调度器开发流程

> 2024.6.23日更新
> 本文介绍了如何对新建K8s的调度器。K8s外扩调度器有两种方式，一种是完全修改Kube-scheduler源码，然后编译，但这种方式需要大量修改代码，而且后续维护困难，本文不采用。第二种是K8s自1.17版本后调度器采用插件化的形式实现，可以方便用户进行定制或者二次开发，我们可以自定义一个调度器并以插件形式和 kubernetes 进行集成。
> 环境配置版本：cri-containerd:v1.6.24 ==,kubeadm kubeclt kubelet:v1.28.2,Golang1.20==
> 配置进行中可能会出现一些问题如，Not superuser、的权限问题，切换到root即可解决。文件权限问题使用 ls -ihl查看，chmod修改，k8s问题请查看日志和其他文档.
> 截至2024年6月7日起中国大陆docker被墙，镜像拉起可通过可翻墙设备下载->上传至阿里云/私有镜像服务器->目标设备docker pull

目录

1K8S调度扩展

1.1调度框架

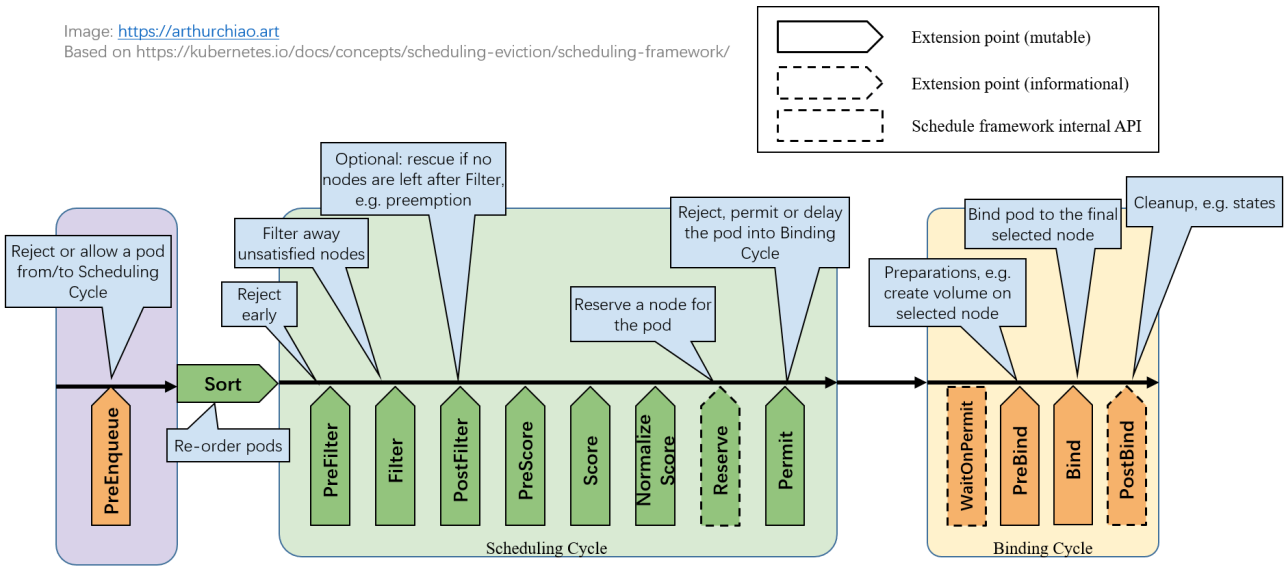
调度框架定义了一组扩展点，用户可以实现扩展点定义的接口来定义自己的调度逻辑（我们称之为**扩展**），并将扩展注册到扩展点上，调度框架在执行调度工作流时，遇到对应的扩展点时，将调用用户注册的扩展。调度框架在预留扩展点时，都是有特定的目的，有些扩展点上的扩展可以改变调度程序的决策方法，有些扩展点上的扩展只是发送一个通知
我们知道每当调度一个 Pod 时，都会按照两个过程来执行：**调度过程**和**绑定过程**
调度过程为 Pod 选择一个合适的节点，绑定过程则将调度过程的决策应用到集群中（也就是在被选定的节点上运行 Pod），将调度过程和绑定过程合在一起，称之为**调度上下文（scheduling context）**。需要注意的是调度过程是 同步 运行的（同一时间点只为一个 Pod 进行调度），绑定过程可异步运行（同一时间点可并发为多个 Pod 执行绑定）
调度过程和绑定过程遇到如下情况时会中途退出：

- 调度程序认为当前没有该 Pod 的可选节点

- 内部错误
  这个时候，该 Pod 将被放回到 **待调度队列**，并等待下次重试

一个 pod 的完整调度过程可以分为两个阶段：

1. `scheduling cycle`：为 pod 选择一个 node，类似于**数据库查询和筛选**；
2. `binding cycle`：落实以上选择，类似于**处理各种关联的东西并将结果写到数据库**。
   例如，虽然 scheduling cycle 为 pod 选择了一个 node，但是在接下来的 binding cycle 中，在这个 node 上给这个 pod 创建 persistent volume 失败了， 那整个调度过程也是算失败的，需要回到最开始的步骤重新调度。 以上两个过程加起来称为一个 `scheduling context`。
   另外，在进入一个 scheduling context 之前，还有一个**调度队列**， 用户可以编写自己的算法对队列内的 pods 进行排序，决定哪些 pods 先进入调度流程。 总流程如下图所示：



   Pod 处于 ready for scheduling 的阶段。 内部工作原理：sig-scheduling/scheduler_queues.md。
   这一步没过就不会进入调度队列，更不会进入调度流程。

1.2扩展点

上图的`scheduling cycle`和**`binding cycle`的两个周期的小方块都是可以扩展的。详细地说

1. `QueueSort` 扩展用于对 Pod 的待调度队列进行排序，以决定先调度哪个 Pod，`QueueSort` 扩展本质上只需要实现一个方法 `Less(Pod1, Pod2)` 用于比较两个 Pod 谁更优先获得调度即可，同一时间点只能有一个 `QueueSort` 插件生效
2. `Pre-filter` 扩展用于对 Pod 的信息进行预处理，或者检查一些集群或 Pod 必须满足的前提条件，如果 `pre-filter` 返回了 error，则调度过程终止
3. `Filter` 扩展用于排除那些不能运行该 Pod 的节点，对于每一个节点，调度器将按顺序执行 `filter` 扩展；如果任何一个 `filter` 将节点标记为不可选，则余下的 `filter` 扩展将不会被执行。调度器可以同时对多个节点执行 `filter` 扩展

