KubeVirt

Streamlining VM creation within KubeVirt

Where are we now?

Lee Yarwood, Software Engineer @ Red Hat

lyarwood@redhat.com
https://github.com/lyarwood
https://blog.yarwood.me.uk/

Agenda

- KubeVirt
- Why?
- Goals
- How?
- What's New?
- What's Next!
- Q&A

KubeVirt @ devconf.cz 2024

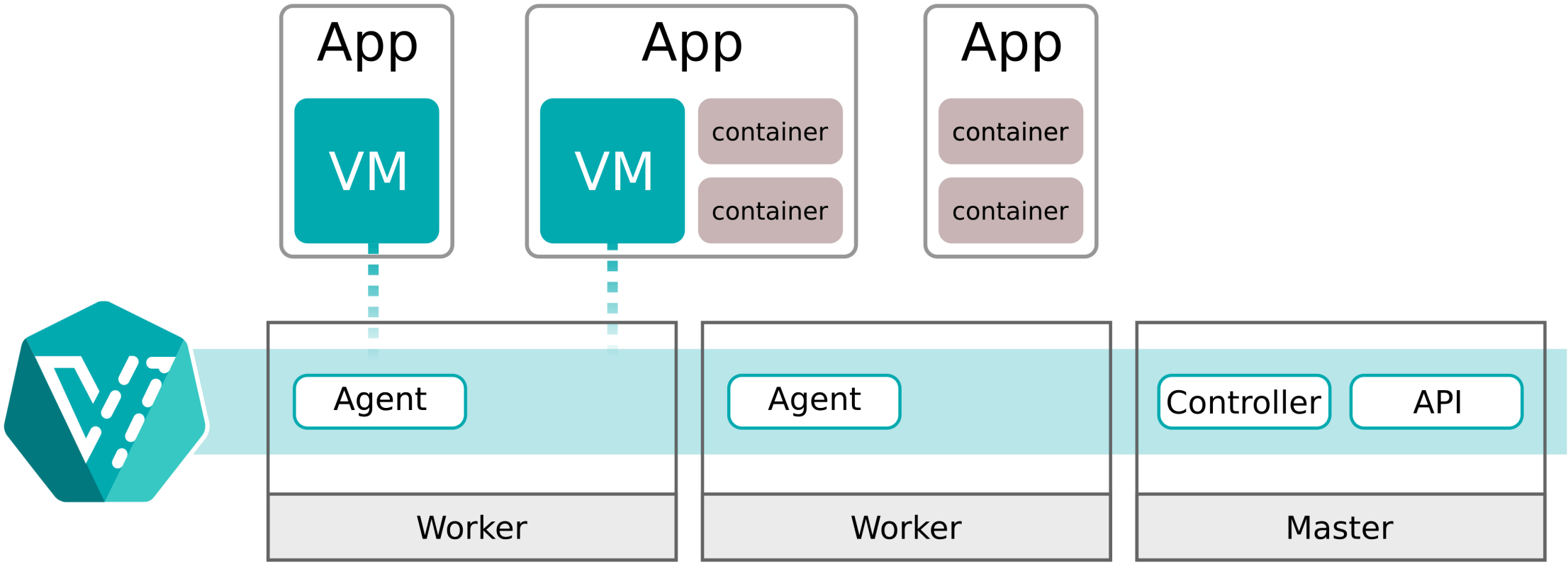
- Come and visit our Booth!
- Device passthrough in KubeVirt - Victor Toso @ 14/06 14:00 D105
  - https://pretalx.com/devconf-cz-2024/talk/JWGCKX/
- What the swap?! Swap on k8s - Itamar Holder @ 14/06 16:00 D105
  - https://pretalx.com/devconf-cz-2024/talk/KUD3WB/

KubeVirt?

