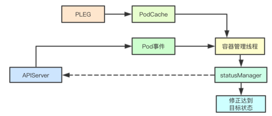
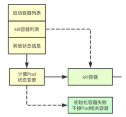
pod在k8s运行机制

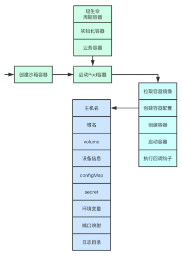
- 客户端创建
  - 用户通过 REST API 提交 Pod`描述文件到 API Server；
    API进行认证，鉴权和准入控制检查
    API Server 将 Pod 对象的信息存入 Etcd；
    Pod 的创建会生成事件，返回给 API Server；
  - 认证（Authentication）：
    - 证书认证、基于令牌的认证
  - 鉴权（Authorization）
    - 鉴权机制通常基于 RBAC（Role-Based Access Control）策略
  - 准入控制（Admission Control）
- Scheduler 监听到 Pod Update 事件；
  - Scheduler 会为 Pod 选择 Node；
    进行预选和优选，资源可用性、污点亲和性检查
    如有满足条件的 Node，Scheduler 会绑定 Pod 和 Node，并将绑定关系告知 API Server；
    API Server 将绑定信息写入 Etcd；
    生成 Pod Update 事件；
- Kubelet 监听到 Pod Update 事件，创建 Pod；
  - PLEG循环控制发现pod事件
    - PLEG PLEG 监听节点上的容器运行时，向 kubelet 提供监控节点上运行的 Pod 的生命周期事件，帮助 kubelet 及时了解节点上 Pod 状态的变化。
    - kubelet预先为每个创建过的pod启动一个容器管理线程（PodWorker）消费生命周期管道里面的事件。



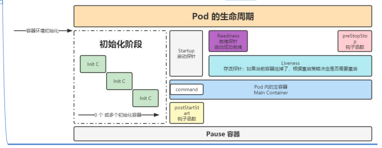
    - Pod 事件发生时，容器管理线程会根据事件通过statusManager组件管理修正目标状态，比如更新PodCache、更新状态、调整容器等。
      - *kubelet创建的podwork线程与ContainerRuntime时时交互*
      - ContainerRuntime 负责直接管理容器的创建、启动、停止和销毁等基本操作
        - 计算Pod容器变更
          - 1、需要启动的容器列表
          - 2、需要被kill掉的容器列表
          - 3、如果初始化容器未完成
            - 初始化失败尝试终止，Kill操作pod
            - 未知状态容器补偿，一般kill掉pod



  - kubelet为pod创建做准备
    - 准入检查（硬性条件）
      - 是否有足够的资源分配
      - 是否满足亲和性规则
      - 资源压力检测
    - PodCache中获取pod最新状态
      - 在通过准入检查之后，会调用statusManager（负责将Pod状态及时更新到Api-Server）来进行Pod最新状态的同步，同步给apiserver
    - kubelet检查runtime版本（软件环境），如果这里检查失败，则会将对应的容器状态设置为Blocked



  - kubelet创建事件管道与容器管理主线程，同步最新状态
    - Kubelet 作为grpc client，CRI作为grpc server，kubelet通过ImageManager和PodManager，使用CRI protobuf协议调用CRI相关功能
      - CRI创建容器沙箱
        - 创建一个Pod sandbox 容器（也叫pause容器），pause容器作为pod中所有容器的"父容器"实现了namespace共享UTS、IPC、PID、NETWORK并为每个pod充当PID 1，并接收僵尸进程
          - 拉取sandbox容器
          - SecurityContext安全上下文
          - 初始化环境
          - 创建容器
          - 创建checkpoint
            - checkpoint主要是将当前sandbox的配置信息进行序列化，并且存储其当前的快照信息
          - 启动sandbox容器
          - CRI 通过CNI Client 调用 CNI(容器网络接口) 配置容器网络；
      - CRI被 Kubelet ImageManager调用，从而继续调用ImageService下载镜像；
        - *获取镜像*
      - CRI 调用运行时服务RuntimeService（例如Docker Runtime、containerd、cri-o、rkt）调用容器服务
        - 启动Pod相关容器
          - 如果对于的容器创建失败，则会通过backoff机制来延缓容器的创建
          - 临时容器（EphemeralContainer）
            - 参考下面业务容器创建过程
          - 初始化容器（initContainer）
            - 参考下面业务容器创建过程，主要用于在主容器启动之前执行一些初始化任务，确保主容器启动时所需的条件得到满足。
          - 业务容器（MainContainer）
            - 检查容器镜像是否拉取
              - 镜像的拉取首先会进行对应容器镜像的拼接，然后将之前获取的拉取的秘钥信息和镜像信息，一起交给CRI运行时来进行底层容器镜像的拉取，当然这里也会各种backoff机制，从而避免频繁拉取失败影响kubelet的性能
            - 创建容器配置
              - Pod的主机名、域名、挂载的volume、挂载的设备信息、configMap、secret、环境变量、要挂载的目录信息、端口映射信息、日志目录信息、根据环境生成执行的命令等信息
            - 创建容器
            - 启动容器
            - 执行容器的回调钩子
              - PostStart
                - 在容器启动后立即执行。这个钩子用于在容器启动后执行一些初始化或配置任务。以确保容器启动后的稳定性和可用性。
                  - 数据库初始化和迁移
                  - 缓存预热
                  - 执行一些定时任务
              - PreStop
                - 在容器终止之前执行。这个钩子用于在容器终止之前执行一些清理或收尾任务。确保容器能够优雅地关闭。例如，可以在容器终止之前执行某些资源的释放或状态的保存。
                  - 优雅终止，如服务注销
                  - 资源释放：如链接释放
                  - 写入日志



    - kubelet使用probManager模块创建探针
      - liveness
        - 用于确定容器是否仍然处于运行状态。失败将认为容器不再处于正常状态，并会尝试重新启动容器。
      - readiness
        - 用于确定容器是否准备好接收流量。失败，Kubernetes 将暂时从服务负载均衡中剔除该容器，直到探测成功。
      - startup
        - 确保容器已经启动并可以正常工作，以防止将流量引导到尚未准备好的容器。
  - 为对应的Pod根据其QoS等级来进行Cgroup配置的更新
    - kubelet 会在 node 上创建了 4 个 cgroup 层级
      - *kubelet管理*
        - Node 级别
        - QoS 级别
          - Guaranteed > Burstable > BestEffort
        - Pod 级别
      - *container runtime 管理*
        - Container 级别
          - 容器运行时管理
  - 这个步骤与上面是并行的：Kubelet 通过 Volume Manager，调用 CSI，将盘挂载到 Node 同时挂载到 Pod；
    - *获取密钥信息*
  - kubelet为pod创建其他的运行环境条件
    - 镜像秘钥的拉取
    - 创建 Pod 运行所需要的Pod、插件、volume目录
      - *等待Volume的完成*
    - 等待volume挂载完成