



下载APP



19 | Kubernetes基础应用：创建一个Pod背后etcd发生了什么？

2021-03-03 唐聪

etcd实战课

进入课程 >



讲述：王超凡

时长 20:45 大小 19.01M



你好，我是唐聪。

今天我将通过在 Kubernetes 集群中创建一个 Pod 的案例，为你分析 etcd 在其中发挥的作用，带你深入了解 Kubernetes 是如何使用 etcd 的。

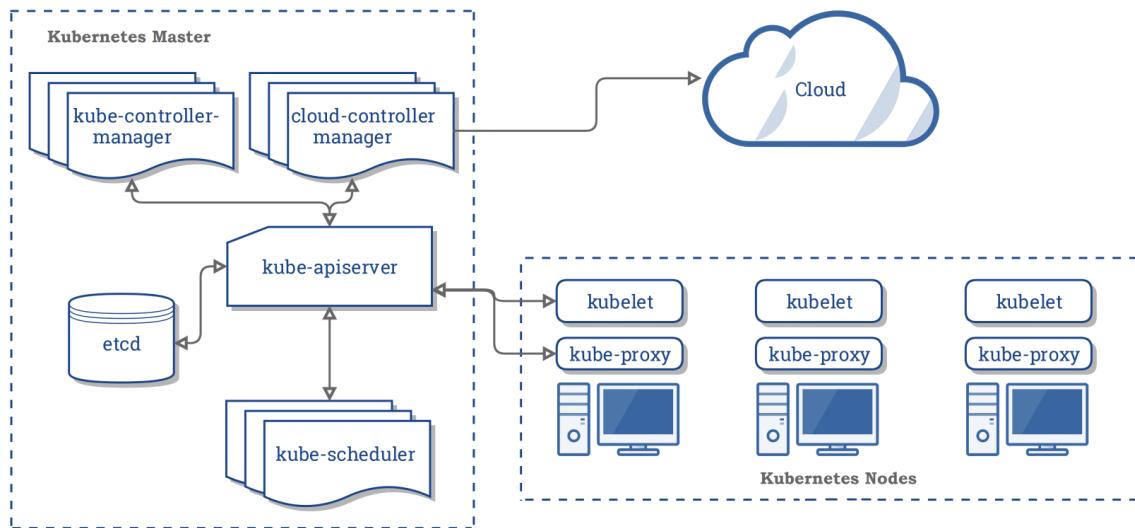
希望通过本节课，帮助你从 etcd 的角度更深入理解 Kubernetes，让你知道在 Kubernetes 集群中每一步操作的背后，etcd 会发生什么。更进一步，当你在 Kubernetes 集群中遇到 etcd 相关错误的时候，能从 etcd 角度理解错误含义，高效进行故障诊断。



Kubernetes 基础架构

在带你详细了解 etcd 在 Kubernetes 里的应用之前，我先和你简单介绍下 Kubernetes 集群的整体架构，帮你搞清楚 etcd 在 Kubernetes 集群中扮演的角色与作用。

下图是 Kubernetes 集群的架构图（[引用自 Kubernetes 官方文档](#)），从图中你可以看到，它由 Master 节点和 Node 节点组成。



控制面 Master 节点主要包含以下组件：

kube-apiserver，负责对外提供集群各类资源的增删改查及 Watch 接口，它是 Kubernetes 集群中各组件数据交互和通信的枢纽。kube-apiserver 在设计上可水平扩展，高可用 Kubernetes 集群中一般多副本部署。当收到一个创建 Pod 写请求时，它的基本流程是对请求进行认证、限速、授权、准入机制等检查后，写入到 etcd 即可。

kube-scheduler 是调度器组件，负责集群 Pod 的调度。基本原理是通过监听 kube-apiserver 获取待调度的 Pod，然后基于一系列筛选和评优算法，为 Pod 分配最佳的 Node 节点。

kube-controller-manager 包含一系列的控制器组件，比如 Deployment、StatefulSet 等控制器。控制器的核心思想是监听、比较资源实际状态与期望状态是否一致，若不一致则进行协调工作使其最终一致。

etcd 组件，Kubernetes 的元数据存储。

Node 节点主要包含以下组件：

kubelet，部署在每个节点上的 Agent 的组件，负责 Pod 的创建运行。基本原理是通过监听 APIServer 获取分配到其节点上的 Pod，然后根据 Pod 的规格详情，调用运行时组件创建 pause 和业务容器等。

kube-proxy，部署在每个节点上的网络代理组件。基本原理是通过监听 APIServer 获取 Service、Endpoint 等资源，基于 Iptables、IPVS 等技术实现数据包转发等功能。

从 Kubernetes 基础架构介绍中你可以看到，kube-apiserver 是唯一直接与 etcd 打交道的组件，各组件都通过 kube-apiserver 实现数据交互，它们极度依赖 kube-apiserver 提供的资源变化**监听机制**。而 kube-apiserver 对外提供的监听机制，也正是由我们基础篇 [08](#) 中介绍的 etcd **Watch 特性**提供的底层支持。

创建 Pod 案例

接下来我们就以在 Kubernetes 集群中创建一个 nginx 服务为例，通过这个案例来详细分析 etcd 在 Kubernetes 集群创建 Pod 背后是如何工作的。

