Prometheus Adapter

Prometheus Adapter — это инструмент, позволяющий использовать метрики из Prometheus для масштабирования приложений в среде Kubernetes на основе динамических метрик. Он выступает связующим звеном между сервером метрик Kubernetes (Metrics API) и Prometheus, преобразуя метрики, собранные Prometheus, в формат, понятный Kubernetes, что обеспечивает автоматическое горизонтальное масштабирование (HPA — Horizontal Pod Autoscaler).

1. Предварительные условия

✓ Вы ознакомились со статьёй и создали все необходимые ресурсы.

2. Основные функции и принципы работы:

- **Адаптация метрик:** Prometheus Adapter преобразует метрики Prometheus в формат, который может быть использован Kubernetes для принятия решений о масштабировании. Это осуществляется с помощью пользовательских настроек и правил, которые сопоставляют метрики Prometheus с API Kubernetes.
- **Конфигурация:** Пользователи могут определить, какие именно метрики из Prometheus будут использоваться для масштабирования и как они должны быть интерпретированы. Это делается через файлы конфигурации YAML, где задаются правила трансформации методов, такие как соотношения, суммирования или модификации метрик.
- **Metrics API:** Prometheus Adapter реализует Metrics API, который предоставляет два типа метрик для *HPA: Custom Metrics API* и *External Metrics API*.
  Custom Metrics API используется для масштабирования на основе метрик, специфичных для кластера Kubernetes, таких как количество запросов на pod.
  External Metrics API используется для метрик, которые находятся за пределами кластера, например, количество записей в очереди сообщений.
- **Horizontal Pod Autoscalers:** С помощью адаптера можно настроить Kubernetes для автоматического изменения количества pod’ов (горизонтального масштабирования) на основе изменяющейся нагрузки, создаваемой приложениями. Метрики, предоставляемые адаптером, помогают HPA более точно определять нагрузки и, соответственно, масштабировать ресурсы.

3. Настройка Prometheus Adapter

По умолчанию Prometheus Adapter настраивается только на метрики CPU и Memory. Для добавления своих метрик выполните следующие шаги:

1. В веб-консоли Nova Container Platform перейдите на вкладку **Administration** > **CustomResourceDefinitions** и найдите ресурс *Kustomization*. Далее перейдите на вкладку **Экземпляры** и найдите `nova-release-prometheus-adapter-main`.
2. На вкладке **YAML** добавьте патч, который будет содержать текущие настройки из *ConfigMap* `nova-prometheus-adapter` в Namespace `nova-monitoring` и ваши изменения.

   Пример:

```yaml
spec:
  patches:
    - patch: |
        - op: replace
          path: /data/config.yaml
          value: |
            rules:
            - seriesQuery:
'container_cpu_usage_seconds_total{namespace!="",pod!="",container!="POD"}'
              resources:
                overrides:
                  namespace:
                    resource: namespace
                  pod:
                    resource: pod
              name:
                matches: ".*"
                as: "cpu_per_5m"
              metricsQuery: 'sum(irate(<<.Series>>{<<.LabelMatchers>>,
container!="POD"}[5m])) by (<<.GroupBy>>)'
            - seriesQuery:
'container_memory_working_set_bytes{namespace!="",pod!="",container!="POD"}'
              resources:
                overrides:
                  namespace:
                    resource: namespace
                  pod:
                    resource: pod
              name:
                matches: ".*"
                as: "memory_per_5m"
              metricsQuery: 'sum(avg_over_time(<<.Series>>
{<<.LabelMatchers>>, container!="POD"}[5m])) by (<<.GroupBy>>)'
            - seriesQuery:
'kafka_controller_ControllerEventManager_Count{name="EventQueueTimeMs"}'
              resources:
                overrides:
```

```
                namespace:
                  resource: namespace
                pod:
                  resource: pod
            name:
              matches: ".*"
              as:
"kafka_controller_ControllerEventManager_Count_EventQueueTimeMs"
            metricsQuery:
'avg(increase(kafka_controller_ControllerEventManager_Count{name="EventQueue
TimeMs"}[5m])) by (namespace, pod)'
      target:
        kind: ConfigMap
        name: nova-prometheus-adapter
        namespace: nova-monitoring
```