A virtualization API for Kubernetes

Built on the KVM, QEMU and libvirt virtualization stack

Containerized and virtualized workloads in the same cluster



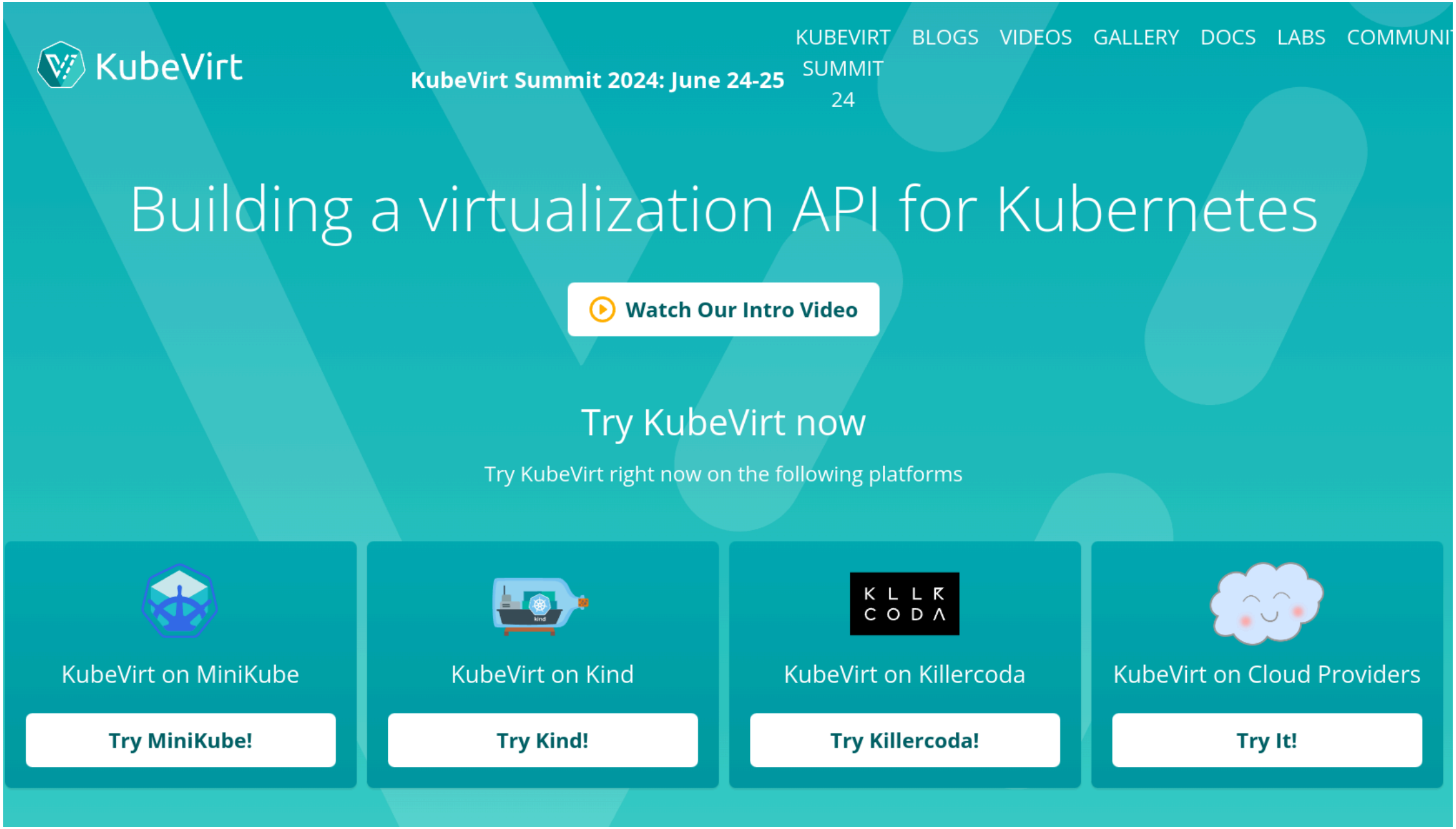
`CustomResourceDefinition` (CRD)

Provides a user facing `VirtualMachine` CRD

Provides a runtime `VirtualMachineInstance` CRD

`$USER` -> `VirtualMachine` -> `VirtualMachineInstance` -> `Guest`

kubevirt.io


KubeVirt
KubeVirt Summit 2024: June 24-25
KUBEVIRT SUMMIT 24
BLOGS
VIDEOS
GALLERY
DOCS
LABS
COMMUNI
Building a virtualization API for Kubernetes
Watch Our Intro Video
Try KubeVirt now
Try KubeVirt right now on the following platforms
KubeVirt on MiniKube
Try MiniKube!
KubeVirt on Kind
Try Kind!
KLLR CODA
KubeVirt on Killercoda
Try Killercoda!
KubeVirt on Cloud Providers
Try It!

Why?

