Falcon Container Sensor for Linux Deployment

Last updated: Dec. 6, 2021

Contents:

Overview

The Falcon Container sensor for Linux extends runtime security to container workloads in Kubernetes clusters that don't allow you to deploy the kernel-based Falcon sensor for Linux. The Falcon Container sensor runs as an unprivileged container in user space with no code running in the kernel of the worker node OS. This allows it to secure Kubernetes pods in clusters where it isn't possible to deploy the kernel-based Falcon sensor for Linux on the worker node, as with AWS Fargate where organizations don't have access to the kernel and where privileged containers are disallowed. The Falcon Container sensor can also secure container workloads on clusters where worker node security is managed separately from application security.

The Falcon Container sensor runs inside each application pod in the cluster as a container alongside application containers in the pod. It tracks activity in all of the pod's application containers and sends telemetry to the CrowdStrike cloud. While the Falcon Container sensor is scoped to the pod where it runs, its functionality is

otherwise similar to that of the kernel-based Falcon sensor for Linux. In particular, it generates detections and performs prevention operations for activity in those application containers.

Installation of the Falcon Container in a Kubernetes cluster is automated so that each pod in the cluster is automatically injected with a Falcon Container sensor when the pod is scheduled to run on the cluster.

> **Note**: The Falcon Container allows privilege escalation by the application container (allowPrivilegeEscalation: true) by temporarily setting these attributes in the injected pod spec. These additional attributes are removed before the application starts:

- `runAsUser : 0, runAsGroup: 0`
- `SYS_ADMIN,SETUID,SETGID,CHOWN,MKNOD`

> **Note:** In Kubernetes clusters where kernel module loading is supported by the worker node OS, we recommend using the existing Falcon sensor for Linux to secure both worker nodes and containers with a single sensor. on each worker node. See Falcon sensor for Linux Deployment Guide [/documentation/20/falcon-sensor-for-linux] and Falcon Discover for Cloud and Containers [/documentation/114/falcon-discover-for-cloud-and-containers#containers] .

For more info about Falcon Container detections and prevention operations, see Container Security [/documentation/145/container-security#falcon-container] .

Falcon Container components

Falcon Container consists of these main components:

- Falcon sensor: At runtime, the Falcon Container sensor is injected into each pod of the Kubernetes cluster as a container. It uses unique technology to run in the application context.

  Note: `crowdstrike-falcon-container` is a CrowdStrike-distinguished container name for the Falcon Container sensor for Linux. If you have an application container with this name in a monitored pod, the deployment will fail and an error message will appear in kubectl events and in the CrowdStrike injector logs.

- Falcon stub: The Falcon Container sensor uses a stub program, Falcon stub, that acts as an entry point for each workload. At runtime, the Falcon stub is transparently injected before the entry point of the application container. If a pod's spec has more than one application container, a Falcon stub is injected before the entry point into each container (except in Init containers). At container start, each Falcon stub connects to the Falcon Container sensor in the pod to enable the Falcon Container sensor to perform monitoring and prevention actions on the application container.

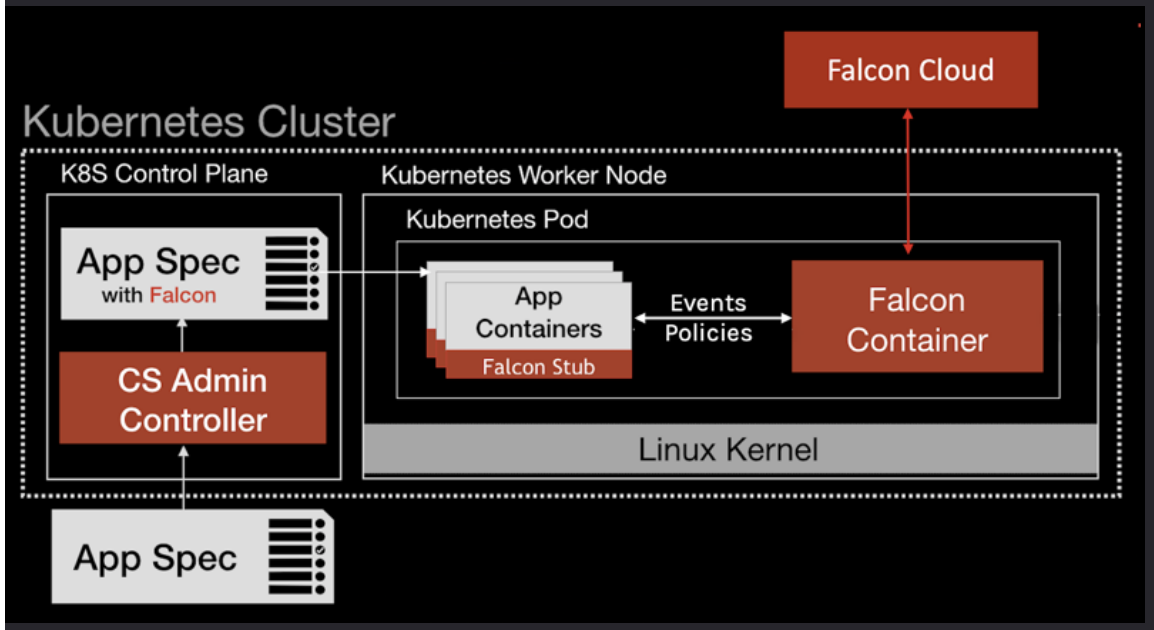
- Falcon injector service: When Falcon Container is installed in a cluster, an injector service is deployed into the cluster as an admission controller in a new namespace named falcon-system. This injector updates the pod specifications to inject the falcon-sensor container into each pod and the falcon-stub as the entry point of each application container.

