Kubernetes 源码分析

文档信息

文档编写人	姬文刚	编写日期	2018-01-18
文档评审人		评审日期	

版本修订历史记录

版本号	作者	参与者	起止日期	备注
V1.0				

内容修订历史记录

章节	修改内容	修订人	修订日期	修订原因

apiserver 组件源码分析

版本： **v1.9.0**

先走证书认证、再走授权、再走准入控制（插件方式实现，根据配置的插件进行）

1、初始化配置参数

（1）、启动http的默认参数

（2）、etcd默认配置

（3）、https Server的参数配置

（4）、非安全的方式8080端口启动的默认配置

（5）、审计日志

（6）、新的默认功能是否开放，集中放置

（7）、准入控制插件参数（初始化时候默认always，默认全部允许）

（9）、认证默认参数，包括证书认证、tocken认证、bootstrap认证、用户密码认证等等（初始化时候是链式调用，链式调用逆向执行）

（10）、授权默认配置

（11）、kube-apiserver自身参数，如是否允许最大权限运行、最大master数量、serviceip范围、servicenodeport范围等

（12）、接受cmd参数传入

2、执行 run 将所有的事情加载到 httpServer 当中

一、CreateKubeAPIServerConfig：

（1）、构建apserver启动的参数对象-defaultOptions。包括准入插件全部加载、默认参数二次设置、检测证书若无则自签证书，serviceaccount的认证，etcd初始化

（2）、参数构建完成进行拦截检测-Validate。并不做全部检测，核心功能包括etcd是否可用、culsterip是否指定、servicenodeport范围是否指定、证书验证、认证验证、非安全服务验证、master最大数验证这几项。

（3）、真正构建apiserver启动所需要的对象数据-BuildGenericConfig。其中有几项重要构建认证链BuildAuthenticator，构建授权链BuildAuthorizer。

二、createAPIExtensionsConfig、createAPIExtensionsServer：

扩展server参数及服务构建，提供三方服务使用。

三、重要函数CreateKubeAPIServer

（1）、Complete。设置serviceiprange，apiserverip等默认参数、设置服务发现默认地址、设置监控参数等。相对简单，主要赋值。

（2）、New。核心复杂方法，最终得到kube-apiserver对象：

A： 构建GenericAPIServer ：

BuildHandlerChainFunc 构建所有apiserver中的handler链条，链条方式（其中构建每个hander都要经过链条处理作为下一个hander的前提参数进行构建下一hander，最后一步授权，通过授权构建起handler HTTP服务，从下网上逆序，这其中构建了授权、认证、审计、跨域请求、panic、永久执行方法等）；

NewAPIServerHandler 根据上面构建的链条进行httpHander真正创建。go-restful框架，构建container（后续将所有pod、service等webservice注册到gorestfulcontainer中）这里一共有几个container需要进一步明确。目前知道goresetfulcontainer是一个，为主要webservice对象的container，listedcontainer为curl构建的container等。

installAPI：安装部分资源的api。

B：InstallAPIs （未分析）

（3）、PrepareRun。启动HTTP服务。包括openapi、健康检查的一些配置进行完整的run。

A：NonBlockingRun非阻塞方式-serveSecurely-RunServer

B：最终 ServerHTTP 里会运行 director 和授权认证 handler。 director 包含 gorestfulcontainer，主要的 webservice-api 都在这里。nogorestfulecontainer 监控请求 api 在这里。

Kube-controller-manager 组件源码分析

源码主要的步骤包括：

创建CMServer对象 -进行AddFlags处理 -日志初始化 -执行主Run函数，启动程序，在创建CMServer时会对Controller-Manager的所有参数赋默认值。在AddFlags参数处理时，会根据程序启动参数对参数赋上实际值。

1、初始化-GetDefaultControllerOptions

略

2、启动-run

一、 基本检查和配置处理

A：-Validate 对设置的controller进行名称检测

B：-configz.New创建对应的配置对象，并进行配置处理

二、启动监控线程 startHTTP

启动监控server，这些都是系统监控的api，如/metrics可以记录api调用次数

三、Controllers 启动-run := func(stop <-chan struct{})核心关键方法

A：CreateControllerContext

B：serviceAccountTokenControllerStarter 启动tockenController，作为其他controller的启动参数

C：StartControllers-核心方法

（1）、NewControllerInitializers 。加载30多个controller，结构为MAP，key为controller的名称，value为启动方法。