下面是创建一个 nginx 服务的 YAML 文件，Workload 是 Deployment，期望的副本数是 1。

 复制代码

```
1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: nginx-deployment
5   labels:
6     app: nginx
7 spec:
8   replicas: 1
9   selector:
10    matchLabels:
11      app: nginx
12   template:
13     metadata:
14       labels:
15         app: nginx
16     spec:
17       containers:
18         - name: nginx
19           image: nginx:1.14.2
20         ports:
21           - containerPort: 80
```

假设此 YAML 文件名为 nginx.yaml，首先我们通过如下的 kubectl create -f nginx.yaml 命令创建 Deployment 资源。

 复制代码

```
1 $ kubectl create -f nginx.yaml
2 deployment.apps/nginx-deployment created
```

创建之后，我们立刻通过如下命令，带标签查询 Pod，输出如下：

 复制代码

```
1 $ kubectl get po -l app=nginx
2 NAME                               READY   STATUS    RESTARTS   AGE
3 nginx-deployment-756d9fd5f9-fkqnf  1/1     Running   0          8s
```

那么在 kubectl create 命令发出，nginx Deployment 资源成功创建的背后，kube-apiserver 是如何与 etcd 打交道的呢？它是通过什么接口**安全写入**资源到 etcd 的？

同时，使用 kubectl 带标签查询 Pod 背后，kube-apiserver 是直接从**缓存读取**还是向 etcd 发出一个**线性读或串行读**请求呢？若同 namespace 下存在大量的 Pod，此操作性能又是怎样的呢？

接下来我就和你聊聊 kube-apiserver 收到创建和查询请求后，是如何与 etcd 交互的。

kube-apiserver 请求执行链路

kube-apiserver 作为 Kubernetes 集群交互的枢纽、对外提供 API 供用户访问的组件，因此保障集群安全、保障本身及后端 etcd 的稳定性的等重任也是非它莫属。比如校验创建请求发起者是否合法、是否有权限操作相关资源、是否出现 Bug 产生大量写和读请求等。

下图是 kube-apiserver 的请求执行链路（引用自 sttts 分享的 PDF），当收到一个请求后，它主要经过以下处理链路来完成以上若干职责后，才能与 etcd 交互。

核心链路如下：

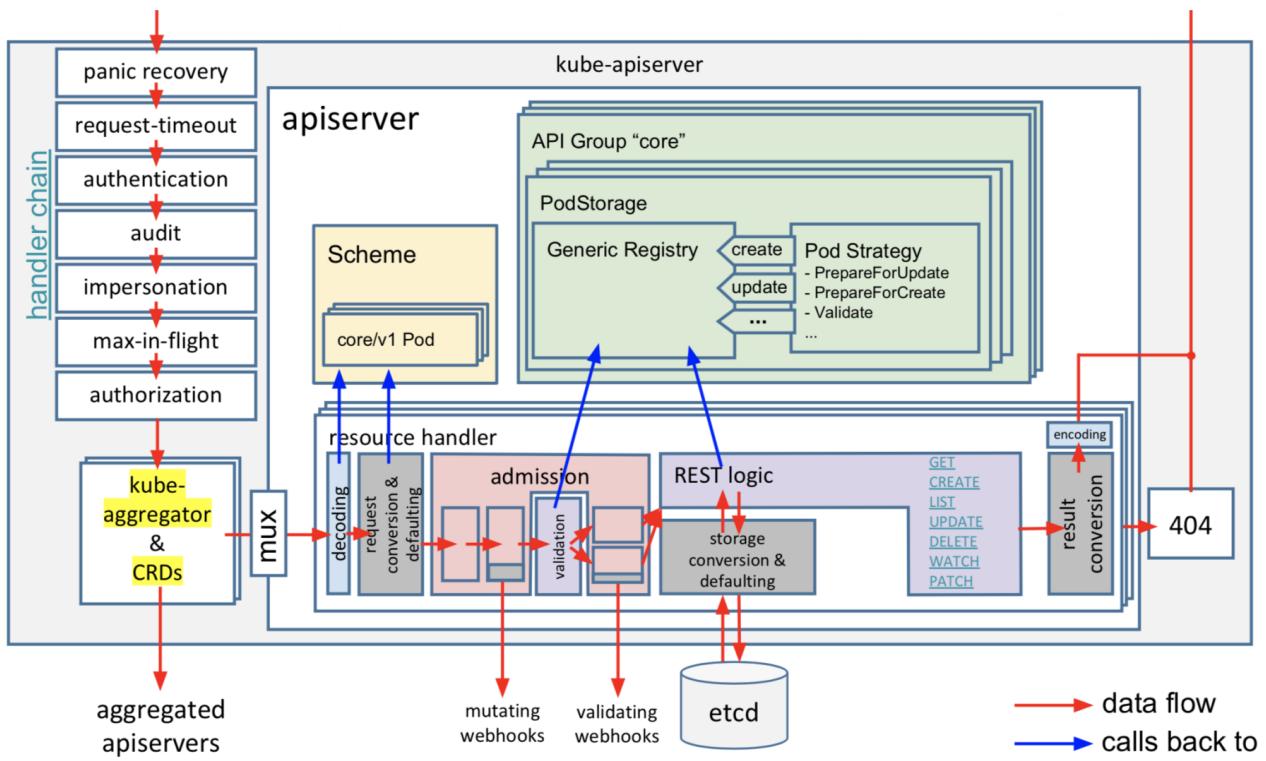
认证模块，校验发起的用户身份是否合法。支持多种方式，比如 x509 客户端证书认证、静态 token 认证、webhook 认证等。