  If the **command** field is defined in a pod spec, the Falcon injector uses it to inject the falcon-stub as the container entry point. If a container in a pod spec doesn't have a **command** field, the Falcon injector can retrieve the **EntryPoint** and **Cmd** of a container image directly from the container registry and set the

falcon-stub as the container entry point.

For a private registry, the injector needs to access the registry using the image pull secrets specified in the pod. To do this, the installer generates a manifest that includes new RBAC rules. These rules allow the injector service account to access the secret resources cluster-wide.

For details about RBAC rules, see https://kubernetes.io/docs/reference/access-authn-authz/rbac/#rolebinding-and-clusterrolebinding [https://kubernetes.io/docs/reference/access-authn-authz/rbac/#rolebinding-and-clusterrolebinding]



Falcon Container sensor image components

The Falcon Container sensor image includes these major components:

- Installer: Generates the manifest for the injector deployment.
- Injector: Updates pod specifications at runtime so that the falcon-sensor container is injected into each created pod and the falcon-stub is injected into each application container.

  > **Note:** Existing pods do not get a sensor unless they are deleted and re-created.

- Sensor (falcon-sensor): The sensor application.

Requirements

Subscription: Cloud Workload Protection (CWP)

Supported Kubernetes environments: You must have a running Kubernetes cluster in one of these supported Kubernetes environments:

Environment	Minimum Kubernetes version	Minimum Falcon Container sensor version
Amazon Elastic Kubernetes Service (EKS) with pods on Linux nodes such as Amazon Linux 2 or Bottlerocket	Kubernetes v1.17 or later	6.18 or later (6.27 or later for Bottlerocket)
Amazon Elastic Kubernetes Service (EKS) with pods on AWS Fargate	Kubernetes v1.17 or later	6.18 or later
Google Kubernetes Engine (GKE) with pods on cos_containerd	Kubernetes v1.18 or later	6.23 or later
Azure Kubernetes Service (AKS) with pods on Linux nodes such as Ubuntu 18.04	Kubernetes v1.18 or later	6.23 or later
OpenShift 4.7 with pods on Red Hat CoreOS	Kubernetes 1.20 or later	6.28 or later

Roles: Users can see pod information in the Activity, Host Management, Prevention Policies, and Cloud Security areas of the console

Installation prerequisites

Before installing Falcon Container, make sure these prerequisites are in place:

- Admin rights: You must have knowledge of, and be familiar with managing, Kubernetes environments, and you must have admin rights to create new resources on Kubernetes. Specifically, you must have permissions to create these Kubernetes resources:
    - Namespace
    - ConfigMap
    - Secret
    - Deployment (ReplicaSet and Pod)
    - Service

- MutatingWebhookConfiguration
- ClusterRole and ClusterRoleBinding
    - These resource names are reserved for Falcon Container sensor:

Resource type	Reserved resource name
Namespace	falcon-system
Pod.spec.volumeMounts	crowdstrike-falcon-volume
Pod.spec.initContainers	crowdstrike-falcon-init-container
Pod.spec.containers	crowdstrike-falcon-container
MutatingWebhookConfiguration	injector.falcon-system.svc
ClusterRole	secret-reader
ClusterRoleBinding	read-secrets-global
Secret (All monitored Namespace)	crowdstrike-falcon-pull-secret

- Correctly configured kubectl: Your kubectl must be able to connect to the cluster. To check the connection, run this command:

  ```
  kubectl get ns
  ```

  You have a valid connection if the namespaces available in the cluster are listed in the output.