Meet casual KubeVirt user `$USER`

```
$
```

`$USER` has a legacy workload they want to deploy within a **Fedora** `VirtualMachine`

```
$
```

`$USER` opens up `$EDITOR` to write their first `VirtualMachine` definition...

```
$ ${EDITOR} vm.yaml
```

`$USER` starts with the basics

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: teleportation-test-vm
spec:
```

`$USER` adds their bootable `fedora` PVC

```yaml
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: teleportation-test-vm
spec:
  template:
    spec:
      volumes:
      - name: fedora
        persistentVolumeClaim:
          claimName: fedora
```

`$USER` then assigns some resources based on the legacy workload they want to run... /s

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: teleportation-test-vm
spec:
  template:
    spec:
      domain:
        cpu:
          sockets: 1
        memory:
          guest: 4Mi
      volumes:
      - name: fedora
        persistentVolumeClaim:
          claimName: fedora
```

`$USER` then discovers `spec.template.spec.domain.devices` is required, even when empty

```
The request is invalid:
spec.template.spec.domain.devices in body is required
```

`$USER` then discovers `spec.template.spec.domain.devices` is required, even when empty

```yaml
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: teleportation-test-vm
spec:
  template:
    spec:
      domain:
        devices: {}
        cpu:
          sockets: 1
        memory:
          guest: 4Mi
      volumes:
      - name: fedora
        persistentVolumeClaim:
          claimName: fedora
```

`$USER` then discovers

`spec.runStrategy` is also required

```
The request is invalid:
spec.running: One of Running or RunStrategy must be specified
```

`$USER` then discovers

`spec.runStrategy` is also required

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: teleportation-test-vm
spec:
  runStrategy: Always
  template:
    spec:
      domain:
        devices: {}
        cpu:
          sockets: 1
        memory:
          guest: 4Mi
      volumes:
      - name: fedora
        persistentVolumeClaim:
          claimName: fedora
```

The `VirtualMachine` is finally created!

```
virtualmachine.kubevirt.io/teleportation-test-vm created
```

However it never starts...

```
$ kubectl get vms
NAME                     AGE     STATUS     READY
teleportation-test-vm    3m28s   Starting   False
```

$USER's colleague attempts to help by providing an example `Windows VirtualMachine`....

```yaml
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: windows-ql8vf1
spec:
  dataVolumeTemplates:
    - apiVersion: cdi.kubevirt.io/v1beta1
      kind: DataVolume
      metadata:
        annotations:
          cdi.kubevirt.io/storage.bind.immediate.requested: 'true'
        creationTimestamp: null
        name: windows-ql8vf1
      spec:
        source:
          blank: {}
        storage:
          resources:
            requests:
              storage: 60Gi
  running: false
  template:
    spec:
      domain:
        clock:
          timer:
            hpet:
              present: false
            hyperv: {}
            pit:
              tickPolicy: delay
            rtc:
              tickPolicy: catchup
          utc: {}
        cpu:
          cores: 1
          sockets: 1
          threads: 1
        devices:
          disks:
            - disk:
                bus: sata
              name: rootdisk
          inputs:
            - bus: usb
              name: tablet
              type: tablet
          interfaces:
            - macAddress: '02:3e:ee:00:00:00'
              masquerade: {}
              model: e1000e
              name: default
        features:
          acpi: {}
          apic: {}
          hyperv:
            frequencies: {}
            ipi: {}
            reenlightenment: {}
            relaxed: {}
            reset: {}
            runtime: {}
            spinlocks:
              spinlocks: 8191
            synic: {}
            synictimer:
              direct: {}
            tlbflush: {}
            vapic: {}
            vpindex: {}
        machine:
          type: pc-q35-rhel9.0.0
        resources:
          requests:
            memory: 4Gi
      networks:
        - name: default
          pod: {}
      volumes:
        - dataVolume:
            name: windows-ql8vf1
          name: rootdisk
```

Why?

- The `VirtualMachine` CRD is rich but overwhelming
- Users shouldn't need to hand craft simple definitions
- Users shouldn't need to know all of the best practices for a given workload

Goals

- Reduce the VM creation decision matrix
- Ideally down to a single choice of workload
- Focus on always providing a valid runnable VM

How?

- Provide `CRDs` to encapsulate resource sizing and workload preferences
- Provide consistent examples of these across KubeVirt deployments
- Wrap manifest creation best practices in an easy to use CLI
- Allow workload (image) owners to set requirements and sane defaults

User Workloads
Lifecycle
Basic use
Creating VirtualMachines
Download and Install the virtctl Command Line Interface
Accessing Virtual Machines
Booting From External Source
Startup Scripts
Windows virtio drivers

Monitoring
Component monitoring
Guest Agent information
Guest Operating System Information
Liveness and Readiness Probes

Workloads
Instance types and preferences
Deploy common-instancetypes
Hook Sidecar Container
Presets
Templates
VirtualMachinePool
VirtualMachineInstanceReplica...
Virtual Machines Instances
VM Rollout Strategies

Instance types and preferences

FEATURE STATE:

- `instancetype.kubevirt.io/v1alpha1` (Experimental) as of the `v0.56.0` KubeVirt release
- `instancetype.kubevirt.io/v1alpha2` (Experimental) as of the `v0.58.0` KubeVirt release
- `instancetype.kubevirt.io/v1beta1` as of the `v1.0.0` KubeVirt release

See the Version History section for more details.

Introduction

KubeVirt's `VirtualMachine` API contains many advanced options for tuning the performance of a VM that goes beyond what typical users need to be aware of. Users have previously been unable to simply define the storage/network they want assigned to their VM and then declare in broad terms what quality of resources and kind of performance characteristics they need for their VM.

Instance types and preferences provide a way to define a set of resource, performance and other runtime characteristics, allowing users to reuse these definitions across multiple `VirtualMachines`.

VirtualMachineInstancetype

