kubelet

简介

- 容器生命周期控制
  - 运行和健康检查
  - 资源管理
- 提供 api 接口
  - apiserver
  - stats
- 依赖 cri ， os(cgroup)

sync

- 首先介绍pod CRUD 操作，将以从此延申到不同模块内部实现
- 同步过程，主要涉及 containers, kuberuntime, pleg 等目录内代码
  - container目录：定义 containre 接口和数据结构，如 runtime，stream，image 等
  - kuberuntime：具体container接口实现
- 同步过程中主要使用的数据结构
  - podCache：实际pod状态，从 runtime 定期或 pleg 事件触发更新
  - podManager：存储应运行的且已接纳 pod 和镜像 pod。从不同的源（apiserver、本地文件系统或静态 http）接收
  - podWorker：存储实际运行的 pod，注意当 apiserver 强制删除 pod，会将其从 podManager 中删除，但 pod 可能仍正被 podWorkers 跟踪管理
    - statusManager：从 podWorker 异步接收更新的 pod 状态更新，并更新服务器端 pod 状态以匹配，但因为是异步更新，所以如需要检查 Pod 是否在运行的应直接咨询 Worker，但这个字段对于其他管理器基本够用，是事实上的 pod 状态存储
- PLEG：周期(10s)从cache拉取容器信息，并对比前后状态生成事件
  - Running：产生 Started 事件
  - Exited：产生 Died 事件
  - Unknown：产生 Changed 事件
  - NonExist：前一状态是 Exist时，产生 Remove；否则产生 Died
- 同步先进行大循环，内部进行小循环
  - 大循环：检查 cri 状态，不正常则睡眠并下次尝试，注意 status 返回有 network 和 status，这里只检查 status。network 不正常没关系
  - 小循环：监听多个来源，但只处理一个，之后退出进入大循环
    - source: apiserver 和 static path 获取事件，事件包括类型和pods
      - 类型：Add，Update，Remove，Reconcile，Delete
    - pleg: 实际pod状态，产生类型和podid 事件
      - Died 事件：放入 containerDelete 队列，通常会保留非运行的最后一个用于调查问题，至于是否全部删除，则由 podWorkers 确定
      - 其他：执行 SyncPod
    - probe: 分 readinesss，liveiness，startup 三种，均执行 ProbeSync
      - startup：若没执行成功，则不会继续创建容器，可以用于启动顺序
    - syncCh：执行 SyncPod，跳过 mirror 类型，向 PodWorker 创建 sync 事件

    - housekeep: 周期，任务是终止 PodWorker、终止不需要的 Pod 以及删除孤立的volume / Pod 目录

- source 管道的事件详细介绍
    - Add：对于mirror类型时，创建 Update；其他时，从 podManager 和 podWorker 中过滤出正运行的actives，判断 pod 和active 是不是可以准入，创建 Create
    - Update或Delete：创建 Update；
    - Sync：对于mirror 类型时跳过，其他类型创建 Update
    - Remove: 对于mirror类型时，创建 Update; 其他类型时，创建 kill ；
    - Reconcile：这是全量信息，因此更新 PodManager 内数据，之后判断是否是从 Ready 变化的，创建 SyncPod；若pod是被驱逐，则执行 containerDelete(从cache中获取ID，并放入 worker 中，即异步删除)。但后两条是否在这里还有疑问，见注释
    - ProbeSync：执行上面提到的 Sync
- 当source 处理完成后，产生的事件信息给 PodWorker 缓存，事件有以下元数据
    - 类型：create, update，kill，sync
    - 时间，kill 选项
    - pod：预期状态，来自 podManager
    - mirrorpod：只有静态pod才有，和pod进行计算后来决策，因此必须 pod 设置
    - runtimepod：运行时信息，和 pod 字段互斥。如 pod 设置，则忽略本字段；若 pod 未设置，则本字段必须设置

