Grafana

Grafana — это платформа с открытым исходным кодом для визуализации и анализа данных, которая позволяет создавать динамические и интерактивные дашборды. Она поддерживает интеграцию с различными источниками данных, такими как базы данных, системы мониторинга и облачные сервисы. Grafana позволяет пользователям строить графики, диаграммы и настраивать предупреждения в реальном времени, что делает её популярным инструментом для мониторинга и анализа системных метрик, бизнес-аналитики и других видов данных.

1. Основные особенности Grafana

- **Интерактивные дашборды:** Возможность создавать настраиваемые панели управления с широким спектром визуальных элементов.
- **Поддержка множества источников данных:** Платформа может подключаться к различным источникам данных, включая Prometheus, Graphite, InfluxDB, Elasticsearch и многим другим.
- **Оповещения и уведомления:** Настройка триггеров для отправки уведомлений при достижении определённых условий.
- **Гибкость и расширяемость:** Поддержка плагинов, которые расширяют функциональность Grafana, добавляя новые типы визуализаций и поддержки дополнительных источников данных.
- **Совместная работа:** Возможность совместного использования дашбордов с коллегами или в больших командах.

2. Основные сведения

Grafana устанавливается в виде ресурса *Deployment* в Namespace `nova-monitoring`. Для доступа к Grafana из внешней сети установлен ресурс *Ingress*.

В Grafana предустановлен источник Prometheus и дашборды. Данные из этих дашбордов отображаются на странице **Observe > Панели мониторинга** в веб-консоли Nova Container Platform. Данные с других дашбордов не будут выведены.

Рекомендуется добавлять новые дашборды и источники данных через ConfigMap. Grafana автоматически начнёт с ними работать.

3. Добавление нового плагина

В настоящее время установка плагинов на постоянной основе невозможна. После перезапуска pod'а настройки возвращаются к исходному состоянию.
Данный функционал будет реализован в одной из будущих версий платформы.

4. Добавление нового источника данных (Datasource)

Для добавления нового источника данных создайте ресурс *ConfigMap* в Namespace `nova-monitoring` с именем `nova-grafana-datasources-<имя_источника>` и меткой `nova-platform.io/cluster-dashboard-datasource: 'true'`, например:

```yaml
kind: ConfigMap
apiVersion: v1
metadata:
  name: nova-grafana-datasources-test ①
  namespace: nova-monitoring ②
  labels:
    nova-platform.io/cluster-dashboard-datasource: 'true' ③
data: ④
  datasources_prometheus_new.yaml: |-
    apiVersion: 1
    datasources:
    - name: nova-prometheus-new
      type: prometheus
      url: http://prometheus-operated.nova-monitoring.svc:9090
      isDefault: false
```

1. Имя ресурса: должно быть `nova-grafana-datasources-<имя_источника>`.
2. Имя Namespace: не должно быть изменено.
3. Метка, по которой ресурс автоматически добавляется в Grafana.
4. Пример файла конфигурации. Имя файла должно быть уникально.

5. Добавление нового дашборда

Для добавления нового дашборда создайте ресурс *ConfigMap* в Namespace `nova-monitoring` с именем `nova-grafana-dashboard-<имя_источника>` и меткой `grafana_dashboard: '1'`, например:

```yaml
kind: ConfigMap
apiVersion: v1
```

```
metadata:
  name: nova-grafana-dashboard-alertmanager-overview ①
  namespace: nova-monitoring ②
  labels:
    grafana_dashboard: '1' ③
data: ④
  alertmanager-overview.json: |-
    {
      "annotations": {
        "list": [
          {
            "builtIn": 1,
            "datasource": {
              "type": "grafana",
              "uid": "-- Grafana --"
            },
            ...
        ]
    }
```

1. Имя ресурса: должно быть `nova-grafana-dashboard-<имя_дашборда>`.
2. Имя Namespace: не должно быть изменено.
3. Метка, по которой ресурс автоматически добавляется в Grafana.
4. Пример файла конфигурации. Имя файла должно быть уникально. Можно заранее настроить дашборд в Grafana и взять его конфигурацию из `JSON Model`.

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'
```