- Host PID namespace: The application containers can't be configured to use the host's PID namespace.
- Private container registry: You must have pull and push access to the registry where the Falcon Container sensor image will be hosted. Kubernetes worker nodes must be able to access it and pull an image from it.
- OpenShift Security Context Constraints (SCC) have these requirements for falcon-container-sensor:
  - Deploy the injector in SCC *restricted* (default SCC).
  - falcon-container-sensor requirements change based on the application container configuration:
    - If all application containers within a pod have runAsUser/runAsGroup explicitly mentioned in the pod spec with the same non-root values, make this update to the SCC used by the application:

      `allowedCapabilities:`

      `- SYS_PTRACE`
    - In all other cases, make these updates to the SCC used by the application:
      - `allowedCapabilities:`

        `-SYS_PTRACE`
      - `runAsUser:`

        `type: RunAsAny`
- In an OpenShift cluster with both Linux and Windows worker nodes, falcon-container-sensor is injected only if the `nodeSelector` value is `linux`. If the `nodeSelector` is not specified, the injector assumes it is `linux`.

Installing the Falcon Container sensor for Linux

You can install the Falcon Container sensor for Linux from either of these locations:

- Sensor Downloads page [/documentation/146/falcon-container-sensor-for-linux#installing-the-sensor-from-sensor-downloads]
- CrowdStrike Container Registry [/documentation/146/falcon-container-sensor-for-linux#installing-the-sensor-from-crowd-strike-container-registry]

Installing the sensor from Sensor Downloads

Note: The sensor is not installed on existing pods. Restart or re-create all pods using a previous sensor version to use a new sensor version.

1. To tag and push the image to the registry used by the cluster nodes, set a variable named REGISTRY_BASE_URL for your registry base URL. For example:REGISTRY_BASE_URL=123456789012.xyz.ecr.us-east-1.amazonaws.com

2. If the registry requires authentication, a Kubernetes docker pull secret must be present in all of the namespaces where you want to install the Falcon Container sensor, otherwise the pods might fail to pull the Falcon Container sensor image. You must have a base64 encoded string of the docker config json. This is used to create the Kubernetes secret based on the docker credentials. This secret is used to pull images from your private registry. To get the base64 string, run this command: `cat ~/.docker/config.json | base64 -w 0`

   Note: This option isn't required for EKS, GCR, or AKS deployments with Falcon container images and application images hosted on ECR, GCR or ACR, respectively.

3. Set a variable named IMAGE_PULL_TOKEN for your pull secret base64 encoded string and a variable named REGISTRY_USER for the registry username. For example:

   ```
   IMAGE_PULL_TOKEN=UmVhbGx5IHJlYWxseSByZWVlZWVlZWFhYWFhYWFhYWFhYWFhYWFhYWFhYWxsbGxsbGxsbGxsbGxsbGxsbGxsbGxsbGx5eXl5eXl5eXl5eXl5eXl5eSBsbGxsbGxsbGxsbG9vb29vb29vb29vb29vb29vb29vb25ubm5ubm5ubm5ubm5ubm5ubm5ubmdnZ2dnZ2dnZ2dnZ2dnZ2cgYXV0aCBrZXlzCg==REGISTRY_USER=dockeruser
   ```

   Note: For more information, see https://kubernetes.io/docs/tasks/configure-pod-container/pull-image-private-registry/#registry-secret-existing-credentials [https://kubernetes.io/docs/tasks/configure-pod-container/pull-image-private-registry/#registry-secret-existing-credentials].

   Note: This option isn't required for EKS, GCR, or AKS deployments with Falcon container images and application images hosted on ECR, GCR, or ACR, respectively.

4. Set a variable named NAMESPACES for the namespaces you want to monitor. Use a comma-separated list. For example: `NAMESPACES=ns1,ns2,ns3`

   Note: If the NAMESPACES variable is not set, the docker pull secret is only created in the default namespace.

   Note: This option isn't required for EKS, GCR, or AKS deployments with Falcon container images and application images hosted on ECR, GCR, or ACR, respectively.

5. Download the Falcon Container sensor image from **Hosts > Sensor Downloads [/hosts/sensor-downloads]**, or using the sensor download API. This container image is a tarball (.tar) file.

