RabbitMQ TIG Monitoring

Content

1. Changes

Date	Version	Author	Changes
18/09/2019	1.0	Hichem BOUSSETTA	Initial version

2. Overview

This document shows how to monitor RabbitMQ cluster using a TIG (Telegraf, InfluxDb, Grafana) stack.

The installation of the different components is done on Kubernetes. We use stable helm chart for each component.

Link to this file:

https://grouperenault-my.sharepoint.com/:w:/r/personal/hichem_boussetta_renault_com/Documents/AKS/RabbitMQ-TIG-Monitoring.docx?d=w8bc3bd6495384b68b493280da676223a&csf=1&e=2NFvop

3. RabbitMQ Installation on Kubernetes

- Install local (minikube) or hosted (azure) Kubernetes cluster
- Install helm / tiller on the cluster to be able to install official community stable helm chart of RabbitMQ HA
- Install RabbitMQ-HA helm chart using the following command. This will install RabbitMQ with 3 stateful nodes and will bind the service with a load balancer (in azure or equivalent, a load balancer will be created and bound to a public ip address, however when testing locally, the load balancer must be emulated using **minikube tunnel** command)

```
helm install --name rabbitmq-ha -set
rabbitmqUsername=guest,rabbitmqPassword=guest,rabbitmqErlangCookie=secretcookie,service.cluste
rIP="",service.type=LoadBalancer,persistentVolume.enabled=true stable/rabbitmq-ha
```

- Export one of RabbitMQ pods using the following command:

```
export POD_NAME=$(kubectl get pods --namespace default -l "app=rabbitmq-ha" -o
jsonpath="{.items[0].metadata.name}")
```

- Enable queue replication / mirroring on the RabbitMQ cluster. HA modes are explained in RabbitMQ documentation:
  https://www.rabbitmq.com/ha.html

```
# Enable mirroring on all nodes
kubectl exec $POD_NAME --namespace "$NAMESPACE" -- \
  rabbitmqctl set_policy ha-all "." \
    '{"ha-mode":"all", "ha-sync-mode":"automatic"}' --apply-to all --priority 0

# Enable mirroring on 2 nodes
kubectl exec $POD_NAME --namespace "$NAMESPACE" -- \
  rabbitmqctl set_policy ha-two "." \
    '{"ha-mode":"exactly", "ha-params":2,"ha-sync-mode":"automatic","ha-sync-batch-size":5}'
--apply-to all --priority 0
```

- Make sure the RabbitMQ instances are running using **kubectl** command

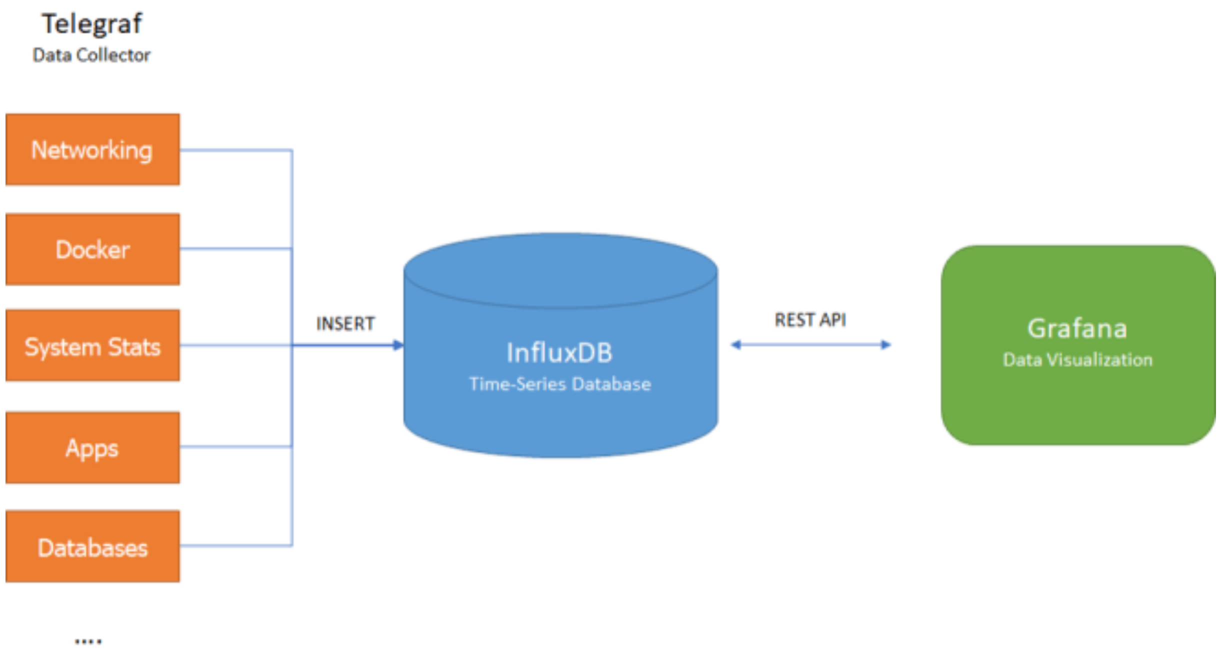
```
kubectl -n services get pods

rabbitmq-ha-0                 1/1     Running    0          4h56m
rabbitmq-ha-1                 1/1     Running    0          4h55m
rabbitmq-ha-2                 1/1     Running    0          4h54m
```

4. TIG Stack Installation

The TIG stack is made of the following components:

- Telegraf: telemetry collector
- InfluxDB: scalable time-series database for metrics, events and real time analytics
- Grafana: Data visualization and exploration tool

The TIG stacks works as described in the picture below:



In the next sections, we describe how to install TIG stack on Kubernetes service using helm installer.

4.1. Telegraf

The Telegraf helm chart for Kubernetes can be referred to at the following github page: https://github.com/helm/charts/tree/master/stable/telegraf

4.1.1. Install on Kubernetes

Run the following helm command to install Telegraf in the default namespace:

```
helm install --name telegraf stable/telegraf
```

4.1.2. Configure for RabbitMQ

Telegraf must be configured to collect mertrics data from RabbitMQ broker. This can be done by using RabbitMQ pluging of telegraf.

The list of all plugins supported by telegraf are listed in the following page:
https://github.com/influxdata/telegraf/tree/master/plugins/inputs

The configuration related to rabbitmq plugin can be found in the following page and provides all the metrics and data collected by telegraf and stored in influxdb:
https://github.com/influxdata/telegraf/tree/master/plugins/inputs/rabbitmq

Telegraf must be configured using the following yaml file:

```
apiVersion: v1
data:
  telegraf.conf: |2

    [agent]
      collection_jitter = "0s"
      debug = false
      flush_interval = "10s"
      flush_jitter = "0s"
      hostname = "$HOSTNAME"
      interval = "10s"
      logfile = ""
      metric_batch_size = 1000
      metric_buffer_limit = 10000
      omit_hostname = false
      precision = ""
      quiet = false
      round_interval = true

    [[inputs.rabbitmq]]
      ## Management Plugin url. (default: http://localhost:15672)
      url = "https://rabbitmq-ha.default.svc.cluster.local:15672"
      ## Tag added to rabbitmq_overview series; deprecated: use tags
      # name = "rmq-server-1"
      ## Credentials
      username = "guest"
      password = "guest"

      ## Optional TLS Config
      # tls_ca = "/etc/telegraf/ca.pem"
      # tls_cert = "/etc/telegraf/cert.pem"
      # tls_key = "/etc/telegraf/key.pem"
      ## Use TLS but skip chain & host verification
      insecure_skip_verify = true

      ## Optional request timeouts
      ##
      ## ResponseHeaderTimeout, if non-zero, specifies the amount of time to wait
      ## for a server's response headers after fully writing the request.
      # header_timeout = "3s"
      header_timeout = "10s"
      ##
      ## client_timeout specifies a time limit for requests made by this client.
      ## Includes connection time, any redirects, and reading the response body.
      # client_timeout = "4s"
      client_timeout = "10s"

      ## A list of nodes to gather as the rabbitmq_node measurement. If not
      ## specified, metrics for all nodes are gathered.
      # nodes = ["rabbit@node1", "rabbit@node2"]

      ## A list of queues to gather as the rabbitmq_queue measurement. If not
      ## specified, metrics for all queues are gathered.
      #queues = ["car", "user"]

      ## A list of exchanges to gather as the rabbitmq_exchange measurement. If not
      ## specified, metrics for all exchanges are gathered.
      # exchanges = ["telegraf"]

      ## Queues to include and exclude. Globs accepted.
      ## Note that an empty array for both will include all queues
      # queue_name_include = []
      # queue_name_exclude = []

    [[outputs.influxdb]]
```

```
      database = "telegraf"
      urls = [
        "http://influxdb.default.svc:8086"
      ]
    [[inputs.statsd]]
      allowed_pending_messages = 10000
      metric_separator = "_"
      percentile_limit = 1000
      percentiles = [
        50,
        95,
        99
      ]
      service_address = ":8125"
kind: ConfigMap
metadata:
  name: telegraf
```

4.1.3. Telemetry Data

Telegraf collects the telemetry data listed below from RabbitMQ management endpoint.

