintainers are allowed to take in code with varying quality for as long as the project's maintainability is not at stake and all required criteria are met (especially the testing and architectural criteria) to be open and inclusive.

The lowest bar for acceptable **coding styles** is enforced via automation:

- goimports to enforce a common coding style for go code
- shfmt to enforce a common coding style for bash scripts

The lowest bar for acceptable golang **coding standards** (anti-patterns, coding errors, …) is enforce via automation:

- nogo from bazel is used and applies a huge set of code analyzers when one builds kubevirt. If a check fails the build fails.
- In addition to standard nogo analyzers the project added ineffassign to its nogo execution.

Even good reviews and automation can't catch all. When common mistakes are identified, actions are considered and precautions get implemented to avoid them in the future. For example: The project repeatedly saw regressions where some inefficient REST calls got in where a watcher should have been used. To identify and fix these issues, efforts are going on in KubeVirt's SIG scaling:

- PRs fixing issues identified by sig scaling: #6226, #6168
- Automation worked on to identify such issues on the PR level already: #38, #1211

Dependencies

General Overview:

- The cluster-level control plane consists of pure go code. All dependencies are vendored into the codebase.
- The node-level infrastructure has additional dependencies (RPM based):
  - System tools (nftables, iptables, …)
  - The main virtualization components (libvirt, qemu)
- To control the dependency chain of the node-level infrastructure the project builds "distroless" containers based on Fedora RPMs (soon Centos8 stream). This allows to minimise the image size and reduces the risk of being exposed to CVEs. More information is available in the dependency-update guide.
- The project protects itself from losing external dependencies (like RPMs, cross-compilers, ...) by mirroring external dependencies to a GCS bucket.
- All dependencies are defined in the codebase and contain a checksum and can be exchanged by any developer within a PR to make feature development as seamless, self directed and easy as possible.
  In order to stay safe, automation is in place which verifies that new dependencies are trustworthy (for instance checking that a RPM signature is from a trusted source and checking shasums). Further the project does not make use of Dockerfiles for reproducibility and safety reasons: The containers are built with bazel without a base image.

Locations where the dependencies are defined:

- Go dependencies: go.mod and the corresponding vendor folder
- RPM dependencies: WORKSPACE contains all referenced RPMs and rpm/BUILD.bazel shows which containers will contain which rpms. RPM resolution can be done in hack/rpm-deps.sh.

The integration with Kubernetes depends on CRDs, API Server Aggregation, Device Plugins, and to a certain extent on CRI and CSI

CI/CD Status

KubeVirt uses Prow as its CI/CD system, both for running tests on PRs and for automating all sorts of tasks. Deck UI is available here https://prow.ci.kubevirt.io/ and there are links on the PR statuses to the results of each of the test jobs. We also have public grafana dashboards that show several aspects of our CI infrastructure, like metrics about the merge queue https://grafana.ci.kubevirt.io/d/WZU1-LPGz/merge-queue?orgId=1, the failure rates of different lanes e2e presubmits or the infrastructure usage https://grafana.ci.kubevirt.io/d/qFDyvVinx/ci-infrastructure-utilization?orgId=1

We have explicit coverage metrics sent to coveralls in a Prow job, they are available here https://coveralls.io/github/kubevirt/kubevirt FOSSA checks, covering compliance, security and quality are integrated too in our CI pipeline https://app.fossa.io/projects/git%2Bgithub.com%2Fkubevirt%2Fkubevirt/refs/branch/master/e740040fd9b26b8372e486b251a3fbe6f5217e82/browse/dependencies

There are different levels of tests in our suite:

- unit tests (some of them can be considered as integration tests given that involve several components), and
- full end to end involving test cluster creation and execution of test cases covering specific product features

We execute the e2e tests for the last 3 supported kubernetes versions and the cases are split by SIG (virtualization, network, storage and operator). We also have specific lanes that cover features like non-root execution, vGPU support and execution on different architectures, like ARM.

Security

KubeVirt is actively following a security in depth approach. It happens that this is also naturally evolving due to the nature of its architecture.

