Install Runtime Sensor for Kubernetes

Use this guide to install Wiz Runtime Sensors in your Kubernetes clusters.

Before you begin

The prerequisites are:

- The operating Kubernetes user must be authorized to create Kubernetes resources in the target Kubernetes cluster(s).
- Make sure your Kubernetes platform and architecture are supported (refer to our list of supported platforms).
- The Kubernetes cluster on which you install the Runtime Sensor allows outbound HTTPS connectivity. Grant access to the required URLs.
- Access to Wiz as a Global Contributor or Global Admin in order to create a service account. Learn about role-based access control.
- Make sure you have the required permission.
- Helm ↗ must be installed locally for the deployment examples in this document. More advanced deployment methods might not require this (terraform helm module, in-cluster operators like Flux or ArgoCD, etc.).
- If you are using a Wiz Admission Controller, you might need to add the Runtime Sensor to the allowed list. Learn how.
- If you are installing the Runtime Sensor on RedHat OpenShift, make sure you configure the Helm chart with `openshift=true`. See all helm chart configurable variables ↗.
- If you are installing the Runtime Sensor on Talos Linux, make sure to override the Pod Security Admission ↗ configuration of the namespace you install in. Learn how
- If you are installing the Runtime Sensor on GKE Autopilot:
  - Make sure you configure the Helm chart with `gkeAutopilot=true`. See all helm chart configurable variables ↗.
  - Add privileged-partner workloads to Google's allow list. Learn how.

GKE Autopilot allow list prerequisite

Before installing the Sensor, note that GKE Autopilot requires privileged-partner workloads to be allow listed by Google. See Google's documentation ↗ for more information on the allow-listing process.

1. Copy the following text and save it in a file, for example `/tmp/wiz-allowlistsynchronizer.yaml`.

```yaml
apiVersion: auto.gke.io/v1
kind: AllowlistSynchronizer
metadata:
  name: wiz-sensor-allowlist-synchronizer
spec:
  allowlistPaths:
  - "Wiz/wiz-sensor/v1/*"
```

2. Run the following command to install the synchronizer:

```
kubectl apply -f /tmp/wiz-allowlistsynchronizer.yaml
```

3. Wait for the synchronizer to be ready:

```
kubectl wait --for=condition=Ready allowlistsynchronizer/wiz-sensor-allowlist-synchronizer
```

4. Validate the `WorkloadAllowlist` objects are present with `kubectl get WorkloadAllowlist` (a number of allowlist versions should be available).
5. You can now delete the `/tmp/wiz-allowlistsynchronizer.yaml` file and proceed with the installation as described above.

Failure to apply the steps above will result in errors such as follows:

```
GKE Warden rejected the request because it violates one or more constraints. Violations details: {"[denied by autogke-default-linux-capabilities]":["linux capability 'SYS_ADMIN,SYS_RESOURCE,SYS_RAWIO,DAC_READ_SEARCH,NET_ADMIN,IPC_LOCK' on container 'wiz-sensor' not allowed; Autopilot only allows the capabilities: 'AUDIT_WRITE,CHOWN,DAC_OVERRIDE,FOWNER,FSETID,KILL,MKNOD,NET_BIND_SERVICE,NET_RAW,SETFCAP,SETGID,SETPCAP,SETUID,SYS_CHROOT,SYS_PTRACE'."],"[denied by autogke-disallow-hostnamespaces]":["enabling hostPID is not allowed in Autopilot.","enabling hostIPC is not allowed in Autopilot."],"[denied by autogke-no-write-mode-hostpath]":["hostPath volume sensor-host-cache in container wiz-sensor is accessed in write mode; disallowed in Autopilot."]}
```

Talos Linux override namespace pod security

By default, Talos Linux enhances cluster security by applying a restricted Pod Security Admission (PSA) ↗ policy to all namespaces (except `kube-system`). This policy prevents pods from running with elevated permissions. Learn more about this practice ↗.

The Runtime Sensor, however, requires privileged access to effectively monitor the host and kernel. To enable this, you need to create a dedicated namespace for the Sensor and specifically override the default PSA policy for that namespace.

> ⚠ Security Warning
>
> It is critical to install the Runtime Sensor in a dedicated, isolated namespace. Overriding the PSA policy to privileged relaxes security constraints for all resources within that namespace. Sharing this namespace with other applications could grant them unintended privileges and expose your cluster to significant security risks.

