Implementing a CNI plugin

Learn how to implement a CNI plugin.

We'll cover the following

Overview

The CNI is a Kubernetes networking framework that allows users to dynamically configure the container networking through a configuration file and a group of executable binaries. These binaries can be written with any language, such as Go, Bash, and so on. With the help of the CNI, we can opt-in our customizations to meet our business needs.

Various vendors and open-source projects provide dozens of network solutions and plugins to address different infrastructures and underlying environments. Most of the time, we don't need to implement a CNI plugin on our own. Appropriate plugins can be found that meet our needs. Even if we do need to implement one, we don't normally build it from scratch. We can build it on top of other chained plugins, such as `ptp`, `vlan`, `bridge`, `portmap`, `firewall`, etc. These chained plugins are already implemented and provided with the CNI.

In this lesson, we'll implement a simple CNI plugin called debug, which will help with debugging and troubleshooting CNI plugins. This debug plugin will also help us better understand how CNI executable plugins are invoked and how they work.

Let's get started.

The scaffold

We can use any programming language we're familiar with when implementing a CNI plugin, such as Golang or shell scripts.

Below is a piece of skeleton code provided by project CNI. We can easily hook up our Golang implementations. From the skeleton, we can see there are four subcommands to be implemented: `ADD`, `DEL`, `CHECK` and `VERSION`.

```
// PluginMain is the core "main" for a plugin which includes automatic error handling.
//
// The caller must also specify what CNI spec versions the plugin supports.
```

```
//
// The caller can specify an "about" string, which is printed on stderr
// when no CNI_COMMAND is specified. The recommended output is "CNI plugin <foo> v<version>"
//
// When an error occurs in either cmdAdd, cmdCheck, or cmdDel, PluginMain will print the error
// as JSON to stdout and call os.Exit(1).
//
// To have more control over error handling, use PluginMainWithError() instead.
func PluginMain(cmdAdd, cmdCheck, cmdDel func(_ *CmdArgs) error, versionInfo version.PluginInfo, about s
    if e := PluginMainWithError(cmdAdd, cmdCheck, cmdDel, versionInfo, about); e != nil {
        if err := e.Print(); err != nil {
            log.Print("Error writing error JSON to stdout: ", err)
        }
        os.Exit(1)
    }
}
```

CNI Golang scaffold

For simplicity, we can use shell scripts as well. Below is a code snippet for implementing a CNI plugin with shell scripts:

```
case $CNI_COMMAND in
ADD)
    …
DEL)
    …
GET)
    …
VERSION)
    …
esac
```

CNI shell scaffold

Implement a plugin

Let's implement such a CNI plugin with the Golang scaffold, where we can directly use lots of utility functions.

We'll implement a function `addDebugLog` that will log all the CNI arguments and environments to a log file that we specify. These debug logs will help us know exactly what happens, such as the invoking parameters, CNI environments, returning values, etc. For the subcommands `ADD`, `DEL`, `CHECK`, and `VERSION`, we just need to hook them to the function `skel.PluginMain`, which is provided by the package `containernetworking/cni`. The code for this implementation is below:

main.go

```
package main

import (
```

Search in directory...

/

10-educative-plugin.conflist

go.sum

main.go

go.mod

```
4       "encoding/json"
5       "fmt"
6       "io"
7       "os"
8
9       "github.com/containernetworking/cni/pkg/skel"
10      "github.com/containernetworking/cni/pkg/types"
11      type100 "github.com/containernetworking/cni/pkg/types/100"
12      "github.com/containernetworking/cni/pkg/version"
13  )
14
15  type NetConf struct {
16      types.NetConf
17      CNIOutput string `json:"cniOutput,omitempty"`
18  }
19
20  func main() {
21      skel.PluginMain(cmdAdd, cmdCheck, cmdDel, version.All, fmt.Sp
22  }
23
24  func outputCmdArgs(fp io.Writer, subcommand string, args *skel.Cm
25      fmt.Fprintf(fp, `
26  CNI Command: %s
27  ContainerID: %s
28  Netns: %s
29  IfName: %s
30  Args: %s
31  Path: %s
```

Implement our CNI plugin

Let's build out an executable binary called `educative` with the command below.

```
1  go build -o /opt/cni/bin/educative main.go
```

Build our CNI plugin

To enable our binary `educative`, we need to prepare a CNI configuration file that includes our plugin. Here, we just make a little modification based on `/etc/cni/net.d/10-flannel.conflist`.

```
1  diff /usercode/10-educative-plugin.conflist /etc/cni/net.d/10-flannel.conflist
```

The diff command

Below is the content `diff`:

```
1  13,16d12
2  <       "type": "educative",
3  <       "cniOutput": "/tmp/cni_output.log"
4  <     },
5  <     {
6  24d19
7  <
```

The content diff

Let's back up the original `/etc/cni/net.d/10-flannel.conflist` and replace it. Run the following commands in the terminal above:

```
cp /etc/cni/net.d/10-flannel.conflist /usercode/
cp /usercode/10-educative-plugin.conflist /etc/cni/net.d/10-flannel.conflist
```

Prepare our CNI configuration file

Now, we can test our CNI plugin. Let's create a Pod `nginx` and see what happens. Run the following commands in the terminal above:

```
kubectl taint node --all node-role.kubernetes.io/master-
kubectl run nginx --image=nginx
```

Create a Pod nginx for testing

After a while, the `nginx` Pod's status will become "running" and it will have its own IP. Run the command `kubectl get pod -o wide` in the terminal above. We will get the output shown below:

```
NAME    READY   STATUS    RESTARTS   AGE   IP           NODE             NOMINATED NODE   READINESS GATE
nginx   1/1     Running   0          27s   10.244.0.4   vm-3-21-ubuntu   <none>           <none>
```

View the Pod status

Let's check the log `/tmp/cni_output.log` to see the debug messages. Run the command `tail -f /tmp/cni_output.log` in the terminal above. We'll get the output shown below:

```
CNI Command: ADD
ContainerID: fe2172d7a97df7c45dd3254c373933bafc08e9543e9d5249c97d933442e6d3bd
Netns: /proc/16448/ns/net
IfName: eth0
Args: IgnoreUnknown=1;K8S_POD_NAMESPACE=default;K8S_POD_NAME=nginx;K8S_POD_INFRA_CONTAINER_ID=fe2172d7a9
Path: /opt/cni/bin
StdinData: {"cniOutput":"/tmp/cni_output.log","cniVersion":"0.3.1","name":"cbr0","prevResult":{"cniVersi
----------------------
```

The logs of our CNI plugin

Ta-da! Our CNI plugin works!

Back

How the CNI Works

Next

Quiz on the Container Network Interface