4. `Post-filter` 是一个通知类型的扩展点，调用该扩展的参数是 `filter` 阶段结束后被筛选为**可选节点**的节点列表，可以在扩展中使用这些信息更新内部状态，或者产生日志或 metrics 信息
5. `PreScore` 扩展用于执行 “前置评分（pre-scoring）” 工作，即生成一个可共享状态供 Score 插件使用。 如果 PreScore 插件返回错误，则调度周期将终止
6. `Score` 评分插件用于对通过过滤阶段的节点进行排名。调度器为每个节点调用每个评分插件。 将有一个定义明确的整数范围，代表最小和最大分数。 在标准化评分阶段之后，调度器将根据配置的插件权重 合并所有插件的节点分数
7. `NormalizeScore` 扩展在调度器对节点进行最终排序之前修改每个节点的评分结果，注册到该扩展点的扩展在被调用时，将获得同一个插件中的 `scoring` 扩展的评分结果作为参数，调度框架每执行一次调度，都将调用所有插件中的一个 `normalize scoring` 扩展一次
8. `Reserve` 是一个通知性质的扩展点，有状态的插件可以使用该扩展点来获得节点上为 Pod 预留的资源，该事件发生在调度器将 Pod 绑定到节点之前，目的是避免调度器在等待 Pod 与节点绑定的过程中调度新的 Pod 到节点上时，发生实际使用资源超出可用资源的情况。（因为绑定 Pod 到节点上是异步发生的）。这是调度过程的最后一个步骤，Pod 进入 reserved 状态以后，要么在绑定失败时触发 Unreserve 扩展，要么在绑定成功时，由 Post-bind 扩展结束绑定过程
9. `Permit` 扩展在每个 Pod 调度周期的最后调用，用于阻止或者延迟 Pod 与节点的绑定。Permit 扩展可以做下面三件事中的一项：
    - approve（批准）：当所有的 permit 扩展都 approve 了 Pod 与节点的绑定，调度器将继续执行绑定过程
    - deny（拒绝）：如果任何一个 permit 扩展 deny 了 Pod 与节点的绑定，Pod 将被放回到待调度队列，此时将触发 `Unreserve` 扩展
    - wait（等待）：如果一个 permit 扩展返回了 wait，则 Pod 将保持在 permit 阶段，同时该 Pod 的绑定周期启动时即直接阻塞直到得到批准，如果超时事件发生，wait 状态变成 deny，Pod 将被放回到待调度队列，此时将触发 Unreserve 扩展
10. `Pre-bind` 扩展用于在 Pod 绑定之前执行某些逻辑。例如，pre-bind 扩展可以将一个基于网络的数据卷挂载到节点上，以便 Pod 可以使用。如果任何一个 `pre-bind` 扩展返回错误，Pod 将被放回到待调度队列，此时将触发 Unreserve 扩展
11. `Bind` 扩展用于将 Pod 绑定到节点上：
    - 只有所有的 pre-bind 扩展都成功执行了，bind 扩展才会执行
    - 调度框架按照 bind 扩展注册的顺序逐个调用 bind 扩展
    - 具体某个 bind 扩展可以选择处理或者不处理该 Pod
    - 如果某个 bind 扩展处理了该 Pod 与节点的绑定，余下的 bind 扩展将被忽略
    - 如果失败，则执行 Unreverse 扩展点将预先消费的资源释放掉(如 PVC 和 PV)，并将 Pod 从调度队列中删除
12. `Post-bind` 是一个通知性质的扩展，是整个调度的最后一步：
    - Post-bind 扩展在 Pod 成功绑定到节点上之后被动调用

- Post-bind 扩展是绑定过程的最后一个步骤，可以用来执行资源清理的动作

13. `Unreserve` 是一个通知性质的扩展，如果为 Pod 预留了资源，Pod 又在被绑定过程中被拒绝绑定，则 unreserve 扩展将被调用。Unreserve 扩展应该释放已经为 Pod 预留的节点上的计算资源。在一个插件中，reserve 扩展和 unreserve 扩展应该成对出现

官方实现了一些扩展点插件：kubernetes/pkg/scheduler/framework/plugins

如果我们要实现自己的插件，必须向调度框架注册插件并完成配置，另外还必须实现扩展点接口，参考项目：kubernetes-sigs/scheduler-plugins

扩展的调用顺序如下：

- 如果某个扩展点没有配置对应的扩展，调度框架将使用默认插件中的扩展
- 如果为某个扩展点配置且激活了扩展，则调度框架将先调用默认插件的扩展，再调用配置中的扩展
- 默认插件的扩展始终被最先调用，然后按照 `KubeSchedulerConfiguration` 中扩展的激活 `enabled` 顺序逐个调用扩展点的扩展
- 可以先禁用默认插件的扩展，然后在 `enabled` 列表中的某个位置激活默认插件的扩展，这种做法可以改变默认插件的扩展被调用时的顺序