podWorker

- 上文中介绍时间会进入 podWorker 中，调用的是 podWorker 的 UpdatePod 方法，参数就是事件信息，接下来描述又会如何处理，注意 podWorker 是记录实际运行的 pod
- 创建 podWorker 时有以下参数
    - podSyncer：实际对runtime操作的接口定义，最后指向的是 kubelet 结构
    - podCache：实际 container 状态
    - workQueue：内部使用的队列
    - resyncInterval, backOffPeriod：重试和退避
- 对于第一次的 pod，会准备协程(通信 channel) 并通知协程去处理，注意只处理从 source 过来的 pod
    - 事件信息最终为三类 Terminated, Terminating, Sync，并会调用对应的 podSyncer 方法，通常流转过程是 sync -> terminating -> terminated ，对于 terminated 之后的状态则不再会执行 podWorkerLoop，此时会被清理，其他的会加入到 workQueue 中，以待下次同步
- 介绍 podSyncer 中方法
    - Terminated：状态是 podCache 确定已经结束，因此这主要执行清理，包括更新 statusManager 中状态；waitUnmount；cgroup；secret/cm

    - Terminating：同步调用runtime的kill 方法，以及更新 statusManager 和停止 probe 检查，如在 apiserver 强制删除则可能会导致这里不被调用
    - TerminatingRuntime：没在上文的通用流转内介绍，因为在 apiserver 强制删除时，因此该操作是对孤儿pod 的处理，这里只调用 runtime 的 kill
    - Sync：创建pod的流程，包括准入检查，更新statusManager，cgroup 准备，卷准备，probe添加，执行 kuberuntime 创建 （以上都是可重入的操作，保证create/update 是相同的路径）
        - 计算 sandbox ，container 变化来决定创建与否，是否变化的计算数据信息来自 runtime，probe，spec 等
- TODO 图

控制器

- 介绍相对独立的控制器

evict

- 驱逐是为确保级别更多的稳定运行，和上文中 可分配是强相关，1.31 beta版本
- 资源校验有以下，操作符号为小于，值以是百分比, 数字
    - memory.available 等于 node.status.capacity - node.stats.memory.workingSet
    - nodefs.available 等于 node.stats.fs.available
    - nodefs.inodesFree 等于 node.stats.fs.inodesFree
    - imagefs.available 等于 node.stats.runtime.imagefs.available
    - imagefs.inodesFree 等于 node.stats.runtime.imagefs.inodesFree
    - containerfs.available 等于 node.stats.runtime.containerfs.available
    - containerfs.inodesFree等于 node.stats.runtime.containerfs.inodesFree
    - pid.available 等于 node.stats.rlimit.maxpid - node.stats.rlimit.curproc
- 驱逐顺序，使用量超出请求的 BestEffort 或 Burstable 按照优先级删除；按照优先级，使用量少于请求的 Guaranteed 和 Burstable 最后被驱逐
    - pod 的资源使用超过请求的
    - Pod 优先级
    - Pod 相对于请求的资源使用情况
- 根据驱逐策略，所以对于不同控制器，最好有以下
    - 对于静态 pod，建议设置 priorityClassName 字段
    - 对于 ds 的容器，通过指定合适的 priorityClassName 为这些 pod 提供足够高的优先级。 也可以使用较低优先级或默认优先级，仅允许该 DaemonSet 中的 pod 在有足够资源时运行
- 对于 kubelet 无法立即观察到内存压力，建议设置 kernel-memcg-notification 标志
- 有关配置