   Note: For info about sensor download APIs, see Sensor Download APIs [/documentation/109/sensor-download-apis&sa=D&ust=1611864598139000&usg=AOvVaw34xKjoO4H59zc7N5G5NrXF].

6. Save the path of the downloaded image as the variable IMAGE_TAR_PATH. For example:

   ```
   IMAGE_TAR_PATH=~/Downloads/falcon-sensor-xxxx.container.x86_64.tar.bz2
   ```

7. Copy your Customer ID Checksum (CID). This is displayed on the **Sensor Downloads** page. Save it as the variable CID: For example: CID=aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa-aa

   Note: The CID must be lowercase.

8. Load the falcon image tarball into your local system: `docker load -i $IMAGE_TAR_PATH`

9. Extract the image name and create a variable named FALCON_IMAGE:

   ```
   FALCON_IMAGE=$(docker images --filter=reference='falcon-sensor' --format '{{.Repository}}:{{.Tag}}')
   ```

10. Ensure the image is loaded in your local docker images store: `echo $FALCON_IMAGE`

11. Tag the image with the URI to your registry with these commands:

    - `FALCON_IMAGE_URI=$REGISTRY_BASE_URL/$REGISTRY_USER/$FALCON_IMAGE`
    - `docker tag $FALCON_IMAGE $FALCON_IMAGE_URI`

12. Push the image to your private registry: `docker push $FALCON_IMAGE_URI`

13. Install the Falcon Container sensor injector into each of your Kubernetes clusters. You can run the installer command from the Falcon image to generate a full yaml manifest with all resources required to set up the sensor injector in the Kubernetes cluster. These resources can be piped to the kubectl create command.

    ```
    docker run --rm $FALCON_IMAGE_URI \
    -cid $CID -image $FALCON_IMAGE_URI  \
    -pulltoken “$IMAGE_PULL_TOKEN” -namespaces “$NAMESPACES” \
    | tee /tmp/falcon-container.yaml | kubectl create -f -
    ```

    Note: For AKS without the -pulltoken option, use `-azure-config "$AZURE_CONFIG_FILE"` where `AZURE_CONFIG_FILE` = `/etc/kubernetes/azure.json"`

    Note: You must install the Falcon Container sensor injector into each cluster you want to monitor.Note: In cases where the Falcon Container sensor injector is configured to monitor all namespaces, explicitly label the kube-public and kube-system namespace to not be monitored. This ensures that new pod creation

requests pertaining to Kubernetes control plane are not forwarded to the injector:

- `kubectl label namespace kube-system sensor.falcon-system.crowdstrike.com/injection=disabled`
- `kubectl label namespace kube-public sensor.falcon-system.crowdstrike.com/injection=disabled`

14. Ensure the injector service is running using this command: `kubectl get pod -n falcon-system`

Installing the sensor from CrowdStrike Container Registry

The Falcon Container image can be pulled directly from the CrowdStrike container registry. The Falcon Container registry requires authentication credentials that are retrieved from the Falcon API.

> Note: The registry download is available for these clouds and versions:

- US-1, US-2, EU-1: 6.28.1101 and later
- US-GOV-1: 6.31.1409 and later

> **Note**: The sensor is not installed on existing pods. Restart or re-create all pods using a previous sensor version to use a new sensor version.

1. In the Falcon console, go to **Support > API Clients and Keys**.
2. Click **Add new API client**.
3. For the **Falcon Images Download** option in the list, select **Read**.
4. Click **Add** and note the Client ID and Client Secret.
5. Log in to CrowdStrike's API gateway using `curl` to get an OAuth2 token:

```
[YOUR CLOUD] US-1 = api
[YOUR CLOUD] US-2 = api.us-2
[YOUR CLOUD] EU-1 = api.eu-1
[YOUR CLOUD] GOV-1 = api.laggar.gcw
RESPONSE=$(curl \
--data "client_id=${CS_CLIENT_ID}&client_secret=${CS_CLIENT_SECRET}" \
--request POST \
--silent \
https://[YOUR CLOUD].crowdstrike.com/oauth2/token)
```

6. Use the token returned as a Bearer Token to the API:

```
curl -i -X GET -H "authorization: Bearer ${BEARER_TOKEN}" \
https://[YOUR CLOUD].crowdstrike.com/container-security/entities/image-registry-credentials/v1
```

7. Copy your Customer ID (CID) from the **Sensor Downloads** page and save it as the variable CID. For example: CID=aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

   Note: The CID must be lowercase.