2调度器源码编写

2.1kube-scheduler源码

如果我们要实现自己的插件，必须向调度框架注册插件并完成配置，另外还必须实现扩展点接口，对应的扩展点接口我们可以在源码 `pkg/scheduler/framework/v1alpha1/interface.go` 文件中找到。如下所示

```
// Plugin is the parent type for all the scheduling framework plugins.
type Plugin interface {
        Name() string
}

type QueueSortPlugin interface {
        Plugin
        Less(*PodInfo, *PodInfo) bool
}

// PreFilterPlugin is an interface that must be implemented by "prefilter"
plugins.
// These plugins are called at the beginning of the scheduling cycle.
type PreFilterPlugin interface {
        Plugin
```

```
        PreFilter(pc *PluginContext, p *v1.Pod) *Status
}

// FilterPlugin is an interface for Filter plugins. These plugins are called
at the
// filter extension point for filtering out hosts that cannot run a pod.
// This concept used to be called 'predicate' in the original scheduler.
// These plugins should return "Success", "Unschedulable" or "Error" in
Status.code.
// However, the scheduler accepts other valid codes as well.
// Anything other than "Success" will lead to exclusion of the given host from
// running the pod.
type FilterPlugin interface {
        Plugin
        Filter(pc *PluginContext, pod *v1.Pod, nodeName string) *Status
}

// PostFilterPlugin is an interface for Post-filter plugin. Post-filter is an
// informational extension point. Plugins will be called with a list of nodes
// that passed the filtering phase. A plugin may use this data to update
internal
// state or to generate logs/metrics.
type PostFilterPlugin interface {
        Plugin
        PostFilter(pc *PluginContext, pod *v1.Pod, nodes []*v1.Node,
filteredNodesStatuses NodeToStatusMap) *Status
}

// ScorePlugin is an interface that must be implemented by "score" plugins to
rank
// nodes that passed the filtering phase.
type ScorePlugin interface {
        Plugin
        Score(pc *PluginContext, p *v1.Pod, nodeName string) (int, *Status)
}

// ScoreWithNormalizePlugin is an interface that must be implemented by
"score"
// plugins that also need to normalize the node scoring results produced by
the same
// plugin's "Score" method.
type ScoreWithNormalizePlugin interface {
        ScorePlugin
        NormalizeScore(pc *PluginContext, p *v1.Pod, scores NodeScoreList)
*Status
}
```

```
// ReservePlugin is an interface for Reserve plugins. These plugins are called
// at the reservation point. These are meant to update the state of the
plugin.
// This concept used to be called 'assume' in the original scheduler.
// These plugins should return only Success or Error in Status.code. However,
// the scheduler accepts other valid codes as well. Anything other than
Success
// will lead to rejection of the pod.
type ReservePlugin interface {
        Plugin
        Reserve(pc *PluginContext, p *v1.Pod, nodeName string) *Status
}

// PreBindPlugin is an interface that must be implemented by "prebind"
plugins.
// These plugins are called before a pod being scheduled.
type PreBindPlugin interface {
        Plugin
        PreBind(pc *PluginContext, p *v1.Pod, nodeName string) *Status
}

// PostBindPlugin is an interface that must be implemented by "postbind"
plugins.
// These plugins are called after a pod is successfully bound to a node.
type PostBindPlugin interface {
        Plugin
        PostBind(pc *PluginContext, p *v1.Pod, nodeName string)
}

// UnreservePlugin is an interface for Unreserve plugins. This is an
informational
// extension point. If a pod was reserved and then rejected in a later phase,
then
// un-reserve plugins will be notified. Un-reserve plugins should clean up
state
// associated with the reserved Pod.
type UnreservePlugin interface {
        Plugin
        Unreserve(pc *PluginContext, p *v1.Pod, nodeName string)
}

// PermitPlugin is an interface that must be implemented by "permit" plugins.
// These plugins are called before a pod is bound to a node.
type PermitPlugin interface {
        Plugin
```

```
        Permit(pc *PluginContext, p *v1.Pod, nodeName string) (*Status,
time.Duration)
}

// BindPlugin is an interface that must be implemented by "bind" plugins. Bind
// plugins are used to bind a pod to a Node.
type BindPlugin interface {
        Plugin
        Bind(pc *PluginContext, p *v1.Pod, nodeName string) *Status
}
```

其实要实现一个调度框架的插件，并不难，我们只要实现对应的扩展点，然后将插件注册到调度器中即可，下面是默认调度器在初始化的时候注册的插件：

```
func NewRegistry() Registry {
        return Registry{
                // FactoryMap:
                // New plugins are registered here.
                // example:
                // {
                //  stateful_plugin.Name:
stateful.NewStatefulMultipointExample,
                //  fooplugin.Name: fooplugin.New,
                // }
        }
}
```

但是可以看到默认并没有注册一些插件，所以要想让调度器能够识别我们的插件代码，就需要自己来实现一个调度器了，当然这个调度器我们完全没必要完全自己实现，直接调用默认的调度器，然后在上面的 `NewRegistry()` 函数中将我们的插件注册进去即可。在 `kube-scheduler` 的源码文件 `kubernetes/cmd/kube-scheduler/app/server.go` 中有一个 `NewSchedulerCommand` 入口函数，其中的参数是一个类型为 `Option` 的列表，而这个 `Option` 恰好就是一个插件配置的定义：

```
// Option configures a framework.Registry.
type Option func(framework.Registry) error

// NewSchedulerCommand creates a *cobra.Command object with default parameters
and registryOptions
func NewSchedulerCommand(registryOptions ...Option) *cobra.Command {
  ......
}
```

所以我们完全就可以直接调用这个函数来作为我们的函数入口，并且传入我们自己实现的插件作为参数即可，而且该文件下面还有一个名为 `WithPlugin` 的函数可以来创建一个 `Option` 实例：

```
// WithPlugin creates an Option based on plugin name and factory.
func WithPlugin(name string, factory framework.PluginFactory) Option {
        return func(registry framework.Registry) error {
                return registry.Register(name, factory)
        }
}
```

所以最终我们的入口函数如下所示：

```
func main() {
        rand.Seed(time.Now().UTC().UnixNano())

        command := app.NewSchedulerCommand(
                app.WithPlugin(sample.Name, sample.New),
        )

        logs.InitLogs()
        defer logs.FlushLogs()

        if err := command.Execute(); err ≠ nil {
                _, _ = fmt.Fprintf(os.Stderr, "%v\n", err)
                os.Exit(1)
        }

}
```

其中 `app.WithPlugin(sample.Name, sample.New)` 就是我们接下来要实现的插件，从 `WithPlugin` 函数的参数也可以看出我们这里的 `sample.New` 必须是一个 `framework.PluginFactory` 类型的值，而 `PluginFactory` 的定义就是一个函数：

```
type PluginFactory = func(configuration *runtime.Unknown, f FrameworkHandle)
(Plugin, error)
```

