KubeVirt总结

背景

KubeVirt 技术满足了已经采用或想要采用 Kubernetes 但现有的基于虚拟机的工作负载无法轻松容器化的开发团队的需求。更具体地说，该技术提供了一个统一的开发平台，开发人员可以在该平台上构建、修改和部署驻留在通用共享环境中的应用程序容器和虚拟机中的应用程序。

好处是广泛而显著的。依赖现有基于虚拟机的工作负载的团队能够快速容器化应用程序。通过将虚拟化工作负载直接放置在开发工作流程中，团队可以随着时间的推移分解它们，同时仍然根据需要利用剩余的虚拟化组件。

场景

1、利用 KubeVirt 和 Kubernetes 来管理不能容器化的应用程序的虚拟机。

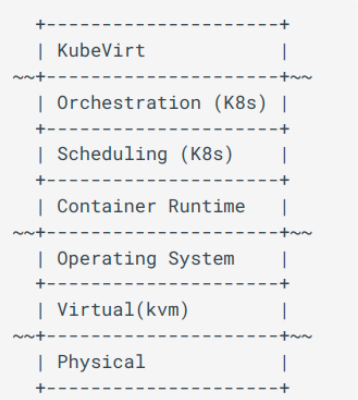
2、在一个平台上将现有的虚拟化工作负载与新的容器工作负载相结合。

3、支持在容器中开发与现有虚拟化应用程序交互的新微服务应用程序。

架构

KubeVirt 使用面向服务的架构和编排模式构建。

分层



需要虚拟化服务的用户正在与虚拟化 API（见下文）通信，而虚拟化 API 又与 Kubernetes 集群通信以调度所请求的虚拟机实例 (VMI)。调度、网络和存储都委托给 Kubernetes，而 KubeVirt 提供虚拟化功能。

附加服务

KubeVirt 为您的 Kubernetes 集群提供附加功能，以执行虚拟机管理。

如果我们回想 Kubernetes 如何处理 Pod，那么我们就会记得 Pod 是通过将 Pod 规范发布到 Kubernetes API 服务器来创建的。然后，该规范被转换为 API 服务器内部的对象，该对象具有特定的类型或种类 - 这就是规范中的调用方式。 Pod 属于 Pod 类型。 Kubernetes 中的控制器知道如何处理这些 Pod 对象。因此，一旦看到新的 Pod 对象，这些控制器就会执行必要的操作来使 Pod 处于活动状态，并匹配所需的状态。

