Monitoring OpenVMS Servers using RabbitMQ, Prometheus and Grafana

Introduction

Options to manage and administer the OpenVMS server estate in the past were quite limited, with T4 being the usual choice. Today there are more technology options to choose from by using the VSI layered product VMSSPI. SPI can send alerts to several monitoring solutions including RabbitMQ, Dynatrace, Slunk, Slack and via a MQTT broker to any monitoring solution that supports MQTT.

This article shows how to monitor OpenVMS using RabbitMQ.

The technology stack to monitor OpenVMS servers via RabbitMQ is quite complicated, see **Figure 1**.

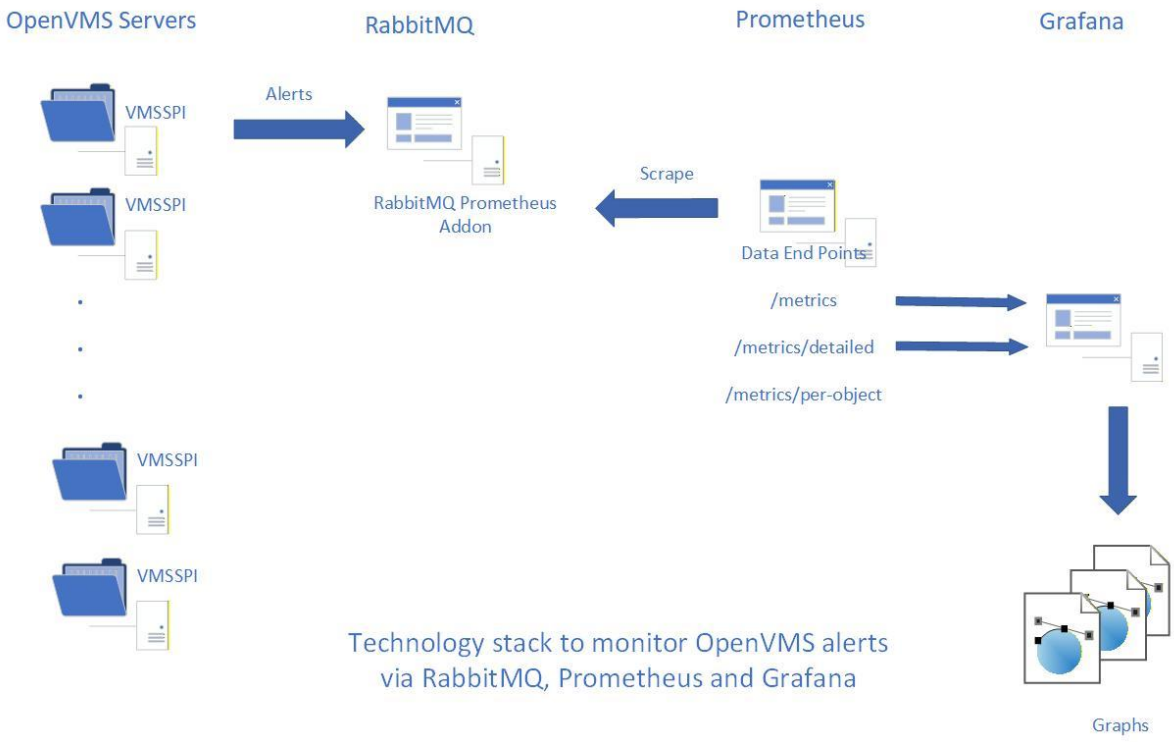

Figure 1

The OpenVMS servers send alerts to RabbitMQ via the VMSSPI layered product. RabbitMQ is an open-source message broker that facilitates communication between different applications by accepting, storing and forwarding messages. Periodically Prometheus scrapes RabbitMQ and receives the alerts in the form of messages.

Prometheus is an open-source monitoring and alerting application. Prometheus usually presents aggregated statistics via the /metrics end point. In this case we need slightly more detail and therefore this configuration will use the /metrics/detailed end point to get aggregated statistics for individual queues. Prometheus can present a per-object endpoint on /metrics/per-object to get individual statistics allowing a deeper investigation into the alerts, but per-object queries have a large adverse affect on performance so we will avoid using that end point in this configuration.