限速模块，对请求进行简单的限速，默认读 400/s 写 200/s，不支持根据请求类型进行分类、按优先级限速，存在较多问题。Kubernetes 1.19 后已新增 Priority and Fairness 特性取代它，它支持将请求重要程度分类进行限速，支持多租户，可有效保障 Leader 选举之类的高优先级请求得到及时响应，能防止一个异常 client 导致整个集群被限速。

审计模块，可记录用户对资源的详细操作行为。

授权模块，检查用户是否有权限对其访问的资源进行相关操作。支持多种方式，RBAC(Role-based access control)、ABAC(Attribute-based access control)、Webhook 等。Kubernetes 1.12 版本后，默认授权机制使用的 RBAC。

准入控制模块，提供在访问资源前拦截请求的静态和动态扩展能力，比如要求镜像的拉取策略始终为 AlwaysPullImages。



经过上面一系列的模块检查后，这时 kube-apiserver 就开始与 etcd 打交道了。在了解 kube-apiserver 如何将我们创建的 Deployment 资源写入到 etcd 前，我先和你介绍下 Kubernetes 的资源是如何组织、存储在 etcd 中。

Kubernetes 资源存储格式

我们知道 etcd 仅仅是个 key-value 存储，但是在 Kubernetes 中存在各种各样的资源，并提供了以下几种灵活的资源查询方式：

按具体资源名称查询，比如 PodName、kubectl get po/PodName。

按 namespace 查询，获取一个 namespace 下的所有 Pod，比如 kubectl get po -n kube-system。

按标签名，标签是极度灵活的一种方式，你可以为你的 Kubernetes 资源打上各种各样的标签，比如上面案例中的 kubectl get po -l app=nginx。

你知道以上这几种查询方式它们的性能优劣吗？假设你是 Kubernetes 开发者，你会如何设计存储格式来满足以上功能点？

首先是按具体资源名称查询。它本质就是个 key-value 查询，只需要写入 etcd 的 key 名称与资源 key 一致即可。

其次是按 namespace 查询。这种查询也并不难。因为我们知道 etcd 支持范围查询，若 key 名称前缀包含 namespace、资源类型，查询的时候指定 namespace 和资源类型的组合的最小开始区间、最大结束区间即可。

最后是标签名查询。这种查询方式非常灵活，业务可随时添加、删除标签，各种标签可相互组合。实现标签查询的办法主要有以下两种：

方案一，在 etcd 中存储标签数据，实现通过标签可快速定位（时间复杂度 $O(1)$ ）到具体资源名称。然而一个标签可能容易实现，但是在 Kubernetes 集群中，它支持按各个标签组合查询，各个标签组合后的数量相当庞大。在 etcd 中维护各种标签组合对应的资源列表，会显著增加 kube-apiserver 的实现复杂度，导致更频繁的 etcd 写入。

方案二，在 etcd 中不存储标签数据，而是由 kube-apiserver 通过范围遍历 etcd 获取原始数据，然后基于用户指定标签，来筛选符合条件的资源返回给 client。此方案优点是实现简单，但是大量标签查询可能会导致 etcd 大流量等异常情况发生。

那么 Kubernetes 集群选择的是哪种实现方式呢？

下面是一个 Kubernetes 集群中的 coredns 一系列资源在 etcd 中的存储格式：

 复制代码

```

1 /registry/clusterrolebindings/system:coredns
2 /registry/clusterroles/system:coredns
3 /registry/configmaps/kube-system:coredns
4 /registry/deployments/kube-system:coredns
5 /registry/events/kube-system:coredns-7fcc6d65dc-6njlg.1662c287aabf742b
6 /registry/events/kube-system:coredns-7fcc6d65dc-6njlg.1662c288232143ae
7 /registry/pods/kube-system:coredns-7fcc6d65dc-jvj26
8 /registry/pods/kube-system:coredns-7fcc6d65dc-mgvb
9 /registry/pods/kube-system:coredns-7fcc6d65dc-whzq9
10 /registry/replicasets/kube-system:coredns-7fcc6d65dc
11 /registry/secrets/kube-system:coredns-token-hpqbt
12 /registry/serviceaccounts/kube-system:coredns

```

从中你可以看到，一方面 Kubernetes 资源在 etcd 中的存储格式由 prefix + "/" + 资源类型 + "/" + namespace + "/" + 具体资源名组成，基于 etcd 提供的范围查询能力，非常简单地支持了按具体资源名称查询和 namespace 查询。

kube-apiserver 提供了如下参数给你配置 etcd prefix，并支持将资源存储在多个 etcd 集群。

 复制代码

```

1 --etcd-prefix string      Default: "/registry"
2 The prefix to prepend to all resource paths in etcd.
3 --etcd-servers stringSlice
4 List of etcd servers to connect with (scheme://ip:port), comma separated.
5 --etcd-servers-overrides stringSlice
6 Per-resource etcd servers overrides, comma separated. The individual override
7 semicolon separated.