KubeVirt 使用相同的机制。因此，KubeVirt 做了三件事来提供新功能：

1、添加类型 - 所谓的自定义资源定义 (CRD) - 已添加到 Kubernetes API 中。

2、为与这些新类型关联的集群范围逻辑添加控制器（controller）。

3、为与这些新类型关联的节点特定逻辑添加守护进程。

这些可以让用户能够进行如下操作：

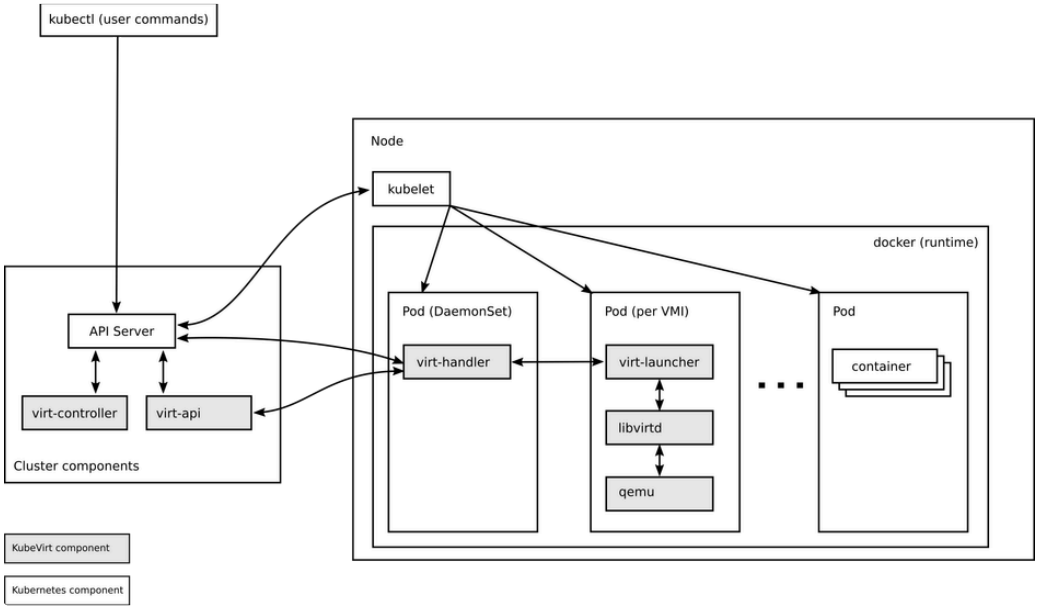
1、在kubernetes中创建这些新类型的对象（vmis）

2、进而控制器负责将vmis调度到一些节点上

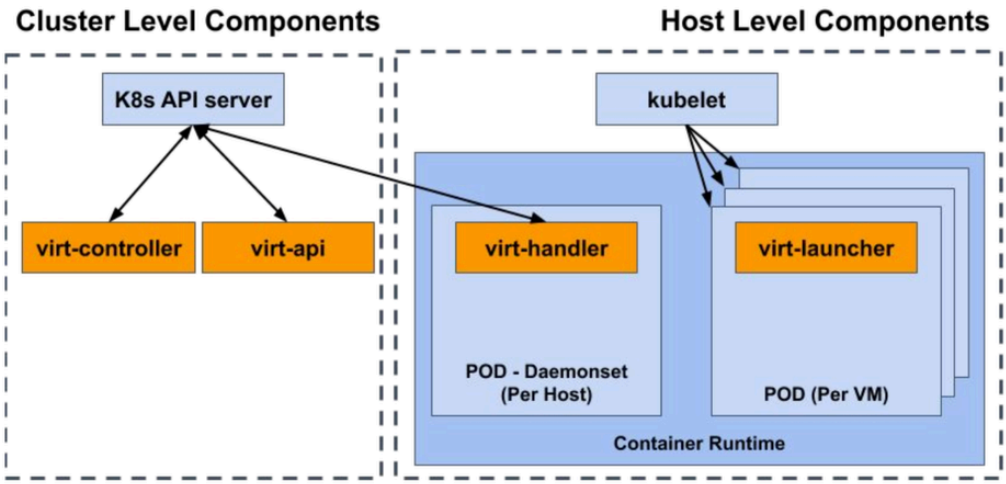
3、进而节点上的守护进程virt-handler与kubelet一起启动并配置vmis，直到其达到预期状态

最后一点；控制器和守护进程都作为 Pod（或类似的）运行在 Kubernetes 集群之上，并且不与集群一起安装。正如之前所说，该类型甚至是在 Kubernetes API 服务器内部定义的。这允许用户与 Kubernetes 对话，但修改 VMI。

下图说明了附加控制器和守护程序如何与 Kubernetes 通信以及附加类型的存储位置：



组件



virt-api：该组件提供 HTTP RESTful 入口点来管理集群内的虚拟机。
virt-controller：该组件是一个 Kubernetes 控制器，用于管理 Kubernetes 集群中虚拟机的生命周期。
virt-handler：这是一个在每个 Kubernetes 节点上运行的守护进程。它负责根据 Kubernetes 监控 VMI 的状态，并确保相应的 libvirt 域相应地启

动或停止。为了执行这些操作，virt-handler 与每个 virt-launcher 都有一个通信通道，用于管理 virt-launcher pod 内 qemu 进程的生命周期。
virt-launcher：每个正在运行的 VMI 都有一个。该组件直接管理 VMI pod 内 qemu 进程的生命周期，并从 virt-handler 接收生命周期命令。

更多组件

- VMI（CRD）：VMI 定义作为自定义资源保存在 Kubernetes API 服务器内。

VMI 定义定义了虚拟机本身的所有属性，例如

机器的种类
CPU类型
RAM 和 vCPU 数量
NIC 的数量和类型

- libvirtd：每个 VMI Pod 中都存在一个 libvirtd 实例。 virt-launcher 使用 libvirtd 来管理 VMI 进程的生命周期。
- Storage Controller：KubeVirt 可能会定义存储 CRD 以及流程描述，这将允许此类控制器无缝集成到 KubeVirt 中。
- 网络接口：https://kubevirt.io/user-guide/virtual_machines/interfaces_and_networks/

应用布局

VirtualMachineInstance (VMI) 是代表实例的基本临时构建块的自定义资源。在很多情况下，该对象不会由用户直接创建，而是由高级资源创建。 VMI 的高级资源可以是：

1、虚拟机 (VM) - 有状态 VM，可以在保留 VM 数据和状态的同时停止和启动。
2、VirtualMachineInstanceReplicaSet (VMIRS) - 与 Pod ReplicaSet 类似，是一组在模板中定义的具有相似配置的临时 VMI。

本地负载

KubeVirt 部署在 Kubernetes 集群之上。这意味着可以继续在通过 KubeVirt 管理的 VMI 旁边运行 Kubernetes 原生工作负载。