Measurements & Fields:

- rabbitmq_overview
    - channels (int, channels)
    - connections (int, connections)
    - consumers (int, consumers)
    - exchanges (int, exchanges)
    - messages (int, messages)
    - messages_acked (int, messages)
    - messages_delivered (int, messages)
    - messages_delivered_get (int, messages)
    - messages_published (int, messages)
    - messages_ready (int, messages)
    - messages_unacked (int, messages)
    - queues (int, queues)
    - clustering_listeners (int, cluster nodes)
    - amqp_listeners (int, amqp nodes up)
    - return_unroutable (int, number of unroutable messages)
    - return_unroutable_rate (float, number of unroutable messages per second)
- rabbitmq_node
    - disk_free (int, bytes)
    - disk_free_limit (int, bytes)
    - disk_free_alarm (int, disk alarm)
    - fd_total (int, file descriptors)
    - fd_used (int, file descriptors)
    - mem_limit (int, bytes)
    - mem_used (int, bytes)
    - mem_alarm (int, memory a)
    - proc_total (int, erlang processes)
    - proc_used (int, erlang processes)

  - run_queue (int, erlang processes)
  - sockets_total (int, sockets)
  - sockets_used (int, sockets)
  - running (int, node up)
  - uptime (int, milliseconds)
  - health_check_status (int, 1 or 0)
  - mnesia_disk_tx_count (int, number of disk transaction)
  - mnesia_ram_tx_count (int, number of ram transaction)
  - mnesia_disk_tx_count_rate (float, number of disk transaction per second)
  - mnesia_ram_tx_count_rate (float, number of ram transaction per second)
  - gc_num (int, number of garbage collection)
  - gc_bytes_reclaimed (int, bytes)
  - gc_num_rate (float, number of garbage collection per second)
  - gc_bytes_reclaimed_rate (float, bytes per second)
  - io_read_avg_time (float, number of read operations)
  - io_read_avg_time_rate (int, number of read operations per second)
  - io_read_bytes (int, bytes)
  - io_read_bytes_rate (float, bytes per second)
  - io_write_avg_time (int, milliseconds)
  - io_write_avg_time_rate (float, milliseconds per second)
  - io_write_bytes (int, bytes)
  - io_write_bytes_rate (float, bytes per second)
  - mem_connection_readers (int, bytes)
  - mem_connection_writers (int, bytes)
  - mem_connection_channels (int, bytes)
  - mem_connection_other (int, bytes)
  - mem_queue_procs (int, bytes)
  - mem_queue_slave_procs (int, bytes)
  - mem_plugins (int, bytes)
  - mem_other_proc (int, bytes)
  - mem_metrics (int, bytes)
  - mem_mgmt_db (int, bytes)
  - mem_mnesia (int, bytes)
  - mem_other_ets (int, bytes)
  - mem_binary (int, bytes)
  - mem_msg_index (int, bytes)
  - mem_code (int, bytes)
  - mem_atom (int, bytes)
  - mem_other_system (int, bytes)
  - mem_allocated_unused (int, bytes)
  - mem_reserved_unallocated (int, bytes)
  - mem_total (int, bytes)

- rabbitmq_queue
  - consumer_utilisation (float, percent)
  - consumers (int, int)

  - idle_since (string, time - e.g., "2006-01-02 15:04:05")
  - memory (int, bytes)
  - message_bytes (int, bytes)
  - message_bytes_persist (int, bytes)
  - message_bytes_ram (int, bytes)
  - message_bytes_ready (int, bytes)
  - message_bytes_unacked (int, bytes)
  - messages (int, count)
  - messages_ack (int, count)
  - messages_ack_rate (float, messages per second)
  - messages_deliver (int, count)
  - messages_deliver_rate (float, messages per second)
  - messages_deliver_get (int, count)
  - messages_deliver_get_rate (float, messages per second)
  - messages_publish (int, count)
  - messages_publish_rate (float, messages per second)
  - messages_ready (int, count)
  - messages_redeliver (int, count)
  - messages_redeliver_rate (float, messages per second)
  - messages_unack (integer, count)

- rabbitmq_exchange

  - messages_publish_in (int, count)
  - messages_publish_in_rate (int, messages per second)
  - messages_publish_out (int, count)
  - messages_publish_out_rate (int, messages per second)

Tags:

- All measurements have the following tags:

  - url

- rabbitmq_overview

  - name

- rabbitmq_node

  - node
  - url

- rabbitmq_queue

  - url
  - queue
  - vhost
  - node
  - durable

    - auto_delete

- rabbitmq_exchange
    - url
    - exchange
    - type
    - vhost
    - internal
    - durable
    - auto_delete

4.2. InfluxDB

The InfluxDB helm chart documentation can be found in the following github project: https://github.com/helm/charts/tree/master/stable/influxdb

4.2.1. Install on Kubernetes

Install InfluxDB helm chart on Kubernetes using the following command:

```
helm install --name influxdb stable/influxdb
```

4.2.2. Create Database

In this step, we create an influxdb database for Telegraf to store its metrics data. In order to create this database, follow these steps:

Expose influxdb server installed on Kubernetes in your local machine using this command:

```
kubectl port-forward --namespace default $(kubectl get pods --namespace default -l
app=influxdb -o jsonpath='{ .items[0].metadata.name }') 8086:8086
```

Connect to InfluxDB server using the influx client (to be installed in your machine) by running the following command:

```
influxdb -host localhost -port 8086

Connected to http://localhost:8086 version 1.7.6
InfluxDB shell version: v1.7.7
>
```

Create a database named “**telegraf**” by executing the following SQL command in influx console.

```
> CREATE DATABASE telegraf
>
```

When the command executes properly by influx nothing is returned, otherwise, an error will be returned. You can check the created database by running this command:

```
> SHOW DATABASES
name: databases
name
----
_internal
```

```
telegraf
>
```

4.3. Grafana

Grafana's official Kubernetes helm chart can be found at the following location:
https://github.com/helm/charts/tree/master/stable/grafana

4.3.1. Install on Kubernetes

Install Grafana's helm chart using the following helm command:

```
helm install --name grafana stable/grafana -set persistence.enabled=true,persistence.type=pvc
```

