Red Hat Advanced Cluster Security

With Kasten K10

Quick Start Guide

January 2024

Red Hat Advanced Cluster Security with Kasten K10

Below is an example guide of how someone may be able to deploy Red Hat Advanced Cluster Security in conjunction with Kasten K10 for Kubernetes in an OCP cluster to ensure a security posture fit for your deployment that correlates and heightens events that are associated with custom Kasten K10 resources.

Setting up RHACS and Kasten K10 on an OCP Cluster

In this example, we'll deploy a Red Hat OpenShift cluster on AWS. Once our cluster is ready, we can install Red Hat Advanced Cluster Security either via Helm or the OpenShift Operator. Let's go ahead and install via Helm by getting the chart:

```
helm repo add rhacs https://mirror.openshift.com/pub/rhacs/charts/
```

We need to ensure we have access to the Red Hat registry at https://catalog.redhat.com/ before proceeding or else the image pull will fail. For example, if using docker you can run the following command with your Red Hat username and password:

```
docker login   -u "$username" -p "$password"
```

Next, let's create a `stackrox` namespace and secret to hold our Red Hat registry credentials:

```
kubectl create ns stackrox
kubectl create secret generic redhatcreds -n stackrox \
  --from-file=.dockerconfigjson="$HOME/.docker/config.json" \
  --type=kubernetes.io/dockerconfigjson
```

Now, let's install the central services with port forwarding (you can also use an exposed route or a load balancer, see here for help):

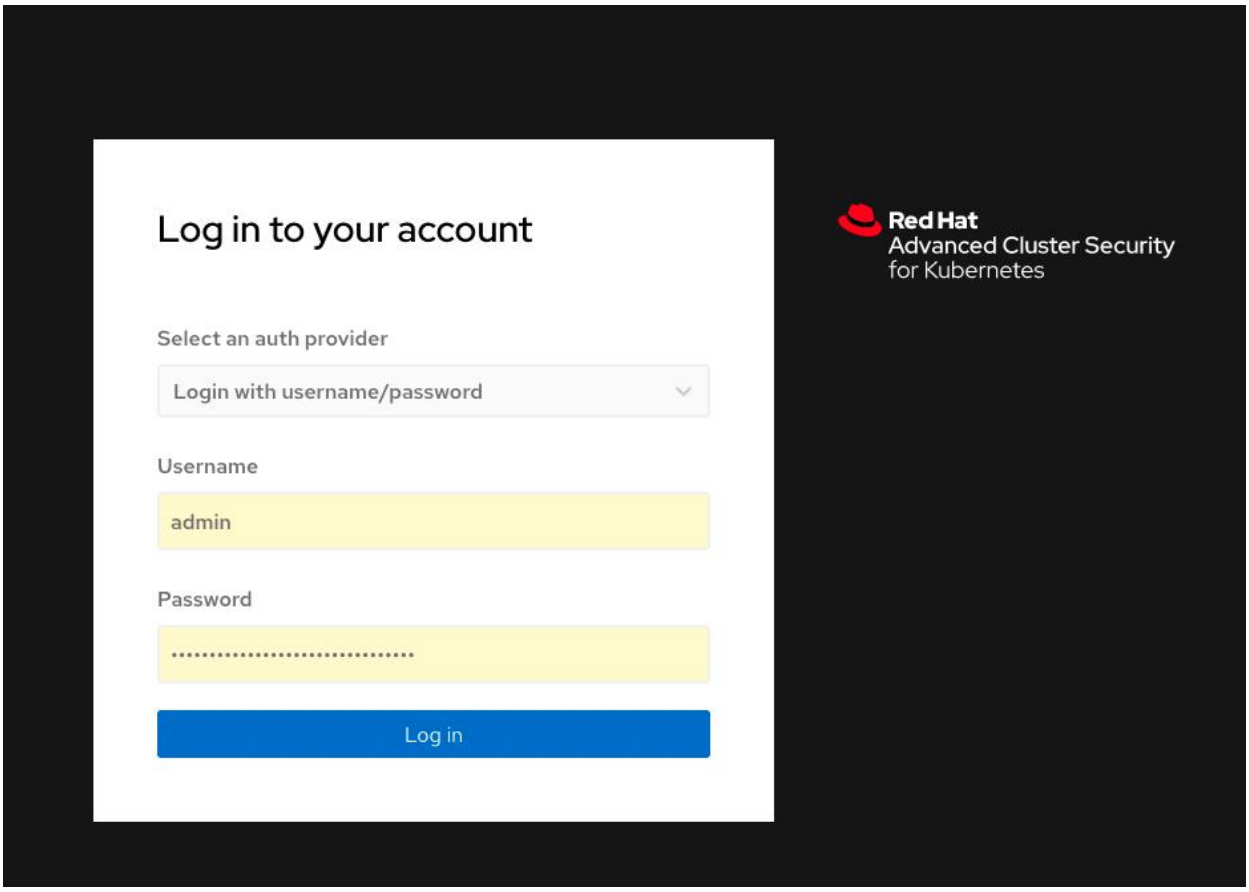
```
helm install -n stackrox \
  --create-namespace stackrox-central-services rhacs/central-services \
  --set imagePullSecrets.useExisting=redhatcreds
```