所以 `sample.New` 实际上就是上面的这个函数，在这个函数中我们可以获取到插件中的一些数据然后进行逻辑处理即可。

2.2官方示例扩展源码

官方在源码文件的pkg\scheduler\framework\plugins\examples下面给出了几个扩展的示例，可以参考这些格式进行扩展,下面介绍一下代码逻辑。首先的代码将本文件封装为一个子包即package

multipoint。然后导入了必须的k8s扩展文件，这一步类似python的import。下面定义了类CommunicatingPlugin，在这个类实现了ReservePlugin和PreBindPlugin扩展。之后再k8s扩展框架中注册了CommunicatingPlugin。命名本扩展调度器，注意这个名字必须和后面k8s部署文件的KubeSchedulerConfiguration的扩展名相同。后面实现了类中必须的方法和实现。函数签名可以再源文件的framework和interface文件中找到，注意保持一致即可。

```
package multipoint//本文件封装为一个子包

//导入了必须的k8s扩展文件
import (
    "context"

    v1 "k8s.io/api/core/v1"
    "k8s.io/apimachinery/pkg/runtime"
    "k8s.io/kubernetes/pkg/scheduler/framework"
)


// CommunicatingPlugin is an example of a plugin that implements two
// extension points. It communicates through state with another function.
type CommunicatingPlugin struct{}//定义类
//注册ReservePlugin和PreBindPlugin扩展
var _ framework.ReservePlugin = CommunicatingPlugin{}
var _ framework.PreBindPlugin = CommunicatingPlugin{}

// Name is the name of the plugin used in Registry and configurations
const Name = "multipoint-communicating-plugin" //命名本扩展调度器

// Name returns name of the plugin. It is used in logs, etc.
func (mc CommunicatingPlugin) Name() string {
    return Name
}

type stateData struct {
    data string
}

func (s *stateData) Clone() framework.StateData {
    copy := &stateData{
        data: s.data,

    }

    return copy
```

```
}



// Reserve is the function invoked by the framework at "reserve" extension
point.

func (mc CommunicatingPlugin) Reserve(ctx context.Context, state
*framework.CycleState, pod *v1.Pod, nodeName string) *framework.Status {

    if pod == nil {

        return framework.NewStatus(framework.Error, "pod cannot be nil")

    }

    if pod.Name == "my-test-pod" {

        state.Write(framework.StateKey(pod.Name), &stateData{data: "never
bind"})

    }

    return nil

}



// Unreserve is the function invoked by the framework when any error happens

// during "reserve" extension point or later.

func (mc CommunicatingPlugin) Unreserve(ctx context.Context, state
*framework.CycleState, pod *v1.Pod, nodeName string) {

    if pod.Name == "my-test-pod" {

        // The pod is at the end of its lifecycle -- let's clean up the
allocated

        // resources. In this case, our clean up is simply deleting the key
written

        // in the Reserve operation.
```

```
        state.Delete(framework.StateKey(pod.Name))

    }

}



// PreBind is the function invoked by the framework at "prebind" extension
point.

func (mc CommunicatingPlugin) PreBind(ctx context.Context, state
*framework.CycleState, pod *v1.Pod, nodeName string) *framework.Status {

    if pod == nil {

        return framework.NewStatus(framework.Error, "pod cannot be nil")

    }

    if v, e := state.Read(framework.StateKey(pod.Name)); e == nil {

        if value, ok := v.(*stateData); ok && value.data == "never bind" {

            return framework.NewStatus(framework.Unschedulable, "pod is not
permitted")

        }

    }

    return nil

}



// New initializes a new plugin and returns it.

func New(_ context.Context, _ *runtime.Unknown, _ framework.Handle)
(framework.Plugin, error) {

    return &CommunicatingPlugin{}, nil

}
```

2.3自定义修改示例

这里给出扩展score模块的示例，这里给节点名字为piworknode001的节点打高分，期望pod被调度到该节点。值得说明的是这里不光有score代码还有NormalizeScore代码，是因为interface文件中framework.ScorePlugin中的签名中包含2个函数，因此扩展中也需要扩展2个函数，后续扩展其他插件时需要注意。

```
package main
import (
    "context"
    "fmt"
    "os"

    v1 "k8s.io/api/core/v1"
    "k8s.io/apimachinery/pkg/runtime"
    "k8s.io/component-base/logs"
    "k8s.io/klog/v2"
    "k8s.io/kubernetes/cmd/kube-scheduler/app"
    "k8s.io/kubernetes/pkg/scheduler/framework"
)



func main() {
    command := app.NewSchedulerCommand(
        app.WithPlugin(Name, createNew),
    )
    logs.InitLogs()
    defer logs.FlushLogs()
    if err := command.Execute(); err ≠ nil {
        _, _ = fmt.Fprintf(os.Stderr, "%v\n", err)
        os.Exit(1)
    }
}


// NewScoringPlugin is an example of a plugin that implements two
// extension points. It communicates through state with another function.
type NewScoringPlugin struct {
}

var _ framework.ScorePlugin = &NewScoringPlugin{}
// Name is the name of the plugin used in Registry and configurations.
const Name = "tonodename-scoring-plugin"
// Name returns name of the plugin. It is used in logs, etc.
```

```go
func (mc NewScoringPlugin) Name() string {
    return Name
}
type stateData struct {
    data string
}
func (s *stateData) Clone() framework.StateData {
    copy := &stateData{
        data: s.data,
    }
    return copy
}

func (s *NewScoringPlugin) Score(ctx context.Context, state
*framework.CycleState, p *v1.Pod, nodeName string) (int64, *framework.Status)
{
    if nodeName == "piworknode001" {
        return 80, framework.NewStatus(framework.Success)
    }
    return 20, framework.NewStatus(framework.Success)
}

func (s *NewScoringPlugin) ScoreExtensions() framework.ScoreExtensions {
    return s
}

func (s *NewScoringPlugin) NormalizeScore(ctx context.Context, state
*framework.CycleState, p *v1.Pod, scores framework.NodeScoreList)
*framework.Status {
    var min, max int64 = 0, 0
    // 求出最小分数和最大分数区间
    for _, score := range scores {
        if score.Score < min {
            min = score.Score
        }
        if score.Score > max {
            max = score.Score
        }
    }
    if max == min {
        min = min - 1
    }

    for i, score := range scores { // 每个节点的分数归一化处理
        scores[i].Score = (score.Score - min) * framework.MaxNodeScore / (max
- min)
```

```
        klog.Infof("节点: %v, Score: %v   Pod:  %v", scores[i].Name,
scores[i].Score, p.GetName())
    }
    return framework.NewStatus(framework.Success, "")
}

// New initializes a new plugin and returns it.
func createNew(ctx context.Context, configuration runtime.Object, f
framework.Handle) (framework.Plugin, error) {
    return &NewScoringPlugin{}, nil
}
```