此外：如果可以运行本机工作负载，并且安装了 KubeVirt，那么也应该能够运行基于 VM 的工作负载。例如，与在普通 Pod 中使用该功能相比，应用程序操作员不应需要额外的权限来使用 VM 的集群功能。

从安全角度来看，安装和使用 KubeVirt 不得向用户授予他们尚未拥有的有关本机工作负载的任何权限。例如，非特权应用程序操作员绝不能使用 KubeVirt 功能来访问特权 Pod。

扩展话题

我们喜欢虚拟机，认为它们非常重要，并努力让它们在 Kubernetes 中易于使用。但与虚拟机相比，我们更喜欢良好的设计和模块化、可重用的组件。我们经常面临一个困境：我们应该以最适合虚拟机优化的方式解决 KubeVirt 中的问题，还是应该采取更长的路径并将解决方案引入基于 Pod 的工作负载？

为了解决这些困境，我们提出了 KubeVirt Razor：“如果某件事对 Pod 有用，我们不应该只为 VM 实现它”。

例如，我们讨论了如何将虚拟机连接到外部网络资源。最快的方法似乎是引入 KubeVirt 特定的代码，将虚拟机附加到主机桥。然而，我们选择了与 Multus 和 CNI 集成并改进它们的更长路径。

VirtualMachine

VirtualMachine 为集群内的 VirtualMachineInstance 提供额外的管理功能。包括了：

1、API稳定性

2、控制器级别的启动/停止/重启功能

3、离线配置更改与 VirtualMachineInstance 重新创建的传播

4、确保 VirtualMachineInstance 正在运行（如果应该运行）

它侧重于控制器实例和虚拟机实例之间的 1:1 关系。在很多方面，它与 spec.replica 设置为 1 的 StatefulSet 非常相似。

如何使用VirtualMachine

如果spec.running设置为true，VirtualMachine将确保集群中存在具有相同名称的VirtualMachineInstance对象。此外，如果 spec.running 设置为 false，它将确保 VirtualMachineInstance 将从集群中删除。

存在一个字段spec.runStrategy，它也可用于控制关联的VirtualMachineInstance对象的状态。为了避免混乱和矛盾的状态，这些字段是相互排斥的。

可以在运行策略中找到spec.runStrategy与spec.running的扩展解释。

启动和关闭

```
# Start the virtual machine:
virtctl start vm

# Stop the virtual machine:
virtctl stop vm
```

控制器状态

```
kubectl get vms -A
```

Stopped

Provisioning

Starting

Running

Paused

Migrating

Stopping

Terminating

Unknown

重启

```
＃重启
virtctl restart vm
＃强制重启
virtctl restart vm --force --grace-period=0
```

注意

VirtualMachine不会重启或重建VirtualMachineInstance，直到vmi的当前实例从集群中删除。

暴露为服务

VirtualMachine可以作为服务公开。 VirtualMachineInstance 启动后，实际服务将可用，无需额外交互。

例如，在VirtualMachine创建之后、启动之前，使用 virtctl 将 SSH 端口 (22) 公开为 ClusterIP 服务：

```
virtctl expose virtualmachine vmi-ephemeral --name vmiservice --port 27017 --target-port 22
```

何时使用VirtualMachine

当重新启动之间需要 API 稳定性时¶
VirtualMachine 确保 VirtualMachineInstance API 配置在重新启动之间保持一致。一个典型的例子是绑定到虚拟机固件 UUID 的许可证。

VirtualMachine 确保 UUID 始终保持不变，而无需用户处理。

主要好处之一是用户仍然可以使用默认逻辑，尽管需要稳定的 API。

下次重启时何时应获取配置更新¶
如果 VirtualMachineInstance 配置应该在集群内可修改，并且这些更改应该在下一次 VirtualMachineInstance 重新启动时生效。这意味着不涉及热插拔。

当你想让集群管理你个人的VirtualMachineInstance时¶
Kubernetes 作为声明式系统可以帮助您管理 VirtualMachineInstance。你告诉它你希望这个 VirtualMachineInstance 与你的应用程序一起运行，VirtualMachine 将尝试确保它保持运行。

