SR-IOV Network Device Plugin

FeatureGates and global config

Apr 2021

What are the goals?

- Introduce FeatureGates – a way for user to disable/enable specific device plugin's features (similar to FeatureGates available in Kubernetes) that would be introduced in the future.
- FeatureGates should be configurable using ConfigMap and CLI arguments (precedence: default < ConfigMap < CLI)
- Introduce global config that could be used by features introduced in the future

FeatureGates – proposed implementation

Globally accessible singleton contained within 'features' package initialized within init() function.

Exported components:

- FeatureGate struct
- Available maturity levels (Alpha, Beta, GA, Deprecated)
- List of supported (known) features
- (*featureGate) **SetFromMap(map[string]bool)** - sets FeatureGates according to the provided map of featureName:bool
- (*featureGate) **SetFromString(string)** - sets FeatureGates according to provided string in format of featureName1=bool,featureName2=bool,... (used for CLI parameters)
- (*featureGate) **Enabled(feature)** - returns enablement status for provided feature

Internal components:

- defaultSriovDpFeatureGates - list of default values for known features
- init()
- newFeatureGate() - creates new featureGate with default settings
- (*featureGate) isFeatureSupported(feature) - checks if feature is among known features
- (fg *featureGate) set(feature, bool) - sets status of selected feature

How to use FeatureGates?

1. Add feature to the list of supported features and provide default enablement value.
2. (optional) Provide enablement value for SR-IOV Network Device Plugin deployment via ConfigMap or CLI arguments:

```
{
   "resourceList": [{
         "resourceName": "intel_sriov_netdevice",
         "selectors": {
            "vendors": ["8086"],
            "devices": ["154c", "10ed"],
            "drivers": ["i40evf", "iavf", "ixgbevf"]
         }
      }
   ],
   "featureGates": {
      “featureName1": true,
      "featureName2": false
   }
}

./sriovdp --feature-gates aphaFeature=true,betaFeature=false
```

3. Import features package and use conditional statement to check whether the feature is enabled or disabled:

```
if features.FeatureGate.Enabled(features.FeatureName) ...
```

Global config – proposed implementation

Globally accessible singleton contained within ‘config' package initialized within init() function.

Exported components:

- (cfg *config) ReadConfig(configFile string) - read config from the specified file (path)
- config struct that will hold feature-specific config (for now only FeatureGates)

```
type config struct {
   FeatureGates map[string]bool  `json:"featureGates,omitempty"`
}
```

Internal components:

- init()
- newConfig() - creates new config instance (used by init())

Concerns

- FeatureGate configuration precedence - should CLI args override ConfigMap parameters? When DP is running as a pod it seems to be easier to change ConfigMap than CLI args.
- Deprecation message – should emitted deprecation message be generic (e.g. WARNING: feature ABC is deprecated), or should be feature-specific (provided in config for example)?
- Should there be special idioms like e.g. ‘allAlphaFeatures’ that would disable/enable groups of features?
- Should there be a way to block the feature disabling/enabling?


intel