2.4编译-构建镜像

然后编译该代码为二进制文件，注意由于引用了大量k8s源文件，因此注意k8s文件的版本信息，下面是经过测试可以使用的go.mod文件。go编译注意事项再网上资料很多这里不赘述。

```
module scheduler



go 1.21



toolchain go1.22.2



require (

    k8s.io/component-base v0.29.4

    k8s.io/kubernetes v1.29.4

)



require (
```

```
    github.com/Azure/go-ansiterm v0.0.0-20210617225240-d185dfc1b5a1 //
indirect

    github.com/NYTimes/gziphandler v1.1.1 // indirect

    github.com/antlr/antlr4/runtime/Go/antlr/v4 v4.0.0-20230305170008-
8188dc5388df // indirect

    github.com/asaskevich/govalidator v0.0.0-20190424111038-f61b66f89f4a //
indirect

    github.com/beorn7/perks v1.0.1 // indirect

    github.com/blang/semver/v4 v4.0.0 // indirect

    github.com/cenkalti/backoff/v4 v4.2.1 // indirect

    github.com/cespare/xxhash/v2 v2.2.0 // indirect

    github.com/coreos/go-semver v0.3.1 // indirect

    github.com/coreos/go-systemd/v22 v22.5.0 // indirect

    github.com/davecgh/go-spew v1.1.1 // indirect

    github.com/distribution/reference v0.5.0 // indirect

    github.com/emicklei/go-restful/v3 v3.11.0 // indirect

    github.com/evanphx/json-patch v5.6.0+incompatible // indirect

    github.com/felixge/httpsnoop v1.0.3 // indirect

    github.com/fsnotify/fsnotify v1.7.0 // indirect

    github.com/go-logr/logr v1.3.0 // indirect

    github.com/go-logr/stdr v1.2.2 // indirect

    github.com/go-logr/zapr v1.2.4 // indirect
```

```
    github.com/go-openapi/jsonpointer v0.19.6 // indirect

    github.com/go-openapi/jsonreference v0.20.2 // indirect

    github.com/go-openapi/swag v0.22.3 // indirect

    github.com/gogo/protobuf v1.3.2 // indirect

    github.com/golang/groupcache v0.0.0-20210331224755-41bb18bfe9da //
indirect

    github.com/golang/protobuf v1.5.4 // indirect

    github.com/google/cel-go v0.17.7 // indirect

    github.com/google/gnostic-models v0.6.8 // indirect

    github.com/google/go-cmp v0.6.0 // indirect

    github.com/google/gofuzz v1.2.0 // indirect

    github.com/google/uuid v1.3.1 // indirect

    github.com/grpc-ecosystem/go-grpc-prometheus v1.2.0 // indirect

    github.com/grpc-ecosystem/grpc-gateway/v2 v2.16.0 // indirect

    github.com/imdario/mergo v0.3.12 // indirect

    github.com/inconshreveable/mousetrap v1.1.0 // indirect

    github.com/josharian/intern v1.0.0 // indirect

    github.com/json-iterator/go v1.1.12 // indirect

    github.com/mailru/easyjson v0.7.7 // indirect

    github.com/matttproud/golang_protobuf_extensions v1.0.4 // indirect
```

```
    github.com/moby/sys/mountinfo v0.6.2 // indirect

    github.com/moby/term v0.0.0-20221205130635-1aeaba878587 // indirect

    github.com/modern-go/concurrent v0.0.0-20180306012644-bacd9c7ef1dd //
indirect

    github.com/modern-go/reflect2 v1.0.2 // indirect

    github.com/munnerz/goautoneg v0.0.0-20191010083416-a7dc8b61c822 //
indirect

    github.com/opencontainers/go-digest v1.0.0 // indirect

    github.com/opencontainers/selinux v1.11.0 // indirect

    github.com/pkg/errors v0.9.1 // indirect

    github.com/prometheus/client_golang v1.16.0 // indirect

    github.com/prometheus/client_model v0.4.0 // indirect

    github.com/prometheus/common v0.44.0 // indirect

    github.com/prometheus/procfs v0.10.1 // indirect

    github.com/spf13/cobra v1.7.0 // indirect

    github.com/spf13/pflag v1.0.5 // indirect

    github.com/stoewer/go-strcase v1.2.0 // indirect

    go.etcd.io/etcd/api/v3 v3.5.10 // indirect

    go.etcd.io/etcd/client/pkg/v3 v3.5.10 // indirect

    go.etcd.io/etcd/client/v3 v3.5.10 // indirect


go.opentelemetry.io/contrib/instrumentation/google.golang.org/grpc/otelgrpc
```

```
v0.42.0 // indirect

    go.opentelemetry.io/contrib/instrumentation/net/http/otelhttp v0.44.0 //
indirect

    go.opentelemetry.io/otel v1.21.0 // indirect

    go.opentelemetry.io/otel/exporters/otlp/otlptrace v1.21.0 // indirect

    go.opentelemetry.io/otel/exporters/otlp/otlptrace/otlptracegrpc v1.19.0 //
indirect

    go.opentelemetry.io/otel/metric v1.21.0 // indirect

    go.opentelemetry.io/otel/sdk v1.21.0 // indirect

    go.opentelemetry.io/otel/trace v1.21.0 // indirect

    go.opentelemetry.io/proto/otlp v1.0.0 // indirect

    go.uber.org/multierr v1.11.0 // indirect

    go.uber.org/zap v1.25.0 // indirect

    golang.org/x/crypto v0.21.0 // indirect

    golang.org/x/exp v0.0.0-20220827204233-334a2380cb91 // indirect

