Особенности работы Prometheus в Nova Container Platform

В Nova Container Platform вы можете настроить Prometheus для сбора метрик с определённых pod’ов в Namespace. Для этого используется Prometheus Operator и ServiceMonitor.

ServiceMonitor — это специальный пользовательский ресурс (*Custom Resource* / *CR*) в Kubernetes, который используется в связке с Prometheus Operator для автоматизации настройки сбора метрик от сервисов, работающих в Kubernetes.

1. Настройка пространства имён

Чтобы Prometheus Operator начал работать с ServiceMonitor в Namespace, необходимо добавить метку при создании или затем в существующий Namespace:

```yaml
  labels:
    nova-platform.io/cluster-monitoring: 'true'
```

Пример:

```yaml
kind: Namespace
apiVersion: v1
metadata:
  name: test
  labels:
    nova-platform.io/cluster-monitoring: 'true'
```

2. Создание ServiceMonitor

Далее необходимо содать ServiceMonitor, с которым Prometheus Operator начнет работу.

```yaml
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: example
spec: {}
```

Пример:

```yaml
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: service-monitor-test
  namespace: test
spec:
  endpoints:
    - interval: 30s
      port: metrics
  namespaceSelector:
    matchNames:
      - test
  selector:
    matchLabels:
      app: test-exporter
```

Подсказки для настройки ServiceMonitor можно найти в веб-консоли Nova на странице **Administration > CustomResourceDefinitions**. Найдите *ServiceMonitor*, перейдите на вкладку **Экземпляры** и нажмите **[Создать ServiceMonitor]** .
Далее следуйте подсказкам на боковой панели. Если боковая панель не отображается, нажмите на **[Посмотреть боковую панель]**.

3. Настрока pod'а

Создайте pod, который будет содержать сервис и встроенный экспортер. Pod должен содержать метку, указанную в ServiceMonitor.

Пример:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: jmx
  namespace: test
  labels:
    app: test-exporter
spec:
  ports:
  - name: metrics
    port: 5556
    targetPort: 5556
  selector:
    app: test-exporter
  type: ClusterIP
---
apiVersion: v1
kind: Service
```

```yaml
metadata:
  name: kafka
  namespace: test
  labels:
    app: test-exporter
spec:
  ports:
  - name: broker
    port: 9092
    targetPort: 9092
  - name: controller
    port: 9093
    targetPort: 9093
  selector:
    app: test-exporter
  type: ClusterIP
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: test-exporter-config
  namespace: test
  labels:
    app: test-exporter
data:
  jmx-exporter-config.yaml: |
    rules:
    - pattern: ".*"
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: kafka
  namespace: test
  labels:
    app: test-exporter
spec:
  serviceName: "kafka"
  replicas: 1
  selector:
    matchLabels:
      app: test-exporter
  template:
    metadata:
      labels:
        app: test-exporter
    spec:
      volumes:
      - name: test-config
        configMap:
          name: test-exporter-config
```