- EvictionHard map[string]string 定义硬驱逐阈值的数量，如：{"memory.available": "300Mi"}
- EvictionSoft map[string]string 定义软驱逐阈值的数量
- EvictionSoftGracePeriod map[string]string 软驱逐信号宽限期的数量的映射，如：{"memory.available": "30s"}
- EvictionPressureTransitionPeriod metav1.Duration 退出驱逐压力条件之前必须等待的持续时间
- EvictionMaxPodGracePeriod int32 终止 pod 时允许使用的最大宽限期，单位秒
- EvictionMinimumReclaim map[string]string 将回收的给定资源量, 当该资源面临压力

allocate

- https://kubernetes.io/docs/tasks/administer-cluster/reserve-compute-resources/
- 可分配资源： all - kubereversed - systemreversed - hardevict
- 节点可分配资源，根据 enforceNodeAllocatable 选择计算方式，默认方式是 pods 类型，还可以追加以下，将在 pods 计算基础上再减
  - kube-reserved 要求配置 kubeReservedCgroup
  - system-reserved 要求配置 systemReservedCgroup
- 使用 pods 时，资源预留有关配置如下
  - SystemReserved / KubeReserved map[string]string 保留资源，可以设置的资源类型有 cpu, memory, ephemeral-storage, pid
  - SystemReservedCgroup / KubeReservedCgroup 当enforceNodeAllocatable中含有相同的配置时，会向对应 cgoup 中写入 reserve 资源
  - ReservedSystemCPUs：为主机线程和 k8s 线程保留的cpu列表, 会覆盖 SystemReserved / KubeReserved 提供的 CPU
  - ReservedMemory： NUMA 节点的内存预留的逗号分隔列表
- 更多的详细操作将在之后 cm 一节介绍

os

- 启动时，检查会设置sysctl，iptable，veth 模式等，为容器铺平道路
- iptable 自 1.24 之后，会准备以下，并且不创建 KUBE-MARK-DROP|MASK，和KUBE-POSTROUTING 链，以下这些操作会周期检查来确保
  - mangle / filter / nat 表上创建 KUBE-KUBELET-CANARY 链
  - mangle 表上创建 KUBE-IPTABLES-HINT 链，被其他插件用于确定 iptable 版本 (?)
  - filter 表上创建 KUBE-FIREWALL 链，并从前插入
    - output / input 链，跳转到 firewall
  - KUBE-FIREWALL 链添加规则，原因见 https://issue.k8s.io/90259
    - 目的是127，源地址不是127，drop

lifecycle

- 容器生命周期，通常按顺序有 admit，lifecycle，probe 等
- 特性
  - SidecarContainers：alpha: v1.28 beta: v1.29，一种initcontainer类型，会一直运行直到 pod 被删除，这和之前init工作方式不同，之前是 init 必须成功且退出
- probe 检查
  - liveness：失败则 pod 将重启
  - readiness：成功则 Ready 被设置
  - startup：成功则初始化完成，在完成之前不会执行其他探测。如失败，则将重启
- admit 过滤不可用时，阻塞创建，包括 evictionAdmitHandler, sysctlsAllowlist, AllocateResources, PredicateAdmit, AppArmorAdmit, shutdownAdmit。下面将分别介绍可能发生的问题，即错误原因
  - evictionAdmit：
    - 优先级大于 system-node-critical 或静态pod，或非BestEffort 时直接通过
    - 校验是否有memory 不可调度污点，且pod能容忍，如不行，则是 evicted 原因
  - sysctlsAllowlist： 校验 pod.spec.SecurityContext.Sysctls 合规，仅以下配置
    - kernel.shm.
    - kernel.msg.
    - fs.mqueue.
    - net.
  - AllocateResources：实际使用是 topology 的准入检查
  - PredicateAdmit：分为以下几部分
    - 确定标签 kubernetes.io/os 符合
    - 使用非校验pod以外组建 nodeinfo，再用 framework 判断
      - cpu, memory, EphemeralStorage, external 超限则错误是 Outofxxx，不过当pod优先级在 system-node-crictl 之上的，会尝试进行驱逐已满足该pod运行
      - 节点亲和性，节点名称，hostport
      - 非静态pod，会判断匹配污点和容忍
  - AppArmorAdmit: TODO
  - shutdown: 关机时禁止调度

```
// pod.status.conditions 字段例子
- lastTransitionTime: "2024-10-12T14:47:54Z"
  status: "True"
  type: Ready
- lastTransitionTime: "2024-10-12T14:47:50Z"
```

```
  status: "True"
  type: Initialized
```