    golang.org/x/net v0.23.0 // indirect

    golang.org/x/oauth2 v0.11.0 // indirect

    golang.org/x/sync v0.5.0 // indirect

    golang.org/x/sys v0.18.0 // indirect

    golang.org/x/term v0.18.0 // indirect

    golang.org/x/text v0.14.0 // indirect
```

```
    golang.org/x/time v0.3.0 // indirect

    google.golang.org/appengine v1.6.7 // indirect

    google.golang.org/genproto v0.0.0-20230822172742-b8732ec3820d // indirect

    google.golang.org/genproto/googleapis/api v0.0.0-20230822172742-
b8732ec3820d // indirect

    google.golang.org/genproto/googleapis/rpc v0.0.0-20230822172742-
b8732ec3820d // indirect

    google.golang.org/grpc v1.59.0 // indirect

    google.golang.org/protobuf v1.33.0 // indirect

    gopkg.in/inf.v0 v0.9.1 // indirect

    gopkg.in/natefinch/lumberjack.v2 v2.2.1 // indirect

    gopkg.in/yaml.v2 v2.4.0 // indirect

    gopkg.in/yaml.v3 v3.0.1 // indirect

    k8s.io/api v0.30.2 // indirect

    k8s.io/apiextensions-apiserver v0.29.4 // indirect

    k8s.io/apimachinery v0.29.4 // indirect

    k8s.io/apiserver v0.29.4 // indirect

    k8s.io/client-go v0.29.4 // indirect

    k8s.io/cloud-provider v0.29.4 // indirect

    k8s.io/component-helpers v0.29.4 // indirect

    k8s.io/controller-manager v0.29.4 // indirect
```

```
    k8s.io/csi-translation-lib v0.29.4 // indirect

    k8s.io/dynamic-resource-allocation v0.29.4 // indirect

    k8s.io/klog/v2 v2.110.1 // indirect

    k8s.io/kms v0.29.4 // indirect

    k8s.io/kube-openapi v0.0.0-20231010175941-2dd684a91f00 // indirect

    k8s.io/kube-scheduler v0.29.4 // indirect

    k8s.io/kubelet v0.29.4 // indirect

    k8s.io/mount-utils v0.29.4 // indirect

    k8s.io/utils v0.0.0-20230726121419-3b25d923346b // indirect

    sigs.k8s.io/apiserver-network-proxy/konnectivity-client v0.28.0 //
indirect

    sigs.k8s.io/json v0.0.0-20221116044647-bc3834ca7abd // indirect

    sigs.k8s.io/structured-merge-diff/v4 v4.4.1 // indirect

    sigs.k8s.io/yaml v1.3.0 // indirect

)



replace (


go.opentelemetry.io/contrib/instrumentation/google.golang.org/grpc/otelgrpc ⇒
go.opentelemetry.io/contrib/instrumentation/google.golang.org/grpc/otelgrpc
v0.46.0

    go.opentelemetry.io/otel ⇒ go.opentelemetry.io/otel v1.21.0
```

```
    go.opentelemetry.io/otel/exporters/otlp/otlptrace/otlptracegrpc ⇒
go.opentelemetry.io/otel/exporters/otlp/otlptrace/otlptracegrpc v1.21.0

    go.opentelemetry.io/otel/metric ⇒ go.opentelemetry.io/otel/metric v1.21.0

    go.opentelemetry.io/otel/sdk ⇒ go.opentelemetry.io/otel/sdk v1.21.0

    go.opentelemetry.io/otel/trace ⇒ go.opentelemetry.io/otel/trace v1.21.0

    k8s.io/api ⇒ k8s.io/api v0.29.4

    k8s.io/apiextensions-apiserver ⇒ k8s.io/apiextensions-apiserver v0.29.4

    k8s.io/apimachinery ⇒ k8s.io/apimachinery v0.29.4

    k8s.io/apiserver ⇒ k8s.io/apiserver v0.29.4

    k8s.io/cli-runtime ⇒ k8s.io/cli-runtime v0.29.4

    k8s.io/client-go ⇒ k8s.io/client-go v0.29.4

    k8s.io/cloud-provider ⇒ k8s.io/cloud-provider v0.29.4

    k8s.io/cluster-bootstrap ⇒ k8s.io/cluster-bootstrap v0.29.4

    k8s.io/code-generator ⇒ k8s.io/code-generator v0.29.4

    k8s.io/component-base ⇒ k8s.io/component-base v0.29.4

    k8s.io/component-helpers ⇒ k8s.io/component-helpers v0.29.4

    k8s.io/controller-manager ⇒ k8s.io/controller-manager v0.29.4

    k8s.io/cri-api ⇒ k8s.io/cri-api v0.29.4

    k8s.io/csi-translation-lib ⇒ k8s.io/csi-translation-lib v0.29.4

    k8s.io/dynamic-resource-allocation ⇒ k8s.io/dynamic-resource-allocation
v0.29.4
```

```
    k8s.io/endpointslice => k8s.io/endpointslice v0.29.4

    k8s.io/kms => k8s.io/kms v0.29.4

    k8s.io/kube-aggregator => k8s.io/kube-aggregator v0.29.4

    k8s.io/kube-controller-manager => k8s.io/kube-controller-manager v0.29.4

    k8s.io/kube-proxy => k8s.io/kube-proxy v0.29.4

    k8s.io/kube-scheduler => k8s.io/kube-scheduler v0.29.4

    k8s.io/kubectl => k8s.io/kubectl v0.29.4

    k8s.io/kubelet => k8s.io/kubelet v0.29.4

    k8s.io/kubernetes => k8s.io/kubernetes v1.29.4

    k8s.io/legacy-cloud-providers => k8s.io/legacy-cloud-providers v0.29.4

    k8s.io/metrics => k8s.io/metrics v0.29.4

    k8s.io/mount-utils => k8s.io/mount-utils v0.29.4

    k8s.io/pod-security-admission => k8s.io/pod-security-admission v0.29.4

    k8s.io/sample-apiserver => k8s.io/sample-apiserver v0.29.4

)
```