Once installed you should see an admin password listed to be used with the Red Hat Advanced Cluster Security console. Copy this as we'll need it.

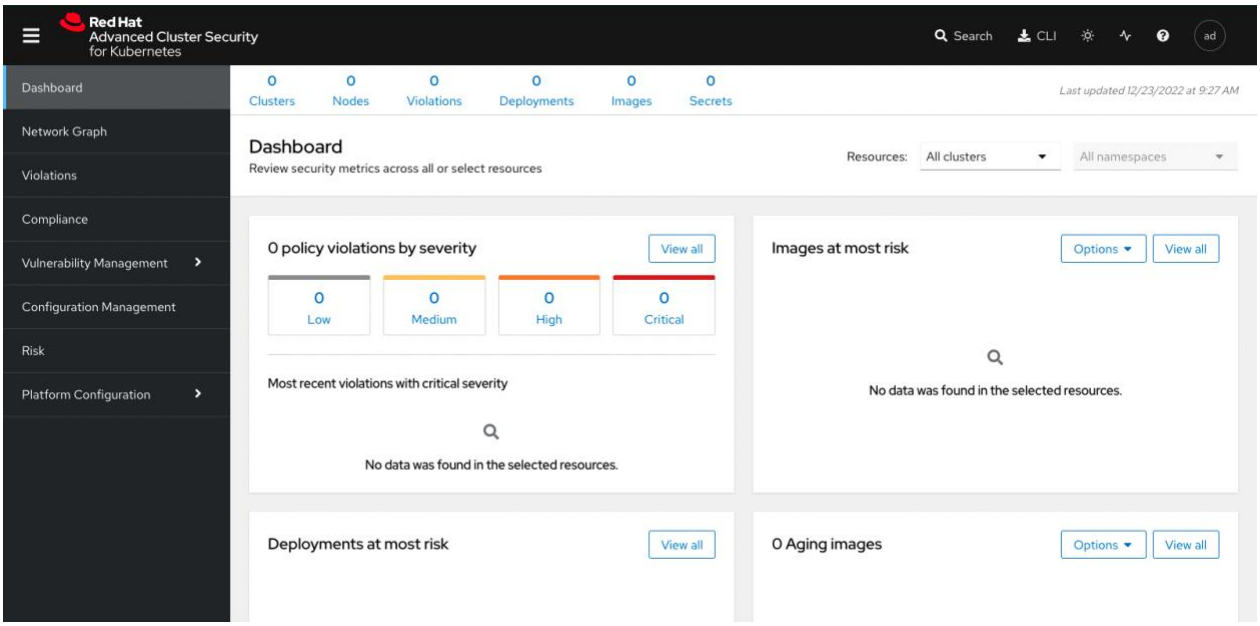
At this point pods for the centralized services should be creating. Once these are created, make sure to port forward the central services:

```
kubectl -n stackrox port-forward svc/central 18443:443
```

