Kubernetes device plug-in for IBM Crypto Express (CEX) cards
Version 1.1.0

Installation and User Guide

Contents

Figures

Release Notes

Version 1.0

Version 1.0 of the Kubernetes device plug-in for IBM Crypto Express (CEX) cards is the initial release of this plug-in. It provides containerized applications access to IBM Crypto Express (CEX) cards on IBM Z® and IBM® LinuxONE (s390).

Features

The following features are included in the initial release:

- Enable CEX cards for pods
- Configure available crypto sets by using a ConfigMap
- Static copy of the sysfs for the pod
- Overcommitment of CEX Resources
- Hot plug and hot unplug of APQNS

Version 1.0.2

Version 1.0.2 of the Kubernetes device plug-in for IBM Crypto Express (CEX) cards contains one new feature and one bug fix.

Features

This update includes one new feature:

- The OVERCOMMIT limit can be specified per configset with an `overcommit:` field.

Resolved issues

- The code has been rebuilt with updated libraries because of a CVE finding in the protobuf package:
  - https://nvd.nist.gov/vuln/detail/CVE-2021-3121
  - http://github.com/gogo/protobuf.

Version 1.1.0

Version 1.1.0 of the Kubernetes device plug-in for IBM Crypto Express (CEX) cards includes the exploration of Prometheus metrics around the crypto resources managed by the plug-in. This release adds support for *Red Hat OpenShift Container Platform (RHOCP)* with enforced *Security Context Constraints (SCC)*. By default the CEX device plug-in now installs into it's own namespace *cex-device-plugin* and all the deployments have been rearranged for *Kustomize* support to make it easier to create and update the CEX device plug-in entities.

Features

- Prometheus metrics support
- New namespace 'cex-device-plugin' (See Migrating from kube-system to cex-device-plugin Namespace)
- Enabled for RHOCP with enforced SCC support
- Kustomize support

Resolved issues

The code has been rebuilt with updated libraries because of CVE findings in libraries the CEX device plug-in depends on:

- CVE-2022-3172 (Medium) in k8s.io/apiMachinery-v0.20.4
- CVE-2022-21698 (High) in github.com/prometheus/client_goLang-v1.11.0
- CVE-2022-32149 (High) in golang.org/x/text-v0.3.4

Known issues

There are no known issues. See Limitations for the list of current limitations.

Inclusive language

While IBM values the use of inclusive language, terms that are outside of IBM's direct influence are sometimes required for the sake of maintaining user understanding. As other industry leaders join IBM in embracing the use of inclusive language, IBM will continue to update the documentation to reflect those changes.

To learn more about this initiative, read the Words matter blog on ibm.com.

Introduction

This Kubernetes device plug-in provides access to IBM Crypto Express (CEX) cards for containers running on IBM Z and LinuxONE. Throughout this publication, the term 'CEX device plug-in' is used to refer to this Kubernetes device plug-in.

Overview

The Kubernetes CEX device plug-in provides IBM Crypto Express cards to be made available on Kubernetes nodes for use by containers.

The CEX device plug-in groups the available CEX resources (*APQNs*) into *CEX config sets*. Containers can request **one** resource from **one** *CEX config set*. Thus, from a container perspective, the APQNs within one *CEX config set* should be equivalent, which means that each APQN can be used interchangeably for any crypto workload.

See Considerations for equally configured APQNs for details.

The CEX config sets are described in a cluster-wide `ConfigMap`, which is maintained by the cluster administrator.

The CEX device plug-in instances running on all compute nodes:

- Check if the existing crypto resources are available on the nodes.
- Handle CEX resource allocation requests from the containers.
- Claim the resource.
- Ensure containers are scheduled on the correct compute node with the requested CEX crypto resources.

The CEX device plug-in instances running on each compute node:

- Screen all the available CEX resources on the compute node and provide this information to the Kubernetes infrastructure service.
- Allocate and deallocate a CEX resource on request of the Kubernetes infrastructure based on the requirement of a pod asking for CEX support.

The application container only has to specify that it needs a CEX resource from a specific *CEX config set* with a Kubernetes resource limit declaration. The cluster system and the CEX device plug-in handle the details, claim a CEX resource, and schedule the pod on the correct compute node.

The following sections provide more information about CEX resources on IBM Z and LinuxONE, the CEX device plug-in details, the CEX crypto configuration as *CEX config sets* in a cluster, and application container handling details.

Setting up the environment for the CEX device plug-in

CEX resources on IBM Z and LinuxONE

IBM Z and LinuxONE machines can have multiple Crypto Express cards (CEX) plugged in. The CEX device plug-in supports Crypto Express generations *CEX4* to *CEX7*. For each card, a mode of operation must be chosen: *Accelerator* mode, *CCA mode*, or *EP11* mode.

Each card is logically partitioned into independent units, so called crypto domains, which represent independent Hardware Security Modules (HSMs). These independent units within a card share the same mode of operation and the same card generation.

Thus, one HSM unit can be addressed with the *adapter* number (the crypto card number within a machine) and the *domain* number (the crypto partition index). Both values must be numeric in the range 0-255. They act as a link to one HSM unit and are called an "*APQN*".

An LPAR within an IBM Z or LinuxONE machine can have one or more crypto cards assigned and one or more domains. This results in a 2-dimensional table of APQNs.

The Kubernetes cluster is implemented as a KVM host running on an LPAR. The control plane and compute nodes of the cluster are represented by KVM guests running at and controlled by the KVM host. Therefore, some or all of the crypto resources available on the LPAR must be provided for use by the KVM guests running as Kubernetes compute nodes.

The point of view for a KVM guest running as a Kubernetes compute node is similar to the view of the LPAR. A compute node might have zero or more crypto adapters assigned and zero or more domains, which can be seen as a 2-dimensional table of APQNs.

This documentation does not cover the assignment and distribution of crypto resources to LPARs, KVM hosts, and KVM guests. For details, see:

- Section 10.1.3 "Configuring Crypto Express7S" in the IBM Redbook IBM z15 Configuration Setup
- Configuring Crypto Express Adapters for KVM Guests

For more information on Crypto Express cards, generations, and operation modes see:

- https://www.ibm.com/security/cryptocards

Usually the adapter/domain pair is sufficient to identify an APQN. However, if the compute nodes of a cluster are distributed over multiple IBM Z or LinuxONE machines a unique machine identification (*machine-id*) is needed in addition to the adapter and domain information.

An HSM contains a *secret* which must not get exposed to anyone. The secret, and potential additional settings of the HSM, are maintained by the *Security Administrator* of the system. These settings are typically done out-of-band, are properly maintained, and relatively static. On IBM Z and LinuxONE everything regarding crypto cards is typically done by the Security Administrator with the help of a Trusted Key Entry (TKE) workstation. For details, see the IBM Redbook System Z Crypto and TKE Update.

The *secret* is usually the source of a secret key often referred to as the *master key* or *master wrapping key*. Applications working with the HSM use *secure key* objects, which are clear key values encrypted ("wrapped") by the master key. Such a secure key can only be used together with the HSM as only the HSM has the master key to unwrap the secure key blob during a cryptographic operation.

A CEX crypto card in EP11 mode contains one *wrapping key*. A crypto domain on a CCA coprocessor card contains up to four master keys, which can be of type DES, AES, RSA, and ECC. Each of these master keys can wrap any type of clear key into a secure key. A CEX card in accelerator mode does not contain any secrets and can only be used to accelerate RSA clear key operations.

Multiple HSMs can be set up by the security administrator to be used as a backup for each other. Thus, the master keys and additional settings can be *equal*. *Equal* in this context means that an application using secure key methods can fulfill the job with either one of these HSMs, which form an equivalence set.

Spreading these equal APQNs among the compute nodes allows the Kubernetes dispatching algorithm to choose the target node of a crypto load. The algorithm is based on criteria like CPU and memory requirements and the availability of crypto resources.

In version 1 of the CEX device plug-in, a container should not change the configuration, master key, and control points of the HSM resources. Also any changes to crypto resources (mode, master keys, control points) should be performed while the affected APQNs are not available for use within the cluster.

Getting started with the CEX device plug-in

Creating and establishing a CEX resource configuration map

The CEX device plug-in needs to know a valid CEX configuration to start up properly. This section deals with creating a CEX resource configuration.

In the CEX resource configuration, equivalent APQNs are grouped into equivalence sets, called *crypto config sets*. A *crypto config set* has a unique name and comprises of one or more equivalent APQNs. A pod with a crypto application requests a CEX resource by requesting the allocation of one arbitrary APQN from the *crypto config set* by the name of the *crypto config set*.

Considerations for equally configured APQNs

Within each config set, all the APQNs must be set up consistently. For each CEX mode, consider:

- For Common Cryptographic Architecture (CCA) CEX resources, the master keys and access control point settings should be equal.
- For EP11 CEX resources, the EP11 wrapping key and control settings should be equal.
- CEX accelerator resources are stateless and do not need any equal setup.

A container requests exactly **one** *crypto config set* and obtains **one** CEX crypto resource from the CEX device plug-in if an APQN is available, healthy, and not already allocated. The APQN is randomly chosen and is assigned to the container.

The cluster-wide configuration of the CEX crypto resources is kept in a Kubernetes `ConfigMap` within the same namespace as the device plug-in. If the Kustomize base deployment provided in git repository is used, the namespace is called `cex-device-plugin`. The name of the `ConfigMap` must be `cex-resources-config` and the content is a configuration file section in JSON format.

A working sample is provided in the appendix Sample CEX resource configuration map.