```

另一方面，我们未看到任何标签相关的 key。Kubernetes 实现标签查询的方式显然是方案二，即由 kube-apiserver 通过范围遍历 etcd 获取原始数据，然后基于用户指定标签，来筛选符合条件的资源返回给 client（资源 key 的 value 中记录了资源 YAML 文件内容等，如标签）。

也就是当你执行"kubectl get po -l app=nginx"命令，按标签查询 Pod 时，它会向 etcd 发起一个范围遍历整个 default namespace 下的 Pod 操作。

 复制代码

```

1  $ kubectl get po -l app=nginx -v 8
2  I0301 23:45:25.597465 32411 loader.go:359] Config loaded from file /root/.ku
3  I0301 23:45:25.603182 32411 round_trippers.go:416] GET https://ip:port/api/v
4  labelSelector=app%3Dnginx&limit=500

```

etcd 收到的请求日志如下，由此可见当一个 namespace 存在大量 Pod 等资源时，若频繁通过 kubectl，使用标签查询 Pod 等资源，后端 etcd 将出现较大的压力。

 复制代码

```

1  {
2      "level": "debug",
3      "ts": "2021-03-01T23:45:25.609+0800",
4      "caller": "v3rpc/interceptor.go:181",
5      "msg": "request stats",
6      "start time": "2021-03-01T23:45:25.608+0800",
7      "time spent": "1.414135ms",
8      "remote": "127.0.0.1:44664",
9      "response type": "/etcdserverpb.KV/Range",
10     "request count": 0,
11     "request size": 61,
12     "response count": 11,
13     "response size": 81478,
14     "request content": "key: /registry/pods/default/ range_end: /registry/pods
15 }

```

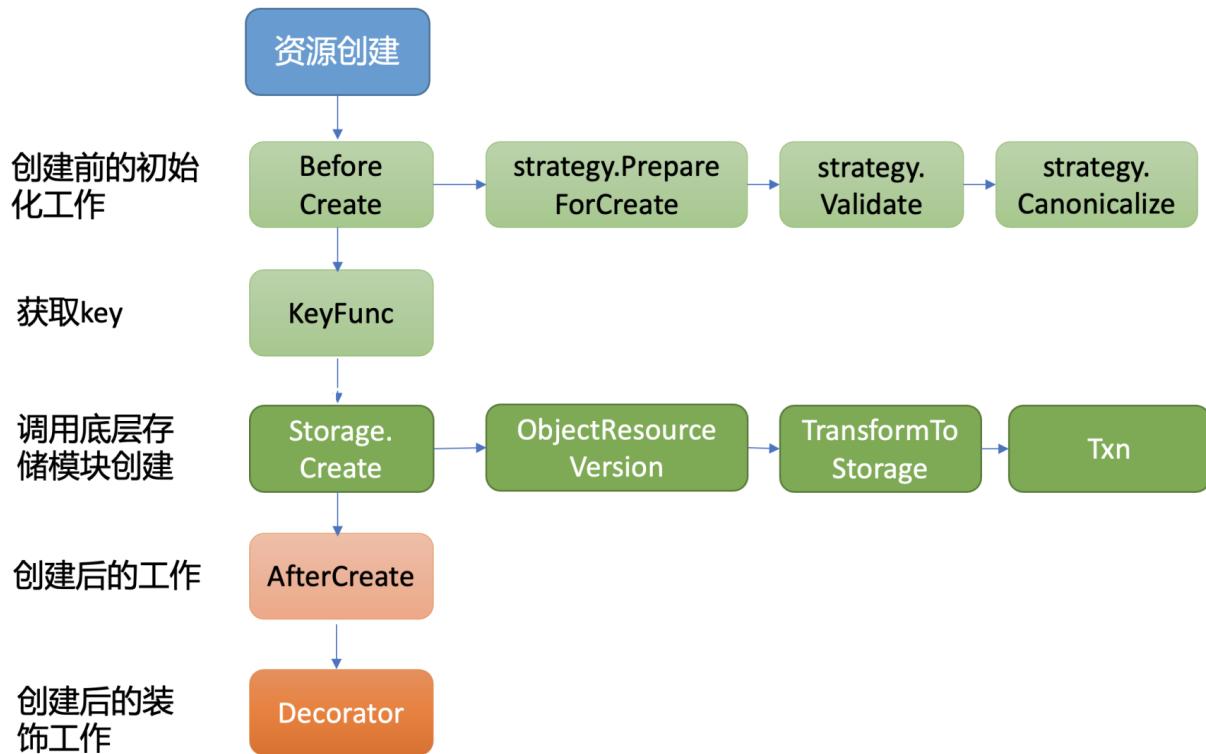
了解完 Kubernetes 资源的存储格式后，我们再看看 nginx Deployment 资源是如何由 kube-apiserver 写入 etcd 的。

通用存储模块

kube-apiserver 启动的时候，会将每个资源的 APIGroup、Version、Resource Handler 注册到路由上。当请求经过认证、限速、授权、准入控制模块检查后，请求就会被转发到对应的资源逻辑进行处理。

同时，kube-apiserver 实现了类似数据库 ORM 机制的通用资源存储机制，提供了对一个资源创建、更新、删除前后的 hook 能力，将其封装成策略接口。当你新增一个资源时，你只需要编写相应的创建、更新、删除等策略即可，不需要写任何 etcd 的 API。