You can configure the namespace using one of the method below. In these examples, we use the `wiz-sensor` namespace, but you can choose any name.

Option 1: Using kubectl Commands

This method uses two `kubectl` commands to create and label the namespace.

1. Create the dedicated namespace:

```
kubectl create namespace wiz-sensor
```

2. Apply labels to override the PSA policy. This step adjusts the policy to allow privileged pods within the wiz-sensor namespace.

```
kubectl label --overwrite namespace wiz-sensor \
   pod-security.kubernetes.io/enforce=privileged \
   pod-security.kubernetes.io/audit=privileged \
   pod-security.kubernetes.io/warn=privileged
```

Option 2: Using a YAML Manifest (Recommended)

1. Create a file named `sensor-namespace.yaml` with the following content. This manifest defines the namespace and includes the necessary labels to set the PSA policy to privileged.

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: wiz-sensor
  labels:
    name: wiz
    # Override the default PSA policy to allow privileged pods
    pod-security.kubernetes.io/enforce: privileged
    pod-security.kubernetes.io/audit: privileged
    pod-security.kubernetes.io/warn: privileged
```

2. Apply the manifest to your cluster:

```
kubectl apply -f sensor-namespace.yaml
```

Next Steps

Once the namespace is configured, proceed to install the Runtime Sensor ensuring you deploy it into the namespace you just created (e.g., `wiz-sensor`).

> The Runtime Sensor is independent of other Kubernetes Deployment types in Wiz, including the Kubernetes Connector and the Wiz Admission Controller. However, to provide Wiz the maximum context on the cluster resources the Runtime Sensor monitors, we highly recommend connecting the cluster to Wiz via a Kubernetes Connector. For a unified installation of all Wiz Kubernetes deployments, see the unified installation guide.

Installation steps

Step 1: Create a Service Account for the Runtime Sensor in Wiz

Step 2: Get the Runtime Sensor Helm Chart

Step 3: Get the Runtime Sensor image pull key from Wiz

Step 4: Install the Runtime Sensor on a cluster

Step 5: Perform a sanity check

Create a Service Account for the Runtime Sensor in Wiz

Create a service account the Runtime Sensor can use to communicate with the Wiz backend.

1. In Wiz, navigate to ⚙ Settings > Access Management > Service Accounts ↗, then click **Add Service Account**.
2. Enter a meaningful **name** for the account.
3. From Type, select **Sensor** or **Complete Kubernetes Integration** (learn about the different types of Service Accounts). Click **Add Service Account**.
4. A dialog window opens with your new OAuth credentials. Copy the **Client ID** and **Client secret**, you will need them later on.

> ⚠ The Client ID and Client Secret are available only once! Be sure to save them in a secure place.

5. Click **Finish**

Get the Runtime Sensor Helm Chart

Run the following commands to add the Wiz Charts repository to Helm:

```
helm repo add wiz-sec https://charts.wiz.io/
helm repo update
```

> ⓘ You can inspect the Helm chart files in the wiz-sec repo ↗.

Get the Runtime Sensor image pull key from Wiz

Obtain the image pull key the Runtime Sensor uses to access the Wiz container registry in order to retrieve new Runtime Sensor versions.

1. In Wiz, go to the Tenant Info ↗ page.
2. Copy the **Wiz Registry Credentials**.

Install the Runtime Sensor on a cluster

This procedure details the steps needed to manually deploy the Runtime Sensor on a single cluster. For large environments, we recommend that you automate this process across multiple clusters.

To install the Runtime Sensor:

1. Connect to your Kubernetes cluster.
2. Run either the Helm or Yaml scripts, and replace the values in the script with the pull-key username, pull-key password, Client ID, and Client Secret which you retrieved earlier.

3. (Optional) To customize certain properties, refer to the list of configurable variables in the Runtime Sensor Helm chart ↗ and add the corresponding `--set <helm chart value>=<your value>` to the command. Another best practice is to create a `values.yaml` file with all required parameters, and add `--values values.yaml` to the command below.
4. Replace the parameters for `PULL_KEY_USERNAME`, `PULL_KEY_PASSWORD`, `CLIENT_ID`, `CLIENT_SECRET`, then run the following command to install the Runtime Sensor in a new namespace named `wiz` and the Runtime Sensor DaemonSet as a Helm chart named `wiz-sensor`. If you are installing the Runtime Sensor on a cluster that will be scanned from a Wiz for Gov tenant, make sure to use the relevant command:

Select an environment 👉 | shell | Windows | Wiz for Gov—Shell | Wiz for Gov—Windows

- Parameter names are case-sensitive in Helm.
- We recommend installing the `wiz-sensor` Helm chart in a separate Kubernetes namespace, adhering to the principle of least privilege as well as isolating the Runtime Sensor Kubernetes resources if another namespace is compromised.
- For security best practices, do not push the image-pull, service account or proxy credentials directly into various CI/CD scripts (i.e. Terraform). The wiz-sensor Helm chart supports receiving the names of pre-created secrets via the `imagePullSecret.name`, `wizApiToken.secret.name` and `httpProxyConfiguration.secretName` values.
- If you prefer, you can install via a YAML file .

5. (Recommended) Validate successful installation on the cluster by running the following command:

```
kubectl get ds -n wiz
```

In the sample output, you can see an up-to-date `wiz-sensor` DaemonSet in the `wiz` namespace:

Output

```
NAME          DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR
AGE
wiz-sensor    1         1         1       1            1           <none>
55s
```

6. (Recommended) Validate that the Runtime Sensor can successfully communicate with Wiz by running the following command:

Linux/Mac | Windows (Powershell)

```
kubectl logs -n wiz `kubectl get pods -n wiz -l "app.kubernetes.io/name=wiz-sensor" -o jsonpath="{.items[0].metadata.name}"` | jq '.' | grep -B30 -A2 "ending
```

Sample output:

Output

```
{
  "timestamp": "2023-06-15 09:23:41.014190933 UTC",
  "level": "INFO",
  "fields": {
    "message": "ending content update - new content downloaded"
  }
}
```

⚠ If this is not the output you see, you most likely have a misconfigured route to `wiz.io`. Make sure you set up the proxy correctly.

7. Go to the Settings > Deployments ↗ page, and select the Sensors tab. It takes between 5-10 minutes for a new Sensor to appear on the page in **Active** status.

Configure TLS proxy for Kubernetes Sensor

To deploy the Sensor in environments with a proxy that performs TLS inspection with a self-signed certificate, you need to use a specific configuration. Learn how to set this up.

⚠ **Runtime Sensor detection interoperability with proxy and proxy-like solutions**

If you are using a proxy or other network proxy-like solutions (such as Istio or Calico), the Runtime Sensor network-based detections might show the wrong Linux process or the wrong Primary Resource since the IP/DNS communications go through the Proxy.

(Optional) Pre-create a secret for the image pull key

Kubernetes uses Docker to pull the Sensor container image from the Wiz registry. Therefore, the image pull key secret is of type `kubernetes.io/dockerconfigjson`, which follows a specific format described here ↗.

Pre-create secret

To pre-create the secret using `kubectl`, use this command:

Select 👉 Wiz commercial Wiz for Gov

If you prefer to use external tools to create the secret, you can utilize the following Bash script to generate the base64-encoded string:

Shell

```
#!/bin/bash

# Read input credentials
read -p "Enter registry username: " username
read -sp "Enter registry password: " password
echo
read -p "Enter registry URL [default: wizio.azurecr.io]: " registry_url
registry_url=${registry_url:-wizio.azurecr.io}

# Encode credentials to base64
auth_string="${username}:${password}"
base64_encoded_auth_string=$(echo -n "${auth_string}" | base64)

# Generate JSON content
json_content="{\"auths\":{\"${registry_url}\":
{\"auth\":\"${base64_encoded_auth_string}\"}}}"

#print decoded version of the result
echo '-----------------------------------------------------------'
echo "This is how your dockerjsonconfig secret should look like:"
echo '---'
echo ${json_content}
echo '---'
echo "You can save this in AWS secret manager"
echo "Assuming you save this with a key named 'image-pull-key' and the
AWS secret name iz 'WizRegistrySecret'."
echo "And assuming you're using external-secrets k8s plugin."
echo "Here's a yaml file that will create a secret named 'sensor-image-
pull-key' in your cluster."
echo "You can than pass this 'sensor-image-pull-key' to the sensor helm
chart."
cat <<EOF
---
apiVersion: external-secrets.io/v1beta1
kind: ExternalSecret
metadata:
  name: example-1
spec:
  refreshInterval: 1h
  secretStoreRef:
    name: secretstore-sample
    kind: SecretStore
  target:
```