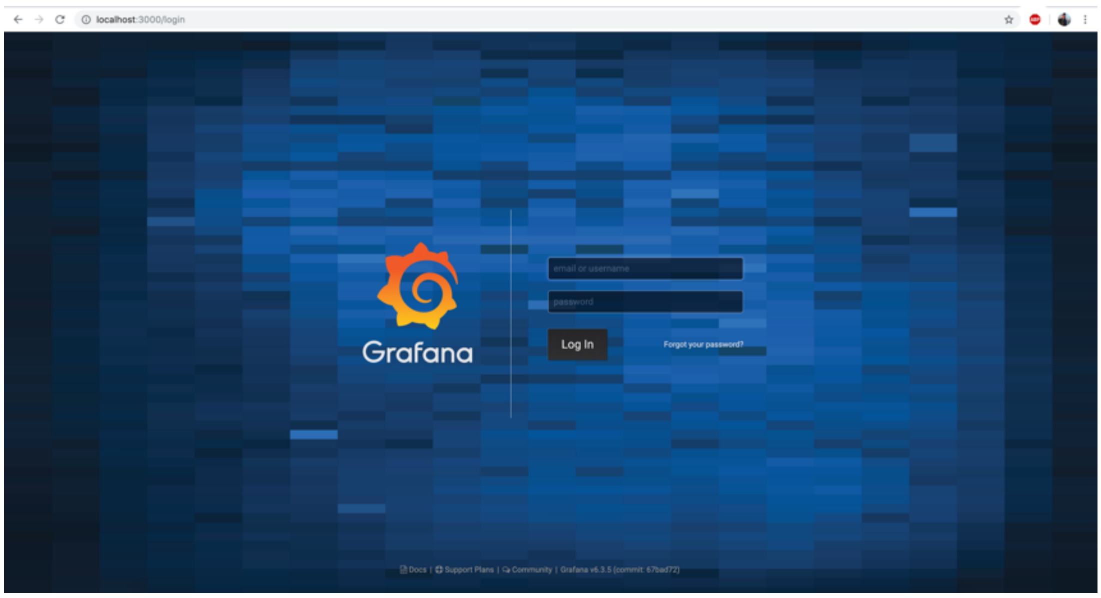
Get Grafana's auto-generated admin password from Grafana secret:

```
kubectl get secret --namespace default grafana -o jsonpath="{.data.admin-password}" | base64
--decode ; echo
```

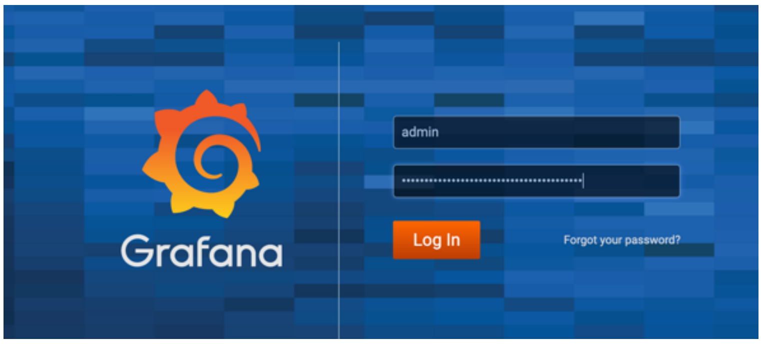
Expose Grafana server to your local machine in order to configure it:

```
kubectl --namespace default port-forward $(kubectl get pods --namespace default -l
"app=grafana,release=grafana" -o jsonpath="{.items[0].metadata.name}") 3000
```

Connect from your browser to Grafana server at the url http://localhost:3000.

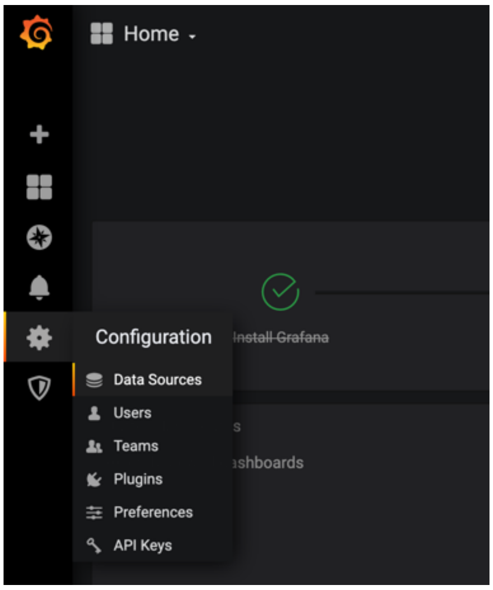


Use the "**admin**" username and the password you got from the Grafana secret to log in.

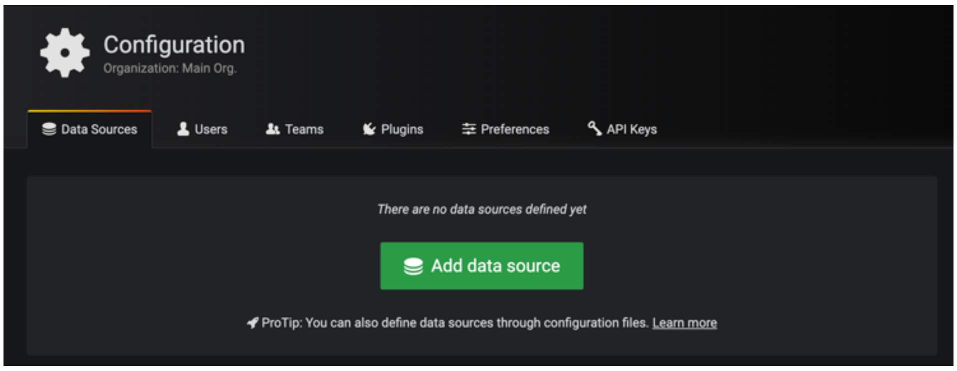


4.3.2. Create InfluxDB Data Source

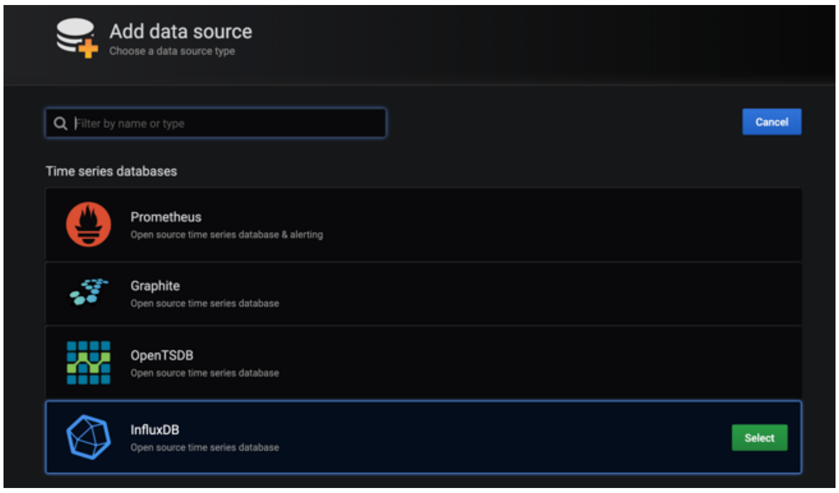
After logging into Grafana, open the Data Sources menu to add a new data source linked to our influxdb server.



Hit the "**Add data source**" button:

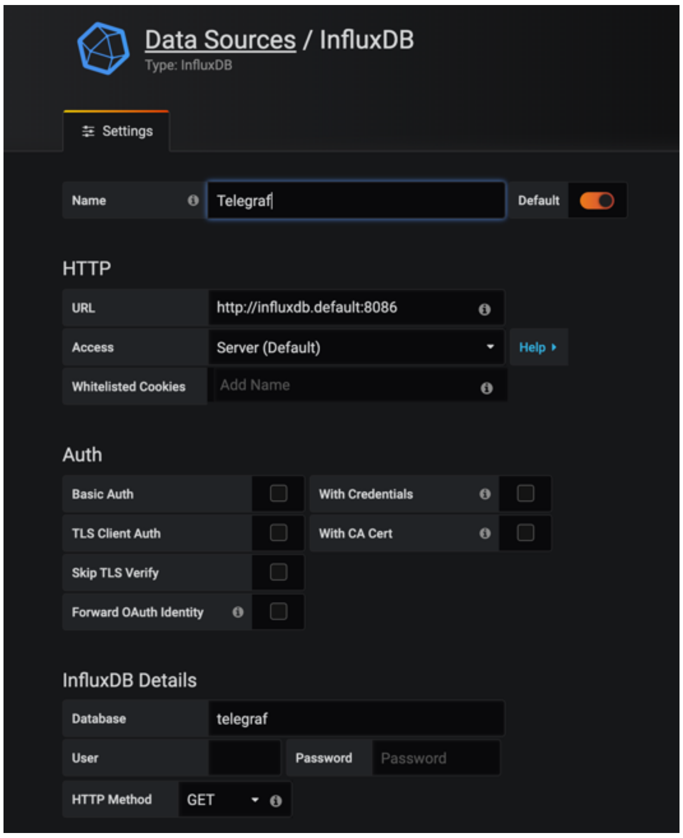

Configuration
Organization: Main Org.
Data Sources
Users
Teams
Plugins
Preferences
API Keys
There are no data sources defined yet
Add data source
ProTip: You can also define data sources through configuration files. Learn more