下面是 kube-apiserver 通用存储模块的创建流程图：



当收到创建 nginx Deployment 请求后，通用存储模块首先会回调各个资源自定义实现的 BeforeCreate 策略，为资源写入 etcd 做一些初始化工作。

下面是 Deployment 资源的创建策略实现，它会进行将 deployment.Generation 设置为 1 等操作。

复制代码

```

1 // PrepareForCreate clears fields that are not allowed to be set by end users
2 func (deploymentStrategy) PrepareForCreate(ctx context.Context, obj runtime.Object)
3     deployment := obj.(*apps.Deployment)
4     deployment.Status = apps.DeploymentStatus{}
5     deployment.Generation = 1
6
7     pod.DropDisabledTemplateFields(&deployment.Spec.Template, nil)
8 }

```

执行完 BeforeCreate 策略后，它就会执行 Storage.Create 接口，也就是由它真正开始调用底层存储模块 etcd3，将 nginx Deployment 资源对象写入 etcd。

那么 Kubernetes 是使用 etcd Put 接口写入资源 key-value 的吗？如果是，那要如何防止同名资源并发创建被覆盖的问题？

资源安全创建及更新

我们知道 etcd 提供了 Put 和 Txn 接口给业务添加 key-value 数据，但是 Put 接口在并发场景下若收到 key 相同的资源创建，就会导致被覆盖。

因此 Kubernetes 很显然无法直接通过 etcd Put 接口来写入数据。

而我们 [09](#) 节中介绍的 etcd 事务接口 Txn，它正是为了多 key 原子更新、并发操作安全性等而诞生的，它提供了丰富的冲突检查机制。

Kubernetes 集群使用的正是事务 Txn 接口来防止并发创建、更新被覆盖等问题。当执行完 BeforeCreate 策略后，这时 kube-apiserver 就会调用 Storage 的模块的 Create 接口写入资源。1.6 版本后的 Kuberne 集群默认使用的存储是 etcd3，它的创建接口简要实现如下：

 复制代码

```

1 // Create implements storage.Interface.Create.
2 func (s *store) Create(ctx context.Context, key string, obj, out runtime.Object
3     .....
4     key = path.Join(s.pathPrefix, key)
5
6     opts, err := s.ttl0pts(ctx, int64(ttl))
7     if err != nil {
8         return err
9     }
10
11    newData, err := s.transformer.TransformToStorage(data, authenticatedDataStr
12    if err != nil {
13        return storage.NewInternalError(err.Error())
14    }
15
16    startTime := time.Now()
17    txnResp, err := s.client.KV.Txn(ctx).If(
18        notFound(key),
19    ).Then(
20        clientv3.OpPut(key, string(newData), opts...),
21    ).Commit

```

从上面的代码片段中，我们可以得出首先它会按照我们介绍的 Kubernetes 资源存储格式拼接 key。

然后若 TTL 非 0，它会根据 TTL 从 leaseManager 获取可复用的 Lease ID。Kubernetes 集群默认若不同 key (如 Kubernetes 的 Event 资源对象) 的 TTL 差异在 1 分钟内，可复用同一个 Lease ID，避免大量 Lease 影响 etcd 性能和稳定性。

其次若开启了数据加密，在写入 etcd 前数据还将按加密算法进行转换工作。

最后就是使用 etcd 的 Txn 接口，向 etcd 发起一个创建 deployment 资源的 Txn 请求。

那么 etcd 收到 kube-apiserver 的请求是长什么样子的呢？

下面是 etcd 收到创建 nginx deployment 资源的请求日志：

 复制代码

```

1  {
2      "level": "debug",
3      "ts": "2021-02-11T09:55:45.914+0800",
4      "caller": "v3rpc/interceptor.go:181",
5      "msg": "request stats",
6      "start time": "2021-02-11T09:55:45.911+0800",
7      "time spent": "2.697925ms",
8      "remote": "127.0.0.1:44822",
9      "response type": "/etcdserverpb.KV/Txn",
10     "request count": 1,
11     "request size": 479,
12     "response count": 0,
13     "response size": 44,
14     "request content": "compare:<target:MOD key:":"/registry/deployments/default/
15 }
```

从这个请求日志中，你可以得到以下信息：

请求的模块和接口，KV/Txn；

key 路径，/registry/deployments/default/nginx-deployment，由 prefix + "/" + 资源类型 + "/" + namespace + "/" + 具体资源名组成；

安全的并发创建检查机制，`mod_revision` 为 0 时，也就是此 key 不存在时，才允许执行 `put` 更新操作。

通过 `Txn` 接口成功将数据写入到 `etcd` 后，`kubectl create -f nginx.yml` 命令就执行完毕，返回给 `client` 了。在以上介绍中你可以看到，`kube-apiserver` 并没有任何逻辑去真正创建 `Pod`，但是为什么我们可以马上通过 `kubectl get` 命令查询到新建并成功运行的 `Pod` 呢？

这就涉及到了基础架构图中的控制器、调度器、`Kubelet` 等组件。下面我就为你浅析它们是如何基于 `etcd` 提供的 `Watch` 机制工作，最终实现创建 `Pod`、调度 `Pod`、运行 `Pod` 的。