cm

- 容器资源管理，除 runtime 以外，其他节点上所有资源都被纳入视野，当前包括有 device，topology，cpu，memory，以及 dra(crd资源)，其基础是 cgroup
- 这一部分涉及到资源的核心模型，所有代码均在 cm 目录内
- feature
  - MemoryQoS：alpha: v1.22，建议和cgroup2 一起使用
  - KubeletCgroupDriverFromCRI：alpha: v1.28
  - CPUManagerPolicyOptions：alpha: v1.22 beta: v1.23
  - TopologyManagerPolicyOptions：alpha: v1.26
  - NodeSwap：alpha: v1.22，beta2: v1.30，要求和cgroup2
  - MemoryManager：alpha: v1.21 beta: v1.22
- 初始化时会准备些常用资源
  - capacity：节点常见资源的容量
  - pidMax：根据配置，设置容器 pids 限制
  - qos: 服务水平，依据 resource 的 request/limit 分组
  - topology: 其内包括 cpu，device，memory

cgroup

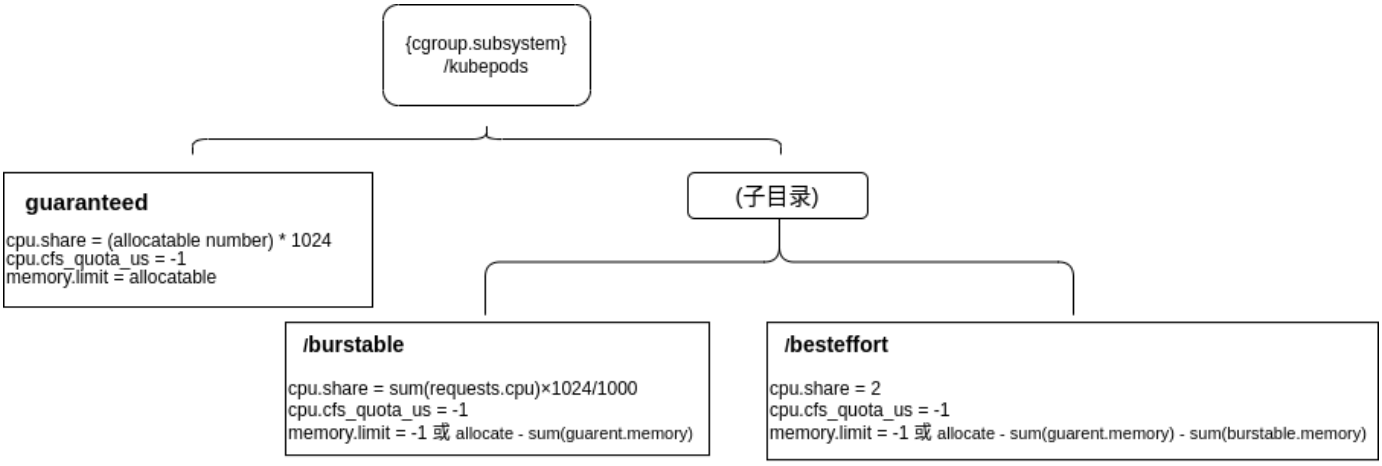
- 驱逐，分配策略都依赖 cgroup，因此了解机制
- cgroup 简介，更详细不在这里说明
  - /proc/cgroups 当前系统支持的子系统
  - /sys/fs/cgroup 默认挂载的位置
    - kubelet 要求节点必须支持: cpuset/memory/cpu.cfs
  - cpuset 来源
    - /sys/devices/system/node/online
  - 不同cgroup driver
    - cgroupfs: {cgroup-root}/{subsystem}/kubepods/
    - systemd: {cgroup-root}/{subsystem}/kubepods.slice

```
// 有关配置
CgroupRoot
CgroupsPerQOS: 默认 true，不同qos分别设置
```

```
CgroupDriver: cgroupfs或systemd，1.xx 之后使用 cri 获取
MemoryThrottlingFactor: 当内存使用达到 limit 百分比后，主动 gc
```