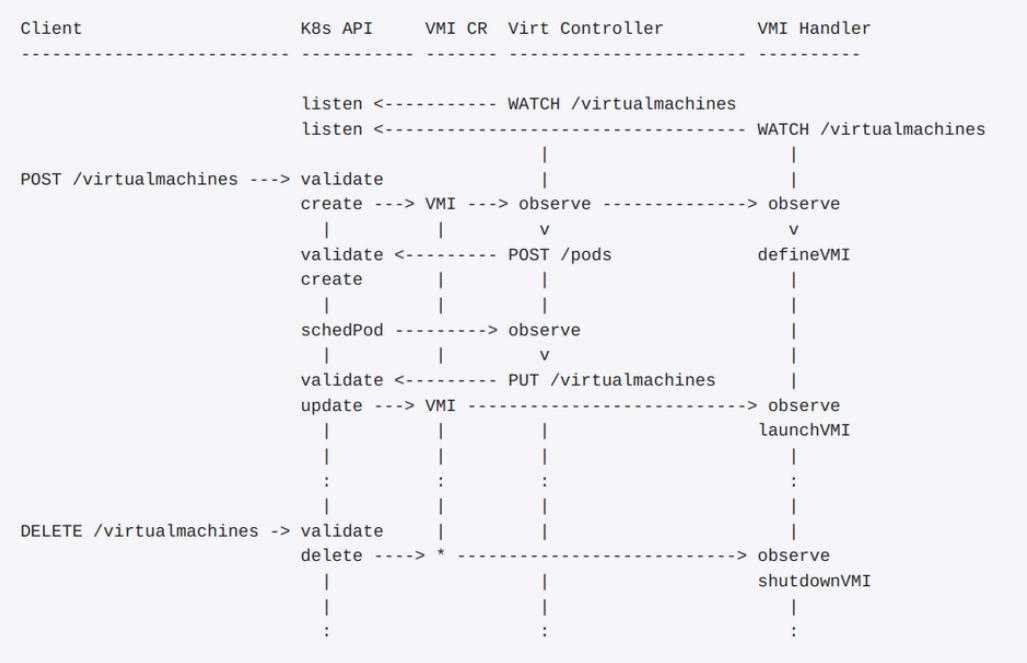
当前的信念是，如果定义 VirtualMachineInstance 应该运行，那么它就应该运行。这与许多经典虚拟化平台不同，在这些平台中，虚拟机在关闭后会保持关闭状态。如果需要，可以添加重新启动策略。

kubectl命令

```
# Define a virtual machine:
   kubectl create -f vm.yaml

   # Start the virtual machine:
   kubectl patch virtualmachine vm --type merge -p \
      '{"spec":{"running":true}}'

   # Look at virtual machine status and associated events:
   kubectl describe virtualmachine vm

   # Look at the now created virtual machine instance status and associated events:
   kubectl describe virtualmachineinstance vm

   # Stop the virtual machine instance:
   kubectl patch virtualmachine vm --type merge -p \
      '{"spec":{"running":false}}'

   # Restart the virtual machine (you delete the instance!):
   kubectl delete virtualmachineinstance vm

   # Implicit cascade delete (first deletes the virtual machine and then the virtual machine instance)
   kubectl delete virtualmachine vm

   # Explicit cascade delete (first deletes the virtual machine and then the virtual machine instance)
   kubectl delete virtualmachine vm --cascade=true

   # Orphan delete (The running virtual machine is only detached, not deleted)
   # Recreating the virtual machine would lead to the adoption of the virtual machine instance
   kubectl delete virtualmachine vm --cascade=false
```

工作流程

以vmi创建和删除流程说明kubevirt工作流程。



1、客户端将新的 VMI 定义发布到 K8s API 服务器。
2、K8s API 服务器验证输入并创建 VMI 自定义资源。
3、virt-controller 观察新 VMI 对象的创建并创建相应的 pod。
4、Kubernetes 正在主机上调度 pod。
5、virt-controller 观察到 VMI 的 pod 已启动，并更新 VMI 对象中的 nodeName 字段。现在，nodeName 已设置，责任将转移到 virt-handler 来执行任何进一步的操作。
6、virt-handler (DaemonSet) 观察到 VMI 已分配给运行它的主机。
7、virt-handler 使用 VMI 规范，并使用 VMI pod 中的 libvirtd 实例发出创建相应域的信号。
8、客户端通过 virt-api-server 删除 VMI 对象。
9、virt-handler 观察到删除并关闭域。

入门

KubeVirt quickstart with cloud providers | KubeVirt.io

Use KubeVirt | KubeVirt.io

Live Migration | KubeVirt.io

安装与使用

https://kubevirt.io/user-guide/operations/installation/

https://kubevirt.io/user-guide/virtual_machines/virtual_machine_instances/

调试

Control libvirt logging for each component - KubeVirt user guide

升级

KubeVirt Upgrades | KubeVirt.io

社区

bug：https://github.com/kubevirt/kubevirt/issues 连接您的 Github 帐户

贡献：Contributing - KubeVirt user guide

API：KubeVirt API Reference

release notes：Release Notes - KubeVirt user guide

参考文献

KubeVirt.io

GitHub - kubevirt/kubevirt: Kubernetes Virtualization API and runtime in order to define and manage virtual machines.