Grafana is an open-source analytics and monitoring application that allows users to query, visualize, and create alerts on metrics and data from various sources, in this case from Prometheus.

OpenVMS Configuration

One provisor is that OpenVMS must be using the native TCPIP services otherwise VMSSPI is not an option. VMSSPI can be downloaded and tested free for 30 days, once that period as expired it requires a license.

Download **<arch>VMS-VMSSPI-V0900-27-1** from the VSI service site. There are three variants of the package, replace <arch> with X86 for intel, I64 for Itanium and AXP for AXP systems. There is also a detailed user manual that guides you through the installation which is a standard procedure for layered products.

VMSSPI for VSI OpenVMS User Guide

Once installed SPI can be controlled with a couple of commands.

1. SET DEF VMSSPI$DATA - Sets the default directory to the SPI data directory.
2. @ SYS$STARTUP:VMSSPI$STARTUP.COM – Starts the three processes the make up SPI.
3. @ SYS$STARTUP:VMSSPI$SHUTDOWN.COM – Stops the three processes.

I noticed that occasionally when stopping the processes, they didn't all stop after issuing the shutdown command. So check that the processes are all stopped when shutting down and all started when starting up.

```
PIPE SHOW SYSTEM | SEARCH SYS$INPUT VMSSPI
```

If any process is still running after shutdown it can be stopped using the stop command and the process id which is listed from the piped show system command above.

```
STOP/ID=<PRO_ID>
```

SPI can be configured using two files, VMSSPI$CONFIGURATION.DAT and MESSAGES.TXT. The installed default configuration will allow all alerts to be monitored. This could overload the system with too many messages, so initially certain classes of event could be disabled in the configuration file by editing the security filter settings. This is fully explained in the VMSSPI User Guide.

To configure monitoring all that is required is to edit the relevant monitoring module entry in the Messages file. To enable monitoring for RabbitMQ the following line needs to be added to the Messages.txt file

```
use interface module "vmsspi$root:[lib]vmsspi$rabbitmq_shr.exe"
        "amqp://<username>:<password>@<rabbit host>:5673/<vhost>";
```

e.g

```
use interface module "vmsspi$root:[lib]vmsspi$rabbitmq_shr.exe"
        "amqp://guest:guest@192.168.1.31:5673/openvms";
```

Only two alerting modules can be enabled at any one time, so check the previous 'use interface' lines to make sure they are commented with a leading '#'.

Once the edit is complete shutdown and startup the SPI processes and OpenVMS configuration is complete.

RabbitMQ Configuration

There are many ways to configure RabbitMQ to filter or separate the OpenVMS Alert types. In this configuration we are separating the alert severities into their own unique queues. If there are only a few OpenVMS hosts we could also configure such that each OpenVMS server has its own queue(s) but this will quickly increase the number of queues to be unmanageable.

The article isn't an in depth RabbitMQ configuration document so some understanding of RabbitMQ is required.

Create a new virtual host (vhost), select the 'Admin' tab and 'Virtual Hosts' from the righthand menu, see **Figure 2**.