At the heart of KubeVirt's security model is the desire - and future fact - that - in the end - basic VMs will run as regular pods, with no elevated capabilities or privileges.
This is not the case today, but while KubeVirt already eliminated privileged mode and almost all capabilities from the virt launcher pod (the pod holding the VM), this work continues at full steam, an important and the imminent next step is to run pods as non-root.

The following article is giving an in depth overview over KubeVirt's security model: https://kubevirt.io/2020/KubeVirt-Security-Fundamentals.html

Security tradeoffs

Initially, in order to make progress in KubeVirt, the VMI workload pods depended on running as root. This allowed certain feature sets to be developed quickly with the tradeoff that the environment launching the `qemu` process (not the `qemu` process itself) runs as root. Over

time the root dependencies have been reduced to the point where we've created a path that completely removes the dependency on root. This non-root VMI pod feature is still being stabilised, but it will become the de facto standard for all KubeVirt deployments in a future release.

Supply chain security

The following aspects of supply chain security are addressed:

- Tracking dependencies
    - Golang: go.mod/go.sum files which is the base for our vendor folder. CI ensures on every PR that the vendor folder is in sync with go.mod, to avoid that stuff gets sneaked into the vendor folder.
    - RPM: Dependencies for all architectures (arm64 and x86_64) are tracked in rpm/BUILD.bazel. We track the origin as well as the checksum. On every PR all referenced RPMs are validated. RPM dependencies are bumped periodically and on-demand. This includes checksum verification as well as verifying GPG signatures.
    - KubeVirt artifacts are built from golang and RPM dependencies tracked like described above. In addition a small set of extra dependencies are part of the build, which are tracked with origin and checksum in our WORKSPACE file. Containers (except for some testing containers) are completely built from scratch where RPMs need to be installed, or based on googles distroless containers.
    - For building KubeVirt, bazel is used. There we almost exclusively rely on libraries written and maintained by google. A small set of additional extensions is used. These extensions are tracked with checksum and location and do not auto-update.
    - TODO: registries like quay.io can not scan our containers for all included dependencies which may delay detecting CVEs. An attempt from Roman Mohr is going on to improve that: https://github.com/quay/claircore/issues/488
- Ensuring availability of dependencies
    - KubeVirt builds are reproducible. One of the advantages of reproducible builds is full dependency tracking.
    - Golang dependencies are vendored from our go.mod/go.sum files and checked in in our VCS.
    - If dependencies are modified, pass our CI checks (see above) and the modifications get merged, a periodic CI job goes through all our tracked dependencies (except golang where we directly vendor), uploads it to the projects GCS bucket and creates a PR where the GCS bucket is added as a mirror location for all new dependencies.
- Developer Security
    - KubeVirt only accepts members which have 2FA on GitHub enabled. This also includes our bot accounts, where the CI team has shared credentials, but each member has an individual device for the second authentication step.

- Releases tags are signed (https://github.com/kubevirt/kubevirt/releases/tag/v0.49.0)
- Git and Github are used for code tracking
- Our CI system ensures that main and release branches do not allow direct push. Only PRs are allowed.
- TODO: Tracking golang CVEs more directly visible for the broad community could be improved. A lot of this happens right now inside Red Hat and bubbles up to the community. CVEs on used libraries are addressed mostly this way.

Roadmap

- **Can be accomplished now:**
  - Stable API - v1 released
  - Accelerate compute intensive workloads (GPU access to single VM)
  - Non disruptive updates of the KubeVirt operator (zero downtime, live)
  - CPU pinning support and NUMA Topology passthrough
  - More robust Live-Migration support
  - Data protection with Offline and online disk snapshots
  - SR-IOV support for high performance networking
  - Improved operations with runbooks and enhanced observability
  - Multus support for multiple network interfaces attached to Virtual Machines
  - Declarative host network configuration

- **Can be accomplished with reasonable additional effort (and are ideally already on the project roadmap):**
  - Backup / Recovery & DR with Velero
  - Additional support for very large VMs - e.g. 3TB+ memory
  - Further scale out testing of KubeVirt API
  - More advanced virtualization configurations - e.g. CPU NUMA topology
  - Advanced security enhancements - e.g. increased workload isolation with non-root VMI Pods
  - VM import/export file format and API