qos

- cpu 限制
  - 默认 cfs_quota_us 为 -1；cfs_period_us 为 100000 (单位微秒, 默认 100ms)
    - request: 配置 cpu.shares 文件，调度内所占比例，默认 1024(最小2)，和同QoS容器平分时间片，所有pod的请求和不超过 cpu_num。如设置100m，share将配置 max(0.1 * 1024 / 1000, 2)，最后为 102
    - limit: 配置 cpu.cfs_quota_us 文件，默认 -1 或 [limit] * cfs_period_us, 含义是在周期(cfs_period_us)内可以使用的 CPU 时间的上限，所有pod的请求和不超过 cpu_num，注意周期 100ms 是单核上。当达到上限，开始限流，只能在下个周期继续执行。因为容器不知道cpu数目，所以 limit 设置是否合理不好确定
- 在k8s中 qos 主要分为三类，导致不同设置 oom，cgroup.cpu, cgroup.mem
- guaranteed: mem/cpu 的 request和limit一致，oom 分数 -998，根 cgroup 配置
  - cpu.shares = allocatable * 1024 # 单位个数，如cpu: "1" 得到 1024
  - cfs_quota_us = -1
  - memory.limit_in_bytes = allocatable
- burstable: mem/cpu 设置至少一个，oom分数 999，根 cgroup 配置
  - cpu.shares = max(sum(resources.requests.cpu)×1024/1000,2)
  - cfs_quota_us = -1
  - memory.limit_in_bytes: 打开memoryQOS时，为allocatable - sum(guaranteed)，否则为 -1
- besteffort: mem/cpu 一个为设置，oom分数 1000，根 cgroup 配置
  - cpu.shares = 2
  - cfs_quota_us = -1
  - memory.limit_in_bytes: 打开memoryQOS时，为 allocatable - sum(guaranteed) - sum(burstable)，否则为系统内存容量



- cpu.share 和 cpu.cfs_quota_us 工作机制
  - 未使用的 cpu 池(capacity) 根据 cpu.share 分配
  - 找到确切配额 cpu.cfs_quota_us 的 cgroup，按其时间保留和限制
  - 将 capacity - sum(cpu.cfs_quota_us) 标记未使用的 cpu 池
  - 从第一步迭代
- 参考代码 xxx 统计并输出进程时间片
- qosmanager 启动
  - 准备 besteffort, burstable 类型
  - 根据 activePods 更新各qos配额
  - 校验 EnforceNodeAllocatable 列表
    - 根据 system-reserved-cgroup 和 system-reserved 设置对应的cgroup
    - 根据 kube-reserved-cgroup 和 kube-reserved 设置对应的cgroup

```
# 例子
cpu: request 100m; limit 2
cpu:{shares:102,quota:200000,period:100000}

memory: request 600Mi; limit 4Gi
memory:{limit:4294967296,swap:4294967296}
```

topology

- 上文中主要是cgroup的 cpu，meory 控制器设置，但针对 numa 节点还是不能很好释放性能，因此有 topology 控制器
- 用于管理 scope (pod/container) 上的 cpu 等资源在 NUMA 节点上的对齐方式，当前能处理的节点数目为 8，从 lscpu 可以看到 nodes 数目
  - restricted：对资源分配进行严格的检查，确保pod在节点上的对齐是最佳的
  - best-effort：优先考虑对齐 CPU 和设备资源的 Pod
  - none：不会对资源对齐进行任何检查或优化
  - single-numa-node：确保 Pod 的所有资源都位于同 NUMA 节点上
  - prefer-closest-numa-nodes：在特性 TopologyManagerPolicyOptions 打开时可配置
- 配置
  - TopologyManagerPolicy：策略，默认 none, 不含prefer
  - TopologyManagerPolicyOptions：当前只能配置 prefer-closest-numa-nodes=true
  - TopologyManagerScope：级别，默认为 container
  - ReservedSystemCPUs: 字符串类型，静态配置 cpu，Kubernetes v1.17 [stable]，这些 cpuset 会给系统进程使用，并且不会分配给 guaranteed

