# Kubernetes device plug-in for IBM Crypto Express (CEX) cards
Version 1.0

*Installation and User Guide*

IBM

# Contents

# Version 1.0 release notes

Version 1.0 of the Kubernetes device plug-in for IBM Crypto Express (CEX) cards is the initial release of this new 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

## 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 s390 Crypto Express (CEX) cards for s390 Kubernetes container loads. 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 version 1 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 life time of the container.

The cluster-wide configuration of the CEX crypto resources is kept in a Kubernetes `ConfigMap` within the `kube-system` namespace. The `ConfigMap` must be named `cex-resources-config`. The content of the `ConfigMap` 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: kube-system
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.

## 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 Kubernetes namespace `kube-system`. To deploy the clusterwide CEX `ConfigMap` using a yaml enter:

```
kubectl create -f <my_cex_resources.yaml>
```

Updates and maintenance steps are described in CEX configuration ConfigMap updates.

# 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

The CEX device plug-in image needs to be run on each compute node with administrator privileges. Kubernetes uses the concept of a *DaemonSet* for this kind of cluster-wide service. The git repository shows a sample daemonset yaml file, which provides all the needed settings and options to run the CEX device plug-in as Kubernetes daemonset. A working sample is provided in the appendix Sample CEX device plug-in daemonset yaml.

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

- `namespace`: The CEX device plug-in instances need to run in namespace `kube-system` as this is 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`: The commands needed to deal with SELinux have been outsourced into an InitContainer. The `InitContainer` entry can be safely removed from the yaml file when there is no need.

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

```
kubectl create -f <my_cex_plug-in_daemonset.yaml>
```

A few seconds later a pod 'cex-plug-in' in namespace `kube-system` 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 kube-system` 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 kube-system
```

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.

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.

## 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.

# Appendix

## Sample CEX resource configuration map

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cex-resources-config
  namespace: kube-system
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": ""
                }
            ]
        },
        {
            "setname":   "CCA_for_customer_2",
            "project":   "customer-2",
            "cexmode":   "cca",
            "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",
            "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-plug-in-daemonset
  namespace: kube-system
spec:
  selector:
    matchLabels:
      name: cex-plug-in
  template:
    metadata:
      labels:
        name: cex-plug-in
    spec:
      priorityClassName: system-cluster-critical
      tolerations:
      - key: CriticalAddonsOnly
        operator: Exists
      initContainers:
      - name: shadowsysfs
        image: 'registry.access.redhat.com/ubi8/ubi-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:
      - image: 'image-registry.openshift-image-registry.svc:5000/kube-system/cex-plugin-
v1:latest'
        imagePullPolicy: Always
        name: cex-plug-in
        securityContext:
          privileged: true
        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-plug-in
            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-plug-in gRPC needs this
        - name: device-plug-in
          hostPath:
            path: /var/lib/kubelet/device-plugins
        # pod-resources lister gRPC needs this
        - name: pod-resources
          hostPath:
            path: /var/lib/kubelet/pod-resources
        # plug-in 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-$n.yaml
        for s in `oc get cm cex-resources-config -n kube-system -o
jsonpath='{.data.cex_resources\.json}'
                  | jq -r ".cryptoconfigsets | .[] | select(.project != \"$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
```

## 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

## Trademarks

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

## Terms and conditions for product documentation

Printed in USA