（2）、core.go 29个controller的各自启动方法：

startEndpointController 启动endpoint

startReplicationController 启动rc

startPodGCController 启动podgc

  startResourceQuotaController 启动resourcequota

  startNamespaceController 启动namespace

  startServiceAccountController 启动serviceaccount

  startGarbageCollectorController 未知

  startDaemonSetController 启动deamonset

  startJobController 启动job

  startDeploymentController 启动deployment

  startReplicaSetController 启动replicaset

  startHPAController 启动hpa

  startDisruptionController 未知

  startStatefulSetController 未知

  startCronJobController 启动cronjob

  startCSRSigningController 启动csr

  startCSRApprovingController 启动csrapprove

  startCSRCleanerController 启动csrcleanr

  startTTLController 启动ttl

  startBootstrapSignerController 启动bootstrapsingner

  startTokenCleanerController 启动tocken

  startServiceController 启动service

  startNodeController 启动node

  startRouteController 启动route

  startPersistentVolumeBinderController 启动volume

  startAttachDetachController 未知

  startVolumeExpandController 未知

  startClusterRoleAggregrationController 启动clusterrole

  startPVCProtectionController 启动pvc

以上方法风格一致，进入对应的start方法后，执行相应的run方法，run方法里包括WaitForCacheSync 同步缓存，同步缓存后以之前配置的worker数启动多个协程。协程做什么工作通过worker方法执行。

deploymentcontroller启动：

（1）、执行processNextWorkItem进行同步，调用synchander方法，其实是调用syncDeployment方法。

A：SplitMetaNamespaceKey 通过传入的key 获取到namespaces

B：dc.dLister.Deployments 根据上面获取的namespace 和name 获取对应的deplyment对象

C：getReplicaSetsForDeployment 获取所有的replicaset对象

D：getPodMapForDeployment 根据之前获取到的replicatlist 获取所有deployment对应的pod的map

E：syncStatusOnly 判断deployment的条件，增删改查syncStatusOnly下的 getAllReplicaSetsAndSyncRevision 关键方法，版本同步。

getNewReplicaSet 创建deploymentcontroller 还有一些同步deployment状态的操作，整体工作结束。

controller在启动过程中，一般是先创建一个对象，然后调用对象的Run函数进行启动。启动过程会先等待Api Object 都同步完成。再启动woker协程，在woker协程中进行实际的处理

（2）、rccontroller的启动及调用

A： rc 和deployment类似，具体工作内容也在到对应的worker中。syncReplicaSet

B：SplitMetaNamespaceKey 通过传入的key 获取到namespaces

C：rsc.rsLister.ReplicaSets 根据上面获取的namespace 和name 获取对应的ReplicaSets对象

D：rsc.podLister.Pods 获取所有的pod对象 IsPodActive运行过滤筛选，只筛选运行中的pod

E：claimPods RecheckDeletionTimestamp 检查pod是否需要进行数量的同步。

F：calculateStatus 获取replicaset状态，更新状态。

G：manageReplicas 核心方法， diff := len(filteredPods) - int(*(rs.Spec.Replicas)) 比较，根据比较结果创建删除pod。 减法<0 CreatePodsWithControllerRef 创建pod ，createPods 创建pod的具体实现，调用apiserver。 减法>0 getPodsToDelete 删除pod 调用apiserver后将rc、pod对象持久化到etcd，接着scheduler watch调度。

四、进行选主处理

leaderelection.RunOrDie 是通用的选主的机制，他们启动的时候都会抢先注册自己为 leader，当然只有一个会成功，其它的节点 watch，如果 leader 挂了后其它节点会成为 leader 启动自己 CM

附：Kubernetes 版本发布历史

kube-scheduler 源码分析

scheduler 运行机制

scheduler 负责安排 Pod 到具体的Node，通过 API Server 提供的接口监听 Pods，获取待调度 pod，根据一系列的预选策略和优选策略给各个 Node 节点打分排序，然后将 Pod 调度到得分最高的 Node 节点上，然后由 kubelet 负责创建 Pod。Kubernetes 调度分为 Predicate（预选）和 Priority（优选），分为两个过程：

