# Spookd - virtual device plugin for Kubernetes

*Release 0.2.2*

Alexander Hill <alexander.hill@csiro.au>

Jul 18, 2022

# CONTENTS

**1 Spookd - virtual device plugin for Kubernetes** **3**

1.1 Introduction . . . . . 3

1.2 Deployment . . . . . 3

1.3 Device configuration . . . . . 4

1.4 Development . . . . . 5

1.5 TODO . . . . . 8

**2 Indices and tables** **9**

This project implements a Device Plugin for Kubernetes that allows arbitrary extended resources to be advertised and allocated based on a configuration file on the filesystem or in a ConfigMap.

This documentation is generated from the project README.md.

The project's home is on GitLab at https://gitlab.com/ska-telescope/ska-ser-k8s-spookd.

CHAPTER

ONE

# SPOOKD - VIRTUAL DEVICE PLUGIN FOR KUBERNETES

Spookd is a Kubernetes device plugin that lets you add virtual devices to your nodes. These devices can be requested and allocated to pods in your cluster in the same way that real host-local devices are. Virtual devices can be used to represent e.g. devices that are accessed over a network.

This device plugin draws inspiration from several others, notably

- https://github.com/hustcat/k8s-rdma-device-plugin
- https://github.com/Xilinx/FPGA_as_a_Service/tree/master/k8s-fpga-device-plugin

The precursor by Piers Harding can be found at https://gitlab.com/piersharding/k8s-ghost-device-plugin.

## 1.1 Introduction

Writing a Kubernetes device plugin allows you to add custom ("extended") resources to your cluster, which pods can request in the same way they do the built-in resources like `cpu` and `memory`. Most device plugins are concerned with managing devices present on the host, like GPUs or FPGAs - the plugin inspects the available hardware on the host and reports it to the Kubelet, and then Kubernetes handles scheduling and allocation of the resource to pods that have requested it.

Instead of dealing with real hardware available on the host, Spookd simply reads a configuration file (stored in a ConfigMap) and advertises the extended resources defined in it. These resources don't have to be network devices, or represent any physical device at all - they can represent any abstract resource your pods need mutually exclusive access to.

At SKAO we use this plugin to coordinate exclusive access to network-connected devices in our clusters.

## 1.2 Deployment

### 1.2.1 Using the Helm chart

First, if you haven't already, add the SKAO Helm repository. The following command will add it with the name `skao`:

```
$ helm repo add skao https://artefact.skao.int/repository/helm-internal
```

Next, create yourself an empty values file:

```
$ touch myvalues.yaml
```

Edit myvalues.yaml, adding a `deviceMapping` key with your initial device configuration (described in detail in the section below), plus any other values you wish to override in the Helm chart. Read the comments in the Chart's values.yaml for details on what you can override.

Then install the latest version of the Helm chart, using your values file:

```
$ helm install --values myvalues.yaml spookd-device-plugin skao/ska-ser-k8s-spookd --
↪version 0.2.2
```

This will install Spookd as a Helm release named `spookd-device-plugin`.

### 1.2.2 From the Helm chart in a local checkout of this git repository

Follow the instructions above, but skip the `helm repo add ...` step, and for the install step, run

```
$ helm install --values myvalues.yaml spookd-device-plugin ./charts/ska-ser-k8s-spookd
```

### 1.2.3 From the example Kubernetes manifest in a local checkout of this git repository

If you don't want to use Helm, you can deploy Spookd by editing the sample manifest `spookd-device-plugin.yaml` in this repository and deploying with

```
$ kubectl apply -f spookd-device-plugin.yaml
```

This manifest is mainly meant to be used as a shortcut during development, so there are a couple of things to bear in mind:

- This manifest is designed for testing during development, so the image in the pod spec is `localhost/spookd-device-plugin:latest`. If you want to deploy a production version, update this to `artefact.skao.int/ska-ser-k8s-spookd:0.2.2`.
- The imagePullPolicy is set to `Always`. You may wish to change this to `IfNotPresent`.
- A sample ConfigMap is defined inline in this manifest. Edit it to suit your needs before deploying.