Select InfluxDB data source:


Add data source
Choose a data source type
Filter by name or type
Cancel
Time series databases
Prometheus
Open source time series database & alerting
Graphite
Open source time series database
OpenTSDB
Open source time series database
InfluxDB
Open source time series database
Select

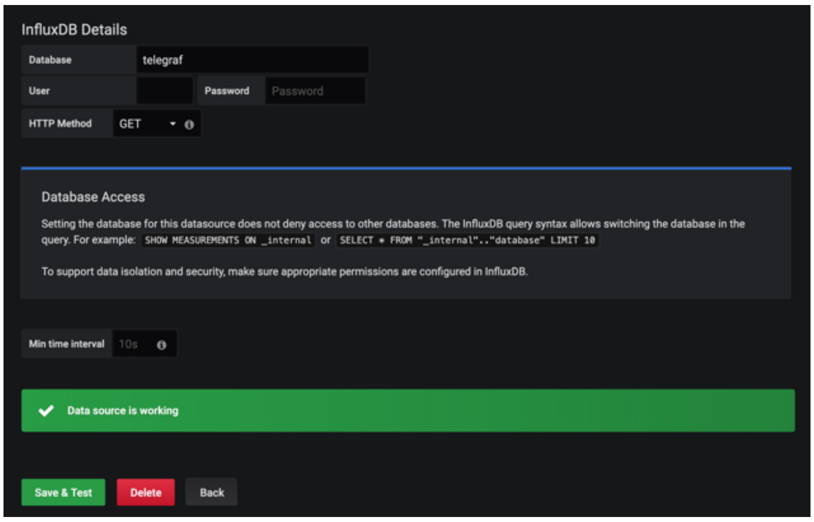
Provide the host of the influxDB service in Kubernetes and the name of the database "**telegraf**". Name your data source "**Telegraf**".

InfluxDB Host	http://influxdb.default:8086 or http://influxdb.default.svc.cluster.local:8086
InfluxDB Database Name	telegraf
Data Source Name	Telegraf



Name: Telegraf
Default: on
URL: http://influxdb.default:8086
Access: Server (Default)
Whitelisted Cookies: Add Name
Basic Auth: ☐
With Credentials: ☐
TLS Client Auth: ☐
With CA Cert: ☐
Skip TLS Verify: ☐
Forward OAuth Identity: ☐
Database: telegraf
User:
Password: Password
HTTP Method: GET

Press the "**Save & Test**" button to create the data source. If the creation is successful, Grafana will display a green message "**Data source is working**"


InfluxDB Details

Database: telegraf

User: Password: Password

HTTP Method: GET

Database Access

Setting the database for this datasource does not deny access to other databases. The InfluxDB query syntax allows switching the database in the query. For example: `SHOW MEASUREMENTS ON _internal` or `SELECT * FROM "_internal".."database" LIMIT 10`

To support data isolation and security, make sure appropriate permissions are configured in InfluxDB.

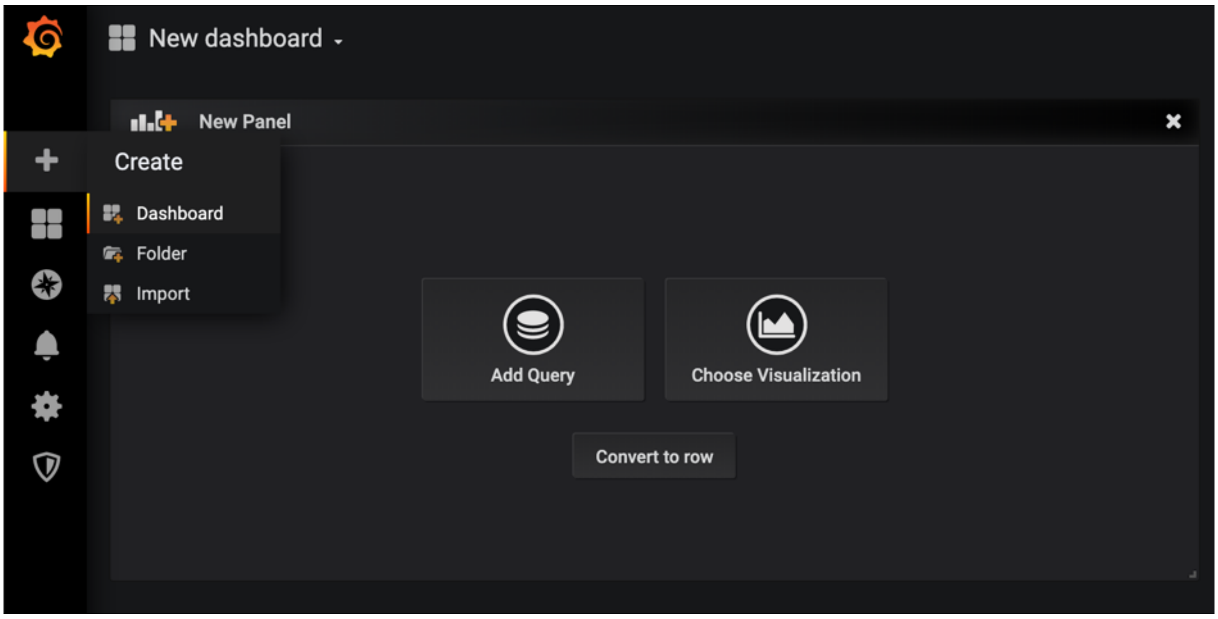
Min time interval: 10s

Data source is working

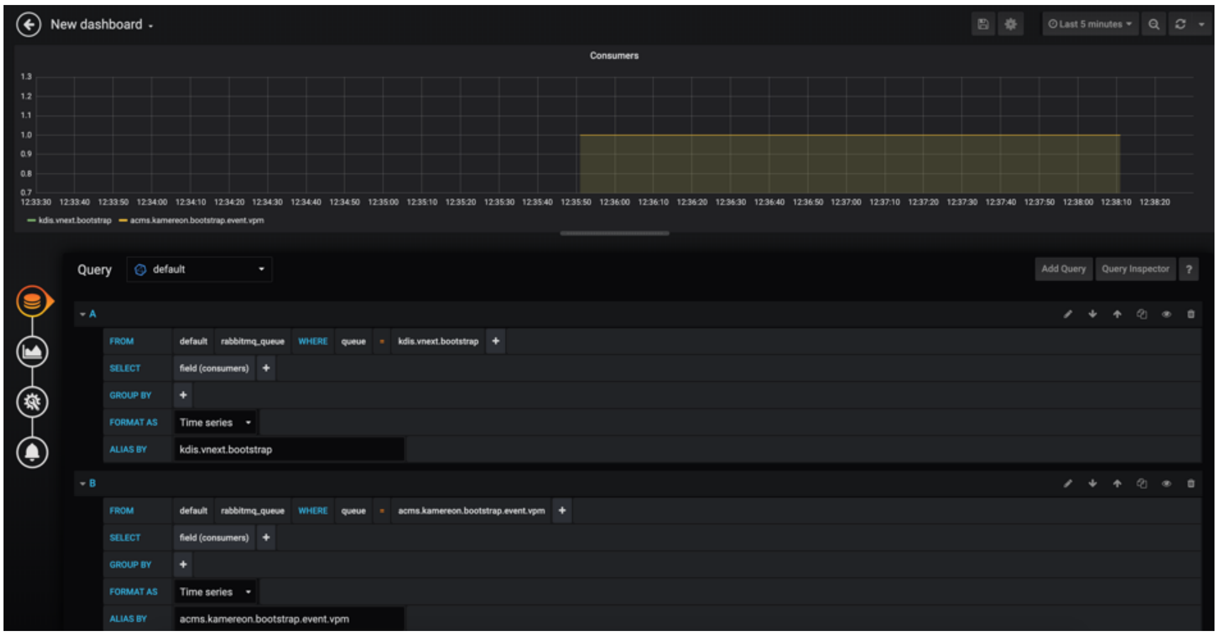
4.3.3. Create Dashboards

A dashboard can be created by constructing a query to influxdb database.