8. Use the token returned as ART_PASSWORD and ART_USERNAME is fc-${CID}

```
[YOUR CLOUD LOWER] US-1 = us-1
[YOUR CLOUD LOWER] US-2 = us-2
[YOUR CLOUD LOWER] EU-1 = eu-1[YOUR CLOUD LOWER] GOV-1 = govcloud
[YOUR REGISTRY] US-1 = registry
[YOUR REGISTRY] US-2 = registry
[YOUR REGISTRY] EU-1 = registry
[YOUR REGISTRY] GOV-1 = registry.laggar.gcw
[YOUR CLOUD] US-1 = US-1
[YOUR CLOUD] US-2 = US-2
[YOUR CLOUD] EU-1 = EU-1
[YOUR CLOUD] GOV-1 = US-GOV-1
docker login --username  ${ART_USERNAME} --password ${ART_PASSWORD} [YOUR
REGISTRY].crowdstrike.com
```

9. Set FALCON_IMAGE_URI to

```
[YOUR REGISTRY].crowdstrike.com/falcon-container/[YOUR CLOUD LOWER]/release/falcon-sensor:
<VERSION>.container.x86_64.Release.[YOUR CLOUD]
```

   Note: See Falcon Container Sensor release notes [https://supportportal.crowdstrike.com/s/release-notes] for the <VERSION> value.

10. Run this command to get a base64 encoded string of the Docker config JSON: `cat ~/.docker/config.json | base64 -w 0`

    This string is used to create the Kubernetes secret based on the Docker credentials, and the secret is used to pull images from your private registry.

    Note: To pull a Falcon Container sensor image for each deployment, the Kubernetes cluster must have authentication configured as a Kubernetes Docker pull secret in all namespaces where you want to install the Falcon Container sensor. If not, the pods might fail to pull the Falcon Container sensor image.

11. Set a variable named IMAGE_PULL_TOKEN. For example:

```
IMAGE_PULL_TOKEN=UmVhbGx5IHJlYWxseSByZWVlZWVlZWVlZWFhYWFhYWFhYWFhYWFhYWFhYWFhYWxsbGxsbGxsbGxsbGxsbGxsbGxsbGxsbGx5eXl5eXl5eXl5eXl5eXl5eSBsbGxsbGxsbGxsbG9vb29vb29vb29vb29vb29vb29vb25ubm5ubm5ubm5ubm5ubm5ubmdnZ2dnZ2dnZ2dnZ2dnZ2cgYXV0aCBrZXlzCg==
```

12. Set a variable named NAMESPACES for the namespaces you want to monitor. Use a comma-separated list. For example: `NAMESPACES=ns1,ns2,ns3`

    Note: If the NAMESPACES variable is not set, the docker pull secret is only created in the default namespace.

13. Install the Falcon Container sensor injector into each of your Kubernetes clusters. You can run the installer command from the Falcon image to generate a full yaml manifest with all resources required to set up the sensor injector in the Kubernetes cluster. These resources can be piped to the kubectl create command.

```
docker run --rm $FALCON_IMAGE_URI \-cid $CID -image $FALCON_IMAGE_URI  \-pulltoken “$IMAGE_PULL_TOKEN” -namespaces “$NAMESPACES” \| tee /tmp/falcon-container.yaml | kubectl create -f -
```

    Note: You must install the Falcon Container sensor injector into each cluster you want to monitor.

    Note: In cases where the Falcon Container sensor injector is configured to monitor all namespaces, explicitly label the kube-public and kube-system namespace to not be monitored. This ensures that new pod creation requests pertaining to Kubernetes control plane are not forwarded to the injector:

    - `kubectl label namespace kube-systemsensor.falcon-system.crowdstrike.com/injection=disabled`
    - `kubectl label namespace kube-publicsensor.falcon-system.crowdstrike.com/injection=disabled`

14. Ensure the injector service is running using this command: `kubectl get pod -n falcon-system`

Additional installation options

To view options available with the installer, run this command:

```
docker run --rm $FALCON_IMAGE_URI -help
```

Always pull installation option

Always pull the Falcon Container image: Use the `-pullpolicy` argument when installing the Falcon Container sensor injector on your Kubernetes cluster:

```
docker run --rm $FALCON_IMAGE_URI \

-cid $CID -image $FALCON_IMAGE_URI  \

-pulltoken "$IMAGE_PULL_TOKEN" -namespaces "$NAMESPACES" \

-pullpolicy Always \

| tee tmp/falcon-container.yaml | kubectl create -f -
```

Falconctl configuration options

Pass falconctl options as environment variables using the `-falconctl-opts` argument. The `-falconctl-opts` argument must use this format:

```
-falconctl-opts "--[option]=[value] --<option>=<value>"
```