## 1.3 Device configuration

Device configuration is defined in YAML and stored in a ConfigMap or file. The device plugin watches for configuration changes, and updates the set of advertised extended resources accordingly.

The configuration consists of a list of mappings, where each mapping defines a set of hostnames and a set of devices those hosts can access. Hosts and devices may appear in multiple mappings.

Devices are uniquely identified by `resourcename` and `instanceid` keys, and may optionally include `env` - a map of environment variables that will be set in containers that are allocated the device. If a device appears more than once, its `env` must be identical or the configuration is considered invalid.

Example:

```
deviceMapping:
  - hosts:
      - lab1-host1
```

(continues on next page)

(continued from previous page)

```
      - lab1-host2
    devices:
      - resourceName: skao.int/oscilloscope
        instanceID: "0001"
        env:
          IP: 10.0.10.215
      - resourceName: skao.int/oscilloscope
        instanceID: "0002"
        env:
          IP: 10.0.10.218
      - resourceName: skao.int/signal-generator
        instanceID: "0001"
        env:
          IP: 10.10.10.80
  - hosts:
      - lab2-host1
    devices:
      - resourceName: skao.int/oscilloscope
        instanceID: "0003"
        env:
          IP: 10.0.10.85
      - resourceName: skao.int/signal-generator
        instanceID: "0002"
        env:
          IP: 10.0.10.80
```

## 1.4 Development

The following instructions assume you have a minikube environment available, and the hostname of your minikube node is “minikube”.

### 1.4.1 Build

1. run `eval $(minikube podman-env)` (or `eval $(minikube docker-env)` if you’re using Docker). This ensures that the image is built by your minikube’s podman/docker daemon and the resulting image is immediately available in your minikube Kubernetes context.
2. Run `make docker`.
   This builds the project, creates an OCI image and tags it as `localhost/spookd-device-plugin:latest`. There is no need to run `make build` first - this happens as part of the OCI multi-stage build.

### 1.4.2 Deploy the sample manifest

Run the following command to deploy Spookd in your cluster's default namespace:

```
$ kubectl apply -f spookd-device-plugin.yaml -n spookd
```

You should see the following:

```
serviceaccount/spookd-device-plugin created
configmap/spookd-device-plugin created
role.rbac.authorization.k8s.io/spookd-device-plugin:watch-configmaps created
rolebinding.rbac.authorization.k8s.io/spookd-device-plugin created
daemonset.apps/spookd-device-plugin created
```

Verify that the pod is running with:

```
$ kubectl get pods -l app.kubernetes.io/name=spookd-device-plugin
NAME                         READY   STATUS    RESTARTS   AGE
spookd-device-plugin-fw4l9   1/1     Running   0          45s
```

And view logs:

```
$ kubectl logs -l app.kubernetes.io/name=spookd-device-plugin
time="2022-07-05T15:02:52Z" level=debug msg="0x4000138d00: Creating server for: example.
↪com/widget"
time="2022-07-05T15:02:52Z" level=info msg="0x400013d180: Starting to serve on /var/lib/
↪kubelet/device-plugins/spookd-examplecom_widget.sock"
time="2022-07-05T15:02:52Z" level=info msg="0x400013d180: Registered device plugin with␣
↪Kubelet"
time="2022-07-05T15:02:52Z" level=debug msg="0x4000138d00: Creating server for: example.
↪com/thingamajig"
time="2022-07-05T15:02:52Z" level=info msg="0x4000330780: Starting to serve on /var/lib/
↪kubelet/device-plugins/spookd-examplecom_thingamajig.sock"
time="2022-07-05T15:02:52Z" level=info msg="0x4000330780: Registered device plugin with␣
↪Kubelet"
time="2022-07-05T15:02:52Z" level=debug msg="0x4000138d00: Updating server for: example.
↪com/widget"
time="2022-07-05T15:02:52Z" level=debug msg="0x4000138d00: Updating server for: example.
↪com/thingamajig"
time="2022-07-05T15:02:52Z" level=debug msg="0x400013d180: reporting 1 example.com/
↪widget devices"
time="2022-07-05T15:02:52Z" level=debug msg="0x4000330780: reporting 2 example.com/
↪thingamajig devices"
```