Start by adding a new dashboard, and a new panel to this dashboard using a query (press the “**Add Query**” button)



Select the data source “**Telegraf**” and create a SQL query using the Wizard. Give each query an alias which represents the legend to be associated with the result:



Query: default

A
FROM default rabbitmq_queue WHERE queue = kdis.vnext.bootstrap
SELECT field (consumers)
GROUP BY
FORMAT AS Time series
ALIAS BY kdis.vnext.bootstrap

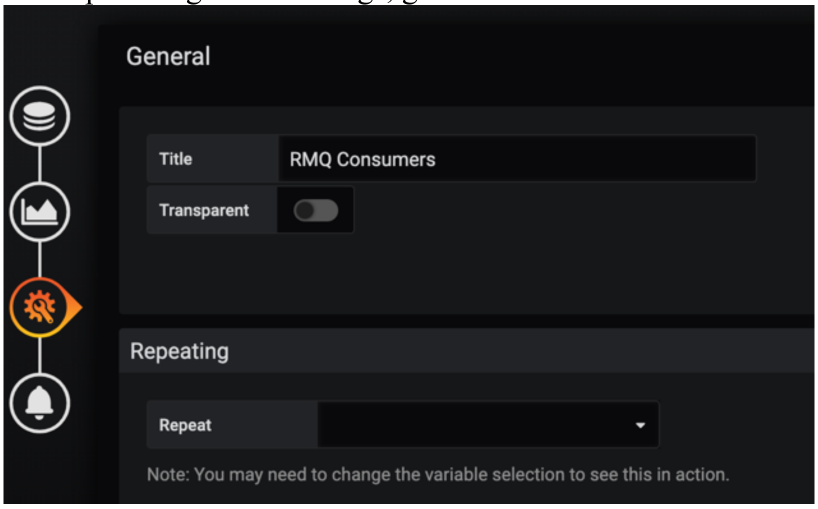
B
FROM default rabbitmq_queue WHERE queue = acms.kamereon.bootstrap.event.vpm
SELECT field (consumers)
GROUP BY
FORMAT AS Time series
ALIAS BY acms.kamereon.bootstrap.event.vpm

Example Queries:

SELECT "consumers" FROM "rabbitmq_queue" WHERE ("queue" = 'kdis.vnext.bootstrap') AND time >= now() - 3h

SELECT "consumers" FROM "rabbitmq_queue" WHERE ("queue" = 'kdis.vnext.bootstrap');
SELECT "consumers" FROM "rabbitmq_queue" WHERE ("queue" = 'acms.kamereon.bootstrap.event.vpm')
SELECT "consumers" FROM "rabbitmq_queue" WHERE ("queue" = 'acms.kamereon.bootstrap.event.vnext')

In the panel's general settings, give it a name that reflects the nature of the query:



General

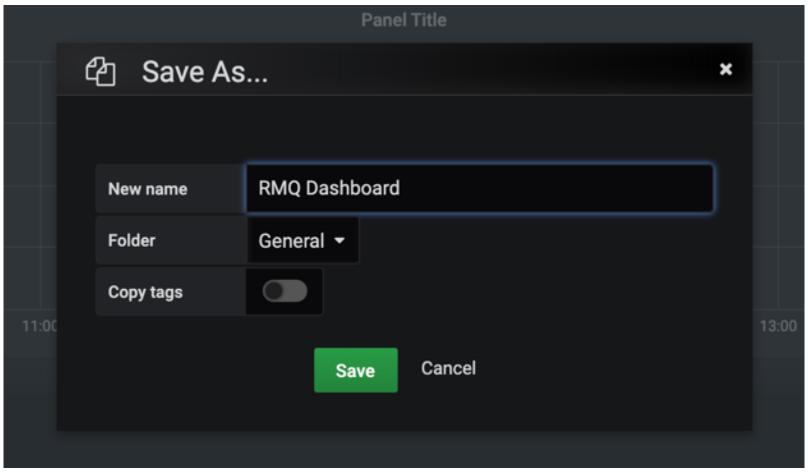
Title: RMQ Consumers
Transparent

Repeating

Repeat

Note: You may need to change the variable selection to see this in action.

Save the dashboard and give it a name:

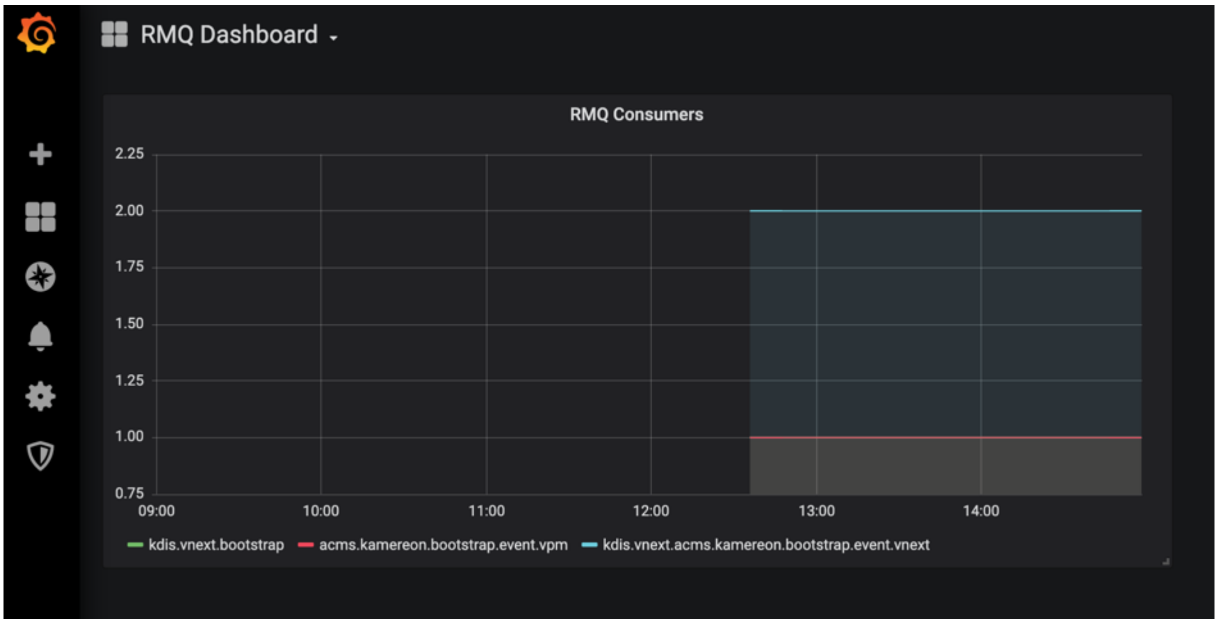

Save As...