预选：遍历所有 Node，按照预选筛选出符合要求的 Node，如果没有 Node 符合 Predicate 策略，那该 Pod 就会被挂起，直到有 Node 能够满足所有策略；

优选，在第一步基础上，按照优选为待选 Node 打分，获取最高分；

scheduler 的ConfigFactory中定义了podQueue用来存储需要被调度的pod，每当新的pod建立后，就会将pod添加到该queue中。

scheduleOne会从queue中选择需要调度的下一个pod进行详细的调度操作。

选择好调度节点后会将该pod bind 到选出的node节点上。

一. 预选算法－predicate

路径 plugin/pkg/scheduler/algorithm/predicates/predicates.go，选出符合预选策略的 node 列表

PodFitsHostPorts: 检查节点的端口可以分配给 pod 使用

PodFitsResources: 检查节点是否有足够的资源，CPU，内存，GPU，storage

NoDiskConflict: 检查节点卷是否冲突，是否已经被挂载，如果已经被其他使用则不能调度到该节点，可以看具体解释，包括 GCE，AWS，Ceph，ISCSI 说明

PodMatchNodeSelector: 检查 pod 的选择器是否匹配到节点 label

CheckNodeLabelPresence:

InterPodAffinityMatches:

CheckNodeMemoryPressurePredicate

CheckNodeConditionPredicate

二. 优选算法－predicate

路径 plugin/pkg/scheduler/algorithm/priorities/priorities.go，结果为一个二维数组，含有节点与各项 优选算法的分数（0-10 之间），其他算法就不分析了

BalancedResourceAllocationMap: 资源使用均衡分数最高，计算公式 score = 10 - abs(cpuFraction-memoryFraction)*10

启动分析：

主函数首先构建并初始化schedulerserver结构体，调度可以分成两个部分，第一部分是调度算法部分，第二部分是调度执行部分。

genericScheduler 调度执行部分的数据结构 FitPredicate函数用来在Node中判断POD请求的资源是否满足需要。

PriorityConfig结构体包括Function函数和Weight整型变量，Function函数用来表示预测算法，Weight变量用来表示预测算法权重。

genericScheduler中Schedule是负责调度的函数，里面用到了findNodesThatFit和PrioritizeNodes两个函数。

初始化-schedulerserver

1、新建schedulerserver结构体

2、传入参数并初始化

3、run

（1）通过kubeconfig创建kubeclient用于访问APIserver

（2）创建htttpserver用于性能分析，性能指标度量

（3）创建scheduler.config对象（定期获取已经调度的pod列表，并从modeler假定队列中删除。监控可用的node，获取service列表、创建genericScheduler对象）

（4）run不停执行scheduleOne（获取下一个待调度的pod，具体由genericScheduler调度）kubernetes调度通过结构体genericScheduler最后选取出了一个Node，但是在这个node上面创建pod的工作并没有做。

（5）Binding，这个结构体的作用是建立将一个对象绑定到另一个对象的关系，比如在调度过程中将POD绑定到一个Node上，在这个结构体中就是将ObjectMeta（POD）绑定到ObjectReference（Node）上。在建立好POD同Node的绑定关系后，执行bindAction函数

（6）把调度结果放入假定调度队列中。

启动 server.go **Run**

1、NewSchedulerServer 初始化参数 略

2、server.Run

（1）、NewFromConfig 注册scheduler

（2）、StartRecordingToSink 待

（3）、HealthzServer.ListenAndServe 启动监听server的健康状态

（4）、MetricsServer.ListenAndServe() 启动性能监控

（5）、go s.PodInformer.Informer().Run(stop) 启动所有的infomer

（6）、InformerFactory.WaitForCacheSync 调度钱等待所有的内存同步

（7）、核心方法。sched.Run() 预选启动

A：WaitForCacheSync 校验方法， 等待同步，如果没有同步的情况直接退出。

B：go sched.config.VolumeBinder.Run 待

C：核心方法 go wait.Until(sched.scheduleOne 正式调度

（A）： 核心方法 sched.schedule(pod) 调度。

kube-scheduler的调度分为predicate和priority两类算法，predicate用于将不符合pod调度条件的node剔除出待调度node列表；而priority从剩下的node中