The following example shows only the head and some possibly *crypto config set* definitions:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cex-resources-config
  namespace: cex-device-plugin
data:
  cex_resources.json: |
    {
    "cryptoconfigsets":
    [
        {
            "setname":   "CCA_for_customer_1",
            "project":   "customer-1",
            "cexmode":   "cca",
            "apqns":
            [
                {
                    "adapter":    1,
                    "domain":     6,
                    "machineid": ""
                },
                {
                    "adapter":    2,
                    "domain":     6,
                    "machineid": ""
                },
                {
                    "adapter":    7,
                    "domain":     6,
                    "machineid": ""
                }
```

```
            ]
        },
```

The `ConfigMap` defines a list of configuration sets. Each configuration set comprises the following entries:

Basic parameters

- `setname`: required, can be any string value, must be unique within all the configuration sets. This is the identifier used by the container to request one of the CEX crypto resources from within the set.
- `project`: required, can be any string value, namespace of the configuration set. Only containers with matching namespace can access CEX crypto resources of the configuration set. For version 1 this is not fully implemented as there are limits on the existing API preventing this. For details, see: Limitations.
- `cexmode`: optional, specifies the CEX mode. If specified, one of the following choices is required: `ep11`, `cca`, or `accel`. Adds an extra verification step every time the APQNs on each node are screened by the CEX device plug-in. All APQNs of the configuration set must match the specified CEX mode. On mismatches, the CEX device plug-in creates a log entry and discards the use of this APQN for the configuration set.
- `mincexgen`: optional, specifies the minimum CEX card generation for the configuation set. If specified, must match to `cex[4-9]`. Adds an extra verification step every time the APQNs on each compute node are screened. All APQNs of the configuration set are checked to have at least the specified CEX card generation. On mismatches, the CEX device plug-in creates a log entry and discards the use of the APQN for the configuration set.
- `overcommit`: optional, specifies the overcommit limit for resources in this ConfigSet. If the parameter is omitted, it defaults to the value specified through the environment variable APQN_OVERCOMMIT_LIMIT. If the environment variable is not specified, the default value for overcommit is 1 (no overcommit).

APQN parameters

- `apqns`: A list of equivalent APQN entries. The exact meaning of *equivalent* depends on the crypto workload to be run with the *crypto config set*. However, it forms a set of APQNs where anyone is sufficient to fulfill the needs of the requesting crypto workload container. See Considerations for equally configured APQNs.

  For example, a CCA application that uses a given AES secure key always relies on APQNs with a master key that wraps this secure key, regardless on which container it runs. In other words the master key setup of the APQNs within a ConfigSet should be the same.

  An APQN must not be member of more than one *crypto config set*. It is valid to provide an empty list. It is also valid to provide APQNs, which might currently not exist but might come into existence sometime in future when new crypto cards are plugged.

  The most simple APQN entry comprises these two fields:

  – `adapter`: required, the CEX card number. Can be in the range of 0-255. Typically referred to as `adapter` number.

  – `domain`: required, the domain on the adapter. Can be in the range of 0-255.

  The tuple of these two numbers uniquely identifies an APQN within one hardware instance. If the compute nodes are distributed over more than one hardware instance, an extra entry is needed to distinguish an APQN(a,d) on hardware instance 1 from APQN(a,d) on hardware instance 2:

  – `machineid`: optional, is only required when the compute nodes are physically located on different hardware instances and the APQN pairs (adapter, domain) are not unique. If specified, the value must be entered as follows: `<manufacturer>-<machinetype>-<sequencecode>` with

    - `<manufacturer>` – value of the `Manufacturer` line from `/proc/sysinfo`
    - `<machinetype>` – value of the `Type` line from `/proc/sysinfo`

- <sequencecode> – value of the Sequence Code line from /proc/sysinfo

For example, a valid value for machineid is IBM-3906-00000000000829E7.

The tuple (a,d) gets extended with the machine id, which is unique per hardware instance and the triple (a,d,maschineid) identifies an APQN again uniquely within the hardware instances.

Establishing the CEX resource configuration map

The CEX resource configuration map is a Kubernetes ConfigMap named cex-resources-config in the same Kubernetes namespace as the CEX device plug-in.

The CEX plug-in git repository contains a Kustomize base deployment that generates a configuration map from a given cex_resources.json file.

1. Download the repository and go to the deployments/configmap folder.
2. Edit the cex_resources.json JSON file in the deployments/configmap folder with your favorite editor.
3. To verify via pretty-print that you made valid JSON entries without errors, run the following command: jq -r . cex_resources.json If you see error messages, you need to fix them before continuing to the next step.
4. To create the configurtion map, run the following command: oc create -k . To update an already existing config map, run the following command: oc apply -k .

Installing and Configuring the CEX device plug-in

Obtaining the CEX device plug-in

The sources of the CEX device plug-in are located on github:

https://github.com/ibm-s390-cloud/k8s-cex-dev-plugin

- To use the certified and supported image from the Red Hat registry, run:

  ```
  podman pull registry.connect.redhat.com/ibm/ibm-cex-device-plugin-cm:<version>
  ```

- To use the community version, run:

  ```
  podman pull quay.io/ibm/ibm-cex-plugin-cm:<version>
  ```

The CEX device plug-in comprises several Golang source files that are built into one static binary, which is embedded into a container image. A sample *Dockerfile* is provided in the git repository to build the Go code and package the binary into a container image.

Next the container image needs to pushed into the image repository of your Kubernetes cluster. This step highly depends on the actual cluster and the cluster configuration and thus is not covered in this documentation.

Installing the CEX device plug-in

For installation of the CEX device plug-in, a set of Kustomize overlays is provided in the `deployments` directory of the repository. The source tree contains overlays for installation on Red Hat OpenShift Container Platform in the following directories:

- `rhocp-create` a configmap
- `rhocp-update` an overlay to update the installation without touching an existing configmap
- `configmap` an overlay to only update or create a configmap

OpenShift Container Platform including a configmap (directory `rhocp-create`), an overlay to update the installation without touching a (possibly existing) configmap (directory `rhocp-update`), and an overlay to only update or create a configmap (directory `configmap`). See the getting started documentation for details on the configuration.

To install the CEX device plug-in via these overlays with an empty configmap, run the following command:

```
oc create -k deployments/rhocp-create
```

By default, this deploys an empty configuration map. For example, it results in a CEX device plug-in that will not provide any devices. To directly create a custom configuration map, edit the file `cex_resources.json` in the `deployments/configmap` directory before running the above command. See Getting started with the CEX device plug-in for details.

Updating an Installation

If a configuration already exists in the `cex-device-plugin` namespace, it should not be overwritten by the installation script. To only update the CEX device plug-in, run the following command:

```
oc apply -k deployments/rhocp-update
```

This will update the cluster to the latest version of the CEX device plug-in without changing the existing configuration.

Installing the CEX device plug-in in details

The CEX device plug-in container image needs to be run privileged on each compute node. Kubernetes uses the concept of a *DaemonSet* for this kind of cluster-wide service. The git repository provides kustomize-based deployments for installation on Red Hat OpenShift Container Platform (directory `deployments/rhcop-create`).

To successfully run the CEX device plug-in the daemonset yaml, consider:

- `namespace`: The CEX device plug-in instances need to run in the same namespace where the CEX ConfigMap resides.
- `securityContext`: Must be *privileged* because the plug-in code needs access to some directories and files on the compute node:
  - To establish an IPC connection to the kubelet.
  - To do administrative tasks. For example, create and destroy zcrypt additional device nodes.
  - To build and provide directory trees to be mounted into the client containers. For example, shadow sysfs.
- `volumes`: The plug-in needs some volumes from the compute node:
  - `/dev` and `/sys` are needed to access the zcrypt device node and to add and remove *zcrypt additional device nodes* to be used by the crypto load containers.
  - The device-plug-in API provided by Kubernetes is accessed via gRPC, which needs the directory `/var/lib/kubelet/device-plugins`.
  - The CEX ConfigMap is accessed as a volume, which provides one file `cex_resources.json` where the cluster-wide CEX configuration is stored.
  - Access to `/var/tmp` is needed to build up the sysfs overlay directories for each container that uses crypto resources. For details on sysfs overlay see: The shadow sysfs.
- `InitContainer`: These commands set the appropriate SELinux labels for the shadow sysfs directory. Required only for nodes that are enabled for SELinux.
- `serviceAccount` and `serviceAccountName` should point to the account running the containers in the pods. This account requires on a few privileges:
  - It requires `get`, `list`, and `watch` access to `pods` to keep track of pods using devices provided by the plugin.
  - It requires `get`, `list`, and `watch` access to `configmaps` to be able to update its own configuration.
  - It also requires `use` access to the `privileged` SCC. This part is specific to RHOCP.

After obtaining the CEX device plug-in deployment files you should screen and maybe update the plug-in image source registry and then apply it with the following command:

```
oc create -k <kustomize_directory>
```

Here, `<kustomize_directory>` is the path to the desired directory you want to use.

A few seconds later a pod whose name starts with 'cex-plugin' in namespace `cex-device-plugin` should run on every compute node.

Further details on the CEX device plug-in

A CEX device plug-in instance is an ordinary application built from Golang code. The application provides a lot of information about what is going on via stdout/stderr. You can generate the output with the `kubectl logs <pod>` command, which should contain the namespace `-n cex-device-plugin` option.

The CEX device plug-in application initially screens all the available APQNS on the compute node, then reads in the CEX configuration. After verifying the CEX configuration, a Kubernetes *device-plug-in* with the

name of the config set is registered for each config set. This results in one device plug-in registration per config set with the full name *cex.s390.ibm.com/\<config-set-name\>*.

- For details about Kubernetes device plug-in's see: https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/device-plugins/
- For details about the Device Plugin Manager (dpm) see: https://pkg.go.dev/github.com/kubevirt/device-plugin-manager/pkg/dpm

After registration the CEX device plug-in is ready for allocation requests forwarded from the kubelet service. Such an allocation request is triggered by a crypto load pod requesting a CEX resource from the config set. The allocation request is processed and creates:

- A new zcrypt device node and forwards it to the container.
- Sysfs shadow directories and makes sure they are mounted on to the correct place within the container.

In addition, there are some secondary tasks to do:

- APQN rescan: Every `APQN_CHECK_INTERVAL` (default is 30s) the available APQNs on the compute node are checked. When there are changes, the plug-in reevaluates the list of available APQNs per config set and reannounces the list of plug-in-devices to the Kubernetes system.
- CEX config map rescan: Every `CRYPTOCONFIG_CHECK_INTERVAL` (default is 120s) the crypto config map is re-read. If the verification of the ConfigMap succeeds, the changes are re-evaluated and eventually result in reannouncements to the Kubernetes system. If verification fails, an error message `Config Watcher: failed to verify new configuration!` is shown. The plug-in continues to run without CEX crypto configuration and is thus unable to satisfy allocation requests. For details see: CEX configuration ConfigMap updates.
- Surveillance of pods with CEX resources allocated: Every `PODLISTER_POLL_INTERVAL` (default is 30s) the list of pods, which have a CEX resource assigned, is examined. This is matched against the list of resources, which are provided by the plug-in. For each allocation request the plug-in creates a zcrypt device node and shadow sysfs directories. These resources must be removed when no longer needed:
  - When the resources (zcrypt device node, shadow sysfs directories), which were created based on an allocation request are not used any more (the pod using the related plug-in device has not been seen any more) for more than `RESOURCE_DELETE_UNUSED` (default is 120s) seconds, these resources are destroyed.
  - When a zcrypt device node and the shadow sysfs directories, which were created based on an allocation request have not been used (there was never seen a running pod with the related plug-in device) for more than `RESOURCE_DELETE_NEVER_USED` (default 1800s) seconds, the zcrypt device node and the shadow sysfs directories are destroyed.

Allocation of CEX resources by containers

A container deployment can request one CEX resource from a CEX config set by specifying a resource statement as part of the container specification.

```
...
spec:
  containers:
  - image ...
    ...
    resources:
      limits:
        cex.s390.ibm.com/<config_set_name>: 1
    ...
```

For example, a container requesting a CEX resource from the config set `CCA_for_customer_1` from the sample ConfigMap in appendix Sample CEX resource Configuration Map needs the following container specification:

```
...
spec:
  containers:
  - image ...
    ...
    resources:
      limits:
        cex.s390.ibm.com/CCA_for_customer_1: 1
    ...
```

Sample CEX crypto load container in the appendix is a simple but complete sample yaml file for a customer load with CEX resource allocation.

When the Kubernetes system tries to run an instance of this container it recognizes the resource limitation. The CEX device plug-in instances should have registered plug-in-devices for each of the config sets, among them plug-in-devices for the `CCA_for_customer_1`. The Kubernetes system does the bookkeeping for all these devices and therefore knows, which devices are free and which devices were announced by the CEX device plug-in instances. The Kubernetes system chooses one compute node where a CEX device plug-in runs that had announced one of the free plug-in devices and forwards an allocation request to this plug-in.

The plug-in instance running on the compute node where the container gets applied, prepares the CEX resource and the sysfs shadow directories for the container, returns these to the Kubernetes system, and then the container is started. The container will have a device node `/dev/z90crypt` customized to have access to the allocated APQN and a customized `/sys/devices/ap` and `/sys/bus/ap` providing a limited view of the AP/zcrypt world.

When the container finally finishes, the CEX device plug-in on the compute node spots this, cleans up the allocated resources, and the Kubernetes system marks the plug-in-device as unused. The allocated resources which are cleaned up are the customized additional zcrypt device node and the sysfs shadow dirs.

Frequently asked questions

Q: What happens when all CEX resources within one config set are assigned to running containers and a new pod/container requesting a CEX resource from this config set is started?

A: Kubernetes will try to start the pod/container but the pod state is shown as `pending`. A `kubectl describe` shows the reason:

```
Warning  FailedScheduling  2m31s  default-scheduler  0/6 nodes are available: 1
  Insufficient cex.s390.ibm.com/<cex_config_set_name> ...
```

When finally a CEX resource from the config set becomes available, the `pending` pod will get started automatically by the Kubernetes system.

Q: What happens when a CEX resource from a not existing or not defined CEX config set is requested by a pod/container?

A: The Kubernetes cluster behaves similar to the out-of-CEX-resources within a config set case. The pod is in `pending` state until a config set with this name and a free CEX resource for this set come into existence. Then the CEX resource is assigned and the container started.

Q: I'd like to assign more than one APQN to the container to provide a backup possibility for the running application. Is this supported?

A: Currently, exactly **one** CEX resource can be requested by **one** container. The idea for backups for cluster applications is to schedule more pods/containers. This keeps the application within a container simple and easy and delegates the backup and performance issues to the cluster system.

Q: I'd like to package an application into a container that uses different kinds of CEX resources, for example one CCA and one EP11 APQN. So I'd like to assign two APQNs from different config sets to one container. Does that work?

A: No. Currently, exactly **one** CEX resource can be assigned to **one** container. This is only a limit to containers, but not to pods. As a pod can contain several containers each container can request one CEX resource from any config set. Split your application into units using only one type of CEX resource and package each unit into it's own container. Now your pod load runs as multiple containers with each having it's own CEX resource.

Technical Concepts and Limitations

CEX configuration ConfigMap updates

From a cluster administration point of view it is desirable to change the CEX configuration in the cluster-wide crypto ConfigMap. For example, to add or remove CEX resources within a config set or even add or remove whole crypto config sets.

This can be done during regular cluster uptime but with some carefulness. Every `CRYPTOCONFIG_CHECK_INTERVAL` (default is 120s) the crypto ConfigMap is re-read by all the CEX device plug-in instances. The new ConfigMap is verified and if valid, activated as the new current ConfigMap. On successful ConfigMap re-read the plug-in logs a message:

```
CryptoConfig: updated configuration
```

If the verification of the new CEX ConfigMap fails, the CEX device plug-in logs an error message. One reason for the verification failure might be the failure to read or parse the ConfigMap resulting in error logs like:

```
CryptoConfig: Can't open config file ...
```

or

```
CryptoConfig: Error parsing config file ...
```

If the verification step fails, the following message is displayed:

```
Config Watcher: failed to verify new configuration!
```

These failures result in running the plug-in instances without any configuration map.

The log messages appear periodically until yet another update of the ConfigMap is finally accepted as valid.

Note: After an update of a configuration map, the cluster needs some time (typically up to 2 minutes) to propagate the changes to all nodes. Another, potentially faster, way to update the configuration map for the plug-in is to restart the rollout of the deployment via:

```
kubectl rollout restart daemonset <name-of-the-cex-plug-in-daemonset> -n cex-device-plugin
```

This triggers a restart of each instance of the daemonset in a coordinated way by Kubernetes.

Overcommitment of CEX resources

By default, a CEX resource (an APQN) maps to exactly one Kubernetes *plug-in-device*. This is the administration unit known by Kubernetes and in fact a container requests such a plug-in device.

By default, the CEX device plug-in maps each available APQN to one plug-in device and as a result one APQN is assigned to a container requesting a CEX resource.

The CEX device plug-in can provide more than one plug-in-device per APQN, which allows some overcommitment of the available CEX resources.

Setting the environment variable `APQN_OVERCOMMIT_LIMIT` to a value greater than 1 (default is 1) allows to control how many plug-in devices are announced to the Kubernetes system for each APQN. For example, with three APQNs available within a config set and an overcommit value of 10, 30 CEX plug-in devices are allocatable and up to 30 containers could successfully request a CEX resource. The environment variable is specified in the DaemonSet YAML file via the `env` parameter.

You can specify the optional ConfigSet parameter "overcommit" to control the overcommit limit at config set level. If this parameter is omitted, the value defaults to the environment variable.

Eventually, more than one container will share one APQN with overcommitment enabled. This exposes no security weakness, but might result in lower performance for the crypto operations within each container.

Note: Dynamically changing the overcommit value, either by changing the environment variable, or by changing the overcommit parameter of a config set, changes the number of available CEX resources. If the number of available resources increases, containers waiting for resources might be able to run. Whereas already running containers continue to run, even if a used resource is no more available because of the decreased number of available resources. Due to lack of resources, those containers cannot be restarted.

The device node z90crypt

On a compute node, the device node `/dev/z90crypt` offers access to all zcrypt devices known to the compute running as a KVM guest. The application of a container, which requests a CEX resource will also see and use the device node `/dev/z90crypt`. However, what is visible inside the container is in fact a newly constructed z90crypt device with limited access to only the APQN assigned.

On the compute node, these constructed z90crypt devices are visible in the `/dev` directory as device nodes `zcrypt-apqn-<card>-<domain>-<overcommitnr>`. With the start of the container the associated device node on the compute node is mapped to the `/dev/z90crypt` device inside the container.

These constructed z90crypt devices are created on the fly with the CEX allocation request triggered with the container start and deleted automatically when the container terminates.

With version 1 of the CEX device plug-in, the constructed zcrypt device nodes limit access to exact one APQN (adapter, usage domain, no control domain), allowing all ioctls.

Note: These settings allow both usage and control actions, which are restricted to the underlying APQN with the `/dev/z90crypt` device that is visible inside the container, even with overcommited plug-in devices.

The shadow sysfs

The CEX device plug-in manipulates the AP part of the sysfs that a container can explore. The sysfs tree within a container contains two directories related to the AP/zcrypt functionality: `/sys/bus/ap` and `/sys/devices/ap`.

Tools working with zcrypt devices, like `lszcrypt` or `ivp.e`, need to see the restricted world, which is accessible via the `/dev/z90crypt` device node within the container.

The CEX device plug-in creates a *shadow sysfs* directory tree for each of these paths on the compute node at `/var/tmp/shadowsysfs/<plug-in-device>`. With the start of the container, both directories `/sys/bus/ap` and `/sys/devices/ap` are overlayed (overmounted) with the corresponding shadow directory on the compute node.

These shadow directory trees are simple static files that are created from the original sysfs entries on the compute node. They loose their sysfs functionality and show a static view of a limited AP/zcrypt world. For example, `/sys/bus/ap/ap_adapter_mask` is a 256 bit field listing all available adapters (crypto cards). The manipulated file that appears inside the container only shows the adapter that belongs to the assigned APQN. All load and counter values in the corresponding sysfs attributes, for example `/sys/devices/ap/card<xx>/<xx>.<yyyy>/request_count`, show up as 0 and don't get updates when a crypto load is running.

This restricted sysfs within a container should be sufficient to satisfy the discovery tasks of most applications (`lszcrypt`, `ivp.e`, opencryptoki with CCA or EP11 token) but has limits. For example, `chzcrypt` will fail to change sysfs attributes, offline switch of a queue will not work, and applications inspecting counter values might get confused.

An administrator logged into a Kubernetes compute node could figure out the assignment of a CEX resource and a requesting container. For example, by reading the log messages from the plug-ins. Without overcommitment the counters of an APQN on the compute node reflect the crypto load of the associated container and `lszcrypt` can be used.

Hot plug and hot unplug of APQNs

The CEX device plug-in monitors the APQNs available on the compute node by default every 30 seconds. This comprises the existence of APQNs and their *online* state. When the compute node runs as a KVM guest it is possible to *live* modify the devices section of the guest's xml definition at the KVM host, which results in APQNs appearing or disappearing. The AP bus and zcrypt device driver inside the Linux system recognizes this as hot plug or unplug of crypto cards and/or domains.

It is also possible to directly change the *online* state of a card or APQN within a compute node. For example, an APQN might be available but switched to *offline* by intention by an system administrator.

A dialog on the HMC offers the possibility to *configure off* and *configure on* CEX cards assigned to an LPAR. A CEX card in *config off* state is still visible in the LPAR and thus in the compute node but similar to the *offline* state no longer usable.

All this might cause the CEX device plug-in to deal with varying CEX resources. The plug-in code is capable of handling hot plug, hot unplug, the *online* state changes of CEX resources, and reports changes in the config set to the Kubernetes system. Because of this handling, APQNs can be included into the CEX config sets, which might not exist at the time of first deployment of the CEX configuration map. At a later time the card is hot plugged and assigned to the running LPAR. The cluster will spot this and make the appearing APQNs, which are already a member in a config set, available for allocation requests.

The handling of the *online* state is done by reporting the relevant plug-in devices as *healthy* (online) or *unhealthy* (offline). An *unhealthy* plug-in device is not considered when a CEX resource allocation takes place.

Note: It might happen that a CEX resource becomes unusable (hot unplug or offline state) but is assigned to a running container. The plug-in recognizes the state change, updates the bookkeeping, and reports this to the Kubernetes system but does **not** stop or kill the running container. It is assumed that the container load fails anyway because the AP bus or zcrypt device driver on the compute node reacts with failures on the attempt to use such a CEX resource device. A well designed cluster application terminates with a bad return code causing Kubernetes to re-establish a new container, which will claim a CEX resource and the situation recovers automatically.

SELinux and the Init Container

The CEX device plug-in prepares various files and directories that become mounted to the pod at an allocation request. Among those mounts are the directories descibed under The shadow sysfs. These folders are generated on the compute node and mounted into the new pod. In some cases, special actions are needed for such a mount to be accessible inside the newly created pod. For example, SELinux where the folder, or one of its parent folders, must have the appropriate SELinux label. Other security mechanisms might have different requirements.

Because the security mechanisms and their configuration depend on the cluster instance, the CEX device plug-in does not provide any support for such mechanisms. Instead, in the SELinux case, an Init Container can be used to set the correct label on the shadow sysfs root folder `/var/tmp/shadowsysfs` that contains all the sub-folders that are mapped into pods. See Sample CEX device plug-in daemonset yaml for an example of a daemonset deployment of the CEX device plug-in that contains an init container to set up `/var/tmp/shadowsysfs` for use in a SELinux-enabled environment.

Limitations

Namespaces and the project field

The `project` field of a CEX config set should match the namespace of the container requesting a member of this set. This results in only *blue* applications being able to allocate *blue* APQNs from the *blue* config set.

Unfortunately, the allocation request forwarded from the Kubernetes system to the CEX device plug-in does not provide any namespace information. Therefore, the plug-in is not able to check the namespace affiliation.

When the container runs, the surveillance loop of the CEX device plug-in detects this mismatch and displays a log entry:

```
PodLister: Container <aaa> in namespace <bbb> uses a CEX resource <ccc> marked for project
<ddd>!.
```

This behavior can be a security risk as this opens the possibility to use the HSM of another group of applications. However, to really exploit this, more is needed. For example, a secure key from the target to attack or the possibility to insert a self made secure key into the target application.

As a workaround, you can set quotas for all namespaces except for the one that is allowed to use the resource. See the following example:

```
    apiVersion: v1
    kind: ResourceQuota
    metadata:
      name: cex-blue-quota-no-red
      namespace: blue
    spec:
      hard:
        requests.cex.s390.ibm.com/red: 0
        limits.cex.s390.ibm.com/red: 0
```

This yaml snippet restricts the namespace *blue* to allocate zero CEX resources from the crypto config set *cex.s390.ibm.com/red*. The result is that all containers, which belong to the *blue* namespace, are not able to allocate *red* CEX resources any more.

Sample CEX quota restriction script in the appendix shows a bash script that produces a yaml file, which establishes these quota restrictions.

Prometheus Support

Starting with version 1.1.0 the CEX device plug-in provides metrics data about the CEX device resources. The metrics are offered in a Prometheus compatible way.

The following metrics are exposed:

- Metric `cex_plugin_plugindevs_available`:

  A vector of integer literals showing the number of CEX plug-in devices currently available, grouped by configset name.

  For example:

```
# TYPE cex_plugin_plugindevs_available gauge
cex_plugin_plugindevs_available{setname="Accels"} 6
cex_plugin_plugindevs_available{setname="CCA_for_customer_1"} 2
cex_plugin_plugindevs_available{setname="CCA_for_customer_2"} 4
cex_plugin_plugindevs_available{setname="EP11_for_customer_1"} 2
cex_plugin_plugindevs_available{setname="EP11_for_customer_2"} 3`
```

- Metric `cex_plugin_plugindevs_used`:

  A vector of integer literals showing the number of CEX plug-in devices currently in use, grouped by configset name.

  For example:

```
# TYPE cex_plugin_plugindevs_used gauge
cex_plugin_plugindevs_used{setname="Accels"} 1
cex_plugin_plugindevs_used{setname="CCA_for_customer_1"} 2
cex_plugin_plugindevs_used{setname="CCA_for_customer_2"} 1
cex_plugin_plugindevs_used{setname="EP11_for_customer_1"} 0
cex_plugin_plugindevs_used{setname="EP11_for_customer_2"} 3
```

- Metric `cex_plugin_request_counter`:

  A vector of integer literals showing the sum of request counter values of all CEX resources managed by the CEX device plug-in daemonset, grouped by configset name.

  For example:

```
# TYPE cex_plugin_request_counter gauge
cex_plugin_request_counter{setname="Accels"} 44700
cex_plugin_request_counter{setname="CCA_for_customer_1"} 36505
cex_plugin_request_counter{setname="CCA_for_customer_2"} 40428
cex_plugin_request_counter{setname="EP11_for_customer_1"} 24127
cex_plugin_request_counter{setname="EP11_for_customer_2"} 21655
```

- Metric `cex_plugin_total_plugindevs_available`:

  A simple integer literal showing the total number of CEX plug-in devices currently available in in the cluster. This metric gives the sum of all `cex_plugin_plugindevs_available` over all crypto configsets and is provided only for convenience.

  For example:

```
# TYPE cex_plugin_total_plugindevs_available gauge
cex_plugin_total_plugindevs_available 17
```

- Metric `cex_plugin_total_plugindevs_used`:

  A simple integer literal showing the total number of CEX plug-in devices currently in use in the cluster. This metric gives the the sum of all `cex_plugin_plugindevs_used` over all crypto configsets and is provided only for convenience.

For example:

```
# TYPE cex_plugin_total_plugindevs_used gauge
cex_plugin_total_plugindevs_used 7
```

- Metric `cex_plugin_total_request_counter`:

  A simple integer literal showing the total sum of all request counter values of all CEX resources managed by all CEX plug-in instances. This metric gives the sum of all `cex_plugin_request_counter` over all crypto configsets and is provided only for convenience.

  For example:

```
# TYPE cex_plugin_total_request_counter gauge
cex_plugin_total_request_counter 167415
```

Sample use cases that exploit these metrics are shown at the end of this section in paragraph Some sample use cases.

Details about the Prometheus support

The *CEX Prometheus exporter* is a Prometheus exporter that acts as a proxy between the CEX device plug-in instances running on each cluster node and the Prometheus server, or a similar service. The CEX Prometheus exporter collects and aggregates the raw metrics data from the plug-ins and serves as Prometheus client for the metrics monitoring service.

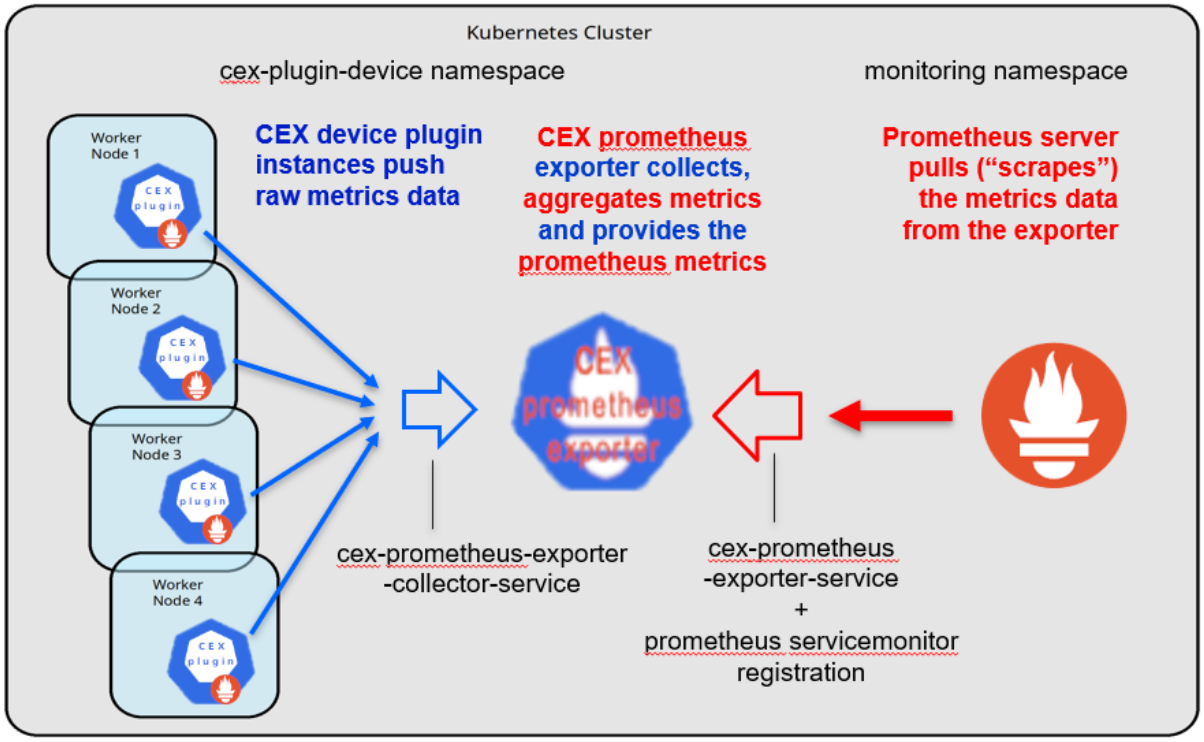

Figure 1. Plug-in and Exporter

The CEX Prometheus exporter runs as a pod inside the cluster and communicates with the CEX device plug-in instances via a cluster network through the *cex-prometheus-exporter-collector-service*. The CEX device plug-in instances use the cluster lookup for the metrics collector service and publish their metrics contributions to the corresponding endpoint.

The Prometheus server, or any other compatible monitoring service, has to be made aware of the new scrape target. This is done with a ServiceMonitor registration and the *cex-prometheus-exporter-service*.

Note: The service port 9939 used here is registered at Prometheus_default port allocations and should not need any adjustment.

Setting up the Prometheus support for the CEX device plug-in

To provide the CEX Prometheus metrics the CEX Prometheus exporter needs to run as another pod.

The prebuilt CEX device plug-in image comes with the following integrated applications:

- The `cex-plugin` application providing the CEX device plug-in functionality
- The Prometheus exporter application `cex-prometheus-exporter`

To specify which application to use, add the `command` parameter to the config yaml file.

- `command: ["/work/cex-plugin"]` starts the CEX device plug-in
- `command: ["/work/cex-prometheus-exporter"]` starts the CEX Prometheus exporter

If `command` is not specified, the default entrypoint application `/work/cex-plugin` is executed.

The appendix contains sample yaml deployments for the CEX device plug-in daemonset and the CEX Prometheus exporter pod. The CEX Prometheus exporter is a simple application, which does not require any extended capabilities, secrets, or special volume mounts. The CEX Prometheus exporter is not required to run permanently, but if stopped the ability to fetch metrics data for the CEX resources is lost.

The CEX device plug-in github repository provides ready-to-use deployment samples for `kubectl` via the `kustomize` extension.

The `cex_prom_exporter_pod.yaml` yaml file provides a pre-customized deployment for the CEX Prometheus exporter pod.

In addition to the CEX Prometheus exporter pod, two service definitions are required to permit TCP traffic. Pre-customized deployments are provided in the CEX github repository:

- For traffic between the CEX device plug-in instances and the exporter, the *cex-prometheus-exporter-collector-service* yaml file.
- For traffic between the Prometheus server and the exporter, the *cex-prometheus-exporter-service* yaml file.

If a new Prometheus client is available for retrieval of metrics, the Prometheus server needs to get informed. A sample *ServiceMonitor* definition for the CEX Prometheus exporter is provided in the appendix Sample CEX Prometheus exporter servicemonitor yaml.

The github repository file `cex_prom_exporter_prometheus_servicemonitor.yaml` provides pre-customized deployment for the ServiceMonitor.

By default the CEX Prometheus exporter pod, the two ClusterIP services, and the ServiceMonitor all live in the *cex-plugin-device* namespace. However, an experienced administator can change the names, the port settings, and the namespace preselection. The only requirement is that the CEX device plug-in instances are able to reach the CEX Prometheus exporter pod to push their raw data. In addition, the monitoring system needs to contact the CEX Prometheus exporter pod to pull the metrics. For details see Environment variables.

Sample Prometheus use cases for the CEX resources

- Prometheus query:

```
`rate(cex_plugin_total_request_counter[60s])`
```

The diagram that is generated by the query provides a basic overview of the CEX crypto activities within the cluster over time. It shows the summarized CEX crypto counters without differentiation about CEX config sets. If you are an experienced administator, you can use this query to discover unexpected crypto counter rates.

Example output:

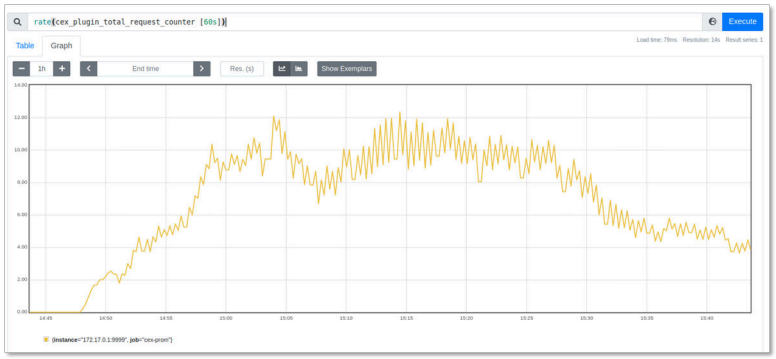

Figure 2. cex_plugin_total_request_counter

- Prometheus query:

```
`30 * rate(cex_plugin_request_counter[30s])`
```

The diagram that is generated by the query shows the rate of the request counters grouped by crypto config sets to provide a basic overview about the use of the individual config set CEX resources over time. However, this query can only express a relative utilization because a request counter increase can have various causes. For example, the cause could be a lightweight crypto operation where 100000s of operations per second per CEX resource are possible or the result of a heavyweight operation with RSA 4K key generations where only about 1000 operations per second per CEX resource are possible.

Example output:

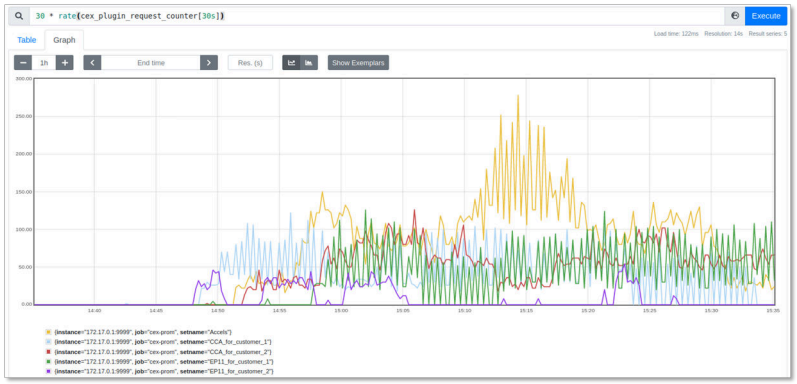

Figure 3. rate cex_plugin_request_counter

- Prometheus query:

```
`30 * rate(cex_plugin_plugindevs_used[30s])`
```

The diagram that is generated by the query shows the rate of used CEX plug-in devices over time differentiated by CEX config set. The diagram gives some hints how often and how long CEX resources of a certain set are used and can be used to adjust the provisioning of APQNs in the cluster.

Example output:

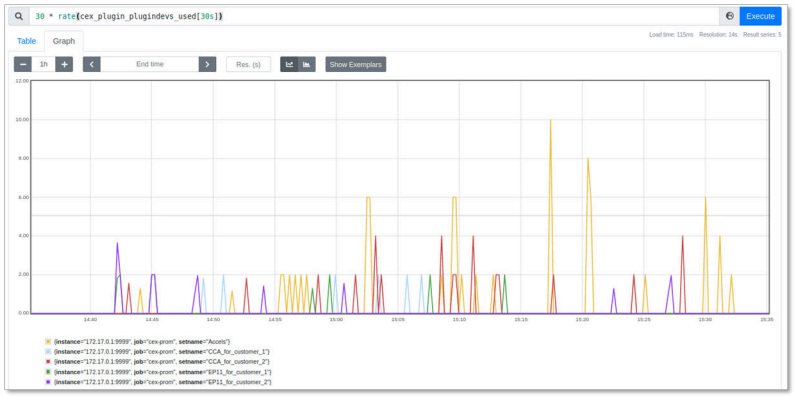

Figure 4. rate cex_plugin_plugindevs_used

- Prometheus query:

```
`100 * cex_plugin_plugindevs_used / cex_plugin_plugindevs_available`
```

The diagram that is generated by the query shows the utilization of the CEX plug-in devices in percent by calculating the fraction of used versus available CEX plug-in devices. A value of 100% means that all available CEX plug-in devices within this crypto config set are in use. Thus when a new container load requests a CEX resource of the given type, it is delayed until a CEX resource is released by a terminating container.

You can use the results to define a threshold when you configure alert rules for the Prometheus server. For example, you can define the threshold to fire when the 90% limit is overshoot for more than 2 minutes. The Prometheus alert rule then triggers an action depending on the configuration of the Alertmanager. For example, an email or SMS is sent to the cluster administrator, to indicate that the number of CEX resourcees for this configset are not sufficient and more are needed.

Example output:

Figure 5. utilisation cex_plugin_plugindevs

Troubleshooting

This section provides information on diagnostics and troubleshooting.

If you experience issues with the CEX device plug-in, you can check the pod status, gather pod diagnostics, and collect debugging data.

Prerequisites

You must log in as a user that belongs to a role with administrative privileges for the cluster. For example, `system:admin` or `kube:admin`.

Verification

The CEX device plug-in runs as a daemonset in namespace `cex-device-plugin`.

The following query should list the CEX device plug-in daemonset:

```
$ kubectl get daemonsets -n cex-device-plugin
NAME                   DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
cex-plugin-daemonset   3         3         3       3            3           <none>          4d
```

The daemonset is realized as a pod with one container per each compute node.

To review the status of pods running in the CEX device plug-in and the kube-system namespace, run the following command:

```
$ kubectl get pods -n cex-device-plugin
NAME                          READY   STATUS    RESTARTS   AGE
cex-plugin-daemonset-bfxt2   1/1     Running   0          3d23h
cex-plugin-daemonset-bhhj8   1/1     Running   0          3d23h
cex-plugin-daemonset-bntsp   1/1     Running   0          3d23h
```

Verify that the pods are running correctly. There should be one pod per each compute node in status `Running`. If one or more of the CEX device plug-in pods do not show up or are not showing a `Running` status, you can collect diagnostic information.

To inspect the status of a pod in detail, use the `describe` subcommand, for example:

```
$ kubectl describe pod cex-plugin-daemonset-bfxt2 -n cex-device-plugin
```

When these requirements are fulfilled, ensure that you have a CEX resource configuration map, which defines the *CEX config sets* deployed in namespace `cex-device-plugin`:

```
$ kubectl get configmap -n cex-device-plugin
NAME                                 DATA   AGE
...                                  ...    ...
cex-resources-config                 1      4d2h
...                                  ...    ...
```

To verify if a configmap has been deployed, run kubectl describe on one of the plug-in pods. If no configmap is deployed the output will show a message that the volume mount failed, for example:

```
MountVolume.SetUp failed for volume "cex-resources-conf" : configmap "cex-resources-config" not
found
```

If the CEX configmap is deployed and the CEX device plug-in instances are running, verify the available and allocated CEX resources on each compute node:

```
$ kubectl describe nodes
...
Allocatable:
```

```
  ...
  cex.s390.ibm.com/Accel:                1
  cex.s390.ibm.com/CCA_for_customer_1:   3
  cex.s390.ibm.com/EP11_for_customer_2:  2
  cpu:                                   3500m
  ephemeral-storage:                     15562841677
  ...
...
Allocated resources:
  Resource                              Requests      Limits
  --------                              --------      ------
  cpu                                   408m (11%)    0 (0%)
  memory                                2213Mi (20%)  0 (0%)
  ephemeral-storage                     0 (0%)        0 (0%)
  hugepages-1Mi                         0 (0%)        0 (0%)
  cex.s390.ibm.com/Accel                0             0
  cex.s390.ibm.com/CCA_for_customer_1   1             1
  cex.s390.ibm.com/EP11_for_customer_2  0             0
  ...
```

Each CEX device plug-in pod provides log messages, which provide details that might explain a possible failure or misbehavior. The logs of each of the CEX device plug-in instances can be extracted with the following command sequence:

```
$ kubectl get pods -n cex-device-plugin --no-headers | grep cex-plugin-daemonset
cex-plugin-daemonset-p5j8h   1/1   Running   0     32m
cex-plugin-daemonset-qdz8r   1/1   Running   0     32m
cex-plugin-daemonset-zxwts   1/1   Running   0     32m
$ kubectl logs -n cex-device-plugin cex-plugin-daemonset-p5j8h
$ kubectl logs -n cex-device-plugin cex-plugin-daemonset-qdz8r
$ kubectl logs -n cex-device-plugin cex-plugin-daemonset-zxwts
```

Here are some important parts of a sample CEX device plug-in log shown with some explanations:

```
 1: 2022/06/07 14:05:18 Main: S390 k8s z crypto resources plugin starting
 2: 2022/06/07 14:05:18 Plugin Version: v1.0.2
 3: 2022/06/07 14:05:18 Git URL:        https://github.com/ibm-s390-cloud/k8s-cex-dev-plugin.git
 4: 2022/06/07 14:05:18 Git Commit:     40fae46c3d3aacff055d5f2fd7e1c580abc850b9
```

Line 2: CEX device plug-in version.

Line 3-4: Source code and commit id base for this CEX device plug-in application.

```
 5: 2022/06/07 14:05:18 Main: Machine id is 'IBM-3906-00000000000DA1E7'
 6: 2022/06/07 14:05:18 Ap: apScanAPQNs() found 4 APQNs: (6,51,cex6,accel,true),
(8,51,cex6,cca,true), (9,51,cex6,cca,true), (10,51,cex6,ep11,true)
 7: 2022/06/07 14:05:18 CryptoConfig: Configuration changes detected
 8: 2022/06/07 14:05:18 CryptoConfig: Configuration successful updated
 9: 2022/06/07 14:05:18 Main: Crypto configuration successful read
10: 2022/06/07 14:05:18 CryptoConfig (3 CryptoConfigSets):
11: 2022/06/07 14:05:18   setname: 'CCA_for_customer_1'
12: 2022/06/07 14:05:18     project: 'customer_1'
13: 2022/06/07 14:05:18     5 equvialent APQNs:
14: 2022/06/07 14:05:18       APQN adapter=4 domain=51 machineid='*'
15: 2022/06/07 14:05:18       APQN adapter=8 domain=51 machineid='*'
16: 2022/06/07 14:05:18       APQN adapter=9 domain=51 machineid='*'
17: 2022/06/07 14:05:18       APQN adapter=12 domain=51 machineid='*'
18: 2022/06/07 14:05:18       APQN adapter=13 domain=51 machineid='*'
19: 2022/06/07 14:05:18   setname: 'EP11_for_customer_2'
20: 2022/06/07 14:05:18     project: 'customer_1'
21: 2022/06/07 14:05:18     3 equvialent APQNs:
22: 2022/06/07 14:05:18       APQN adapter=5 domain=51 machineid='*'
23: 2022/06/07 14:05:18       APQN adapter=10 domain=51 machineid='*'
24: 2022/06/07 14:05:18       APQN adapter=11 domain=51 machineid='*'
25: 2022/06/07 14:05:18   setname: 'Accel'
26: 2022/06/07 14:05:18     project: 'default'
27: 2022/06/07 14:05:18     3 equvialent APQNs:
28: 2022/06/07 14:05:18       APQN adapter=3 domain=51 machineid='*'
29: 2022/06/07 14:05:18       APQN adapter=6 domain=51 machineid='*'
30: 2022/06/07 14:05:18       APQN adapter=7 domain=51 machineid='*'
```

Line 6: The list of APQNs found by the CEX device plug-in instance on the compute node.

Lines 10-30: Condensed view of the CEX resource configuration.

```
...
40: 2022/06/07 14:05:18 PodLister: Start()
41: 2022/06/07 14:05:18 Plugin: Register plugins for these CryptoConfigSets: [Accel
CCA_for_customer_1 EP11_for_customer_2]
42: 2022/06/07 14:05:18 Plugin: Announcing 'cex.s390.ibm.com' as our resource namespace
43: 2022/06/07 14:05:18 Plugin: NewPlugin('EP11_for_customer_2')
44: 2022/06/07 14:05:18 Plugin['EP11_for_customer_2']: Start()
45: 2022/06/07 14:05:18 Plugin: Announcing 'cex.s390.ibm.com' as our resource namespace
46: 2022/06/07 14:05:18 Plugin: NewPlugin('Accel')
47: 2022/06/07 14:05:18 Plugin['Accel']: Start()
48: 2022/06/07 14:05:18 Plugin: Announcing 'cex.s390.ibm.com' as our resource namespace
49: 2022/06/07 14:05:18 Plugin: NewPlugin('CCA_for_customer_1')
50: 2022/06/07 14:05:18 Plugin['CCA_for_customer_1']: Start()
51: 2022/06/07 14:05:18 Plugin['Accel']: Found 1 eligible APQNs: (6,51,cex6,accel,true)
52: 2022/06/07 14:05:18 Plugin['Accel']: Overcommit not specified in ConfigSet, fallback to 1
53: 2022/06/07 14:05:18 Plugin['Accel']: Derived 1 plugin devices from the list of APQNs
54: 2022/06/07 14:05:18 Plugin['EP11_for_customer_2']: Found 1 eligible APQNs:
(10,51,cex6,ep11,true)
55: 2022/06/07 14:05:18 Plugin['EP11_for_customer_2']: Overcommit not specified in ConfigSet,
fallback to 1
56: 2022/06/07 14:05:18 Plugin['EP11_for_customer_2']: Derived 1 plugin devices from the list
of APQNs
57: 2022/06/07 14:05:18 Plugin['CCA_for_customer_1']: Found 2 eligible APQNs:
(8,51,cex6,cca,true), (9,51,cex6,cca,true)
58: 2022/06/07 14:05:18 Plugin['CCA_for_customer_1']: Overcommit not specified in ConfigSet,
fallback to 1
59: 2022/06/07 14:05:18 Plugin['CCA_for_customer_1']: Derived 2 plugin devices from the list of
APQNs
...
```

Lines 51, 54, 57: List of APQNs from the different CEX config sets that have been found on the compute node and are allocatable.

The following example shows a real allocation by a container:

```
...
70: 2022/06/07 14:17:03 Plugin['CCA_for_customer_1']:
Allocate(request=&AllocateRequest{ContainerRequests:
            []*ContainerAllocateRequest{&ContainerAllocateRequest{DevicesIDs:
[apqn-9-51-0],},},})
71: 2022/06/07 14:17:03 Plugin['CCA_for_customer_1']: creating zcrypt device node 'zcrypt-
apqn-9-51-0'
72: 2022/06/07 14:17:03 Zcrypt: Successfully created new zcrypt device node 'zcrypt-apqn-9-51-0'
73: 2022/06/07 14:17:03 Zcrypt: simple node 'zcrypt-apqn-9-51-0' for APQN(9,51) created
74: 2022/06/07 14:17:03 Shadowsysfs: shadow dir /var/tmp/shadowsysfs/sysfs-apqn-9-51-0 created
75: 2022/06/07 14:17:03 Plugin['CCA_for_customer_1']: Allocate()
response=&AllocateResponse{ContainerResponses:

[]*ContainerAllocateResponse{&ContainerAllocateResponse{Envs:map[string]string{},Mounts:
[]*Mount{&Mount{ContainerPath:/sys/bus/ap,HostPath:/var/tmp/shadowsysfs/
sysfs-apqn-9-51-0/bus/ap,ReadOnly:true,},&Mount{ContainerPath:/sys/devices/
ap,HostPath:/var/tmp/shadowsysfs/sysfs-apqn-9-51-0/devices/ap,ReadOnly:true,},},Devices:
[]*DeviceSpec{&DeviceSpec{ContainerPath:/dev/z90crypt,HostPath:/dev/zcrypt-
apqn-9-51-0,Permissions:rw,},},Annotations:map[string]string{},},},}
...
```

About every 30 seconds the list of running containers with allocated CEX resources is listed:

```
...
80: 2022/06/07 14:47:18 PodLister: 1 active zcrypt nodes
81: 2022/06/07 14:47:18 PodLister: 1 active sysfs shadow dirs
82: 2022/06/07 14:47:18 PodLister: Container 'cex-testload-1' in namespace 'default' uses CEX
resource 'apqn-9-51-0' marked for project 'customer_1'!!!
83: 2022/06/07 14:47:18 PodLister: 1 active containers with allocated cex devices
...
```

When containers terminate with an allocated CEX resource there is a cleanup step, which is reported in the log as follows:

```
...
90: 2022/06/07 14:52:18 PodLister: 1 active zcrypt nodes
91: 2022/06/07 14:52:18 PodLister: 1 active sysfs shadow dirs
92: 2022/06/07 14:52:18 PodLister: 0 active containers with allocated cex devices
93: 2022/06/07 14:52:18 PodLister: deleting zcrypt node 'zcrypt-apqn-9-51-0': no container use
```

```
since 120 s
94: 2022/06/07 14:52:18 PodLister: deleting shadow sysfs 'sysfs-apqn-9-51-0': no container use
since 120 s
...
```

Capturing debug data for support

If you submit a support case, provide debugging data. Describe the failure and the expected behavior and collect the logs of **all** CEX device plug-in instances together with the **currently active** CEX resource configuration map. Optionally, you can include the output of `kubectl describe nodes`. Be careful when providing this node data as internals of the load on the cluster might be exposed.

For example, run following commands to collect the required information:

```
$ cd /tmp
$ for p in `kubectl get pods -n cex-device-plugin --no-headers | grep cex-plugin-daemonset |
awk '{print $1}'`; do \
kubectl logs -n cex-device-plugin $p >$p.log; done
$ kubectl get configmap -n cex-device-plugin cex-resources-config -o yaml >cex-resources-
config.yaml
$ kubectl describe nodes >describe_nodes.log
$ zip debugdata.zip cex-plugin-daemonset-*.log cex-resources-config.yaml describe_nodes.log
$ rm cex-plugin-daemonset-*.log cex-resources-config.yaml describe_nodes.log
```

Note: The CEX device plug-in does not have access to any cluster or application secrets. Therefore, only administrative information, related to the APQNs that are managed by the plug-in, is logged. The logs contain the name of the configuration sets and the name and namespace of pods that request and use APQNs. Since no application, cluster, or company secrets are contained within the logs, it is safe to hand over this logging information to technical support.

Migrating from kube-system to cex-device-plugin namespace

This section describes how to move the CEX device plug-in from the `kube-system` namespace to its own namespace `cex-device-plugin`.

Migration details

The migration basically corresponds to a re-installation of the CEX device plugin and the corresponding CEX resource configuration in the new `cex-device-plugin` namespace.

NOTE: To prevent resource conflicts when two plug-ins try to manage the same device, you must first remove the plug-in from the `kube-system` namespace. This will lead to a short interruption in service such that pods with containers requesting a CEX resource cannot be scheduled. Already running containers with allocated CEX resources will not be affected and continue to run.

To assist in the migration, download the deployment kustomize files from the CEX device plug-in github repository and change to the download directory. With these files you can install the CEX devce plug-in with a custom configuration in the new namespace `cex-device-plugin`.

The configuration map is the only part that has to be moved. Everything else can simply be deleted in the `kube-system` namespace and freshly installed in the `cex-device-plugin` namespace. The kustomize templates for installation can be used to directly install the CEX device plug-in with a custom configmap.

Migration sequence

Follow these steps for migration:

1. Download the deployment Kustomize files from the CEX device plug-in github repository and change into the download directory.
2. Copy the `cex_resources.json` file content contained in the configuration map in the `kube-system` namespace into `deployments/configmap/cex_resources.json` by running the following command:

   ```
   oc get cm -n kube-system cex-resources-config -o jsonpath="{.data['cex_resources\.json']}" >
   deployments/configmap/cex_resources.json
   ```

3. Remove the old CEX device plug-in daemonset from the `kube-system` namespace by running the following command:

   ```
   oc delete daemonset cex-plugin-daemonset -n kube-system
   ```

   Now, no new pods with containers requesting CEX devices can be created anymore.
4. Create the installation template by running the following command:

   ```
   oc create -k deployments/rhocp-create
   ```

   After successful installation, new pods with containers requesting CEX devices can be created and the CEX device plug-in should be fully functional again.
5. Optionally, the `cex-resources-config` configmap in the `kube-system` namespace can be cleaned up. It is no longer needed since the new copy in the `cex-device-plugin` namespace is used. To delete the old configmap, run the following command:

   ```
   oc delete cm cex-resources-config -n kube-system
   ```

Appendix

Sample CEX resource configuration map

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cex-resources-config
  namespace: cex-device-plugin
data:
  cex_resources.json: |
    {
    "cryptoconfigsets":
    [
        {
            "setname":   "CCA_for_customer_1",
            "project":   "customer-1",
            "cexmode":   "cca",
            "overcommit": 3,
            "apqns":
            [
                {
                    "adapter":    1,
                    "domain":     6,
                    "machineid": ""
                },
                {
                    "adapter":    2,
                    "domain":     6,
                    "machineid": ""
                },
                {
                    "adapter":    7,
                    "domain":     6,
                    "machineid": ""
                }
            ]
        },
        {
            "setname":   "CCA_for_customer_2",
            "project":   "customer-2",
            "cexmode":   "cca",
            "overcommit": 4,
            "apqns":
            [
                {
                    "adapter":    1,
                    "domain":    11,
                    "machineid": ""
                },
                ...
                {
                    "adapter":    7,
                    "domain":    11,
                    "machineid": ""
                }
            ]
        },
        {
            "setname":   "EP11_for_customer_1",
            "project":   "customer-1",
            "cexmode":   "ep11",
            "apqns":
            [
                {
                    "adapter":    3,
                    "domain":     6,
                    "machineid": ""
                },
                ...
                {
                    "adapter":   11,
                    "domain":     6,
                    "machineid": ""
                }
```

```
            ]
        },
        {
            "setname":   "EP11_for_customer_2",
            "project":   "customer-2",
            "cexmode":   "ep11",
            "apqns":
            [
                {
                    "adapter":    3,
                    "domain":    11,
                    "machineid": ""
                },
                ...
                {
                    "adapter":   11,
                    "domain":    11,
                    "machineid": ""
                }
            ]
        },
        {
            "setname":   "Accel",
            "project":   "default",
            "cexmode":   "accel",
            "overcommit": 5,
            "apqns":
            [
                {
                    "adapter":    4,
                    "domain":     6,
                    "machineid": ""
                },
                ...
                {
                    "adapter":    5,
                    "domain":     6,
                    "machineid": ""
                }
            ]
        }
    ]
    }
```

Sample CEX device plug-in daemonset yaml

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: cex-plugin-daemonset
  namespace: cex-device-plugin
spec:
  selector:
    matchLabels:
      name: cex-plugin
  template:
    metadata:
      labels:
        name: cex-plugin
    spec:
      priorityClassName: system-cluster-critical
      serviceAccount: cex-plugin-sa
      serviceAccountName: cex-plugin-sa
      tolerations:
      - key: CriticalAddonsOnly
        operator: Exists
      # This Init Container defines settings for SELinux to enable the plug-in
      # to provide and modify a temporary file system. The file system can be
      # used to modify some sysfs entries for the container that uses CEX crypto
      # resources.  If the compute nodes do not have SELinux enabled, the Init
      # Container is not needed.
      initContainers:
      - name: shadowsysfs
        image: 'registry.redhat.io/ubi8-minimal'
        command: ["/bin/sh"]
        args: ["-c", "mkdir -p -m 0755 /var/tmp/shadowsysfs && chcon -t
container_file_t /var/tmp/shadowsysfs"]
        securityContext:
          privileged: true
```