    - ReservedMemory: 列表格式，静态配置 numa memory，Kubernetes v1.22 [beta]，该配置要求保留内存需满足以下公式，并且只在 memory 策略为 static 时发生作用，给系统进程使用，并且不会分配给 guaranteed
        - sum(reserved-memory(i)) = kube-reserved + system-reserved + eviction-threshold // i 是 node 索引
- 每个pod/container 的最优拓扑计算通过在 Admit 阶段时计算并缓存拓扑结果
- 问题：https://github.com/kubernetes/kubernetes/issues/115994
    - ReservedMemory 和 ReservedCpus 是保留给系统使用，但不会阻止非保证pod使用这个范围内的

```
// lscpu
CPU(s):                48  # 逻辑核数 = thread*core*socket
Thread(s) per core:    1   # 多少线程在每个核上
Core(s) per socket:    24  # 多少核在每个cpu
Socket(s):             2   # 物理cpu数目
NUMA node(s):          2   # numa 节点数目
NUMA node0 CPU(s):     0-23
NUMA node1 CPU(s):     24-47

// 查看对应关系
numactl -H
available: 2 nodes (0)
node 0 cpus: 0 1 ... 23
node 0 size: 15964 MB
node 0 free: 338 MB
node 1 cpus: 24 25 ... 47
node 1 size: 0 MB
node 1 free: 0 MB
node distances:
node   0   1
  0:  10  20
  1:  20  10

// 配置
当环境中有 --system-reserved cpu=1000m,memory=246462Mi, --eviction-
hard=memory.available<512Mi 时
总内存信息
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 20 21 22 23 24 25 26 27 28 29
node 0 size: 127684 MB
node 0 free: 106985 MB
node 1 cpus: 10 11 12 13 14 15 16 17 18 19 30 31 32 33 34 35 36 37 38 39
node 1 size: 129018 MB
```

```
node 1 free: 123677 MB
```

则得到：

```
--topology-manager-policy=best-effort
--topology-manager-scope=container
--cpu-manager-policy=static
--memory-manager-policy=Static
--reserved-cpus=0-9,20-29
--reserved-memory=0:memory=127684Mi
--reserved-memory=1:memory=119290Mi
```

cpu

- 主要功能
  - 周期获取pod，更新 {kubeletroot}/cpu_manager_state 文件
  - 提供 Admit，以及在 pod 创建前配置 cpuset
  - 这里会和 reversedCPU 联动
- 配置
  - CPUManagerPolicy：可选none(默认), static
  - CPUManagerPolicyOptions：字典类型，要求特性打开，可以配置如下之一
    - full-pcpus-only
    - distribute-cpus-across-numa
    - align-by-socket
  - CPUManagerReconcilePeriod：同步周期

```
// 文件 cpu_manager_state
{"policyName":"static","defaultCpuSet":"0","entries": {"pod":
{"container":"0"}},checksum":1353318690}
```

memory

- 为保证 QoS 类别的 Pod 提供保证内存（和大页）分配的功能
- 主要功能
  - 周期获取pod，更新 memory_manager_state文件
  - 提供 Admit，
- 配置
  - MemoryManagerPolicy：可选none(默认), static
  - ReservedMemory：指定numa节点设置保留的内存

```
// 文件 memory_manager_state
{"policyName":"None","machineState":{},"checksum":4236770233}
```

device

- 主要功能
  - deviceplugin 服务端，使用 device-plugins/kubelet.sock 监听连接
  - 准备 device-plugins/kubelet_internal_checkpoint 文件，记录已分配和容量