- **Are in-scope but beyond the current roadmap for the next six months:**
  - Optimising cost efficiency through GPU sharing via vGPU
  - Improved ArgoCD templating for VMs in a Tekton Pipeline

- **Areas out of scope:**
  - Enabling any other workload than VMs
  - VDI Streaming solution for seamless desktop access

Project

- **Do we believe this is a growing, thriving project with committed contributors?**
  Yes. While Red Hat remains a core contributor according to the numbers, several larger organisations have picked up KubeVirt, putting it in the best spot to see more external contributions.

- **Is it aligned with CNCF's values and mission?**
  Absolutely. Following cloud native principles and aiming for the *right* integration into the ecosystem are key pillars of the project.

- **Do we believe it could eventually meet the graduation criteria?**
  Yes

- **Should it start at the sandbox level or incubation level?**
  KubeVirt is at the Sandbox level, and is looking to move to Incubation.

- **Does the project have a sound, documented process for source control, issue tracking, release management etc.**
  KubeVirt has adopted the “GitHub Flow” for source control

- **Does it have a documented process for adding committers?**
  Yes, it’s part of the membership policy.

- **What is the general quality of informal communication around the project (slack, github issues, PR reviews, technical blog posts, etc)?**
  The project tries to find the balance between to many channels of communication - to avoid communication fragmentation - but at the same time offer enough different channels to reach different audiences.
  Existing communication channels
    - Forum / Mailinglist - https://groups.google.com/g/kubevirt-dev
    - Slack channel for developers - #kubevirt-dev
    - Slack channel for users - #virtualization

Governance

Project is self-governing:

- Governance
- Membership Policy/Contributor Ladder

As a small project with tightly coupled components, Kubevirt has adopted a simple "maintainer council" style governance system, based on the templates supplied by TAG-Contributor-Strategy.

The current maintainers are:

Name	Employer
David Vossel	Red Hat
Vladik Romanovsky	Red Hat
Roman Mohr	Red Hat
Fabian Deutsch	Red Hat
Stu Gott	Red Hat
Vasiliy Ulyanov	SUSE
Chris Calligari	NASA
Ryan Hallisey	Nvidia
Federico Gimenez	Red Hat
Zvi Cahana	IBM

We are in the process of also building a robust set of working groups with their own leads, both as an engine of growth and as a way of recognizing and promoting additional leadership in the community. To date, three of the working groups have leadership independent of the maintainer council, and we plan to eventually have five to six.

Contributions

- **How much time does the core team commit to the project?**
  The core team (maintainers, but also other regular contributors) are usually working full time on KubeVirt.

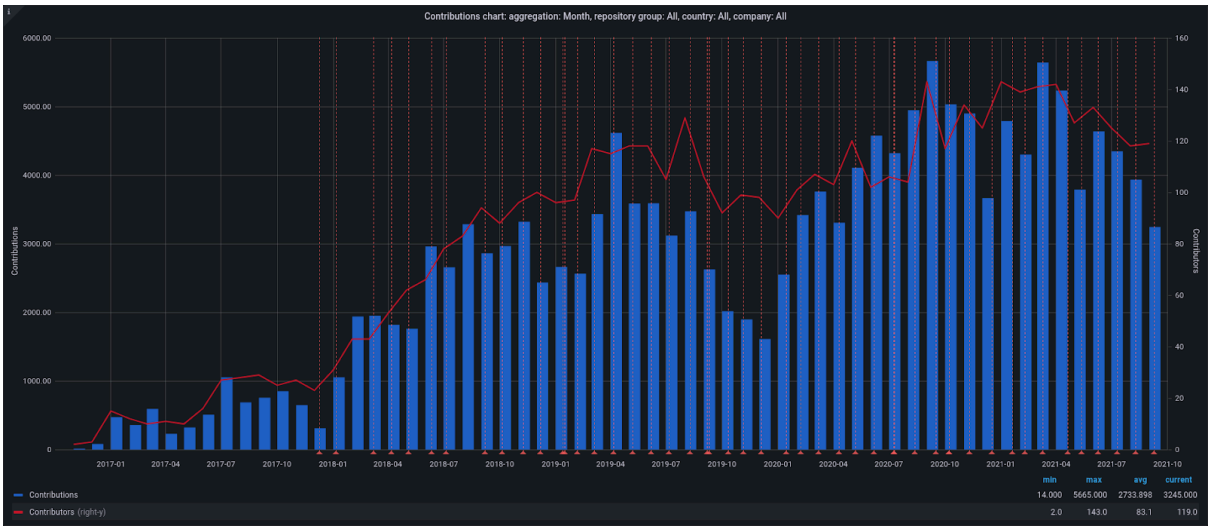
- **How big is the team? Who funds them? Why? How much? For how long?**
  There are approximately 70 contributors to the KubeVirt project with over 100 commits over the course of the last year. About 54 (according to the CNCF devstats) of the 100 contributors of last year are Red Hat employees and are funded by Red Hat. Red Hat, like other vendors contributing to KubeVirt, has an interest in maintaining funding of engineering contributions to KubeVirt to enhance its productized version of the Open Source project.