For example:

```
docker run --rm $FALCON_IMAGE_URI \

-cid $CID -image $FALCON_IMAGE_URI  \

-pulltoken "$IMAGE_PULL_TOKEN" -namespaces "$NAMESPACES" \

-falconctl-opts "--tags='k8s-app,production-env' --billing=metered --provisioning-token=ABCD1234" \

| tee tmp/falcon-container.yaml | kubectl create -f -
```

You can use these falconctl options during Falcon Container installation (**Note**: These options are deprecated for sensor version 6.27 and later):

Environment Variable	Corresponding falconctl option

Environment Variable	Corresponding falconctl option
`FALCONCTL_OPT_APD`	`--apd`
`FALCONCTL_OPT_APH`	`--aph`
`FALCONCTL_OPT_APP`	`--app`
`FALCONCTL_OPT_FEATURE`	`--feature`
`FALCONCTL_OPT_BILLING`	`--billing`
`FALCONCTL_OPT_TAGS`	`--tags`
`FALCONCTL_OPT_PROVISIONING_TOKEN`	`--provisioning-token`

You can get a list of the supported options and help strings by calling falconctl directly from the container with no parameters. For example:

```
docker run --rm $FALCON_IMAGE_URI /usr/bin/falconctl
```

Managing sensor grouping tags

Sensor grouping tags are optional user-defined identifiers you can use to group and filter pods. You can specify sensor grouping tags using the `--tags` falconctl option as the `FALCONCTL_OPT_TAGS` environment variable.

> Note: For information on Falcon grouping tags, which are managed through the Falcon console or CrowdStrike API, see Using grouping tags [/documentation/67/host-and-host-group-management#using-grouping-tags]. Falcon grouping tags can't be specified at the sensor CLI and can't be used with sensor images or templates.

Tags can include these characters:

- letters (`a-z` , `A-Z`)
- numbers (`0-9`)
- hyphens (`-`)
- underscores (`_`)
- forward slashes (`/`)

Tags can't include spaces ( ) or commas (`,`) or the equal sign (`=`).

To assign multiple tags, separate each tag with commas. All tags for a host, including comma separators, cannot exceed 256 characters.

For example:

```
docker run --rm $FALCON_IMAGE_URI \

-cid $CID -image $FALCON_IMAGE_URI  \

-pulltoken “$IMAGE_PULL_TOKEN” -namespaces “$NAMESPACES” \

-falconctl-opts "--tags=' k8s-app,production-env'" \

| tee tmp/falcon-container.yaml | kubectl create -f -
```

Configuring sensor resources

To customize the sensor resources, use the `-sensor-resources` option. Format this option as a json string or as base64.

For example, to specify CPU of 500m and memory of 1Gi, use this command:

```
docker run --rm $FALCON_IMAGE_URI \
-cid $CID -image $FALCON_IMAGE_URI  \
-pulltoken "$IMAGE_PULL_TOKEN" -namespaces "$NAMESPACES" \
-sensor-resources '{"requests":{"cpu":"500m","memory":"1Gi"}}' \
| tee tmp/falcon-container.yaml | kubectl create -f -
```

To remove the default resources altogether, pass an empty object in the command:

```
-sensor-resources '{}'
```

Configuring the injector listen port

Use this installer option if you need to configure the injector pod to use the node network namespace (when you're setting `hostNetwork: true`). By default, the injector pod is set up to run in its own network namespace and you don't need to configure the listen port.

- You can specify a different listen TCP port using this option, in case the default injector port (4433) conflicts with the port used by other processes running on the node.
- Use the flag `-injector-listen-port` to specify any alternative port number between 1024 and 32767.

```
docker run --rm $FALCON_IMAGE_URI \
-cid $CID -image $FALCON_IMAGE_URI  \
-pulltoken "$IMAGE_PULL_TOKEN" -namespaces "$NAMESPACES" \
-injector-listen-port=12345 \
| tee tmp/falcon-container.yaml | kubectl create -f -
```

Restricting Ingress traffic to the injector pod

- We recommend allowing Ingress traffic only from the kube-api-server to the injector pod in the falcon-system namespace. This protects the injector pod from denial-of-service (DOS) attacks.
- To restrict Ingress traffic to the falcon-system namespace running the injector pod, apply a Kubernetes NetworkPolicy. Verify that the Container Networking Interface (CNI) plugin installed in your cluster supports NetworkPolicy.
- If you have a managed solution like EKS, AKS, or GKE, you can select the Ingress traffic to the injector pod using either a DNS hostname (if, for example, you're using a Calico Enterprise plugin) or an IP address (if, for example, you're using a Cilium plugin).