```
---
```

Table of contents
Introduction
VirtualMachineInstancetype
VirtualMachinePreference
VirtualMachine
Creating InstanceTypes, Preferences and VirtualMachines
Versioning
inferFromVolume
common-instancetypes
Examples
Version History
instancetype.kubevirt.io/v1alp... (Experimental)
instancetype.kubevirt.io/v1alp... (Experimental)
instancetype.kubevirt.io/v1bet...

`VirtualMachineInstancetype` & `VirtualMachineClusterInstancetype`

- Resource related attributes of `VirtualMachineInstanceSpec`
- `CPU` and `Memory` required
- Values will conflict with user choices in the `VirtualMachine`
- `VirtualMachine` can reference only one

```yaml
---
apiVersion: instancetype.kubevirt.io/v1beta1
kind: VirtualMachineClusterInstancetype
metadata:
  annotations:
    instancetype.kubevirt.io/description: |-
      The U Series is quite neutral and provides resources for
      general purpose applications.

      *U* is the abbreviation for "Universal", hinting at the universal
      attitude towards workloads.

      VMs of instance types will share physical CPU cores on a
      time-slice basis with other VMs.
    instancetype.kubevirt.io/displayName: General Purpose
  labels:
    instancetype.kubevirt.io/class: general.purpose
    instancetype.kubevirt.io/cpu: "1"
    instancetype.kubevirt.io/icon-pf: pficon-server-group
    instancetype.kubevirt.io/memory: 4Gi
    instancetype.kubevirt.io/vendor: kubevirt.io
    instancetype.kubevirt.io/version: "1"
    instancetype.kubevirt.io/common-instancetypes-version: v1.0.0
  name: u1.medium
spec:
  cpu:
    guest: 1
  memory:
    guest: 4Gi
```

`VirtualMachinePreference` & `VirtualMachineClusterPreference`

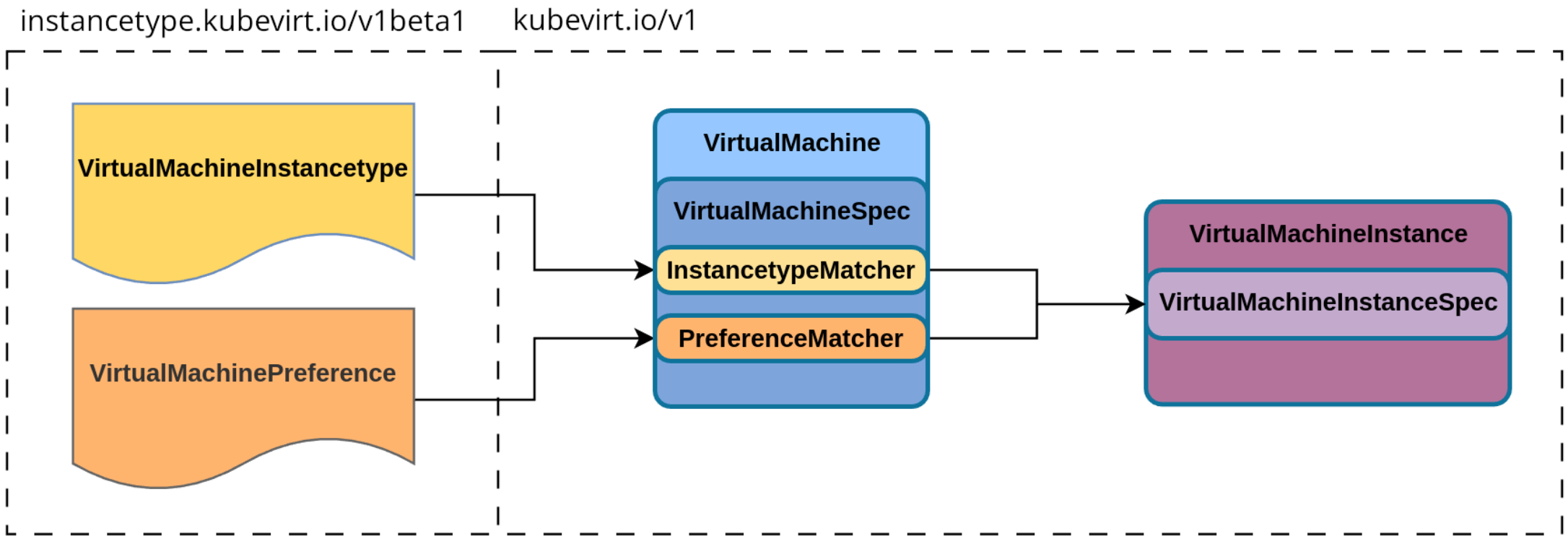
- All remaining attributes of `VirtualMachineInstanceSpec`
- Preferred values, do not overwrite or conflict with user choices in `VirtualMachine`
- Can also provide resource requirements for a given workload
- `VirtualMachine` can reference only one