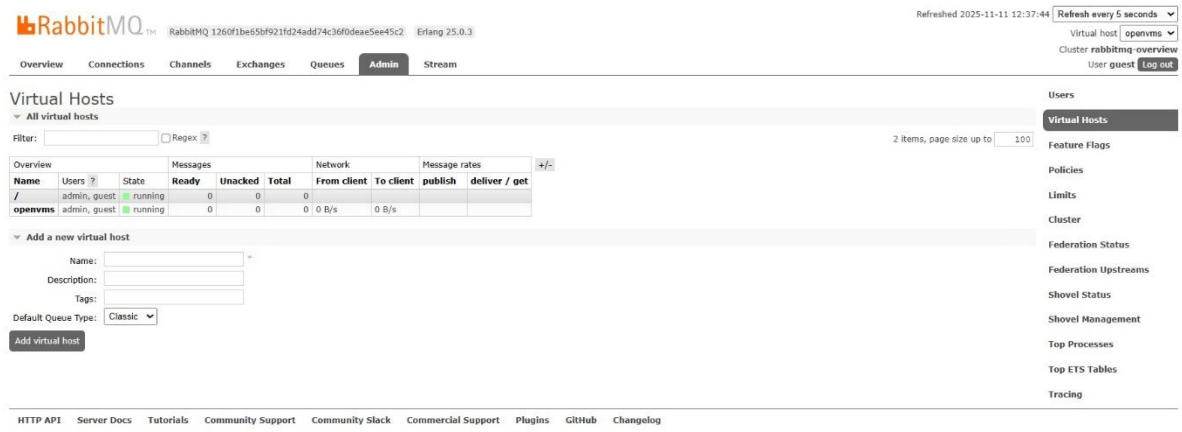

Figure 2

We created the 'openvms' vhost, make sure the user to be used in the VMSSPI connection has access to the vhost.

Create a topic Exchange so that queues can be bound to it, see **Figure 3**.

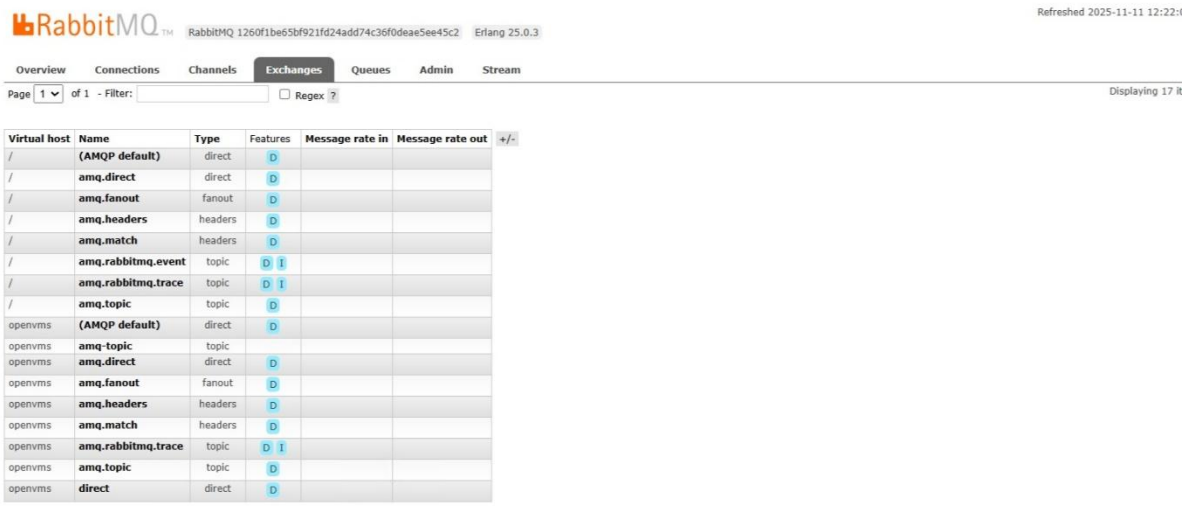

Figure 3

In the figure above, the topic exchange to be used is called amq.topic.

Create queues for the OpenVMS alert types you want to monitor. The severities are critical, major, minor, normal, unknown and none, see **Figure 4**.