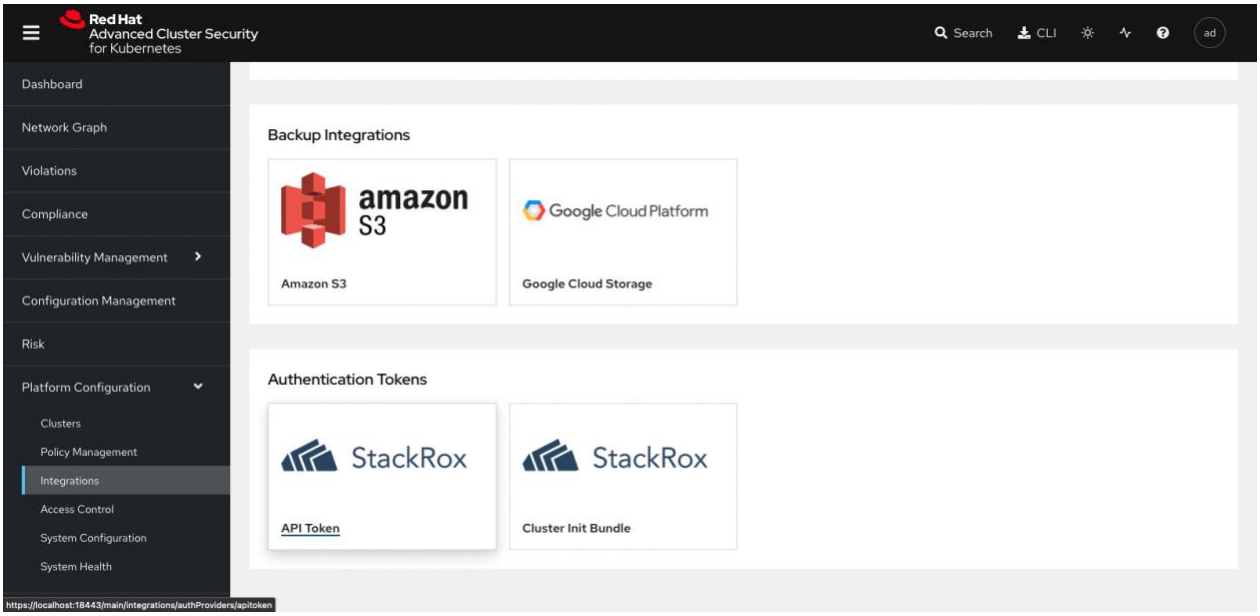
Now open your browser and go to `https://localhost:18443`. You should be prompted for a username and password; use `admin` and the password copied after helm installed the centralized services.



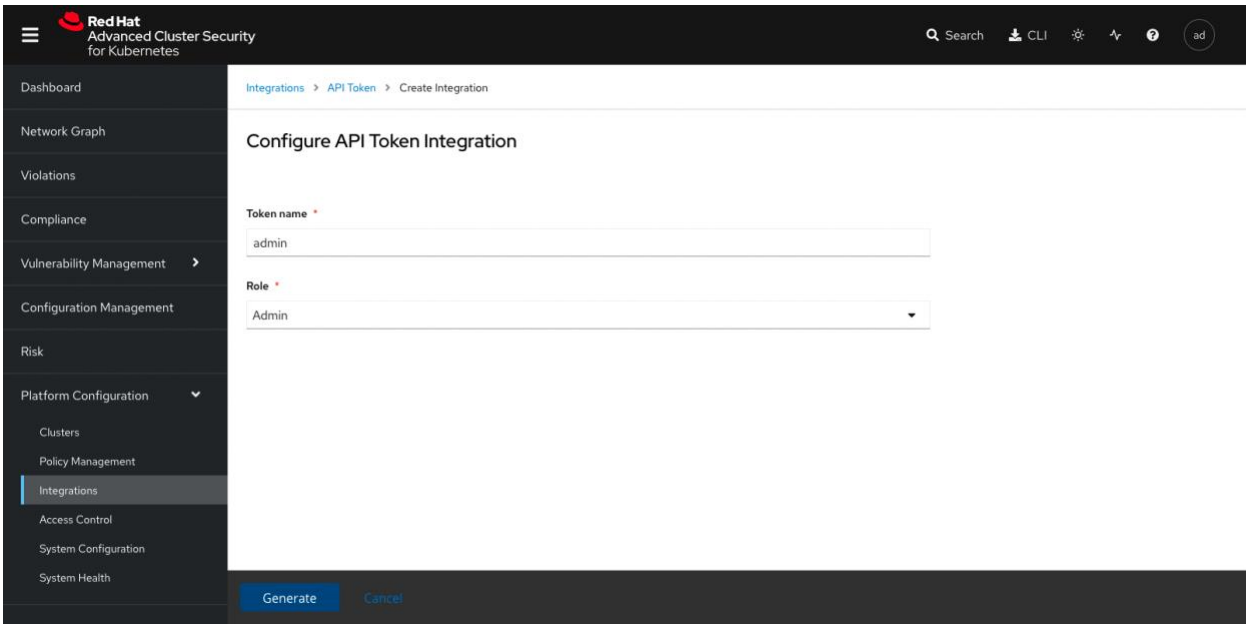
The dashboard should show empty with no clusters monitored since we haven't installed the secured cluster services yet.