```
      containers:
      - name: kafka
        image: bitnami/kafka:latest
        ports:
        - containerPort: 9092
        env:
        - name: KAFKA_CFG_NODE_ID
          value: '0'
        - name: KAFKA_CFG_PROCESS_ROLES
          value: 'controller,broker'
        - name: KAFKA_CFG_LISTENERS
          value: 'PLAINTEXT://:9092,CONTROLLER://:9093'
        - name: KAFKA_CFG_LISTENER_SECURITY_PROTOCOL_MAP
          value: 'CONTROLLER:PLAINTEXT,PLAINTEXT:PLAINTEXT'
        - name: KAFKA_CFG_CONTROLLER_QUORUM_VOTERS
          value: '0@kafka:9093'
        - name: KAFKA_CFG_CONTROLLER_LISTENER_NAMES
          value: CONTROLLER
        - name: JMX_PORT
          value: '5555'
        volumeMounts:
        - name: test-config
          mountPath: /config
      - name: test-exporter
        image: bitnami/jmx-exporter:latest
        ports:
        - containerPort: 5556
          name: metrics
        env:
        - name: JMX_EXPORTER_CONFIG
          value: "/config/jmx-exporter-config.yaml"
        volumeMounts:
        - name: test-config
          mountPath: /config
```

4. Проверка

1. Откройте веб-консоль Nova Container Platform и проверьте статус pod'а `prometheus-main-0` в Namespace `nova-monitoring`.
2. Откройте веб-консоль Prometheus и перейдите на вкладку **Status > Targets**.
3. Найдите нужный таргет, который будет отображаться в формате `serviceMonitor/<имя пространства имён>/<имя ServiceMonitor>/<номер>`.
4. Убедитесь, что Prometheus обнаружил экспортер.

1. Если вы видите таргет, но не видите экспортера, проверьте метки, указанные в *ServiceMonitor* и на pod’е с экспортером.
2. Если вы не видите таргет, проверьте метку в Namespace и убедитесь, что *ServiceMonitor* создан. Если всё выглядит корректно, возможно, метка была добавлена после создания Namespace, и Prometheus Operator еще не обновил данные — в этом случае необходимо подождать.После обновления информации данные о *ServiceMonitor* будут записаны в секрет `prometheus-main` в Namespace `nova-monitoring`.

Архитектура и концепции модуля Neuvector

В данном разделе представлено описание концепций и архитектуры платформы обеспечения безопасности Neuvector.

1. Компоненты модуля

Модуль Neuvector в кластере Nova Container Platform включает компоненты, представленные в таблице ниже:

Компонент	**Категория**	**Количество**
Controller	Кластерный компонент	1 или 3
Enforcer	Хостовой компонент (агент)	1 на узел
Scanner	Кластерный компонент	1 и более
Manager	Кластерный компонент	1 и более
Registry Adapter	Кластерный компонент	1 и более
Updater	Кластерный компонент	1
API docs	Кластерный компонент	1 и более
Provisioner	Кластерный компонент	1
Prometheus Exporter	Кластерный компонент	1

Компоненты Neuvector являются контейнерами, которые взаимодействуют между собой, так и с кластером Kubernetes и его узлами. Все компоненты Neuvector запускаются и функционируют в среде Kubernetes.