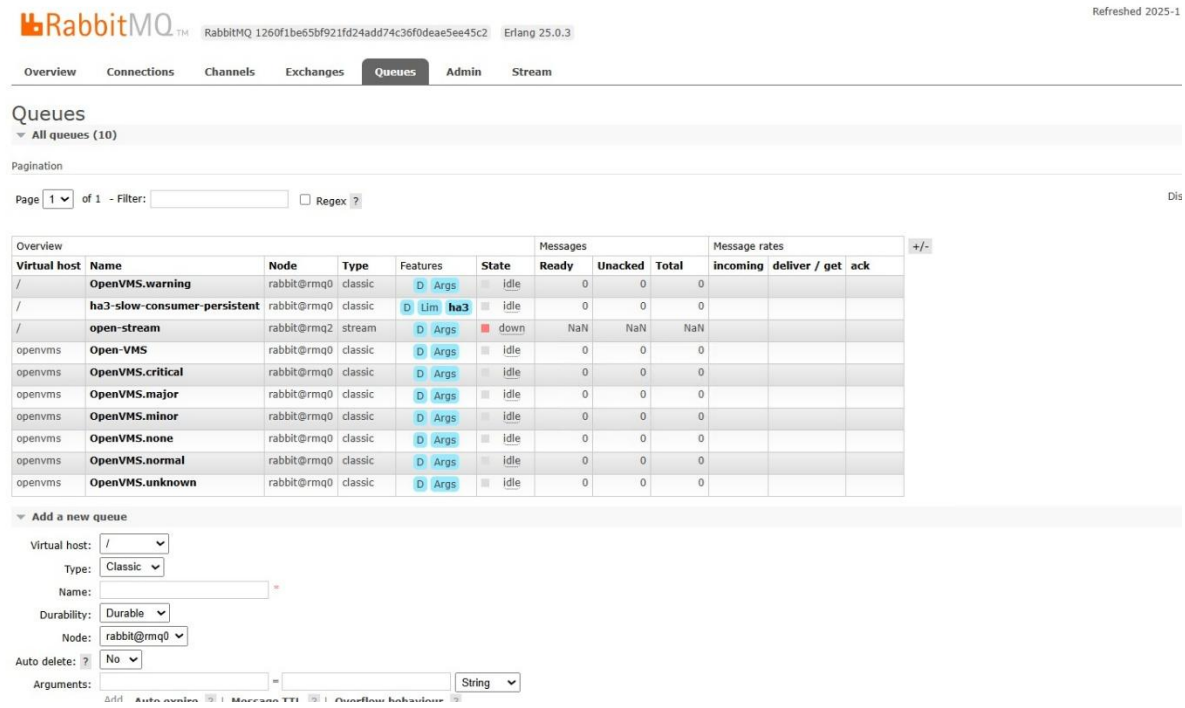

Figure 4

The routing key from VMSSPI used to direct alerts to a particular queue as the form

vmsspi.<org id>.<host>.<severity>

So for the OpenVMS.critical queue a routing key of vmsspi.*.*.critical was specified, see **Figure 5**. This causes all alerts starting with vmsspi and ending in critical to be added to the queue.

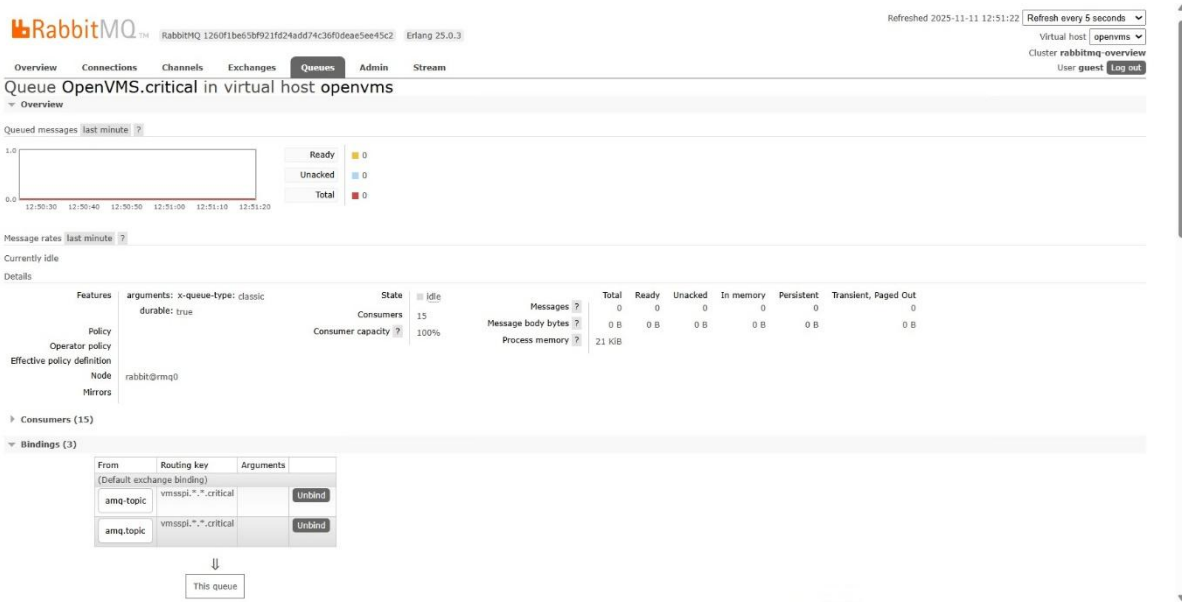

Figure 5

So each queue has one or more routing key(s) created when the queue is added.