New name: RMQ Dashboard

Folder: General

Copy tags: off

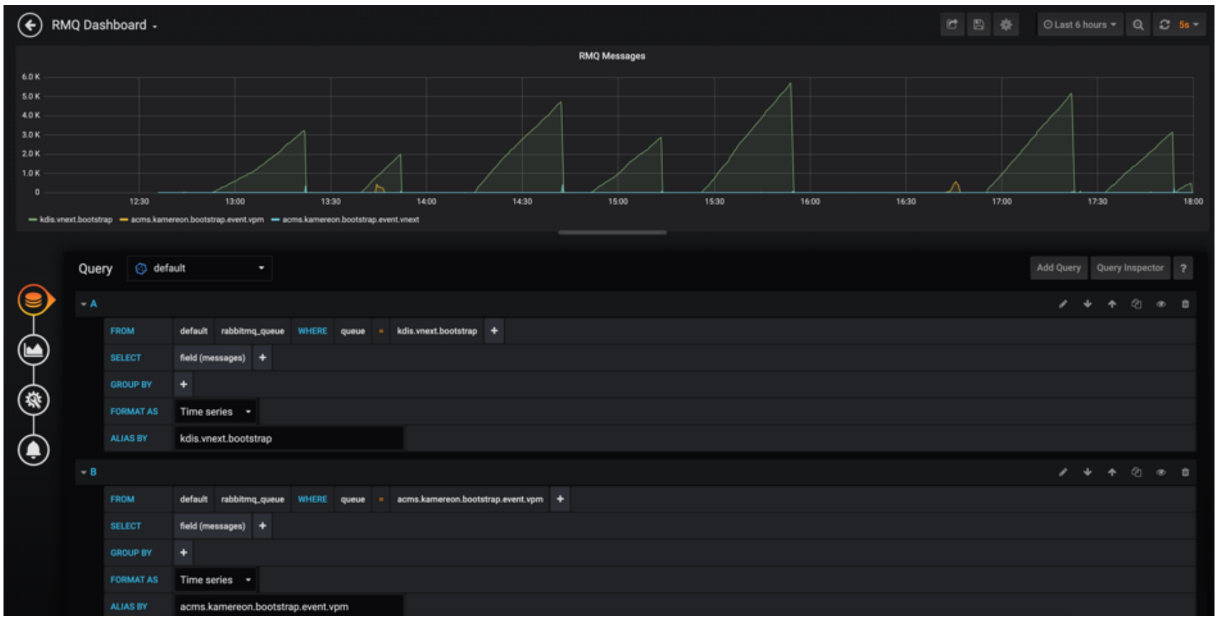
The dashboard will then appear with the first panel we added:



Add another panel to our dashboard to get the messages received per queue. The query template to be used in this case is:

SELECT "messages" FROM "rabbitmq_queue" WHERE ("queue" = 'kdis.vnext.bootstrap') AND time >= now() - 6h

Name this panel “**RMQ Messages**”



A

FROM	default	rabbitmq_queue	WHERE	queue	=	kdis.vnext.bootstrap
SELECT	field (messages)					
GROUP BY						
FORMAT AS	Time series					
ALIAS BY	kdis.vnext.bootstrap					

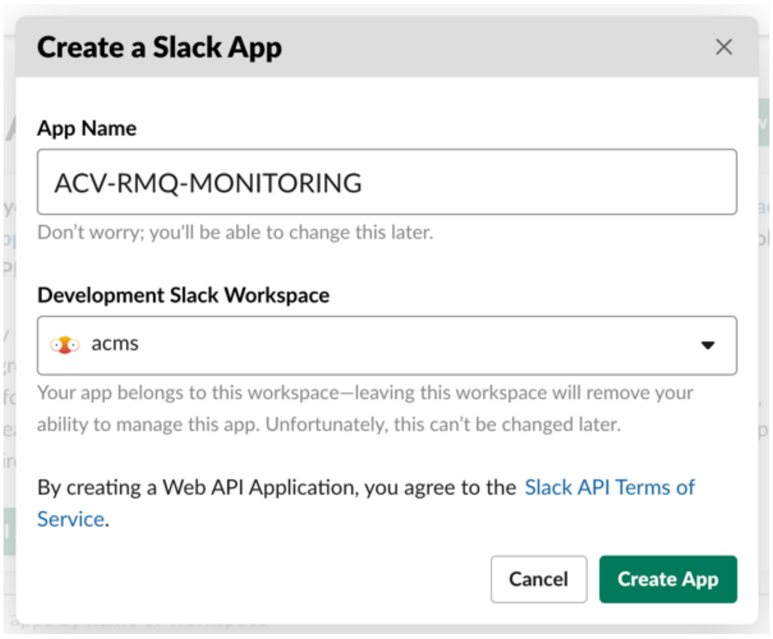
B

FROM	default	rabbitmq_queue	WHERE	queue	=	acms.kamereon.bootstrap.event.vpm
SELECT	field (messages)					
GROUP BY						
FORMAT AS	Time series					
ALIAS BY	acms.kamereon.bootstrap.event.vpm					

4.3.4. Add Slack Notification

In order to connect Grafana to Slack to send notifications, we must go through these steps:

Create a Slack Application and bind it to a workspace (slack organization) at: https://api.slack.com/apps



Create a Slack App

App Name

ACV-RMQ-MONITORING

Don't worry; you'll be able to change this later.

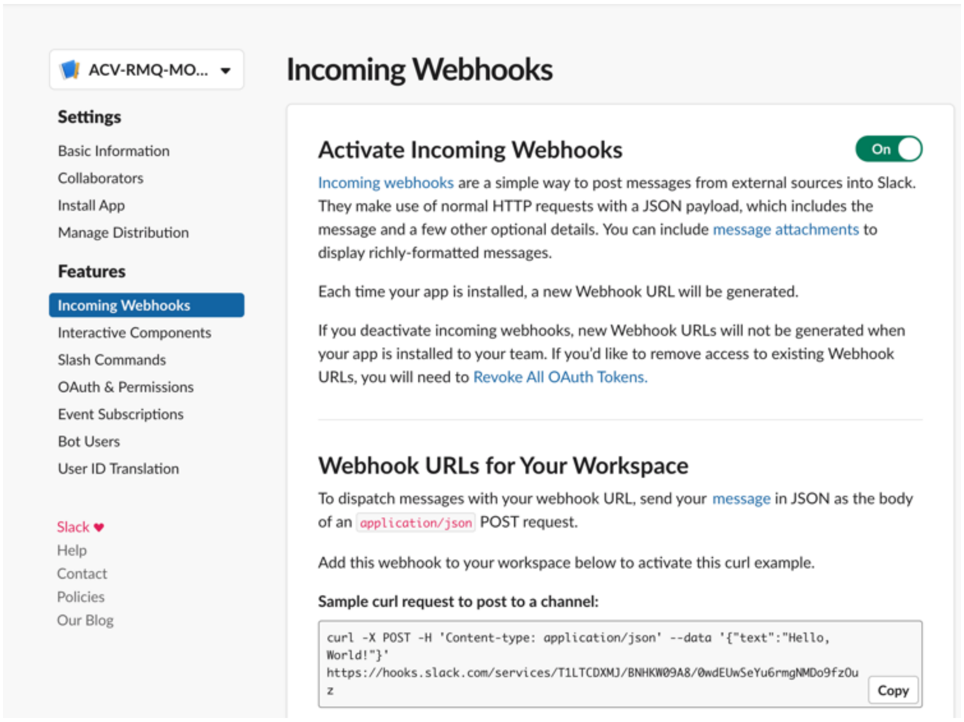
Development Slack Workspace

acms

Your app belongs to this workspace—leaving this workspace will remove your ability to manage this app. Unfortunately, this can't be changed later.

By creating a Web API Application, you agree to the Slack API Terms of Service.

Enable incoming webhooks for the slack application:



Incoming Webhooks

Activate Incoming Webhooks

Incoming webhooks are a simple way to post messages from external sources into Slack. They make use of normal HTTP requests with a JSON payload, which includes the message and a few other optional details. You can include message attachments to display richly-formatted messages.

Each time your app is installed, a new Webhook URL will be generated.

If you deactivate incoming webhooks, new Webhook URLs will not be generated when your app is installed to your team. If you'd like to remove access to existing Webhook URLs, you will need to Revoke All OAuth Tokens.

Webhook URLs for Your Workspace

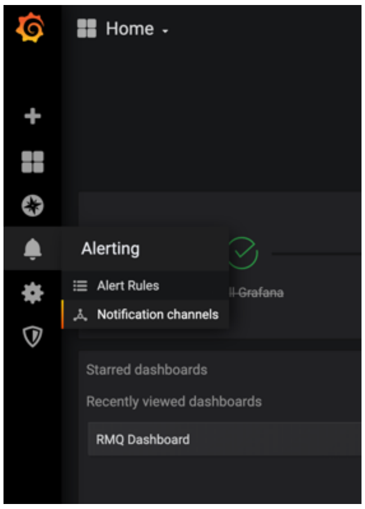
To dispatch messages with your webhook URL, send your message in JSON as the body of an `application/json` POST request.

Add this webhook to your workspace below to activate this curl example.