3. Убедитесь, что pod `nova-prometheus-adapter` успешно перезапустился.
4. Убедитесь, что следующая команда выводит значение метрики:

```
BASH
kubectl get --raw "/apis/custom.metrics.k8s.io/v1beta1/namespaces/<имя
пространства имён>/pods/*/<название метрики из конфига Prometheus Adapter>"
```

Пример:

```
BASH
kubectl get --raw
"/apis/custom.metrics.k8s.io/v1beta1/namespaces/test/pods/*/kafka_controller
_ControllerEventManager_Count_EventQueueTimeMs"

{"kind":"MetricValueList","apiVersion":"custom.metrics.k8s.io/v1beta1","meta
data":{},"items":[{"describedObject":
{"kind":"Pod","namespace":"test","name":"kafka-
0","apiVersion":"/v1"},"metricName":"kafka_controller_ControllerEventManager
_Count_EventQueueTimeMs","timestamp":"2024-10-
08T12:19:22Z","value":"750","selector":null},{"describedObject":
{"kind":"Pod","namespace":"test","name":"kafka-
1","apiVersion":"/v1"},"metricName":"kafka_controller_ControllerEventManager
_Count_EventQueueTimeMs","timestamp":"2024-10-
08T12:19:22Z","value":"750","selector":null}]}
```

4. Настройка Horizontal Pod Autoscalers

После того как необходимая метрика становится доступной в Prometheus Adapter, можно приступать к настройке Horizontal Pod Autoscalers (HPA) для выбранного ресурса. Ниже приведен пример конфигурации:

YAML

```yaml
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v2
metadata:
  name: my-deployment-hpa
  namespace: test
spec:
  scaleTargetRef:
    kind: StatefulSet
    name: kafka
    apiVersion: apps/v1
  minReplicas: 1
  maxReplicas: 2
  metrics:
    - type: Pods
      pods:
        metric:
          name: kafka_controller_ControllerEventManager_Count_EventQueueTimeMs
        target:
          type: AverageValue
          averageValue: '600'
```

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`.

Настройка TLS-Passthrough на ingress-контроллере nginx

1. При использовании TLS-PassThrough на Nginx Ingress-контроллер необходимо учитывать следующие особенности

- для Ingress объектов с активированным TLS-PassThrough не работает path-based routing, так как path зашифрован
- все запросы Ingress отправляет напрямую на IP адрес соответствующего объекта Service, соответственно stickiness и load-balancing настраивается на стороне объекта Service:
  - Stickiness
  - Load balancing
- использование TLS-PassThrough снижает производительность Ingress-контроллера (простой тест при помощи Locust показал 5-10% снижение Average Response Time)
- не работает для клиентов с версией TLS, не поддерживающей SNI

2. Алгоритм активации tls-passthrough

1. На стороне Ingress-контроллер необходимо добавить команду `--enable-ssl-passthrough=true` к строке аргументов, передаваемых nginx. Для этого необходимо изменить кастомизацию `nova-release-ingress-public-main`, добавив в нее следующий патч:

```yaml
patches:
  - patch: |-
      - op: add
        path: /spec/template/spec/containers/0/args/1
        value: '--enable-ssl-passthrough=true'
    target:
      kind: DaemonSet
      name: nova-ingress-public-controller
      namespace: nova-ingress-public
```

2. При последующем создании ingress-обьектов, для активации tls-passthough необходимо добавить следующие аннотации:

```yaml
  annotations:
    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
    nginx.ingress.kubernetes.io/ssl-passthrough: "true"
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
```

3. Для ingress-обьектов с `nginx.ingress.kubernetes.io/ssl-passthrough: "true"` Nginx работает в режиме TCP-proxy, при этом клиентские сертификаты не передаются бэкенду, что препятствует реализации mTLS. Для корректной работы mTLS необходимо добавить следующую аннотацию к ingress-обьекту:

```yaml
nginx.ingress.kubernetes.io/auth-tls-pass-certificate-to-upstream: "true"
```