NetworkPolicies examples

Sample 1

We disallow all Ingress traffic by default to falcon-system namespace, followed by a second policy that allows Ingress traffic only from pods that have the label component: apiserver and provider: kubernetes as shown in this sample.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
    name: default-deny-ingress
    namespace: falcon-system
spec:
    podSelector: {}
    policyTypes:
    - Ingress
---
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
```

```
    namespace: falcon-system
    name: allow-kube-api
spec:
    podSelector: {}
    policyTypes:
    - Ingress
    ingress:
    - from:
        - podSelector:
            matchLabels:
                component: apiserver
                provider: kubernetes
```

Sample 2

We disallow all Ingress traffic by default to falcon-system namespace, followed by a second policy that allows Ingress traffic only from a specific IP address and port number.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
    name: default-deny-ingress
    namespace: falcon-system
spec:
    podSelector: {}
```

```
    policyTypes:
    - Ingress
---
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
    name: allow-kube-api
    namespace: falcon-system
spec:
    podSelector: {}
    ingress:
    - from:
        - ipBlock:
        cidr: <ip_addr>/32
    ports:
        -  port: <port_number>
```

Disabling Falcon Container for specific workloads

By default, the Falcon Container sensor is injected into all pods in all namespaces except kube-system, kube-public, and falcon-system. For cases when you don't want to run the Falcon Container sensor in one or more specific containers or pods, or when you have namespaces that you don't need to monitor, you can disable falcon-sensor or falcon-stub injection.

Disabling injection for all namespaces or all pods

To disable sensor injection in all namespaces, use the `-disable-default-ns-injection` flag in your installer command.

For example:

```
docker run --rm  $FALCON_IMAGE_URI \

-cid $CID -image $FALCON_IMAGE_URI  \

-pulltoken “$IMAGE_PULL_TOKEN” -namespaces “$NAMESPACES” \

-disable-default-ns-injection \

| tee falcon-container.yaml | kubectl create -f -
```

To disable sensor injection for all pods, use the `-disable-default-pod-injection` flag in your installer command.

For example:

```
docker run --rm  $FALCON_IMAGE_URI

\-cid $CID -image $FALCON_IMAGE_URI  \

-pulltoken “$IMAGE_PULL_TOKEN” -namespaces “$NAMESPACES” \

-disable-default-pod-injection \

| tee falcon-container.yaml | kubectl create -f -
```

To disable sensor injection for all pods and all namespaces, use both the `-disable-default-ns-injection` and `-disable-default-pod-injection` flags in your install command.

For example:

```
docker run --rm  $FALCON_IMAGE_URI \
```

```
-cid $CID -image $FALCON_IMAGE_URI  \

-pulltoken “$IMAGE_PULL_TOKEN” -namespaces “$NAMESPACES” \

-disable-default-ns-injection -disable-default-pod-injection \

| tee falcon-container.yaml | kubectl create -f -
```

> **Note:** If you want to enable injection in a namespace or pod after you have disabled injection using the -disable-default-ns-injection or -disable-default-pod-injection flag, you must add a label to the applicable namespace and an annotation to the applicable pod spec. See Allowing injection for only some namespaces or pods [#allowing-injection-for-only-some-namespaces-or-pods] .

Allowing injection for only some namespaces or pods

To allow sensor injection for pods in only some namespaces, you must first disable sensor injection for all namespaces and then enable it for the applicable namespaces:

1. ```
   docker run --rm  $FALCON_IMAGE_URI \

   -cid $CID -image $FALCON_IMAGE_URI  \

   -pulltoken “$IMAGE_PULL_TOKEN” -namespaces “$NAMESPACES” \

   -disable-default-ns-injection \

   | tee falcon-container.yaml | kubectl create -f -
   ```

2. Add a label to each applicable namespace:

   ```
   sensor.falcon-system.crowdstrike.com/injection=enabled
   ```

To allow sensor injection for only some pods, you must first disable sensor injection for all pods and then enable it for the applicable pods:

1. ```
   docker run --rm  $FALCON_IMAGE_URI \

   -cid $CID -image $FALCON_IMAGE_URI  \
   ```