- **Controller**: основной компонент (контроллер), реализующий функции управления и координации действий других компонентов. В Nova Container Platform контроллер разворачивается в виде сущности StatefulSet. В зависимости от количества инфраструктурных узлов в кластере автоматически устанавливается необходимое количество реплик. Также контроллер предоставляет REST API для управления сущностями Neuvector.

- **Enforcer**: хостовый компонент (агент), который разворачивается в виде сущности DaemonSet, тем самым размещаясь на каждом узле кластера Kubernetes. Задача агента Enforcer состоит в том, чтобы отслеживать потоки сетевого трафика, процессы и файловые системы контейнеров и применять установленные политики безопасности.
- **Scanner**: компонент (сканер) выполняет задачи по сканированию образов контейнеров, используя встроенную базу уязвимостей (CVE). Сканеров может быть от 1 и более в кластере Kubernetes для увеличения скорости при сканировании большого количество образов. Сканер имеет политику загрузки собственного образа (ImagePullPolicy) в Kubernetes с типом *Always* и установленный тег *latest*. Тем самым, перезапускаясь, сканер всегда запускается с последней доступной в хранилище версией образа и базой CVE.
- **Manager**: компонент предоставляет веб-интерфейс управления сущностями Neuvector и взаимодействует с контроллером через REST API.
- **Registry Adapter**: адаптер для использования во внешних хранилищах образов (например, Harbor) с целью осуществлять сканирование образов по базе CVE Neuvector.
- **Updater**: компонент реализует процесс обновления базы CVE с помощью запуска CronJob, выполняющей поочередный перезапуск всех сканеров.
- **API docs**: веб-портал с документацией к REST API Neuvector.
- **Provisioner**: компонент, запускаемый однократно в виде сущности Job для первоначальной настройки Neuvector.
- **Prometheus Exporter**: компонент для сборка метрик со всех компонентов Neuvector.

2. Архитектурная схема

На рисунке далее представлена архитектурная схема платформы Neuvector.

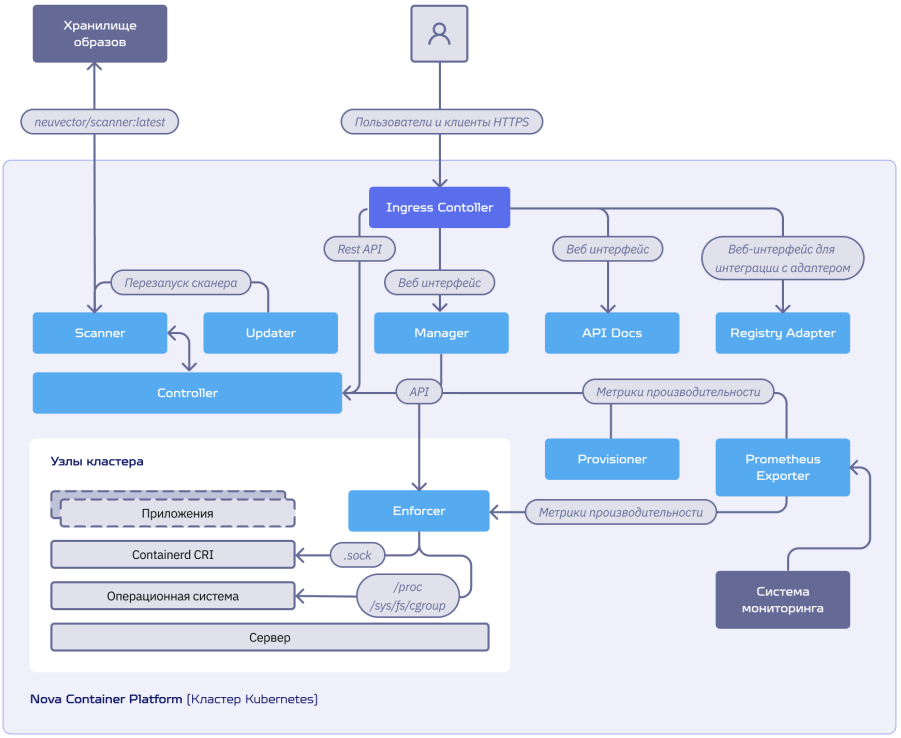

Рисунок 1. Архитектура платформы Neuvector

В кластере Kubernetes модуль Neuvector развертывается с помощью службы непрерывного развертывания FluxCD, которая также обеспечивает дальнейшую консистентность конфигураций компонентов Neuvector в Kubernetes и возможности их кастомизации.

На рисунке далее представлена схема развертывания платформы Neuvector в Nova Container Platform.

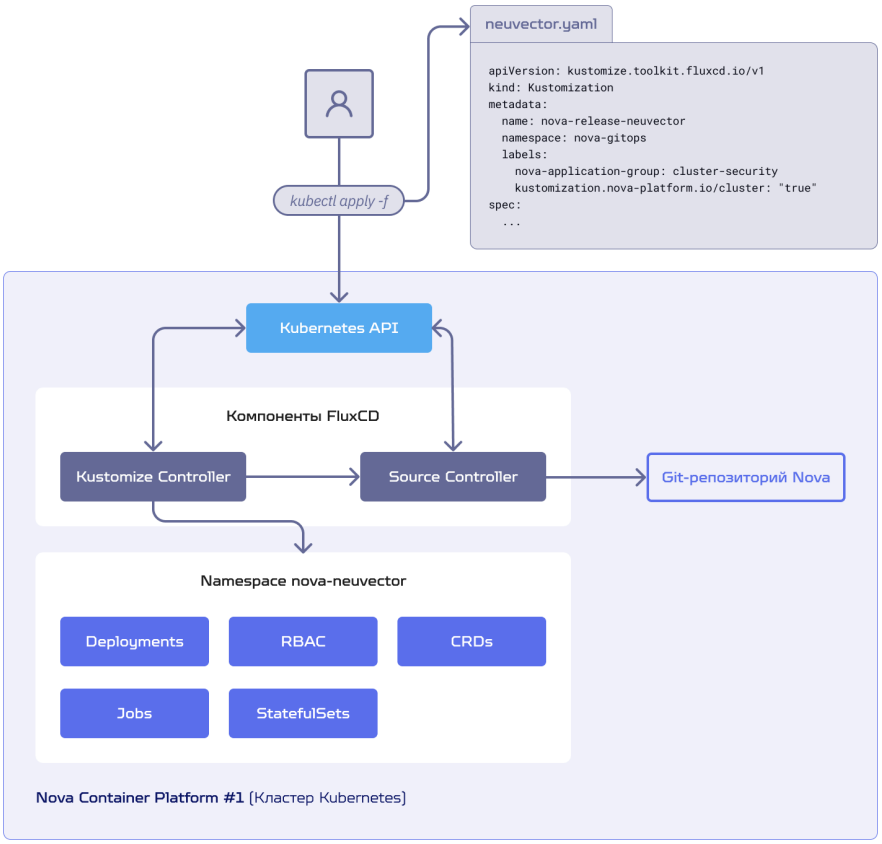

Рисунок 2. Схема развертывания платформы Neuvector

Используемый в Nova Container Platform Git-репозиторий Gitea содержит все необходимые манифесты для развертывания платформы Neuvector. Пользователю платформы необходимо применить в кластере манифесты *Kustomizations*, предназначенные для установки Neuvector. Для этого можно воспользоваться как утилитой `kubectl`, так и веб-консолью Nova.

Модуль Neuvector по умолчанию не установлен в кластере Kubernetes с целью экономии вычислительных ресурсов. Ознакомьтесь с разделом Планирование установки и системные требования для подготовки Nova Container Platform к установке модуля Neuvector.

3. Высокая доступность и непрерывность работы