```
    template:
      type: kubernetes.io/dockerconfigjson
      data:
        .dockerconfigjson: "{{ .image-pull-key | toString }}"
    name: sensor-image-pul-key
    creationPolicy: Owner
  data:
  - secretKey: image-pull-key
    remoteRef:
      key: WizRegistrySecret
      property: image-pull-key
---
EOF


# Encode JSON content to base64
base64_encoded_json_content=$(echo -n "${json_content}" | base64)

# Output the result
echo "In case you manully create the secret and not using the external-secrets k8s plugin - "
echo "This is the base64 encoded secret, it should be stored in a k8s secret in a '.dockerjsonconfig' key"
echo '---'
echo ${base64_encoded_json_content}
echo '---'
```

You can then place the outputted string inside the `.dockerconfigjson` key when creating the secret. An example YAML file that demonstrates pre-creating the secret:

```
apiVersion: v1
kind: Secret
metadata:
  name: <MY_SECRET>
  namespace: wiz
type: kubernetes.io/dockerconfigjson
data:
  .dockerconfigjson: "<BASE64_ENCODED_STRING>"
```

(Optional) Install via YAML file

1. Run the `helm template` command to create a YAML file. This command accepts the same arguments as the regular install command:

shell Windows

```
helm template wiz-sensor wiz-sec/wiz-sensor \
  --namespace=wiz \
  --set imagePullSecret.username=<pull key username> \
  --set imagePullSecret.password=<pull key password> \
  --set wizApiToken.clientId=<client id> \
  --set wizApiToken.clientToken=<client secret> > sensor_install.yaml
```

2. Run the `sensor_install.yaml` file to install on your cluster using `kubectl`:

```
kubectl create namespace wiz
kubectl apply -f sensor_install.yaml
```

(Recommended) Perform a sanity check

You can verify that the Runtime Sensor is enabled and monitoring events in your environment using the Wiz Runtime Sensor Validator tool, which is a benign file that can be executed on an environment with no risk (similar to an EICAR file).

Prerequisites

- Kubernetes version 1.18 or higher, where the Wiz Runtime Sensor was successfully deployed on a cluster.
- A target pod on the Kubernetes cluster (any running pod deployed on one or more nodes).
- Access credentials with permissions to execute commands on the target pod.
- A local installation of `kubectl`.
- Access to the Kubernetes cluster API from your machine with the ability to run `kubectl` commands.

Validation

1. Deploy the Runtime Sensor validator:
   i. Download the Runtime Sensor validator tool ↓.
   ii. Start a pod for testing, e.g. a `centos` pod:

```
kubectl run centos --attach=false --image=docker.io/library/centos:7 --command
-- /bin/sh -c "sleep 3600"
```

iii. Copy the Runtime Sensor validator package to the target pod. Using the `centos` example:

```
kubectl cp [path to sensor-validator.tgz on your machine] centos:/tmp/
```

iv. Extract the archive:

```
kubectl exec -it centos -- tar xvfz /tmp/sensor-validator.tgz -C /tmp/
```

v. Run the Runtime Sensor validator tool for your architecture:

```
kubectl exec -it centos -- sh -c "/tmp/sensor-validator-\`uname -m\`"
```

vi. (Optional) Delete the testing pod:

```
kubectl delete pod centos
```

2. Wait up to 10 minutes following the execution of the commands in the previous section. This might be a good time to make a cup of tea or coffee.
3. In Wiz, go to the Explorer > Cloud Events page. From the Origin filter, select **Wiz Sensor**. (direct link ↗)
4. Search for an event named "Malware Execution".
5. Select an event and click it to open the details drawer. Review the details to see a valid process tree and the details of the associated Runtime Sensor.

Findings on the Sensor DaemonSet

In line with best security practices and transparency, Wiz applies its Pod Security Standards ↗ Cloud Configuration Rules (CCRs) also to the Wiz Runtime Sensor. These CCRs generate Findings on the Kubernetes Sensor DaemonSet in your environment. If you want to hide these Findings from the Wiz portal, you can ignore them directly from the resource details drawer.

Use the following queries to:

- Identify all relevant CCRs ↗.
- Identify all relevant Findings ↗.