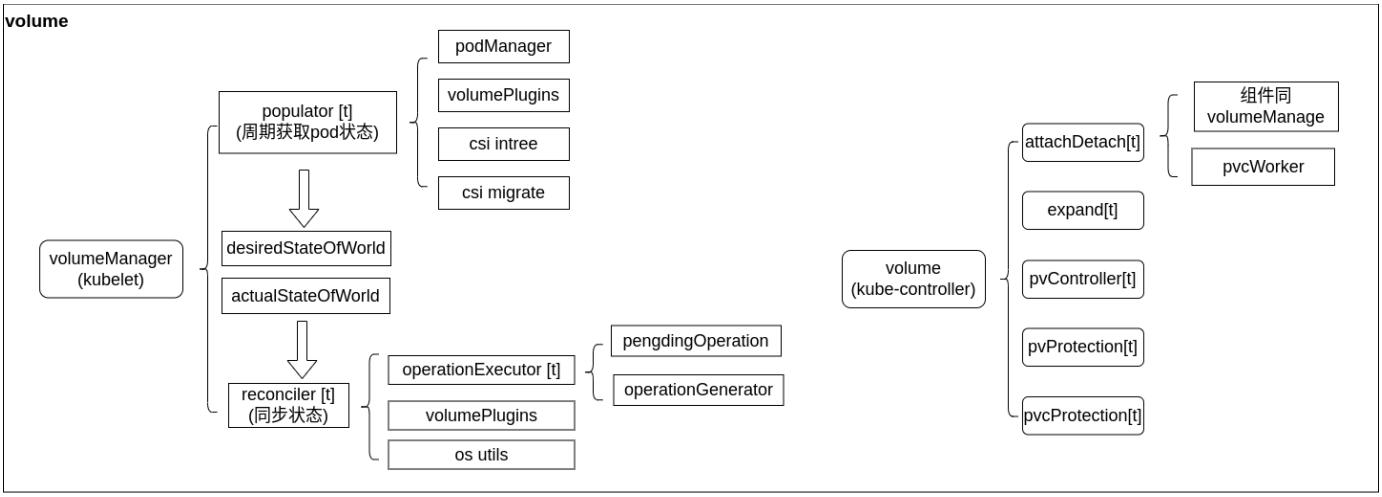
volume

- 卷管理模块，这里主要介绍持久卷的过程，普通的 cm/secret/empty 不在范围内

tmpfs

- cm/secret 为保证有效性，会有三种工作模式
  - watch: 默认模式，监听变化并更新挂载内容
  - cache：pod 被创建或更新时，缓存版本都会被标记为失效
  - get：发现变化后，如pod未重建，则不会更新

pvc



- 卷管理分为 kubelet 和 kube-controller 两部分，在主要卷操作部分都采用相同模式 populator + reconciler 方式进行，如果感兴趣可以描绘 volume 的状态转移图

plugin

device

- pkg/kubelet/cm/devicemanager/manager.go
- 主要功能
  - 向 pluginmanager 注册 处理函数
    - 当有 socket 创建后，回调处理函数向 plugin 设置socketPath/version/name
  - 提供 TopologyHits，用于准入阶段资源可用

    - 提供 node 可分配资源的更新接口
- 文件
    - /var/lib/kubelet/device-plugins/kubelet.sock // socket
    - /var/lib/kubelet/device-plugins/kubelet_internal_checkpoint // 状态文件

api

- pkg/kubelet/server
- NewServer
    - InstallAuthFilter
    - InstallDefaultHandlers
- InstallDefaultHandlers
    - 添加 /healthz 包括ping, log 检查
    - 添加 /pods 接口, 当前所有pod
    - 添加 /stats/summary 接口
    - 添加 metrics/{cadvisor,probes,resource} 接口
- /var/lib/kubelet/pod-resources/kubelet.sock