Watch 机制在 Kubernetes 中应用

正如我们基础架构中所介绍的，`kube-controller-manager` 组件中包含一系列 `WorkLoad` 的控制器。`Deployment` 资源就由其中的 `Deployment` 控制器来负责的，那么它又是如何感知到新建 `Deployment` 资源，最终驱动 `ReplicaSet` 控制器创建出 `Pod` 的呢？

获取数据变化的方案，主要有轮询和推送两种方案组成。轮询会产生大量 `expensive request`，并且存在高延时。而 `etcd` `Watch` 机制提供的流式推送能力，赋予了 `kube-apiserver` 对外提供数据监听能力。

我们知道在 `etcd` 中版本号是个逻辑时钟，随着 `client` 对 `etcd` 的增、删、改操作而全局递增，它被广泛应用在 `MVCC`、事务、`Watch` 特性中。

尤其是在 `Watch` 特性中，版本号是数据增量同步的核心。当 `client` 因网络等异常出现连接闪断后，它就可以通过版本号从 `etcd server` 中快速获取异常后的事件，无需全量同步。

那么在 `Kubernetes` 集群中，它提供了什么概念来实现增量监听逻辑呢？

答案是 `Resource Version`。

Resource Version 与 etcd 版本号

Resource Version 是 Kubernetes API 中非常重要的一个概念，顾名思义，它是一个 Kubernetes 资源的内部版本字符串，client 可通过它来判断资源是否发生了变化。同时，你可以在 Get、List、Watch 接口中，通过指定 Resource Version 值来满足你对数据一致性、高性能等诉求。

那么 Resource Version 有哪些值呢？跟 etcd 版本号是什么关系？

下面我分别以 Get 和 Watch 接口中的 Resource Version 参数值为例，为你剖析它与 etcd 的关系。

在 Get 请求查询案例中，ResourceVersion 主要有以下这三种取值：

第一种是未指定 ResourceVersion，默认空字符串。kube-apiserver 收到一个此类型的读请求后，它会向 etcd 发出共识读 / 线性读请求获取 etcd 集群最新的数据。

第二种是设置 ResourceVersion="0"，赋值字符串 0。kube-apiserver 收到此类请求时，它可能会返回任意资源版本号的数据，但是优先返回较新版本。一般情况下它直接从 kube-apiserver 缓存中获取数据返回给 client，有可能读到过期的数据，适用于对数据一致性要求不高的场景。

第三种是设置 ResourceVersion 为一个非 0 的字符串。kube-apiserver 收到此类请求时，它会保证 Cache 中的最新 ResourceVersion 大于等于你传入的 ResourceVersion，然后从 Cache 中查找你请求的资源对象 key，返回数据给 client。基本原理是 kube-apiserver 为各个核心资源（如 Pod）维护了一个 Cache，通过 etcd 的 Watch 机制来实时更新 Cache。当你的 Get 请求中携带了非 0 的 ResourceVersion，它会等待缓存中最新 ResourceVersion 大于等于你 Get 请求中的 ResourceVersion，若满足条件则从 Cache 中查询数据，返回给 client。若不满足条件，它最多等待 3 秒，若超过 3 秒，Cache 中的最新 ResourceVersion 还小于 Get 请求中的 ResourceVersion，就会返回 ResourceVersionTooLarge 错误给 client。

你要注意的是，若你使用的 Get 接口，那么 kube-apiserver 会取资源 key 的 ModRevision 字段填充 Kubernetes 资源的 ResourceVersion 字段（v1.meta/ObjectMeta.ResourceVersion）。若你使用的是 List 接口，kube-apiserver

会在查询时，使用 etcd 当前版本号填充 ListMeta.ResourceVersion 字段 (v1.meta/ListMeta.ResourceVersion) 。

那么当我们执行 kubectl get po 查询案例时，它的 ResourceVersion 是什么取值呢？查询的是 kube-apiserver 缓存还是 etcd 最新共识数据？

如下所示，你可以通过指定 kubectl 日志级别为 6，观察它向 kube-apiserver 发出的请求参数。从下面请求日志里你可以看到，默认是未指定 Resource Version，也就是会发出一个共识读 / 线性读请求给 etcd，获取 etcd 最新共识数据。

 复制代码

```
1 kubectl get po -l app=nginx -v 6
2 4410 loader.go:359] Config loaded from file /root/.kube/config
3 4410 round_trippers.go:438] GET https://*.*/api/v1/namespaces/default/po
```

这里要提醒下你，在规模较大的集群中，尽量不要使用 kubectl 频繁查询资源。正如我们上面所分析的，它会直接查询 etcd 数据，可能会产生大量的 expensive request 请求，之前我就有见过业务这样用，然后导致了集群不稳定。