```yaml
---
apiVersion: instancetype.kubevirt.io/v1beta1
kind: VirtualMachineClusterPreference
metadata:
  annotations:
    iconClass: icon-fedora
    openshift.io/display-name: Fedora
    openshift.io/documentation-url: https://github.com/kubevirt/common-instancetypes
    openshift.io/provider-display-name: KubeVirt
    openshift.io/support-url: https://github.com/kubevirt/common-instancetypes/issues
    tags: hidden,kubevirt,fedora
  labels:
    instancetype.kubevirt.io/os-type: linux
    instancetype.kubevirt.io/vendor: kubevirt.io
    instancetype.kubevirt.io/common-instancetypes-version: v1.0.0
  name: fedora
spec:
  devices:
    preferredDiskBus: virtio
    preferredInterfaceModel: virtio
    preferredNetworkInterfaceMultiQueue: true
    preferredRng: {}
  features:
    preferredSmm: {}
  firmware:
    preferredUseEfi: true
    preferredUseSecureBoot: true
  requirements:
    cpu:
      guest: 1
    memory:
      guest: 2Gi
```

`InstancetypeMatcher` & `PreferenceMatcher`

- `Name`
- `Kind` (defaults to cluster-wide)
- `RevisionName`
- `InferFromVolume`

```yaml
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: fedora
spec:
  instancetype:
    name: n1.medium
  preference:
    name: fedora
```


instancetype.kubevirt.io/v1beta1
kubevirt.io/v1
VirtualMachineInstancetype
VirtualMachinePreference
VirtualMachine
VirtualMachineSpec
InstancetypeMatcher
PreferenceMatcher
VirtualMachineInstance
VirtualMachineInstanceSpec

`InferFromVolume`

- Looks for labels on the underlying volume `PVC`, `DataSource` or `DataVolume` to determine defaults
- `instancetype.kubevirt.io/default-instancetype`
- `instancetype.kubevirt.io/default-preference`

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: fedora
spec:
  instancetype:
    inferFromVolume: fedora
  preference:
    inferFromVolume: fedora
  template:
    spec:
    [...]
      volumes:
      - name: fedora
        persistentVolumeClaim:
          claimName: fedora
```