Hopefully you can see that particular hosts could have their own queues by adding the hostname rather than a wildcard in that position of the routing key.

In the queue addition, the exchange to bind to is specified. Once the queues are created the bindings can be viewed in the Exchanges tab, see **Figure 6**.

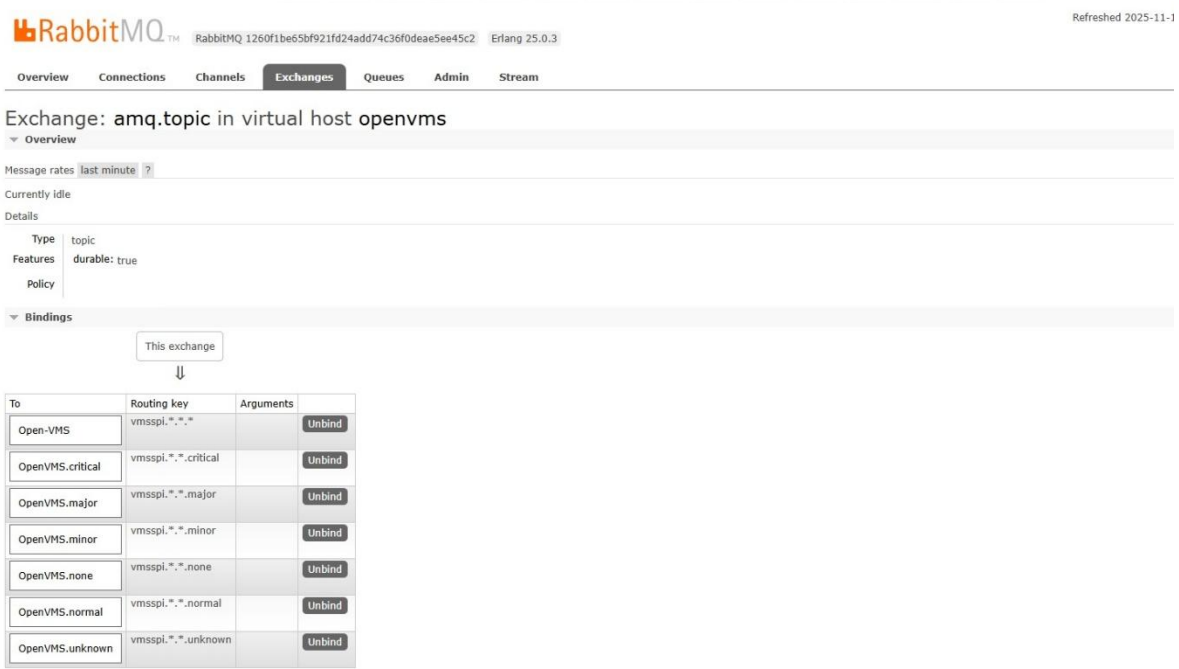

Figure 6

Prometheus Configuration

The Prometheus configuration is mainly carried out in the /etc/prometheus/prometheus.yml file. The article isn't an in depth Prometheus configuration document so some understanding of Prometheus is required.

A sample prometheus.yml file is shown below.

```
---
global:
  scrape_interval: 15s
alerting:
  alertmanagers:
    - static_configs:
        - targets: null
rule_files:
  - /etc/prometheus/rule1.yml
scrape_configs:
  - job_name: prometheus
    static_configs:
      - targets:
          - localhost:9090
  - job_name: docker
    static_configs:
      - targets:
          - docker.for.mac.localhost:9323
  - job_name: node-exporter
    static_configs:
      - targets:
          - node-exporter:9100
  - job_name: rabbitmq-server
    metrics_path: /metrics
    static_configs:
      - targets:
          - rmq0:15692
          - rmq1:15692
          - rmq2:15692
  - job_name: rabbitmq-server-detailed
    metrics_path: /metrics/detailed
    params:
      family:
        - queue_coarse_metrics
        - queue_consumer_count
        - channel_queue_metrics
        - channel_queue_exchange_metrics
        - channel_metrics
        - connection_metrics
        - connection_coarse_metrics
        - channel_exchange_metrics
```

```
    static_configs:
     - targets:
        - rmq0:15692
        - rmq1:15692
        - rmq2:15692
    metric_relabel_configs:
     - source_labels:
        - queue_vhost
       regex: queue_vhost="(.*)"
       action: replace
       target_label: vhost
       replacement: $1
```