介绍完查询案例后，我们再看看 Watch 案例中，它的不同取值含义是怎样的呢？

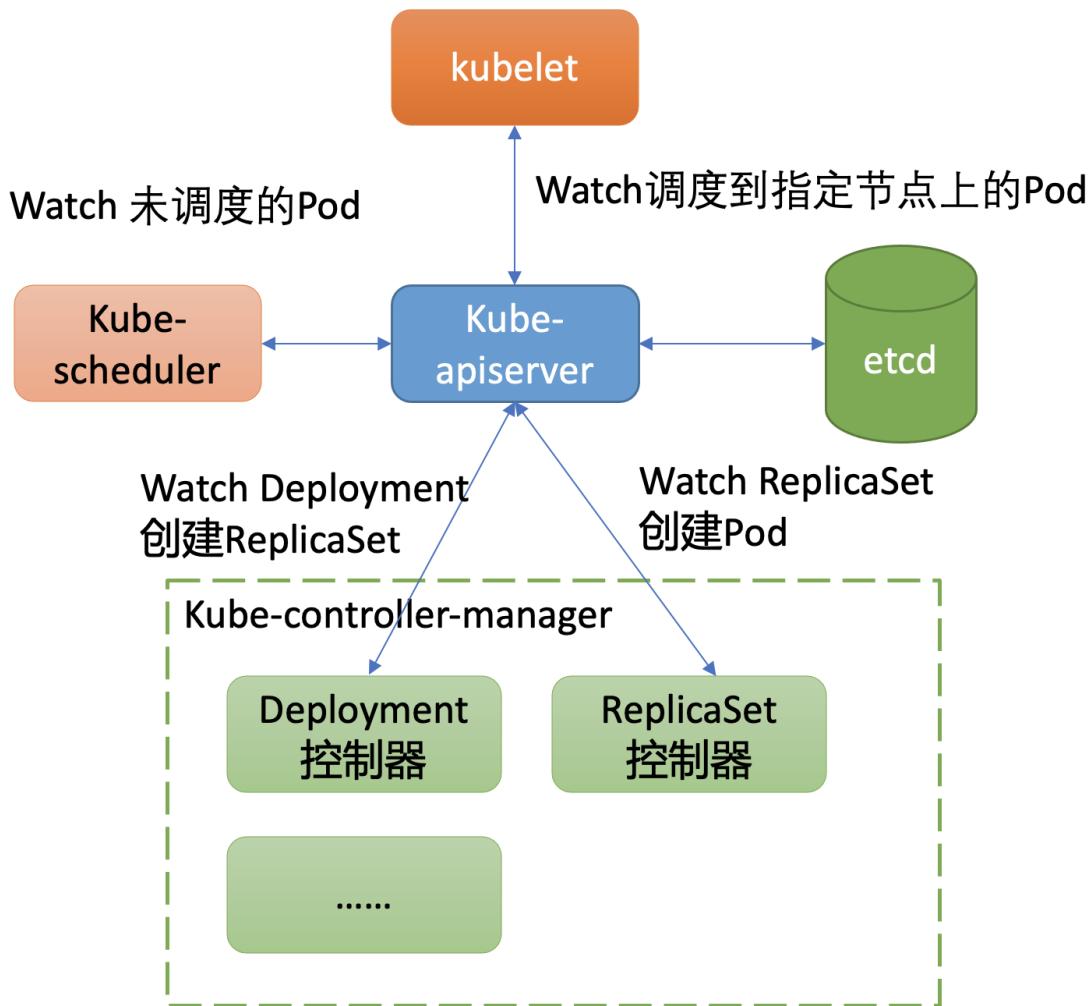
它同样含有查询案例中的三种取值，官方定义的含义分别如下：

未指定 ResourceVersion，默认空字符串。一方面为了帮助 client 建立初始状态，它会将当前已存在的资源通过 Add 事件返回给 client。另一方面，它会从 etcd 当前版本号开始监听，后续新增写请求导致数据变化时可及时推送给 client。

设置 ResourceVersion="0"，赋值字符串 0。它同样会帮助 client 建立初始状态，但是它会从任意版本号开始监听（当前 kube-apiserver 的实现指定 ResourceVersion=0 和不指定行为一致，在获取初始状态后，都会从 cache 最新的 ResourceVersion 开始监听），这种场景可能会导致集群返回陈旧的数据。

设置 ResourceVersion 为一个非 0 的字符串。从精确的版本号开始监听数据，它只会返回大于等于精确版本号的变更事件。

Kubernetes 的控制器组件就基于以上的 Watch 特性，在快速感知到新建 Deployment 资源后，进入一致性协调逻辑，创建 ReplicaSet 控制器，整体交互流程如下所示。



Deployment 控制器创建 ReplicaSet 资源对象的日志如下所示。

复制代码

```

1  {
2    "level": "debug",
3    "ts": "2021-02-11T09:55:45.923+0800",
4    "caller": "v3rpc/interceptor.go:181",
5    "msg": "request stats",
6    "start time": "2021-02-11T09:55:45.917+0800",
7    "time spent": "5.922089ms",
8    "remote": "127.0.0.1:44828",
9    "response type": "/etcdserverpb.KV/Txn",
10   "request count": 1,
11   "request size": 766,
12   "response count": 0,
13   "response size": 44,
  
```

```

14     "request content": "compare:<target:MOD key:/" /registry/replicasets/default/
15 }
```

真正创建 Pod 则是由 ReplicaSet 控制器负责，它同样基于 Watch 机制感知到新的 RS 资源创建后，发起请求创建 Pod，确保实际运行 Pod 数与期望一致。