由于镜像是在arm64的linux系统中运行，注意设置go env为对应配置
由于源码编译成二进制文件，因此构建镜像无需go的任何配置，只需要用最基础镜像busybox即可，下面给出Dockerfile

```
FROM busybox:stable-musl
WORKDIR /bin
ADD . .
RUN chmod 777  tonodename
CMD ["tonodename","--v=3","--config=/etc/kubernetes/scheduler-config.yaml"]
```

tonodename是上面2.3编译后的二进制文件
注意构建镜像时指定平台为arm64

3调度器部署文件

3.1自定义调度器部署文件详解

我们就可以当成普通的应用用一个 `Deployment` 控制器来部署自定义调度器，由于我们需要去获取集群中的一些资源对象，所以当然需要申请 RBAC 权限，下面声明的ClusterRole对象定义了一个拥有查看k8s资源的角色，然后注册一个账号ServiceAccount。使用ClusterRoleBinding文件将二者权限绑定到自定义调度器上。ConfigMap定义的KubeSchedulerConfiguration文件绑定到调度器内部。最后是`Deployment`部署调度器。
注意在调度器内部通过 `--config` 参数来配置，使用 `KubeSchedulerConfiguration` 资源对象配置，可以通过 `plugins` 来启用或者禁用我们实现的插件，也可以通过 `pluginConfig` 来传递一些参数值给插件。

3.2部署文件全文示例

注意替换所有的name标签

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: sample-scheduler-clusterrole
rules:
  - apiGroups:
      - ""
    resources:
      - endpoints
      - events
    verbs:
      - create
      - get
      - update
  - apiGroups:
      - ""
    resources:
      - nodes
    verbs:
      - get
      - list
```

```
    - watch
- apiGroups:
    - ""
  resources:
    - pods
  verbs:
    - delete
    - get
    - list
    - watch
    - update
- apiGroups:
    - ""
  resources:
    - bindings
    - pods/binding
  verbs:
    - create
- apiGroups:
    - ""
  resources:
    - pods/status
  verbs:
    - patch
    - update
- apiGroups:
    - ""
  resources:
    - replicationcontrollers
    - services
  verbs:
    - get
    - list
    - watch
- apiGroups:
    - apps
    - extensions
  resources:
    - replicasets
  verbs:
```

```
    - get
    - list
    - watch
- apiGroups:
    - apps
  resources:
    - statefulsets
  verbs:
    - get
    - list
    - watch
- apiGroups:
    - policy
  resources:
    - poddisruptionbudgets
  verbs:
    - get
    - list
    - watch
- apiGroups:
    - ""
  resources:
    - persistentvolumeclaims
    - persistentvolumes
  verbs:
    - get
    - list
    - watch
- apiGroups:
    - ""
  resources:
    - configmaps
  verbs:
    - get
    - list
    - watch
- apiGroups:
    - "storage.k8s.io"
  resources:
    - storageclasses
```

```
      - csinodes
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - "coordination.k8s.io"
    resources:
      - leases
    verbs:
      - create
      - get
      - list
      - update
  - apiGroups:
      - "events.k8s.io"
    resources:
      - events
    verbs:
      - create
      - patch
      - update
  - apiGroups:
    - "storage.k8s.io"
    resources:
      - csistoragecapacities
      - csidrivers
    verbs:
      - get
      - list
      - watch
  - apiGroups:
    _ ""
    resources:
      - namespaces
    verbs:
      - list
      - watch

---
```

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: sample-scheduler-sa
  namespace: kube-system
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: sample-scheduler-clusterrolebinding
  namespace: kube-system
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: sample-scheduler-clusterrole
subjects:
  - kind: ServiceAccount
    name: sample-scheduler-sa
    namespace: kube-system
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: scheduler-config
  namespace: kube-system
data:
  scheduler-config.yaml: |
    apiVersion: kubescheduler.config.k8s.io/v1
    kind: KubeSchedulerConfiguration
    leaderElection:
      leaderElect: false
    profiles:
    - schedulerName: sample-scheduler#替换为你的scheduler名字
      plugins:
        score:#替换为你的plugins位置
          enabled:
          - name: "tonodename-scoring-plugin"#替换为你的plugins名
---
apiVersion: apps/v1
kind: Deployment
```

```
metadata:
  name: sample-scheduler
  namespace: kube-system
  labels:
    component: sample-scheduler
spec:
  replicas: 1
  selector:
    matchLabels:
      component: sample-scheduler
  template:
    metadata:
      labels:
        component: sample-scheduler
    spec:
      serviceAccount: sample-scheduler-sa
      priorityClassName: system-cluster-critical
      volumes:
        - name: scheduler-config
          configMap:
            name: scheduler-config
      containers:
        - name: scheduler-ctrl
          image: registry.cn-hangzhou.aliyuncs.com/temp-iiip/scheduler:0.0.2#替换为你的镜像
          imagePullPolicy: IfNotPresent
          args:
            - sample-scheduler-framework
            - --config=/etc/kubernetes/scheduler-config.yaml
            - --v=3
          resources:
            requests:
              cpu: "50m"
          volumeMounts:
            - name: scheduler-config
              mountPath: /etc/kubernetes
          command: ["tonodename"]
          args: ["--v=3","--config=/etc/kubernetes/scheduler-config.yaml"]
```

3.3执行成功的结果

部署上述文件后查看pod是否运行，sample-scheduler-579d4464d7-wqmgh显示running

```
NAME                                   READY   STATUS    RESTARTS          AGE
coredns-6554b8b87f-85278               1/1     Running   3 (26d ago)       105d
coredns-6554b8b87f-n2lsc               1/1     Running   3 (26d ago)       105d
etcd-node                              1/1     Running   4 (26d ago)       105d
kube-apiserver-node                    1/1     Running   2 (9d ago)        105d
kube-controller-manager-node           1/1     Running   620 (2d16h ago)   105d
kube-proxy-b5lck                       1/1     Running   4 (26d ago)       105d
kube-proxy-f5grt                       1/1     Running   5 (21d ago)       105d
kube-proxy-qkhg4                       1/1     Running   7 (12d ago)       105d
kube-proxy-z8d5v                       1/1     Running   9 (2d19h ago)     100d
kube-scheduler-node                    1/1     Running   0                 43h
sample-scheduler-579d4464d7-wqmgh      1/1     Running   0                 12h
```