The main sections of the configuration file are:-

global
Sets the scrape interval for Prometheus to scrape the endpoints.

rule_files
Specifies additional yml files that hold rules, some of the Grafana dashboards require specific rules in Prometheus.

scrape_configs
Section that specifies the functionality that will be active. Notice the rabbitmq-server and rabbitmq-server-detailed sub sections. These are the two exposed endpoints that Grafana will use to gather data. Other sections provide data for monitoring Prometheus itself and the node-exporter is an agent that gathers system metrics and exposes them for Prometheus to ingest. In the configuration RabbitMQ, Prometheus and Grafana are all running in Docker containers so its sensible to monitor Docker also.

Once Prometheus is configured and started it can be monitored via the GUI frontend on port 9090 (default port). Selecting Status -> Targets shows the health of the configured endpoints,see **Figure 7**.

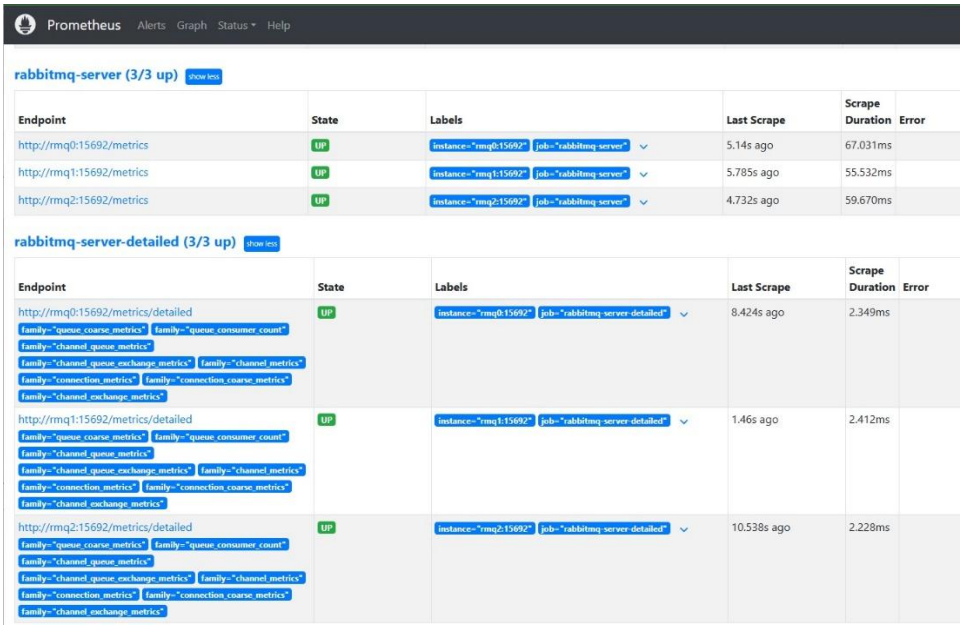
Prometheus
Alerts
Graph
Status
Help
rabbitmq-server (3/3 up)
show less
Endpoint
State
Labels
Last Scrape
Scrape Duration
Error
http://rmq0:15692/metrics
UP
instance="rmq0:15692"
job="rabbitmq-server"
5.14s ago
67.031ms
http://rmq1:15692/metrics
UP
instance="rmq1:15692"
job="rabbitmq-server"
5.785s ago
55.532ms
http://rmq2:15692/metrics
UP
instance="rmq2:15692"
job="rabbitmq-server"
4.732s ago
59.670ms
rabbitmq-server-detailed (3/3 up)
show less
Endpoint
State
Labels
Last Scrape
Scrape Duration
Error
http://rmq0:15692/metrics/detailed
family="queue_coarse_metrics"
family="queue_consumer_count"
family="channel_queue_metrics"
family="channel_queue_exchange_metrics"
family="channel_metrics"
family="connection_metrics"
family="connection_coarse_metrics"
family="channel_exchange_metrics"
UP
instance="rmq0:15692"
job="rabbitmq-server-detailed"
8.424s ago
2.349ms
http://rmq1:15692/metrics/detailed
family="queue_coarse_metrics"
family="queue_consumer_count"
family="channel_queue_metrics"
family="channel_queue_exchange_metrics"
family="channel_metrics"
family="connection_metrics"
family="connection_coarse_metrics"
family="channel_exchange_metrics"
UP
instance="rmq1:15692"
job="rabbitmq-server-detailed"
1.46s ago
2.412ms
http://rmq2:15692/metrics/detailed
family="queue_coarse_metrics"
family="queue_consumer_count"
family="channel_queue_metrics"
family="channel_queue_exchange_metrics"
family="channel_metrics"
family="connection_metrics"
family="connection_coarse_metrics"
family="channel_exchange_metrics"
UP
instance="rmq2:15692"
job="rabbitmq-server-detailed"
10.538s ago
2.228ms