Look at the resources advertised on your nodes with:

```
$ kubectl get nodes -o=jsonpath="{.items[*]['metadata.name', 'status.capacity']}{'\n'}"
minikube {"cpu":"2","ephemeral-storage":"51893228Ki","example.com/thingamajig":"1",
↪"example.com/widget":"1","hugepages-1Gi":"0","hugepages-2Mi":"0","hugepages-32Mi":"0",
↪"hugepages-64Ki":"0","memory":"1954688Ki","pods":"110"}
```

### 1.4.3 Deploy the test consumer Deployment

spookd-test-deployment.yaml is a manifest describing a Deployment that calls for two replicas of a pod which requires one `example.com/widget`, and one `example.com/thingamajig`. Deploy it:

```
$ kubectl apply -f spookd-test-deployment.yaml
deployment.apps/spookd-test created
$ kubectl get pods -l app.kubernetes.io/name=spookd-test
NAME                           READY   STATUS    RESTARTS   AGE
spookd-test-5cd4f987b8-c426x   0/1     Pending   0          2m34s
spookd-test-5cd4f987b8-hhh94   1/1     Running   0          2m34s
```

However, our test node only has one `example.com/widget`, and so one pod will remain in the `Pending` state as expected.

```
$ kubectl get pods spookd-test-5cd4f987b8-c426x -o jsonpath='{.status.conditions[0].
↪message}'
0/1 nodes are available: 1 Insufficient example.com/widget. preemption: 0/1 nodes are␣
↪available: 1 No preemption victims found for incoming pod.
```

Let's examine the environment variables made available by Spookd in the running pod:

```
$ kubectl exec spookd-test-5cd4f987b8-hhh94 -- env | sort
EXAMPLECOM_THINGAMAJIG_DEV0_ID=A1234
EXAMPLECOM_THINGAMAJIG_DEV0_TYPE=example.com/thingamajig
EXAMPLECOM_THINGAMAJIG_ID=A1234
EXAMPLECOM_THINGAMAJIG_NUM_DEVICES=1
EXAMPLECOM_THINGAMAJIG_TYPE=example.com/thingamajig
EXAMPLECOM_WIDGET_DEV0_ID=0001
EXAMPLECOM_WIDGET_DEV0_IP=192.168.0.200
EXAMPLECOM_WIDGET_DEV0_PORT=12345
EXAMPLECOM_WIDGET_DEV0_TYPE=example.com/widget
EXAMPLECOM_WIDGET_ID=0001
EXAMPLECOM_WIDGET_IP=192.168.0.200
EXAMPLECOM_WIDGET_NUM_DEVICES=1
EXAMPLECOM_WIDGET_PORT=12345
EXAMPLECOM_WIDGET_TYPE=example.com/widget
HOME=/root
HOSTNAME=spookd-test-5cd4f987b8-hhh94
KUBERNETES_PORT=tcp://10.96.0.1:443
KUBERNETES_PORT_443_TCP=tcp://10.96.0.1:443
KUBERNETES_PORT_443_TCP_ADDR=10.96.0.1
KUBERNETES_PORT_443_TCP_PORT=443
KUBERNETES_PORT_443_TCP_PROTO=tcp
KUBERNETES_SERVICE_HOST=10.96.0.1
KUBERNETES_SERVICE_PORT=443
KUBERNETES_SERVICE_PORT_HTTPS=443
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
TERM=xterm
```

## 1.5 TODO

- Handle exclusive access between different hosts via CRD and field management
- Clean up resource types with qty 0

CHAPTER

TWO

# INDICES AND TABLES

- genindex
- modindex
- search