应用多个算法将每个node按0-10评分。这样得分最高的node具有最高优先级。

1）、正常

sched.config.Algorithm.Schedule 给pod找到一个适合的node 如果成功，则返回nodename，如果失败则返回并给出原因。

a）：

podPassesBasicChecks PVC 检查

b）：

UpdateNodeNameToInfoMap

c）：核心方法

findNodesThatFit 使用注册的所有predicate算法进行过滤，留下适合待调度pod的node列表以及不适合调度node原因。

（a）：

核心方法 podFitsOnNode 在顺序调用每个predicate算法之前，会首先查看EquivalenceCache是否有记录，其作用是相同controller生成的pod只会用第一个pod的结果，从而

优化了调度执行时间。 如果用户没有指定AlgorithmProvider使用的默认的DefaultProvider，每个predicate算法都会传入三个参数。

d）：核心方法

PrioritizeNodes 使用注册的priority算法对之前过滤后的每个node进行打分

e）：核心方法

selectHost 最终g.selectHost选出得分最高的node（如果多个node得分相同且都是最高，为了调度更加平均g.selectHost进行了优化）。

2）、错误

PodConditionUpdater.Update 待

（B）：SchedulingAlgorithmLatency.Observe 待

（C）：sched.preempt 待

（D）：sched.assumeAndBindVolumes 将调度的Pod信息以及对应的Node信息，写入到SchedulerCache.AssumePod中

（E）：ched.assume assume()主要做了两个事情，首先调用SchedulerCache的AssumePod()，将pod的最新调度结果写入SchedulerCache缓存，并更新schedulerCache中对应node的资源占用和hostPort端口占用，这样的好处是不用等待pod调度结果写入apiserver或者等待pod启动才开始调度下一个pod，可以增加调度效率。当然如果结果写入apiserver失败会清理刚才在缓存中的pod。然后使equivalencePodCache中的GeneralPredicates算法对应的缓存失效。因为GeneralPredicates算法包含了对node资源和端口占用的检查，当有新的pod调度在node上这两项资源都会得到更新，所以原始缓存必须要失效。

（F）：sched.bind bind()是异步进行的，即assume()调用完成就会进入下一轮新的调度循环。sched.config.Binder.Bind(b)将Binding这个对象写入到kube-apiserver中。如果写入发生错误，首先SchedulerCache.ForgetPod(assumed)将之前assume()加入到缓存中的pod删除，清除对应node资源占用。sched.config.Error(assumed, err)跟schedule()的错误处理相同，将失败的pod等一个间隔时间后重新放入未调度pod队列中。生成Event事件和更新pod状态与之前类似。而sched.config.Binder.Bind(b)成功写入kube-apiserver后，会调用SchedulerCache.FinishBinding(assumed)，其主要作用是给SchedulerCache中的那些通过assume的pod加上过期时间ttl，从而可以定时清除他们释放内存占用，而之后调度的时候也并不会完全依赖这些SchedulerCache中assume的pod，因为当pod成功在node上创建后，Scheduler.Config中的NodeLister会定时同步全部node节点状态和资源占用，这些信息才是最准确的。至此，一个完整的调度循环就结束了

（8）、leaderElector.Run() leader选举

整体上Schedule模块，对应Pod的处理，基于list-watch机制，获取哪写Pod需要调度。然

后将信息存入到queuePods缓存队列中，然后在ScheduleOne函数中，将Pod信息取出，利用调度算法进行调度。

完成调度后，将Pod的信息存入ScheduleCache的assumePods队列中再次缓存

再通过list-watch机制监听已经被调度的Pod的信息，放入到Queue队列中，在Controller的协程中，同步的更新assumePods队列中的数据进行对账处理

2015 年 7 月 21 日 Kubernetes 1.0 发布

2016 年 3 月 17 日 Kubernetes 1.2 发布

2016 年 7 月 1 日 Kubernetes 1.3 发布

2016 年 9 月 20 日 Kubernetes 1.4 发布

2016 年 12 月 22 日 Kubernetes 1.5 发布

2017 年 3 月 22 日 Kubernetes 1.6 发布

2017 年 6 月 28 日 Kubernetes 1.7 发布

2017 年 9 月 27 日 Kubernetes 1.8 发布

2017 年 12 月 13 日 k8s v1.9.0 正式版本发布