To do this, navigate to "Platform Configuration" in the left sidebar, and select "Integrations"; scroll to the bottom and click "StackRox API Token":



Create an API Token called "Admin" with the "Admin" Role and copy the generated token



Next, we need to create the `init bundle` to allow the cluster to authenticate with `Central`. You'll need the `roxctl cli` and for a Mac can be installed as follows:

```
curl -O
https://mirror.openshift.com/pub/rhacs/assets/3.70.2/bin/Darwin/roxctl
xattr -c roxctl
chmod +x roxctl
sudo mv roxtl /usr/local/bin
```

Set the `ROX_API_TOKEN` environment variable with the copied API token:

```
export ROX_API_TOKEN=${API_TOKEN}
```

Set the `ROX_CENTRAL_ADDRESS` environment variable:

```
export ROX_CENTRAL_ADDRESS=https://localhost:18443
```

Now, create the `init bundle`:

```
roxctl -e "$ROX_CENTRAL_ADDRESS" central init-bundles generate acs-test --
output cluster_init_bundle.yaml
```

With the `init bundle` created, install the secure cluster services on each cluster you want monitored:

```
helm install -n stackrox \
  --create-namespace stackrox-secured-cluster-services rhacs/secured-cluster-
services \
  -f PATH_TO_INIT_BUNDLE/cluster_init_bundle.yaml \
  --set imagePullSecrets.useExisting=redhatcreds \
  --set clusterName=acs-test \
```

```
  --set centralEndpoint=central.stackrox:443
```

This will create the `admission-control`, `collector`, `scanner-db`, and `sensor` pods. Once these are all running Red Hat Advanced Cluster Security is ready to use.

Lastly, install the latest version of Kasten K10 through the OperatorHub, by following the directions found in the Kasten K10 documentation.

Viewing Kasten K10 Audit Logs In OCP Cluster

The audit policy in a Red Hat OpenShift cluster is customizable in the sense that it allows you to select from three different settings which correlate to the `Metadata`, `Request`, and `RequestReceived` levels that the kube audit logs at.

Running the following command brings up the `config.openshift.io` API group's `APIServer` resource which can be edited:

```
oc edit apiserver cluster
```

Edit the `spec.audit.profile` field to one of `Default`, `WriteRequestBodies`, `AllRequestBodies` to have increasingly more information present in the audit event that's logged.

Let's find the nodes and location of the audit logs by running:

```
oc adm –role=master --path=kube-apiserver/audit.log
```

Depending on the number of nodes, multiple nodes and audit logs should come up. Select the node name and location and run the following to see the audit logs:

```
oc adm node-logs ip-10-0-255-61.us-west-1.compute.internal  --path=kube-
apiserver/audit.log
```

To test this, let's run the following command to list and describe the k10MasterKey:

```
kubectl get passkeys
kubectl describe k10MasterKey
```

Let's search through the logs to find this audit event (you may need to look through multiple nodes and locations to find this):

```
oc adm node-logs ip-10-0-255-61.us-west-1.compute.internal  --path=kube-
apiserver/audit.log | jq 'select(.requestURI |
startswith("/apis/vault.kio.kasten.io/v1alpha1"))'
```

We then see the following audit event:

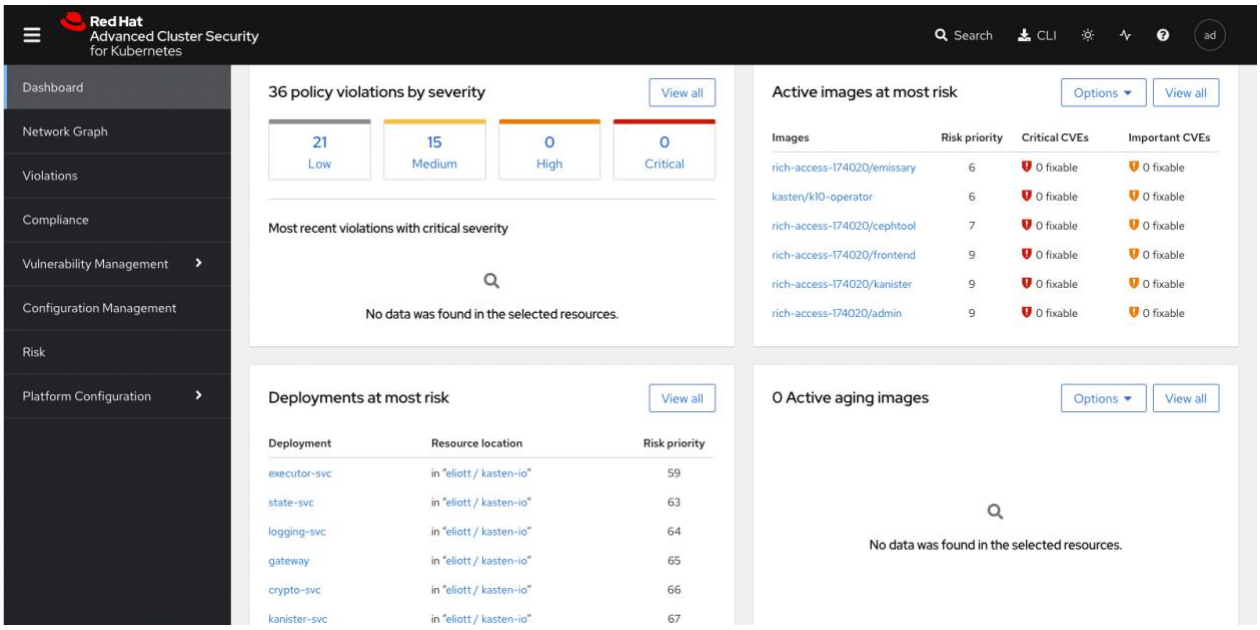

```
{
  "kind": "Event",
  "apiVersion": "audit.k8s.io/v1",
  "level": "Metadata",
  "auditID": "11aa507b-e3a6-4be6-b5d7-458f4d69dc37",
  "stage": "ResponseComplete",
  "requestURI": "/apis/vault.kio.kasten.io/v1alpha1/passkeys/k10MasterKey",
  "verb": "get",
  "user": {
    "username": "system:admin",
    "groups": [
      "system:masters",
      "system:authenticated"
    ]
  },
  "sourceIPs": [
    "10.0.117.141"
  ],
  "userAgent": "kubectl/v1.24.0 (linux/arm64) kubernetes/4ce5a89",
  "objectRef": {
    "resource": "passkeys",
    "name": "k10MasterKey",
    "apiGroup": "vault.kio.kasten.io",
    "apiVersion": "v1alpha1"
  },
  "responseStatus": {
    "metadata": {},
    "code": 200
  },
  "requestReceivedTimestamp": "2022-12-23T20:01:31.603020Z",
  "stageTimestamp": "2022-12-23T20:01:31.605900Z",
  "annotations": {
    "authorization.k8s.io/decision": "allow",
    "authorization.k8s.io/reason": ""
  }
}
```


Actions against Kasten K10's custom resources get processed by the `kube-apiserver` which then logs the audit event based on the audit policy. These can be from custom resources created by CRD's or the Aggregated API; Kasten K10's cloud native architecture natively allows for the `kube audit` to be able to audit interactions with Kasten K10, ensuring protection against unauthorized access and misuse of Kasten K10's underlying Kubernetes structure.