  There are also regular contributions from other well known adopters - including end users and vendors - like SuSE, Apple, and Google. This has been increasing on its own over the last year as more organisations adopt the technology. Additionally, we

are building working groups to allow more contributors to show leadership in the project as well as increasing their contributions. Our maintainers have also been mentoring some non-Red Hat contributors one-on-one with a target of raising them to core contributor status.

- **Who are the clear leaders? Are there any areas lacking clear leadership? Testing? Release? Documentation? These roles sometimes go unfilled.**
    - Code: The Approvers
        - And sub-project owners - like i.e. CDI, knmstate
    - Testing: Daniel Hiller, Red Hat, Roman Mohr, Red Hat
    - Release: David Vossel, Red Hat
    - Community: Fabian Deutsch , Red Hat and David Vossel, Red Hat
    - Scale: Ryan Hallisey, NVIDIA

- **What is the rate of ongoing contributions to the project (typically in the form of merged commits).**
  There is a positive trend of contributions:

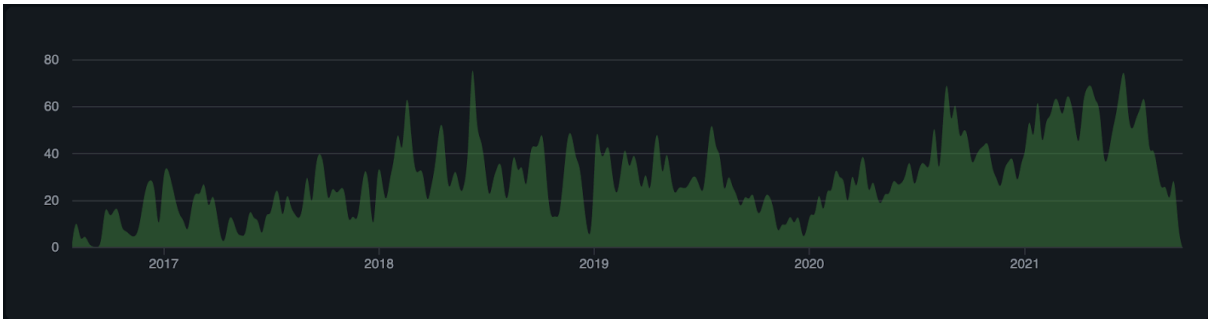


Have a healthy number of committers. A committer is defined as someone with the commit bit; i.e., someone who can accept contributions to some or all of the project.

https://kubevirt.devstats.cncf.io/d/56/company-commits-table?orgId=1&from=now-2y&to=now

In addition to committers from vendor organisations such Red Hat, IBM, SUSE, Giant Swarm - there have been numerous contributions from End User Committers from Amadeus, Apple, Uber, Erickson, Fujitsu, NEC, NVIDIA, and others.

https://github.com/kubevirt/kubevirt/graphs/contributors



Demonstrate a substantial ongoing flow of commits and merged contributions.