复制代码

```

1 {
2     "level": "debug",
3     "ts": "2021-02-11T09:55:46.023+0800",
4     "caller": "v3rpc/interceptor.go:181",
5     "msg": "request stats",
6     "start time": "2021-02-11T09:55:46.019+0800",
7     "time spent": "3.519326ms",
8     "remote": "127.0.0.1:44664",
9     "response type": "/etcdserverpb.KV/Txn",
10    "request count": 1,
11    "request size": 822,
12    "response count": 0,
13    "response size": 44,
14    "request content": "compare:<target:MOD key:/" /registry/pods/default/nginx-d
15 }
```

在这过程中也产生了若干 Event，下面是 etcd 收到新增 Events 资源的请求，你可以看到 Event 事件 key 关联了 Lease，这个 Lease 正是由我上面所介绍的 leaseManager 所负责创建。

复制代码

```

1 {
2     "level": "debug",
3     "ts": "2021-02-11T09:55:45.930+0800",
4     "caller": "v3rpc/interceptor.go:181",
5     "msg": "request stats",
6     "start time": "2021-02-11T09:55:45.926+0800",
7     "time spent": "3.259966ms",
8     "remote": "127.0.0.1:44632",
9     "response type": "/etcdserverpb.KV/Txn",
10    "request count": 1,
11    "request size": 449,
12    "response count": 0,
13    "response size": 44,
14    "request content": "compare:<target:MOD key:/" /registry/events/default/nginx
15 }
```

Pod 创建出来后，这时 kube-scheduler 监听到待调度的 Pod，于是为其分配 Node，通过 kube-apiserver 的 Bind 接口，将调度后的节点 IP 绑定到 Pod 资源上。kubelet 通过同样的 Watch 机制感知到新建的 Pod 后，发起 Pod 创建流程即可。

以上就是当我们在 Kubernetes 集群中创建一个 Pod 后，Kubernetes 和 etcd 之间交互的简要分析。

小结

最后我们来小结下今天的内容。我通过一个创建 Pod 案例，首先为你解读了 Kubernetes 集群的 etcd 存储格式，每个资源的保存路径为 prefix + "/" + 资源类型 + "/" + namespace + "/" + 具体资源名组成。结合 etcd3 的范围查询，可快速实现按 namespace、资源名称查询。按标签查询则是通过 kube-apiserver 遍历指定 namespace 下的资源实现的，若未从 kube-apiserver 的 Cache 中查询，请求较频繁，很可能导致 etcd 流量较大，出现不稳定。

随后我和你介绍了 kube-apiserver 的通用存储模块，它通过在创建、查询、删除、更新操作前增加一系列的 Hook 机制，实现了新增任意资源只需编写相应的 Hook 策略即可。我还重点和你介绍了创建接口，它主要由拼接 key、获取 Lease ID、数据转换、写入 etcd 组成，重点是它通过使用事务接口实现了资源的安全创建及更新。

最后我给你讲解了 Resource Version 在 Kubernetes 集群中的大量应用，重点和你分析了 Get 和 Watch 请求案例中的 Resource Version 含义，帮助你了解 Resource Version 本质，让你能根据业务场景和对一致性的容忍度，正确的使用 Resource Version 以满足业务诉求。

思考题

我还给你留了一个思考题，有哪些原因可能会导致 kube-apiserver 报 “too old Resource Version” 错误呢？

感谢你的阅读，如果你认为这节课的内容有收获，也欢迎把它分享给你的朋友，谢谢。

[提建议](#)

更多课程推荐

Redis 核心技术与实战

从原理到实战，彻底吃透 Redis

蒋德钧

中科院计算所副研究员



涨价倒计时

现仅半价 **¥89** 4月17日涨价至**¥199**

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 18 | 实战：如何基于Raft从0到1构建一个支持多存储引擎分布式KV服务？

下一篇 20 | Kubernetes高级应用：如何优化业务场景使etcd能支撑上万节点集群？

精选留言 (7)

[写留言](#)



唐聪

2021-03-03

kubernetes中创建一个pod工作流程，resource version含义与etcd，通过label/fieldSelector查询性能，是比较常见的面试题。

展开 ▼



6



kxkue

2021-03-09

感觉这篇在讲述的创建pod的过程中，少了一些中间环节，比如介绍list-watch机制和Informer模块

作者回复: 有的哈，kubernetes内容非常多，浓缩成两讲了，下一讲高级应用有介绍informer模块



 ly
2021-03-28

too old Resource Version

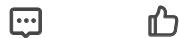
在更新资源的过程中，这个资源已经被其他进程更新的时候



 mmm
2021-03-20

informer watch请求的resource version比kube-apiserver缓存中保存的最小resource version还小，kube-apiserver就会返回“too old Resource Version”，然后触发informer进行list全量数据，导致expensive request

展开 ▼



 types
2021-03-16

你要注意的是，若你使用的Get接口，那么kube-apiserver会取资源key的ModRevision字段填充Kubernetes资源的ResourceVersion字段(v1.meta/ObjectMeta.ResourceVersion)。若你使用的是List接口，kube-apiserver会在查询时，使用etcd当前版本号填充ListMeta.ResourceVersion字段(v1.meta/ListMeta.ResourceVersion)。
请问什么情况下是GET接口 什么是List接口，可以通过kubectl举例说明下吗？

展开 ▼



 kkxue
2021-03-09

有哪些原因可能会导致kube-apiserver报“too old Resource Version”错误呢：有bug的时候





Simon
2021-03-03

思考题：

请求的版本在etcd已经回收了是不是就报"too old Resource Version"？