Figure 7

Grafana Configuration

Once Grafana is installed and running you can manage it via the GUI on port 3000. There are many different dashboards that can be installed from the Official Grafana site, they can be loaded and started from the GUI. These dashboards mainly show aggregated statistics, the official RabbitMQ dashboard is very good at monitoring the status of RabbitMQ itself. It uses the /metrics endpoint which provides aggregated statistics for the queues which makes it difficult for our purpose to see important alerts from OpenVMS, see **Figure 8**.

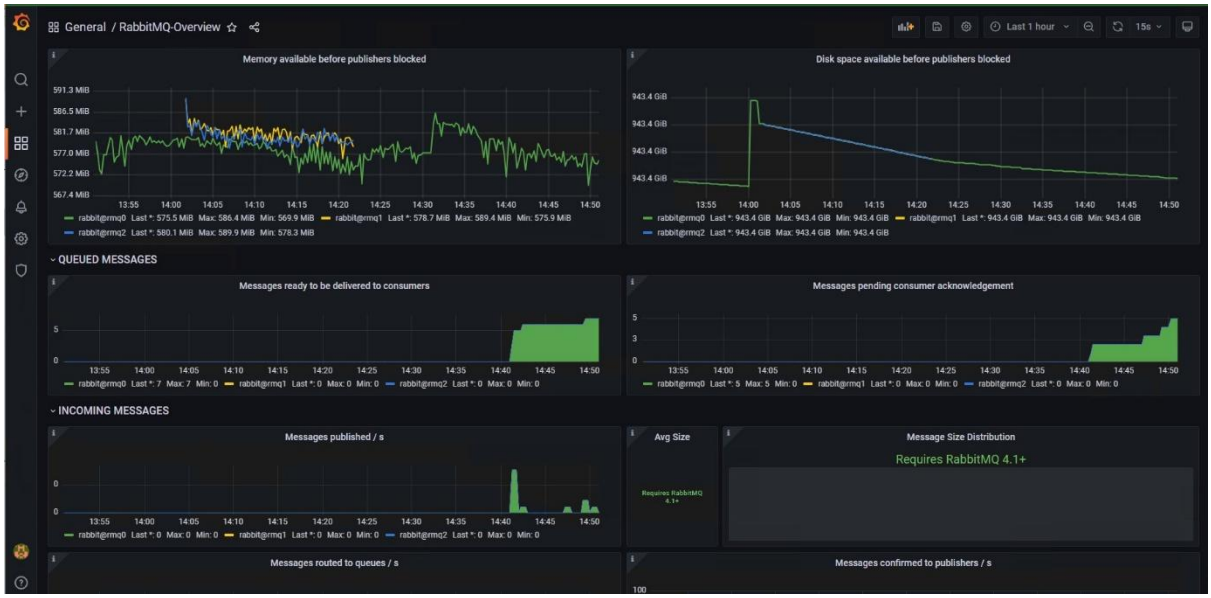

Figure 8

Fortunately, there are a number of other dashboards on the Grafana website that will provide the data we require. I used the Seventh State RabbitMQ Support dashboard which scrapes data from the /metrics/detailed endpoint and therefore gives the detail we require, see **Figure 9**.