For the last year, KubeVirt has averaged more than 350 merged PRs per month from more than 60 contributors. Of those, 10-20 per month come from new contributors, showing ongoing interest in joining the project.

The KubeVirt project has core maintainers from SUSE, NVIDIA and Red Hat. The KubeVirt community would like to focus on increasing the maintainer diversity during incubation to increase the project health and build an even stronger community. Even though the maintainers are represented by 3 companies, the project community itself has had individual contributions from over 68 companies over the last 4 years. Source: https://kubevirt.devstats.cncf.io/d/56/company-commits-table?orgId=1

Community

- **Besides the core team, how active is the surrounding community? Bug reports? Assistance to newcomers? Blog posts etc.**
    - **Newcomer onboarding:** We give new people the opportunity to introduce themselves and talk about what brought them to us. Community team has a well written README that explains how to contribute new material to the website and user-guide. Once a week, maintainers spend 30-60 mins with someone helping them to get started with non-code base contributions.
    - **Blogs:** The community receives about one blog entry per month.
    - **Twitter:** https://twitter.com/kubevirt with 1,965 followers.
    - **Events:** We have a weekly community contribution meeting whose attendance varies. We also have an annual Kubevirt Summit, currently online, which draws around 200 attendees, most of whom are contributors. The project is also starting to participate as a group in third-party events and conferences, such as Kubecon, All Things Open, and various meetups.
    - **GitHub Issues and Pulls:**
        - Metrics are detailed in devstats
        - For the last year, KubeVirt has averaged more than 350 merged PRs per month from more than 60 contributors.
        - Of those, 10-20 per month come from new contributors, showing ongoing interest in joining the project. Kube

        - KubeVirt average Issue age has shown a drastic decrease over the last year
- **Are there any especially difficult personalities to deal with? How is this done? Is it a problem?**

There are seasoned members in the group which are usually trying to moderate these conflicts outside of the primary communication channels. We have not had any serious issues with project conflicts to date.

Adoption

- **Who uses the project? Get a few in-depth references from 2-4 of them who actually know and understand it.**
    - SUSE - SUSE believes KubeVirt is the best open source way to handle Virtual Machines on Kubernetes today. We offer this additional possibility to our customers by leveraging KubeVirt in our products.
    - KUBERMATIC - As a distributor KUBERMATIC runs KubeVirt to enable VM workload on Kubermatic Virtualization.
    - H3C - H3C distributes KubeVirt as part of CloudOS to enable VM workloads on Kubernetes at customer sites.
    - NVIDIA - NVIDIA's latest computing platform is built on open-source projects like Kubernetes and KubeVirt to power products like GeForce NOW with more to come.
    - CoreWeave - A Kubernetes native cloud provider with focus on GPUs at scale. KubeVirt allows us to co-locate non-containerizable workloads such as Virtual Desktops next to compute intensive containers executing on bare metal. All orchestrated via the Kubernetes API leveraging the same network policies and persistent volumes for both VM and containerized workloads.
    - CIVO - CIVO uses KubeVirt as part of their stack to enable tenant cluster provisioning within Civo cloud.

- **What do real users consider to be its strengths and weaknesses? Any concrete examples of these?**

    - A very concrete weakness is our **kubevirt/client-go** repository. We do not yet offer the api definition alone which one can use with **client-gen**. People are repeatedly struggling with updating the dependencies in their projects where they make use of **kubevirt/client-go**. Update Feb. 1 2022: This is mostly resolved with the introduction of **kubevirt/api** which can be used with k8s code generators.
      -Roman Mohr/ Red Hat
    - Strengths - I think the folks that participate in the community are high quality - it's a well of technical expertise, creativity, and openness. People are always interested to discuss new use cases, while also staying well grounded on the core principles that make up KubeVirt and Kubernetes.