`-pulltoken “$IMAGE_PULL_TOKEN” -namespaces “$NAMESPACES” \`

`-disable-default-pod-injection \`

`| tee falcon-container.yaml | kubectl create -f -`

2. Add an annotation to the pod spec of each applicable pod:

`sensor.falcon-system.crowdstrike.com/injection=enabled`

If you have used both the `-disable-default-ns-injection` and `-disable-default-pod-injection` flags in your install command, to allow sensor injection for pods you must add a label to the applicable namespace and an annotation to the applicable pod spec:

Namespace label: `sensor.falcon-system.crowdstrike.com/injection=enabled`

Pod spec annotation: `sensor.falcon-system.crowdstrike.com/injection=enabled`

Disabling injection for one namespace or pod

To disable sensor injection for all pods in one namespace, add a **label** to the namespace:

`sensor.falcon-system.crowdstrike.com/injection=disabled`

To disable sensor injection for one pod, add an **annotation** to the pod spec:

`sensor.falcon-system.crowdstrike.com/injection=disabled`

Disabling injection for containers

To disable falcon-stub injection for one or more containers in a pod, add an **annotation** to the pod spec of the pod and include a comma-separated list of container names:

`sensor.falcon-system.crowdstrike.com/disabled-containers=<container_1>,<container_2>`

Verifying the Falcon Container sensor

After you have created an application pod using `kubectl apply`, or installed baselines, verify the application pod is running with Falcon Container sensor:

```
kubectl describe pod <app-pod-name> -n <your-namespace> | grep falcon
```

If the output lists that the `falcon-init-container`, `falcon-container`, and `falcon-volume` are present, the sensor is running correctly

Uninstalling the Falcon Container sensor for Linux

To uninstall the Falcon Container sensor for Linux from a cluster, use one of these options:

- Use this command with the same options that were used during the install:

  ```
  docker run --rm $FALCON_IMAGE_URI \
  ```

  ```
  -cid $CID -image $FALCON_IMAGE_URI  \
  ```

  ```
  -pulltoken “IMAGE_PULL_TOKEN” -namespaces “$NAMESPACES” \
  ```

  ```
  | kubectl delete -f -
  ```

- If you have saved the manifest file during install, use this command:

  ```
  kubectl delete -f <manifest_file>
  ```

Delete any pods with `falcon-sensor` running to remove the sensor from them.

Updating the Falcon Container sensor for Linux

To update the Falcon Container image, uninstall and then re-install the Falcon Container sensor for Linux. Existing pods are not updated. Restart or re-create all pods that had the previous sensor installed.

Monitoring new namespaces after injector installation

If all namespaces are enabled to be monitored by default, perform either of these actions on all namespaces created after the injector is installed:

- If the new namespace doesn't need to be monitored by the Falcon Container sensor, use this label for the namespace:

  sensor.falcon-system.crowdstrike.com/injection=disabled

  (Example: kubectl label namespace <name>

  sensor.falcon-system.crowdstrike.com/injection=disabled)

- To enable monitoring for the new namespace, copy the pull secret [/documentation/146/falcon-container-sensor-for-linux#copying-pull-image-secret-for-falcon-container-sensor] from the namespace *falcon-system* to a new namespace.

If selective namespaces are monitored, perform these actions to enable monitoring of the new namespaces:

- Label the namespace so that it is monitored by the injector:

  sensor.falcon-system.crowdstrike.com/injection=enabled

  (Example: kubectl label namespace <name>

  sensor.falcon-system.crowdstrike.com/injection=enabled)

- Copy the pull secret [/documentation/146/falcon-container-sensor-for-linux#copying-pull-image-secret-for-falcon-container-sensor] from the namespace *falcon-system* to a new namespace.

Copying the image pull secret for Falcon Container sensor

1. Dump the existing secret: "kubectl get secret crowdstrike-falcon-pull-secret -n falcon-system -o yaml > crowdstrike-falcon-pull-secret.yaml"
2. Update the namespace attribute value from *falcon-system* namespace to the new namespace value in the yaml file: *crowdstrike-falcon-pull-secret.yaml*
3. Create the new pull secret: "kubectl create -f crowdstrike-falcon-pull-secret.yaml"
4. Delete the file: crowdstrike-falcon-pull-secret.yaml

Existing limitations

Device API size limitations

Falcon Container sensor for Linux has these constraints on the host management dashboard:

- Any label or annotation is limited to 64 characters. This limitation is the entire string, including `key=value`. If the label or annotation is longer, it is omitted from the dashboard.
- Only the first 32 annotations from the manifest are represented in the host management dashboard. Additional annotations are omitted.
- Only the first 32 labels from the manifest are represented in the host management dashboard. Additional labels are omitted.