Using RHACS Console to Monitor Kasten K10

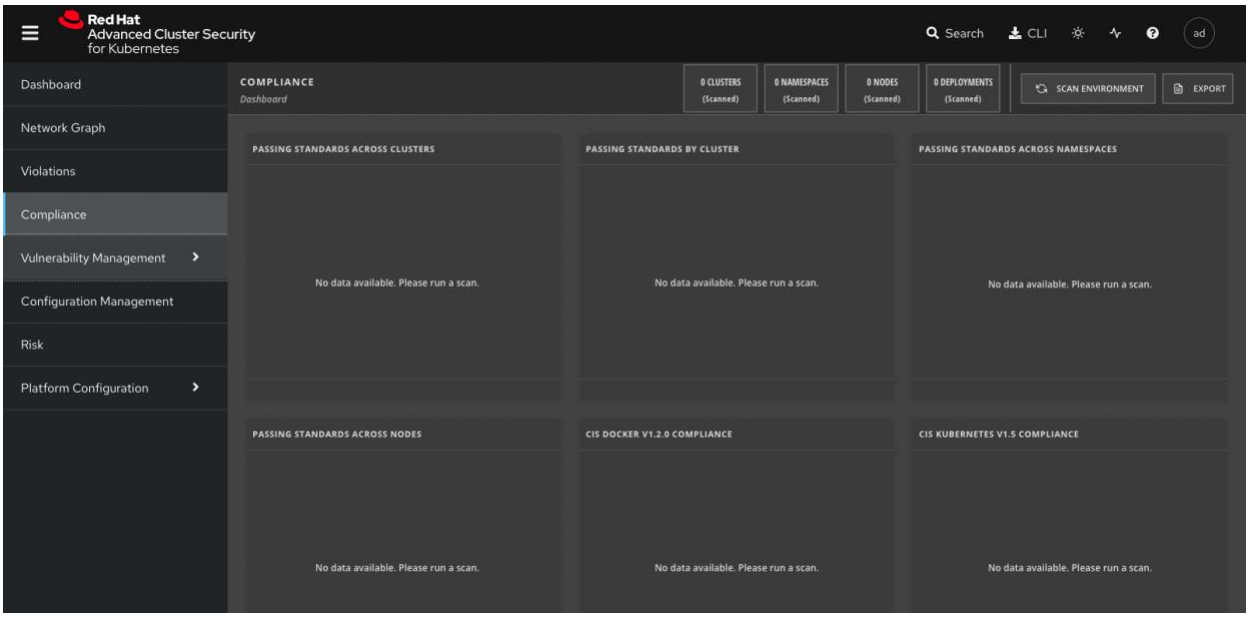
With Kasten K10 installed in our cluster, we can view the Red Hat Advanced Cluster Security dashboard for the `kasten-io` namespace to get high-level information as to the security of Kasten K10; Select `acs-test` from the list of clusters in the top right corner, and then choose `kasten-io` from the list of namespaces.

Here we see policy violations by severity, active images at risk, deployments at most risk, active aging images, policy violations by category, and compliance by standard.



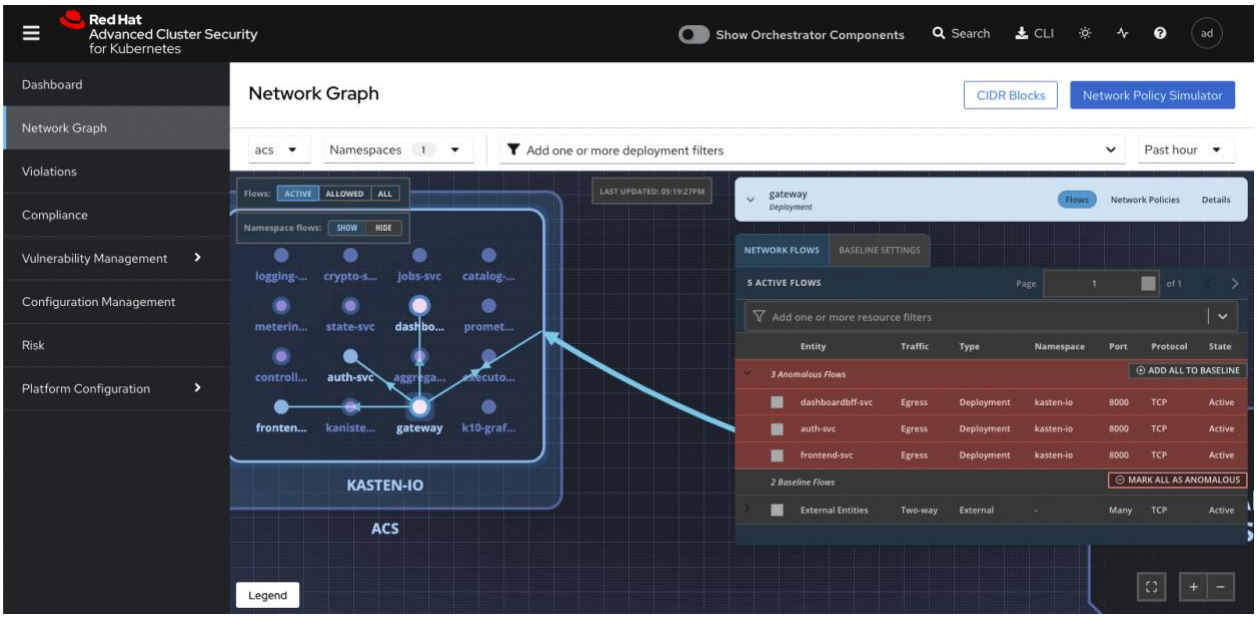
We can follow links to get more information about any of the data presented here.