Под высокой доступностью и непрерывностью работы компонентов Neuvector предполагается использование количества реплик компонентов более 1 для того, чтобы избежать прерывания работы сервисов в период недоступности части узлов платформы или перезапуска части сервисов.

Далее для каждого компонента Neuvector рассмотрены особенности его работы в режиме высокой доступности.

Controller

Для повышения доступности контроллеров необходимо использовать строго нечетное количество реплик. Рекомендуемое и достаточное количество реплик - 3. Контроллеры имеют встроенные механизмы выбора лидера и синхронизируют конфигурацию друг с другом.

Enforcer

Данный компонент разворачивается в виде сущности DaemonSet, тем самым размещаясь на каждом узле кластера Kubernetes. Сценарий с увеличением реплик в данном случае не применим.

Scanner

Компонент требует обязательного увеличения количества реплик в случаях, когда вы работаете с большим количеством образов или время сканирования образов неудовлетворительное. Для сканера в Neuvector поддерживается автоматическое увеличение количества реплик.

Manager

Веб-интерфейс управления сущностями Neuvector взаимодействует с контроллером через REST API и не влияет на работу ключевых сервисов Neuvector. Для повышения доступности веб-интерфейса количество реплик может быть увеличено. Постоянство пользовательских сессий обеспечивается уже установленной директивой метода балансировки в Ingress Controller `nginx.ingress.kubernetes.io/upstream-hash-by: "$binary_remote_addr"`.

Registry Adapter

Количество реплик *Registry Adapter* может быть увеличено при необходимости повышения доступности адаптера(ов), процессов сканирования образов во внешних хранилищах.

Updater

Поскольку компонент Updater является сущностью типа *CronJob* в Kubernetes и запускается только для перезапуска компонента *Scanner*, то высокая доступность и большое количество реплик компоненту не требуются.

API docs

Веб-портал с документацией к REST API не является критичным сервисом для работы Neuvector, поэтому достаточно иметь одну реплику в кластере Kubernetes.

Provisioner

Поскольку компонент Provisioner является сущностью типа *Job* в Kubernetes и запускается однократно при первоначальной установке, то высокая доступность и большое количество реплик компоненту не требуются.

Prometheus Exporter

В кластере Kubernetes в большинстве случаев достаточно иметь одну реплику Prometheus Exporter для сбора метрик Neuvector.

4. Персистентность данных в Neuvector

Конфигурации Neuvector и зарегистрированные события синхронизируются между доступными контроллерами. В случае, если все контроллеры будут недоступны одновременно, то некоторые данные могут быть потеряны, а именно:

- Параметры и конфигурации, политики безопасности установленные вручную через API или веб-интерфейс Neuvector
- Данные о сетевых соединениях, сканированиях
- Зарегистрированные события безопасности

Доступность данных (состояния) обеспечивает как минимум один контроллер Neuvector.

Для возможности постоянного хранения собственных конфигураций Neuvector поддерживает использование сущностей *ConfigMap* в Kubernetes. При этом, Neuvector не сохраняет данные в *ConfigMap* самостоятельно, а только загружает их при первом и каждом последующем запуске.