Sample curl request to post to a channel:

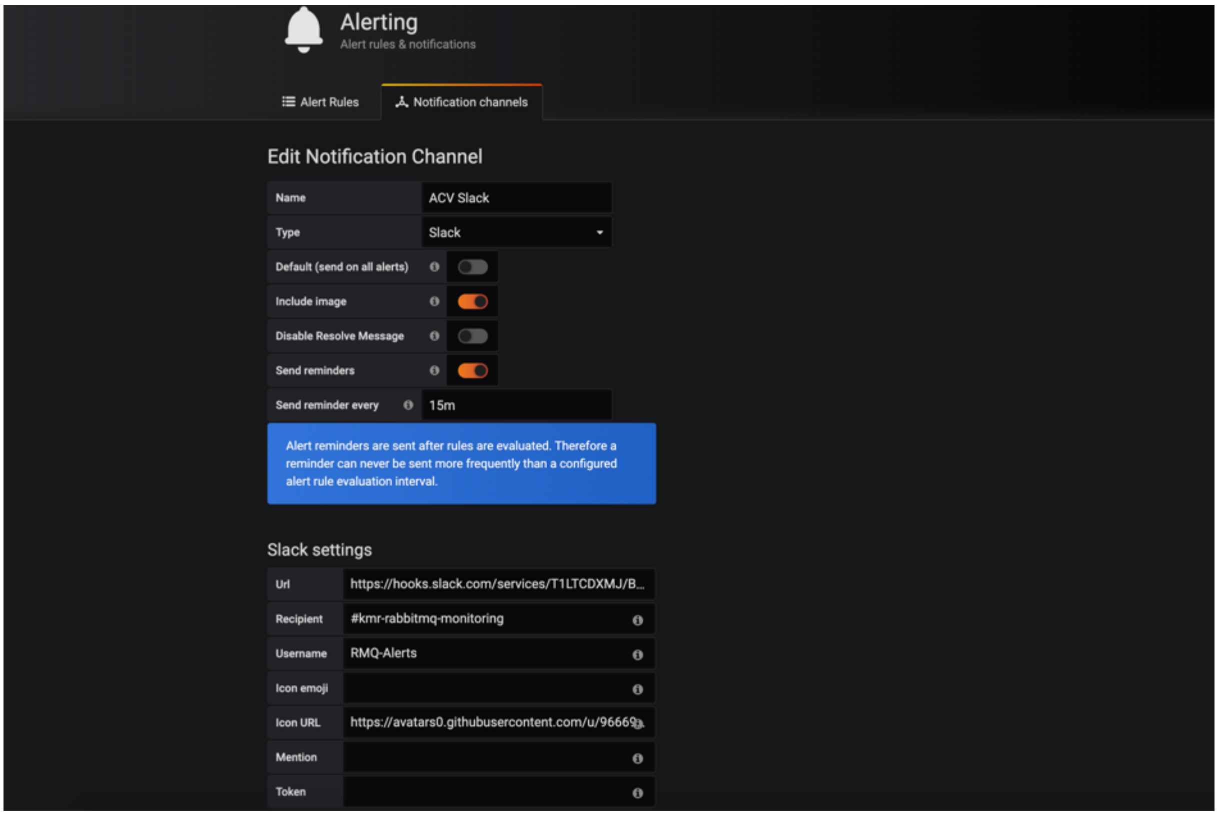
```
curl -X POST -H 'Content-type: application/json' --data '{"text":"Hello,
World!"}'
https://hooks.slack.com/services/T1LTCDXMJ/BNHKW09A8/0wdEUwSeYu6rmgNMDo9fzOu
z
```

Add a Slack notification channel in Grafana. Open Alerting > Notification Channels



Create a new channel named “**ACV Slack**” and provide all required information:

Name	ACV Slack
Type	Slack
Send Reminders	ON
URL	Slack App Webhook URL
Recipient	Slack channel that will receive the alerts



Edit Notification Channel

Name	ACV Slack
Type	Slack
Default (send on all alerts)	off
Include image	on
Disable Resolve Message	off
Send reminders	on
Send reminder every	15m

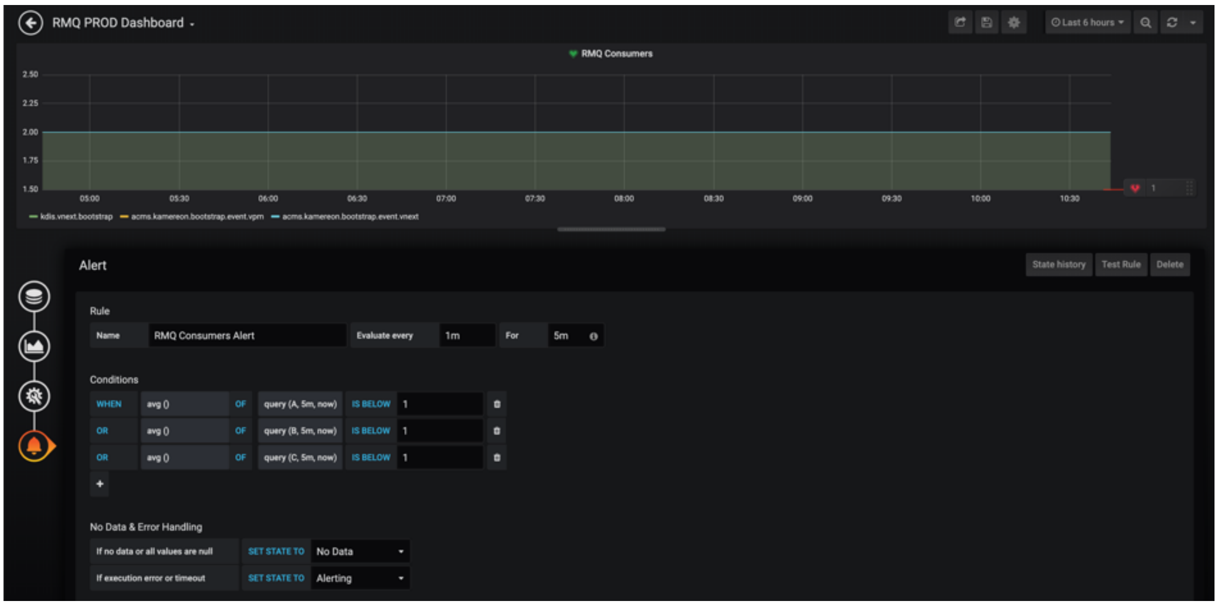
Alert reminders are sent after rules are evaluated. Therefore a reminder can never be sent more frequently than a configured alert rule evaluation interval.

Slack settings

Url	https://hooks.slack.com/services/T1LTCDXMJ/B...
Recipient	#kmr-rabbitmq-monitoring
Username	RMQ-Alerts
Icon emoji	
Icon URL	https://avatars0.githubusercontent.com/u/96669
Mention	
Token	

4.3.5. Create Consumers Alert

To create an alert for a panel, edit the panel settings and open its alert menu:



Rule

Name	RMQ Consumers Alert	Evaluate every	1m	For	5m

Conditions

WHEN	avg ()	OF	query (A, 5m, now)	IS BELOW	1
OR	avg ()	OF	query (B, 5m, now)	IS BELOW	1
OR	avg ()	OF	query (C, 5m, now)	IS BELOW	1

No Data & Error Handling

If no data or all values are null	SET STATE TO	No Data
If execution error or timeout	SET STATE TO	Alerting

5. Security

5.1. OAuth Azure AD

Access to the monitoring cluster can be configured to use Azure AD Single Sign-On. This can be achieved in 2 steps:

1. Create Azure AD Application
2. Configure Grafana to use Azure AD application login URL for oauth2.

Grafana documentation for activation of azure ad oauth2 protocol:
https://grafana.com/docs/auth/generic-oauth/#set-up-oauth2-with-azure-active-directory

