Motivation

As most parts of scheduling framework KEP has been implemented, more and more users will use it to implement their own plugins(either private or public plugins). We'd like to add support for out of tree custom scheduler plugins, then it will be easier for users to use their own plugins.

Proposal

Proposal 1: vendor the scheduler code

As suggested by Custom Scheduler Plugins (out of tree), users could import scheduler's code directly in plugins repository, and use it as the default scheduler with config for plugins.

There has been a PR to implement this proposal. It provides a simple way to register plugins. When writing plugins, users import and use the scheduler code in the following way:

```
import (
  scheduler "k8s.io/kubernetes/cmd/kube-scheduler/app"
)


func main() {
  command := app.NewSchedulerCommand(
      app.WithPlugin("example-plugin1", ExamplePlugin1),
      app.WithPlugin("example-plugin2", ExamplePlugin2))
  if err := command.Execute(); err != nil {
    fmt.Fprintf(os.Stderr, "%v\n", err)
     os.Exit(1)
  }
```

}

It will be better to create a separate repository for this code. When users need to use other public plugins or use plugins in different repositories, users just need import their code in the above code and register their plugins.

Pros

- It is simple to write plugins with it.
- It has high performance.

Cons

- It is not so straightforward way compared with Golang's plugin.

Proposal 2: Golang plugin

A more straightforward way is to implement it using Golang's plugin. Users implement plugins that satisfy scheduler plugin's interface, and build it with plugin mode(`go build -buildmode=plugin`). Then scheduler will load the plugin dynamically if it is enabled in scheduler's config.

Pros

- It is more straightforward and simple.
- It has high performance.

Cons

The Golang plugin has a number of limitations/drawbacks, e.g. https://www.reddit.com/r/golang/comments/b6h8qq/is_anyone_actually_using_go_plugins/. I'd list most important one of them:

> The plugin compiler version must exactly match the program's compiler version. If the program was compiled with 1.11.4, it won't work to compile the plugin with 1.11.5. When distributing a program binary, you must communicate what the compiler version you used is.

Proposal 3: Other plugin mechanism

We investigate some other Go plugins, and finally choose hashicorp/go-plugin, it has stabilized from tens of millions of users using it.

The HashiCorp plugin system works by launching subprocesses and communicating over RPC (using standard `net/rpc`or gRPC). A single connection is made between any plugin and the host process. For net/rpc-based plugins, we use a connection multiplexing library to multiplex any other connections on top. For gRPC-based plugins, the HTTP2 protocol handles multiplexing.

Users need to write plugins like example_client.go, and build it. Scheduler launches the plugin as a subprocess and communicate it over RPC if it is enabled in scheduler's config.

Pros

- Hashicorp/go-plugin has a number of features and has proven to be battle hardened and ready for production use.
- From user's perspective, it is also straightforward.

Cons

- The code that users need write is more complex than the code in proposal 1 and 2 and users need to understand some basic ideas of `hashicorp/go-plugin`.
- Its performance is good, but lower than proposal 1 and 2.

Conclusion

Proposal 2 is not an appropriate way to support out of tree plugins, it is more appropriate to support in tree plugins.

Proposal 3 is a little complex for users.

Although proposal 1 is not obvious that this is the best way to enable out of tree plugins, it is simple and has high performance.

We would prefer to choose proposal 1 to support out of tree plugins.

Acknowledgement

Thanks @ahg-g, @bsalamat, @misterikkit and many others for the suggestions! We are very appreciative to receive more feedback and suggestions for it!