部署一个测试文件

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: test-scheduler
spec:
  replicas: 10
  selector:
    matchLabels:
      app: test-scheduler
  template:
    metadata:
      labels:
        app: test-scheduler
    spec:
      schedulerName: sample-scheduler
      containers:
        - image: registry.cn-hangzhou.aliyuncs.com/temp-iiip/temp:busybox
          name: busybox-try
          command: ["sleep"]
          args: ["infinity"]
```

显示

```
master@master:~/scheduler $ kubectl get pod
NAME                                    READY   STATUS    RESTARTS
test-scheduler-66454d887c-4b8h8         1/1     Running   0
test-scheduler-66454d887c-4znb5         1/1     Running   0
test-scheduler-66454d887c-5p2nt         1/1     Running   0
test-scheduler-66454d887c-dgh4v         1/1     Running   0
test-scheduler-66454d887c-jlgh8         1/1     Running   0
test-scheduler-66454d887c-ksrp4         1/1     Running   0
test-scheduler-66454d887c-r7wtm         1/1     Running   0
test-scheduler-66454d887c-sw25q         1/1     Running   0
test-scheduler-66454d887c-sxbv2         1/1     Running   0
test-scheduler-66454d887c-vhrxb         1/1     Running   0
```

进入sample-scheduler-579d4464d7-wqmgh内部查看日志

```
I0622 14:15:38.613390       15 eventhandlers.go:197] "Add event for scheduled pod" pod="default/test-scheduler-66454d887c-r7wtm"
I0622 14:15:38.613390       15 eventhandlers.go:171] "Delete event for unscheduled pod" pod="default/test-scheduler-66454d887c-r7wtm"
I0622 14:15:38.638508       15 schedule_one.go:302] "Successfully bound pod to node" pod="default/test-scheduler-66454d887c-jlgh8" node="piworknode001" evaluatedNodes=4 feasibleNodes=3
I0622 14:15:38.641020       15 eventhandlers.go:171] "Delete event for unscheduled pod" pod="default/test-scheduler-66454d887c-jlgh8"
I0622 14:15:38.641039       15 eventhandlers.go:197] "Add event for scheduled pod" pod="default/test-scheduler-66454d887c-jlgh8"
I0622 14:15:38.991797       15 eventhandlers.go:126] "Add event for unscheduled pod" pod="default/test-scheduler-66454d887c-dgh4v"
I0622 14:15:38.992016       15 schedule_one.go:98] "Attempting to schedule pod" pod="default/test-scheduler-66454d887c-dgh4v"
I0622 14:15:38.993160       15 main.go:87] 节点: piworknode001, Score: 100   Pod:  test-scheduler-66454d887c-dgh4v
I0622 14:15:38.993252       15 main.go:87] 节点: piworknode002, Score: 25   Pod:  test-scheduler-66454d887c-dgh4v
I0622 14:15:38.993269       15 main.go:87] 节点: piworknode003, Score: 25   Pod:  test-scheduler-66454d887c-dgh4v
I0622 14:15:38.993677       15 default_binder.go:53] "Attempting to bind pod to node" pod="default/test-scheduler-66454d887c-dgh4v" node="piworknode003"
I0622 14:15:39.048748       15 schedule_one.go:302] "Successfully bound pod to node" pod="default/test-scheduler-66454d887c-dgh4v" node="piworknode003" evaluatedNodes=4 feasibleNodes=3
I0622 14:15:39.048916       15 eventhandlers.go:171] "Delete event for unscheduled pod" pod="default/test-scheduler-66454d887c-dgh4v"
I0622 14:15:39.049002       15 eventhandlers.go:197] "Add event for scheduled pod" pod="default/test-scheduler-66454d887c-dgh4v"
I0622 14:15:39.392153       15 eventhandlers.go:126] "Add event for unscheduled pod" pod="default/test-scheduler-66454d887c-4znb5"
I0622 14:15:39.392312       15 schedule_one.go:98] "Attempting to schedule pod" pod="default/test-scheduler-66454d887c-4znb5"
I0622 14:15:39.393589       15 main.go:87] 节点: piworknode001, Score: 100   Pod:  test-scheduler-66454d887c-4znb5
I0622 14:15:39.394288       15 main.go:87] 节点: piworknode002, Score: 25   Pod:  test-scheduler-66454d887c-4znb5
I0622 14:15:39.394424       15 main.go:87] 节点: piworknode003, Score: 25   Pod:  test-scheduler-66454d887c-4znb5
I0622 14:15:39.395428       15 default_binder.go:53] "Attempting to bind pod to node" pod="default/test-scheduler-66454d887c-4znb5" node="piworknode002"
I0622 14:15:39.423268       15 eventhandlers.go:171] "Delete event for unscheduled pod" pod="default/test-scheduler-66454d887c-4znb5"
I0622 14:15:39.423568       15 schedule_one.go:302] "Successfully bound pod to node" pod="default/test-scheduler-66454d887c-4znb5" node="piworknode002" evaluatedNodes=4 feasibleNodes=3
```

4参考文档

自定义 POD 调度 Scheduling Framework - Lain Blog (xuliangtang.github.io)

自定义 Kubernetes 调度器-阳明的博客|Kubernetes|Istio|Prometheus|Python|Golang|云原生 (qikqiak.com)

调度器配置 | Kubernetes

巧用Prometheus来扩展kubernetes调度器 - DEV Community

K8s 调度框架设计与 scheduler plugins 开发部署示例（2024） (arthurchiao.art)

使用scheduler-framework扩展原生k8s调度器-腾讯云开发者社区-腾讯云 (tencent.com)

GitHub - kubernetes-sigs/scheduler-plugins: Repository for out-of-tree scheduler plugins based on scheduler framework.

kubernetes的编译、打包、发布-李佶澳 (lijiaocn.com)