If we select "Compliance" from the left side bar, we can scan our environment to run a compliance check.

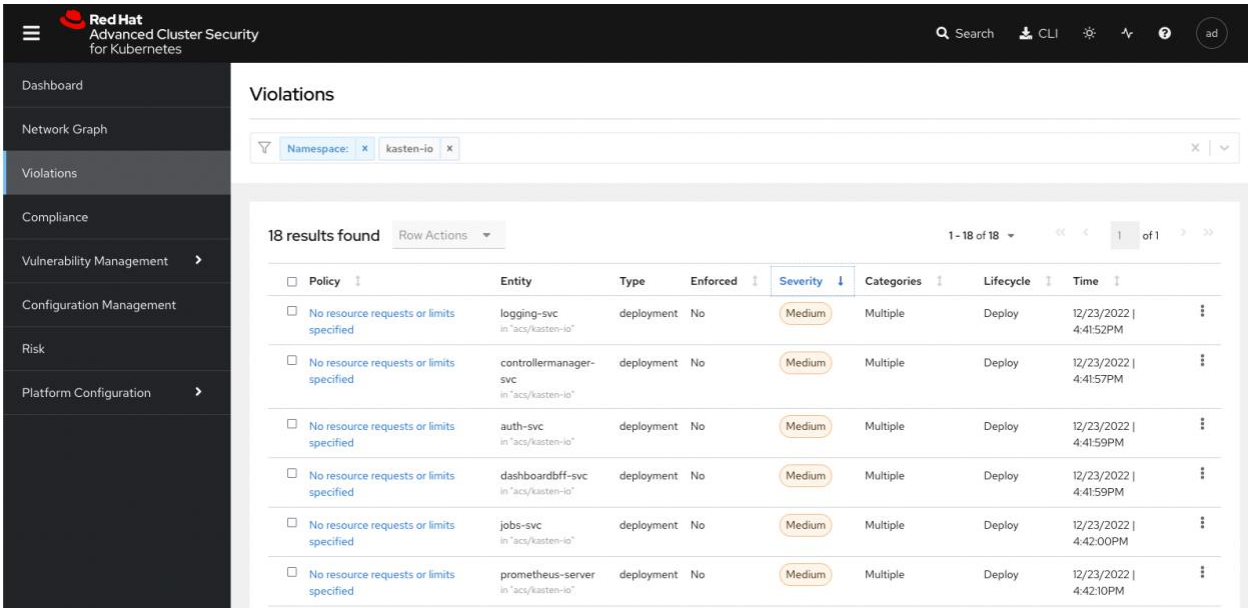


From here we can deep-dive different standards such as NIST SP 800-190 (Application Container Security Guide) and NIST SP 800-53 (Security and Privacy Controls for Information Systems and Organizations).

We can generate the network graph for the `kasten-io` namespace by selecting "Network Graph" from the left sidebar and then selecting this namespace. If we select the `gateway` pod, we can see the network connections and any anomalous flows Red Hat Advanced Cluster Security finds:



Since Red Hat Advanced Cluster Security comes with a lot of security policies by default, such as monitoring `kubectl port-forward...` and `kubectl exec...`, if any of these actions happen on a K10 pod, a violation will show up. These violations are all shown by clicking on "Violations" in the left sidebar.



Conclusion

Red Hat Advanced Cluster Security in conjunction with Kasten K10 allows for an enhanced security posture to further protect Kubernetes data from unknown attacks from malicious users.

Footnotes

1. You cannot install Red Hat Advanced Cluster Security in a local cluster such as with `K3D` or `Minikube`, or in a Lima VM running x86 ubuntu with K3S installed; it must be in one of the listed managed Kubernetes services.