Weaknesses - Something I find KubeVirt struggles with, which isn't very different from many growing open source projects, is more contributions from users. As a user myself, I want to hear how other people are solving their problems. Right now, I hear about how users are using KubeVirt but don't always see them working on it.
-Ryan Hallisey / NVIDIA

- From our perspective KubeVirt provides good flexibility to allow downstream customization. It seamlessly integrates into k8s ecosystem and supports many virtualization features out of the box. It is well maintained by the upstream community promoting collaboration and contributions. [re cons] Cannot really think of concrete weakness examples now. Usually we try to workaround or to upstream fixes whenever we encounter issues. Though nothing really serious so far.
  - Vasiliy Ulyanov / Software Engineer, SUSE Labs Core

- **Perception vs Reality: Is there lots of buzz, but the software is flaky/untested/unused? Does it have a bad reputation for some flaw that has already been addressed?**

  KubeVirt has been adopted by end-users and vendors in production environments, The rate at and the scale of this adoption is not reflected in their presence in the project itself. Thus adopters tend to be silent about their adoption, yet file bugs and request features.
  By looking at the numbers it is clear that Red Hat is the top contributor to the project, but this number is not reflecting the adoption of the project by other vendors. Our conclusion is that KubeVirt is stable and feature rich enough to meet most adopters' demands, not requiring them to step up to close many gaps.
  There are examples of large adopters stepping up to close smaller or larger gaps as well as proposing large architectural changes (ARM presenting KubeVirt with Xen hypervisor).

Context

- **What is the origin and history of the project?**

Kubernetes 1.0 got released. “Wow, it’s a scheduler for compute workloads. As VMs are also compute workloads, can’t we leverage it for VMs as well? Oh and hey - why would you want to have two schedulers in the end if one can rule them all?”

- **Where does it fit in the market and technical ecosystem?**

Just like the CNCF, the KubeVirt project assumes that containers and cloud native principles are here to stay and to solve many new and old problems. However, there will be workloads which can not be moved to containers for technical or non-technical reasons. These workloads will remain important, but require operators to maintain a separate stack for operating these workloads.

KubeVirt is solving this problem, by enabling Kubernetes - the state of the are cloud native platform - to run these legacy workloads as well, reducing the large burden from operators to maintain two stacks (one each for contains and VMs).

- **Is it growing or shrinking in that space? Is that space growing or shrinking?**

Growing, both as a project and as a need. More people adopt containers, and more organisations are faced with the question what to do with their crucial legacy workloads. KubeVirt has also become the default solution for running general VMs on Kubernetes. Two years ago, it was one of several competing solutions.

- **How necessary is it? What do people who don't use this project do? Why exactly is that not adequate, and in what situations?**

Organisations can stick to existing VM platforms for running VMs, this has the main drawbacks that two platforms have to be operated. At the same time organisations get under pressure, because vendors are moving to new container systems, and sometimes discontinue VM management platforms, leaving the

- **Clearly compare and contrast with peers in this space. A summary matrix often helps. Beware of comparisons that are too superficial to be useful, or might have been manipulated so as to favour some projects over others. Most balanced comparisons will include both strengths and weaknesses, require significant detailed research, and usually there is no hands-down winner. Be suspicious if there appears to be one.**

Project	Workload	Focus	Status	Contrast
KubeVirt	VM	VMs in a cloud native world	- Dedicated API, many Virt features, without pod API clashes - K8s workloads controllers can not be natively reused i.e. deployments or DS	
virtlet	VM	Running VMs with a POD API i.e. for stateless apps using CRI	- Almost abandoned - Clashes between VM and pod functionality (i.e. live migration)	Pod API was preferred over a dedicated virtualization API. This prevents having full control over the virtualization stack. Benefit is that VMs can be controlled and defined like VMs.
Rancher VM	VM	Thin wrapper around qemu	Renamed to Harvester and now using KubeVirt	
Kata	Pod	Pod isolation for container	- Can use KVM	No competition, as

		reasons	- Active community	the focus is to increase pod security, not to manage VMs
gVisor	Container	Container isolation for security reasons	- Can use KVM - Active community	
Firecracker	Container	Container isolation for security reasons	- Uses KVM	