`common-instancetypes` `v1.0.0`

- Set of `Kustomize` based instance type and preference resources for KubeVirt
- Provides 39 instance types & 32 preferences
- Both namespaced and cluster-wide resources provided

```
$ kubectl apply -k https://github.com/kubevirt/common-instancetypes.git
virtualmachineclusterinstancetype.instancetype.kubevirt.io/cx1.2xlarge created
virtualmachineclusterinstancetype.instancetype.kubevirt.io/cx1.4xlarge created
virtualmachineclusterinstancetype.instancetype.kubevirt.io/cx1.8xlarge created
virtualmachineclusterinstancetype.instancetype.kubevirt.io/cx1.large created
virtualmachineclusterinstancetype.instancetype.kubevirt.io/cx1.medium created
virtualmachineclusterinstancetype.instancetype.kubevirt.io/cx1.xlarge created
virtualmachineclusterinstancetype.instancetype.kubevirt.io/gn1.2xlarge created
virtualmachineclusterinstancetype.instancetype.kubevirt.io/gn1.4xlarge created
virtualmachineclusterinstancetype.instancetype.kubevirt.io/gn1.8xlarge created
virtualmachineclusterinstancetype.instancetype.kubevirt.io/gn1.xlarge created
[..]
virtualmachineclusterpreference.instancetype.kubevirt.io/alpine created
virtualmachineclusterpreference.instancetype.kubevirt.io/centos.7 created
virtualmachineclusterpreference.instancetype.kubevirt.io/centos.7.desktop created
virtualmachineclusterpreference.instancetype.kubevirt.io/centos.stream8 created
virtualmachineclusterpreference.instancetype.kubevirt.io/centos.stream8.desktop created
virtualmachineclusterpreference.instancetype.kubevirt.io/centos.stream8.dpdk created
virtualmachineclusterpreference.instancetype.kubevirt.io/centos.stream9 created
virtualmachineclusterpreference.instancetype.kubevirt.io/centos.stream9.desktop created
virtualmachineclusterpreference.instancetype.kubevirt.io/centos.stream9.dpdk created
virtualmachineclusterpreference.instancetype.kubevirt.io/cirros created
```

`$ virtctl create vm`

- Creates a `VirtualMachine` manifest a user can modify or submit
- Provides `--instancetype` & `--preference` switches
- Provides `--infer-instancetype` & `--infer-preference` switches
- Handles various `volume` types and import sources
- Allows cloudinit `user-data` and `network-data`

Our hero returns... 

`$USER` can now create their `VirtualMachine` using...

```
$ virtctl create vm \
  --instancetype u1.nano \
  --preference fedora \
  --volume-import type:pvc,src:default/fedora,size:10Gi,name:fedora \
  --name teleportation-test-vm | kubectl apply -f -
[..]
The request is invalid: spec.instancetype: failure checking preference requirements:
insufficient Memory resources of 512Mi provided by instance type, preference
requires 2Gi

$ virtctl create vm \
  --instancetype u1.medium \
  --preference fedora \
  --volume-import type:pvc,src:default/fedora,size:10Gi,name:fedora \
  --name teleportation-test-vm | kubectl apply -f -
virtualmachine.kubevirt.io/teleportation-test-vm created

$ kubectl get vms
NAME                    AGE   STATUS    READY
teleportation-test-vm   6s    Running   True
```

Our hero returns... 

`$USER` can also define defaults for others to infer in the future...

```
$ kubectl label pvc/fedora \
  instancetype.kubevirt.io/default-instancetype=u1.medium \
  instancetype.kubevirt.io/default-preference=fedora

$ virtctl create vm \
  --infer-instancetype \
  --infer-preference \
  --volume-import type:pvc,src:default/fedora,size:10Gi,name:fedora \
  --name teleportation-test-vm-new | kubectl apply -f -
virtualmachine.kubevirt.io/teleportation-test-vm-new created

$ kubectl get vms
NAME                        AGE   STATUS    READY
teleportation-test-vm       40s   Running   True
teleportation-test-vm-new   6s    Running   True
```

What's New? (`v1.1.0` -> `v1.3.0`)

Inference by default with `virtctl`

`inferFromVolumeFailurePolicy` has been introduced to `InstancetypeMatcher`.

```
$ virtctl create vm \
  --volume-import type:pvc,src:default/fedora,size:10Gi,name:fedora \
  --name teleportation-test-vm
[..]
apiVersion: kubevirt.io/v1
kind: VirtualMachine
[..]
spec:
  instancetype:
    inferFromVolume: fedora
    inferFromVolumeFailurePolicy: Ignore
[..]
```

`Reject` is the default, `Ignore` allows us to attempt to infer and ignore any failure.