```
        volumeMounts:
          - name: vartmp
            mountPath: /var/tmp
      containers:
      - name: cex-plugin
        image: 'quay.io/ibm/ibm-cex-plugin-cm:latest'
        imagePullPolicy: Always
        securityContext:
          privileged: true
        command: ["/work/cex-plugin"]
        env:
          # provide NODENAME to the container
          - name: NODENAME
            valueFrom:
              fieldRef:
                fieldPath: spec.nodeName
          # logically overcommit (share) CEX resources (if >1)
          - name: APQN_OVERCOMMIT_LIMIT
            value: "1"
        volumeMounts:
          - name: device-plugin
            mountPath: /var/lib/kubelet/device-plugins
          - name: pod-resources
            mountPath: /var/lib/kubelet/pod-resources
          - name: vartmp
            mountPath: /var/tmp
          - name: dev
            mountPath: /dev
          - name: sys
            mountPath: /sys
          - name: cex-resources-conf
            # the cex_resources.json file is showing up in this dir
            mountPath: /config/
      volumes:
        # device-plugin gRPC needs this
        - name: device-plugin
          hostPath:
            path: /var/lib/kubelet/device-plugins
        # pod-resources lister gRPC needs this
        - name: pod-resources
          hostPath:
            path: /var/lib/kubelet/pod-resources
        # plugin shadow sysfs mounts need this
        - name: vartmp
          hostPath:
            path: /var/tmp
        - name: dev
          hostPath:
            path: /dev
        - name: sys
          hostPath:
            path: /sys
        # cluster wide crypto cex resources config
        - name: cex-resources-conf
          configMap:
            name: cex-resources-config
```

Sample CEX crypto load container

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: testload-cca-for-customer-1
  namespace: customer-1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: testload-cca-for-customer-1
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: testload-cca-for-customer-1
    spec:
      containers:
      - image: 'bash'
        imagePullPolicy: Always
```

```
        name: testload-cca-for-customer-1
        command: ["/bin/sh", "-c", "while true; do echo do-nothing-loop; sleep 30; done"]
        resources:
          limits:
            cex.s390.ibm.com/CCA_for_customer_1: 1
```

Sample CEX quota restriction script

```
#!/bin/bash

# This script produces a yaml file with quota restrictions
# for the cex cryptosets for each given namespace.
# Apply the resulting yaml file and then only the namespace <nnn>
# is allowed  to allocate CEX resources from a crypto set
# marked with project <nnn>.

createquota () {
    QF=quota-$1.yaml
    cat << EOF >> $QF
- apiVersion: v1
  kind: ResourceQuota
  metadata:
    name: cex.$3
    namespace: $1
  spec:
    hard:
      requests.cex.s390.ibm.com/$2: 0
      limits.cex.s390.ibm.com/$2: 0
EOF
}

while ! test -z "$1"; do
    n=$1
    shift
    c=0
    echo "apiVersion: v1" > quota-$n.yaml
    echo "items:" >> quota-$n.yaml
        for s in `oc get cm cex-resources-config -n cex-device-plugin -o
jsonpath='{.data.cex_resources\.json}'
                  | jq -r ".cryptoconfigsets | .[] | select(.project != \"$n\") | .setname"`; do
        c=$(( c + 1 ))
        createquota $n $s $c
    done
    echo "kind: List" >> quota-$n.yaml
    echo "metadata: {}" >> quota-$n.yaml
    ## TODO: apply it
done
```

Sample CEX Prometheus exporter yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cex-prometheus-exporter
  namespace: cex-device-plugin
spec:
  replicas: 1
  selector:
    matchLabels:
      app: cex-prometheus-exporter
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: cex-prometheus-exporter
    spec:
      serviceAccount: cex-prometheus-exporter-sa
      serviceAccountName: cex-prometheus-exporter-sa
      containers:
        - name: cex-prometheus-exporter
          image: 'quay.io/ibm/ibm-cex-plugin-cm:latest'
          imagePullPolicy: Always
          securityContext:
            allowPrivilegeEscalation: false
            capabilities:
```

```
            drop: ["ALL"]
          runAsNonRoot: true
          seccompProfile:
            type: RuntimeDefault
        command: ["/work/cex-prometheus-exporter"]
        ports:
          - containerPort: 9939
            name: prommetrics
          - containerPort: 12358
            name: collector
```

Sample CEX Prometheus exporter collector service yaml

```
apiVersion: v1
kind: Service
metadata:
  name: cex-prometheus-exporter-collector-service
  namespace: cex-device-plugin
  labels:
    app: cex-prometheus-exporter
spec:
  type: ClusterIP
  selector:
    app: cex-prometheus-exporter
  ports:
  - name: collector
    port: 12358
    protocol: TCP
    targetPort: collector
```

Sample CEX Prometheus exporter servicemonitor yaml

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: cex-prometheus-exporter
  namespace: cex-device-plugin
  labels:
    release: prometheus
spec:
  selector:
    matchLabels:
      app: cex-prometheus-exporter
  endpoints:
  - port: metrics
    interval: 15s
    scheme: http
```

Sample CEX Prometheus exporter service yaml

```
apiVersion: v1
kind: Service
metadata:
  name: cex-prometheus-exporter
  namespace: cex-device-plugin
  labels:
    app: cex-prometheus-exporter
spec:
  type: ClusterIP
  selector:
    app: cex-prometheus-exporter
  ports:
  - name: metrics
    port: 9939
    protocol: TCP
    targetPort: prommetrics
```

Environment variables

Environment variables recognized by the CEX plug-in application

Name	Default value	Description
APQN_CHECK_INTERVAL	30	The interval in seconds to check for the node APQNs available and their health state. The minimum is 10 seconds.
APQN_OVERCOMMIT_LIMIT	1	The overcommit limit, 1 defines no overcommit. For details see Overcommitment of CEX resources
CEX_PROM_EXPORTER_COLLECTOR_SERVICE_NAMESPACE		The namespace in which the CEX Prometheus exporter will run. If empty (the default) it is assumed that CEX plug-in instances and the CEX Prometheus exporter run in the same namespace.
CEX_PROM_EXPORTER_COLLECTOR_SERVICE_PORT	12358	The port number where the CEX plug-in instances will contact the CEX Prometheus exporter to deliver their raw metrics data.
CEX_PROM_EXPORTER_COLLECTOR_SERVICE	cex-prometheus-exporter-collector-service	The name of the service where the CEX plug-in instance will contact the CEX Prometheus exporter.
CRYPTOCONFIG_CHECK_INTERVAL	120	The interval in seconds to check for changes on the cluster-wide CEX resource configmap. The minimum is 120 seconds.
METRICS_POLL_INTERVAL	15	The interval in seconds to internally poll base information (like crypto counters) and update the internal metrics data. The minimum is 10 seconds.
NODENAME		The name of the node where the CEX device plug-in instance runs. See the sample CEX plug-in daemonset yaml to set up this environment variable correctly.
PODLISTER_POLL_INTERVAL	30	The interval in seconds to fetch and evaluate the pods within the cluster, which have CEX resources allocated. The minimum is 10 seconds.

Name	Default value	Description
RESOURCE_DELETE_NEVER_USED	1800	The interval in seconds after which an allocated CEX resource requested by a starting pod is freed when the pod never came into the running state. The minimum is 30 seconds.
RESOURCE_DELETE_UNUSED	120	The interval in seconds after which an allocated CEX resource is freed when the pod vanished from the running pods list. The minimum is 30 seconds.
SHADOWSYSFS_BASEDIR	/var/tmp/shadowsysfs	The base directory for the shadow sysfs. For details see The shadow sysfs

Environment variables recognized by the CEX Pometheus exporter application

Name	Default value	Description
COLLECTOR_SERVICE_PORT	12358	The metrics collector listener port, where the CEX plug-in instances will deliver their raw metrics data.
PROMETHEUS_SERVICE_PORT	9939	The Prometheus client port where the Prometheus server will fetch the metrics from.

Additional resources

- IBM z15 Configuration Setup
  https://www.redbooks.ibm.com/abstracts/sg248860.html
- Linux on Z and LinuxONE
  https://www.ibm.com/docs/en/linux-on-systems?topic=linux-z-linuxone
- CryptoCards
  https://www.ibm.com/security/cryptocards
- System z Crypto and TKE Update
  https://www.redbooks.ibm.com/abstracts/sg247848.html
- CCA - Common Cryptographic Architecture functional overview
  https://www.ibm.com/docs/en/linux-on-systems?topic=cca-overview
- Kubernetes Device Plug-ins
  https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/device-plugins
- Device Plugin Manager
  https://pkg.go.dev/github.com/kubevirt/device-plugin-manager/pkg/dpm

Notices

This information was developed for products and services offered in the U.S.A. This material might be available from IBM® in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property rights may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing

IBM Corporation

North Castle Drive, MD-NC119

Armonk, NY 10504-1785

United States of America

For license inquiries regarding double-byte \(DBCS\) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing

Legal and Intellectual Property Law

IBM Japan Ltd.

19-21, Nihonbashi-Hakozakicho, Chuo-ku

Tokyo 103-8510, Japan

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product\(s\) and/or the program\(s\) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who want to have information about it for the purpose of enabling: \(i\) the exchange of information between independently created programs and other programs \(including this one\) and \(ii\) the mutual use of the information which has been exchanged, should contact

IBM Director of Licensing

IBM Corporation

North Castle Drive, MD-NC119

Armonk, NY 10504-1785

United States of America

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

Portions of this information are provided under the Apache v2 license https://www.apache.org/licenses/LICENSE-2.0.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Trademarks

Helm™ and the Helm logo are trademarks of The Linux® Foundation, and use of them as a trademark is subject to The Linux Foundation's Trademark Usage Guidelines at https://www.linuxfoundation.org/trademark-usage/.

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and trademark information at www.ibm.com/legal/copytrade.shtml.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

The registered trademark Linux is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Kubernetes® and the Kubernetes logo are registered trademarks of The Linux Foundation, and use of them as a trademark is subject to The Linux Foundation's Trademark Usage Guidelines at https://www.linuxfoundation.org/trademark-usage/.

Red Hat®, OpenShift®, and Ansible® are registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

UNIX® is a registered trademark of The Open Group in the United States and other countries.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

- **Applicability**

  These terms and conditions are in addition to any terms of use for the IBM® website.

- **Personal use**

  You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM®.

- **Commercial use**

  You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM®.

- **Rights**

  Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

  IBM® reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM®, the above instructions are not being properly followed.

  You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

  IBM® MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Printed in USA