In AKS, we can use the following configmap to configure Grafana for Azure AD OAuth2:

```
apiVersion: v1
data:
  grafana.ini: |
    [analytics]
    check_for_updates = true
    [grafana_net]
    url = https://grafana.net
    [log]
    mode = console
    [paths]
    data = /var/lib/grafana/data
    logs = /var/log/grafana
    plugins = /var/lib/grafana/plugins
    provisioning = /etc/grafana/provisioning
    [server]
    domain = aks-monitor-development.francecentral.cloudapp.azure.com
    #root_url = %(protocol)s://%(domain)s/
    root_url = https://aks-monitor-development.francecentral.cloudapp.azure.com
    [auth.generic_oauth]
    name = Azure AD
    enabled = true
    allow_sign_up = true
    client_id = xxxxxxxxxxxxxxxx
    client_secret = xxxxxxxxxxxxxxxx
    scopes = openid email name
    auth_url = https://login.microsoftonline.com/xxxxxxxxxxxxxxxx/oauth2/authorize
    token_url = https://login.microsoftonline.com/xxxxxxxxxxxxxxxx/oauth2/token
    api_url =
    team_ids =
    allowed_organizations =
kind: ConfigMap
metadata:
  labels:
    app: grafana
  name: grafana
```

5.2. Ingress & TLS

It’s is possible to use Let’s encrypt to manage and issue valid TLS certificates for ingress. This process can be automated in Kubernetes using Cert Manager service:
https://github.com/jetstack/cert-manager

Azure documentation to configuring ingress tls with cert manager:

https://docs.microsoft.com/th-th/azure/aks/ingress-tls
First, install cert manager CRDs and then helm chart in a separate namespace as in below:

```
# Install the CustomResourceDefinition resources separately
kubectl apply -f https://raw.githubusercontent.com/jetstack/cert-manager/release-
0.8/deploy/manifests/00-crds.yaml

# Create the namespace for cert-manager
kubectl create namespace cert-manager

# Label the cert-manager namespace to disable resource validation
kubectl label namespace cert-manager certmanager.k8s.io/disable-validation=true

# Add the Jetstack Helm repository
helm repo add jetstack https://charts.jetstack.io

# Update your local Helm chart repository cache
helm repo update

# Install the cert-manager Helm chart
helm install \
  --name cert-manager \
  --namespace cert-manager \
  --version v0.10.0 \
  jetstack/cert-manager
```

Create a staging (or production) let's encrypt issuer. Test with staging then move to production once the certificate issuing is functional and ready. Create the cluster issuer resource using "*kubectl apply -f*" command

```
apiVersion: certmanager.k8s.io/v1alpha1
kind: ClusterIssuer
metadata:
  name: letsencrypt-prod
  namespace: ingress-basic
spec:
  acme:
    server: https://acme-staging-v02.api.letsencrypt.org/directory
    email: user@contoso.com
    privateKeySecretRef:
      name: letsencrypt-prod
    http01: {}
```

Create an nginx ingress route that requires tls and uses the let's encrypt issuer we created. The ingress below is used with a Grafana service installed on the cluster.

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: grafana
  annotations:
    kubernetes.io/ingress.class: "nginx"
    certmanager.k8s.io/issuer: "letsencrypt-prod"

spec:
  tls:
  - hosts:
    - aks-monitor-development.francecentral.cloudapp.azure.com
    secretName: grafana-tls
  rules:
  - host: aks-monitor-development.francecentral.cloudapp.azure.com
    http:
      paths:
      - path: /
        backend:
          serviceName: grafana
          servicePort: 80
```

Create a certificate resource (defined by cert manager CRD) to generate the let's encrypt certificate for our ingress. This step becomes optional starting from Cert Manager v0.2.2

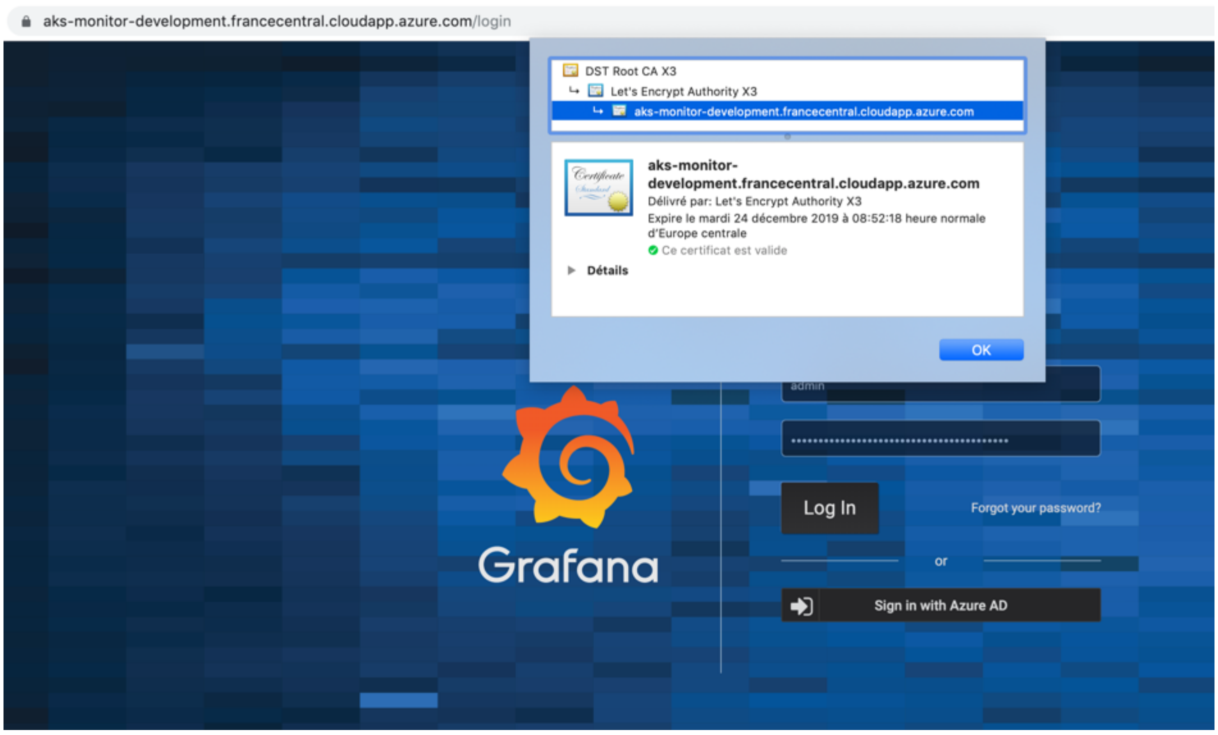
which automatically creates the certificate resource based on ingress and the domain to which it is associated.

```
apiVersion: certmanager.k8s.io/v1alpha1
kind: Certificate
metadata:
  name: grafana-tls
  namespace: default
spec:
  secretName: grafana-tls
  dnsNames:
  - aks-monitor-development.francecentral.cloudapp.azure.com
  acme:
    config:
    - http01:
        ingressClass: nginx
      domains:
      - aks-monitor-development.francecentral.cloudapp.azure.com
  issuerRef:
    name: letsencrypt-prod
    kind: ClusterIssuer
```

Run the command **"kubectl describe certificate <cert-name>"** (cert name is grafana-tls for example) to check the status of the certificate.

```
Type    Reason          Age   From          Message
----    ------          ----  ----          -------
  Normal  CreateOrder     11m   cert-manager  Created new ACME order, attempting validation...
  Normal  DomainVerified  10m   cert-manager  Domain "demo-aks-ingress.eastus.cloudapp.azure.com" verified with "http-01" validation
  Normal  IssueCert       10m   cert-manager  Issuing certificate...
  Normal  CertObtained    10m   cert-manager  Obtained certificate from ACME server
  Normal  CertIssued      10m   cert-manager  Certificate issued successfully
```

Check the status of the tls certificate on the browser. You must see a valid certificate when using let's encrypt production issuer:


aks-monitor-development.francecentral.cloudapp.azure.com/login

DST Root CA X3
↳ Let's Encrypt Authority X3
↳ aks-monitor-development.francecentral.cloudapp.azure.com

aks-monitor-development.francecentral.cloudapp.azure.com
Délivré par: Let's Encrypt Authority X3
Expire le mardi 24 décembre 2019 à 08:52:18 heure normale d'Europe centrale
Ce certificat est valide

6. Terraform Sources

The terraform project for creating the monitoring cluster can be retrieved from the following git repository:

git@ssh.dev.azure.com:v3/acvarchitecture/sample-spring-api/cloud-config