Inference by default with `virtctl`

```
$ virtctl create vm \
  --volume-import type:pvc,src:default/fedora,size:10Gi,name:fedora \
  --name teleportation-auto-infer-vm | kubectl apply -f -
virtualmachine.kubevirt.io/teleportation-test-vm created

$ kubectl get vms
NAME                          AGE   STATUS    READY
teleportation-auto-infer-vm   25s   Running   True
```

`common-instancetypes` now deployed by `virt-operator`

```
$ kubectl get kv/kubevirt -n kubevirt -o=jsonpath='{.spec.configuration.developerConfiguration.featureGates}'
["CommonInstancetypesDeploymentGate"]

$ kubectl get virtualmachineclusterinstancetypes \
  -o=jsonpath='{.items[0].metadata.labels.instancetype\.kubevirt\.io\/common-instancetypes\-version}'
v1.0.0

$ kubectl get virtualmachineclusterpreferences \
  -l app.kubernetes.io/managed-by=virt-operator \
  --no-headers | wc -l
32

$ kubectl get virtualmachineclusterinstancetypes \
  -l app.kubernetes.io/managed-by=virt-operator \
  --no-headers | wc -l
39
```

`containerdisks` can now provide defaults via CDI

Provided as `ENV` variables of the container image:

```
$ podman inspect quay.io/containerdisks/fedora:40 | jq '.[] | .Config.Env'
[
  "INSTANCETYPE_KUBEVIRT_IO_DEFAULT_PREFERENCE=fedora",
  "INSTANCETYPE_KUBEVIRT_IO_DEFAULT_INSTANCETYPE=u1.medium"
]
```

`containerdisks` can now provide defaults via CDI

```
$ kubectl apply -f -<<EOF
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  annotations:
    cdi.kubevirt.io/storage.bind.immediate.requested: "true"
  name: fedora
spec:
  source:
    registry:
      pullMethod: node
      url: docker://quay.io/containerdisks/fedora:40
  storage:
    resources:
      requests:
        storage: 10Gi
EOF
datavolume.cdi.kubevirt.io/fedora created
$ kubectl get datavolume
NAME     PHASE       PROGRESS   RESTARTS   AGE
fedora   Succeeded   100.0%                28s

$ kubectl get pvc/fedora -o json | jq .metadata.labels
{
[..]
  "instancetype.kubevirt.io/default-instancetype": "u1.medium",
  "instancetype.kubevirt.io/default-preference": "fedora"
[..]
}
```

What's next? (~ `v1.4.0`)

`instancetype.kubevirt.io/v1`

- Stop mutating the `Spec` of a `VirtualMachine` after submission
- Allow default inference directly from containerdisks
- Architecture requirements for preferences

`common-instancetypes` deployment by default

- Graduate the feature to GA
- Enable `CommonInstancetypesDeploymentGate` by default

`virtctl`

- Add `--network` and `--interface` support
- Add `--ssh-key` injection support
- Contributions welcome!

Q&A

Thank You!

Reminder - KubeVirt @ devconf.cz 2024

- Come and visit our Booth!
- Device passthrough in KubeVirt - Victor Toso @ 14/06 14:00 D105
  - https://pretalx.com/devconf-cz-2024/talk/JWGCKX/
- What the swap?! Swap on k8s - Itamar Holder @ 14/06 16:16 D105
  - https://pretalx.com/devconf-cz-2024/talk/KUD3WB/

Extra slides cut for time...

What about `VirtualMachinePresets`?

- Produced non-deterministic `VirtualMachine` output
- Based on `PodPresets` concept in k8s now removed with >= `v1.20`
- Deprecated as of KubeVirt `v0.57.0` for removal in `kubevirt.io/v2`

What about `template.openshift.io/v1`?

- Provides a downstream (OpenShift) only `Template` CRD
- Allows the entire `VirtualMachine` to be templated
- Users can't predefine resource sizing without shipping lots of templates

Misc `instancetype.kubevirt.io` updates

- `LiveUpdate` support for instance types (vCPU exposed as Sockets and memory hotplug)
- `SpreadOptions` for the `Spread` `PreferredCPUTopology` preference
- `ControllerRevision` upgrades to `v1beta1` on `VirtualMachine` resync