4.1. Отличие Neuvector в Nova Container Platform

По умолчанию, для хранения данных архитектура Neuvector предполагает использование постоянного тома в Kubernetes с типом доступа RWX (Read Write Many). Данный том подключается в директорию `/var/neuvector/` каждого контроллера, куда сохраняются все данные Neuvector.

Поскольку организация файлового хранилища с множественным доступом (RWX) несет дополнительные накладные расходы (например, отсутствие поддержки в публичном или частном облаке, необходимость организации отдельного NFS-хранилища и т.п.), в Nova Container Platform изменен подход к организации персистентности данных согласно следующим принципам:

1. Контроллеры Neuvector разворачиваются в виде сущности StatefulSet вместо Deployment с привязкой к инфраструктурным узлам.
2. Каждый контроллер Neuvector имеет отдельный смонтированный в директорию `/var/neuvector/` постоянный том из хранилища Local Path.
3. Для всех контроллеров установлен параметр `CTRL_PERSIST_CONFIG`, разрешающий выполнять сохранение всех данных в `/var/neuvector/`.

4. Вместо базовых объектов *ConfigMap* для хранения конфигураций, используются единый объект *Secret*, который инжектируется напрямую в Neuvector с помощью StarVault. Это позволяет безопасно использовать конфигурационные файлы Neuvector в Kubernetes, а также сохраняет возможность декларативного описания данных конфигураций.

4.2. Рекомендации по обеспечению доступности данных в Neuvector

Для обеспечения сохранности данных в Neuvector рекомендуется выполнить следующие шаги:

- Настроить периодическое резервное копирование конфигураций и политик Neuvector.
- Настроить интеграцию с Syslog-сервером и SIEM-системой для экспорта информации о событиях безопасности, системных событиях, атаках, уязвимостях.

В качестве Syslog-сервера и хранилища журналов событий вы можете использовать подключаемый модуль логирования в Nova Container Platform.

5. Интеграция с StarVault

5.1. Единый вход с помощью Nova Oauth

После установки модуля Neuvector возможен вход с помощью учетной записи по умолчанию `kubeadmin`. Для этого между Neuvector и StarVault настроена интеграция по протоколу OIDC.

5.2. Конфигурации контроллера Neuvector

Каждый Pod контроллера Neuvector включает Sidecar-контейнер с StarVault, работающий в режиме агента. Перед запуском основного контейнера Neuvector, Sidecar-контейнер получает из StarVault необходимую информацию и генерирует конфигурационные файлы, а именно:

- `oidcinitcfg` - конфигурация параметров OIDC
- `passwordprofileinitcfg` - конфигурация параметров профилей паролей
- `roleinitcfg` - конфигурация базовых ролей
- `samlinitcfg` - конфигурация параметров SAML
- `ldapinitcfg` - конфигурация параметров LDAP
- `sysinitcfg` - системные параметры Neuvector

- `userinitcfg` - конфигурация системных пользователей Neuvector

Шаблоны конфигурационных файлов без чувствительной информации находятся в объекте *Secret* с именем `neuvector-init-template` в пространстве имен `nova-neuvector`.

6. Следующие шаги

- Планирование установки и системные требования