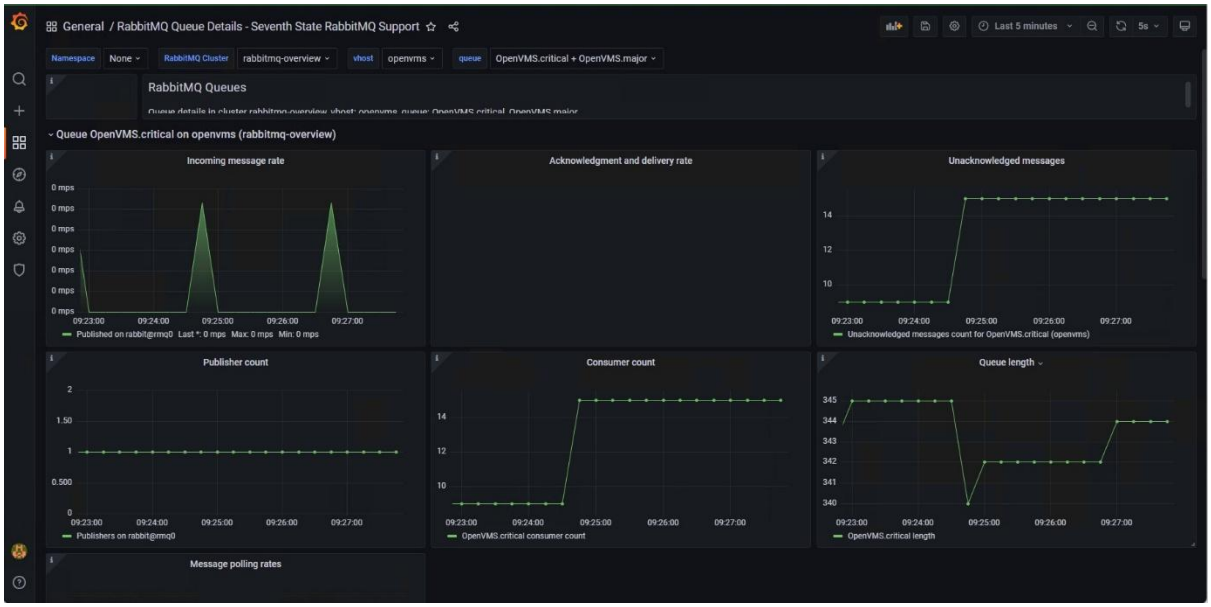

Figure 9

The dashboard needs the Prometheus Configuration to include the RabbitMQ-server-detailed section to enable certain per-object items giving more detailed data. In the dashboard one or more relevant queue(s) can be selected so that only the statistics we require are displayed. In figure 9 both the OpenVMS.critical and the OpenVMS.major queues are selected and there are 344 alerts on those two queues. By selecting one of those queues we can focus on how many critical alerts and how many major alerts require attention.

Seventh State have provided a number of Grafana dashboards for RabbitMQ but the one in figure 9 shows that an OpenVMS server somewhere is generating critical and major alerts.

Further Investigation to Identify the Server(s) generating Alerts

The technology stack described above works best in situations where we are monitoring a large server estate and the monitoring environment must be performant. In those situations, aggregated statistics are the preferred option. If we need individual data, per-object data then the monitoring system performance soon becomes degraded resulting in unreliable monitoring . The aggregated data identified that critical and major alerts are being generated on one or more of our OpenVMS servers.

If the OpenVMS estate is small we could modify the configuration to identify the servers in question. In RabbitMQ, if a queue was created for every server that would provide the identity of the server.

For example we have OpenVMS server VMS923, create a RabbitMQ queue named VMS923 with the following Routing key bindings.

From	Routing Key
amq-topic	vmsspi.*.vms923.critical
amq-topic	vmsspi.*.vms923.major

If we see messages on this queue in the Grafana dashboard, the server VMS923 is the obvious one to investigate. This approach is workable for a number of servers but becomes harder to manage when we have say 50 servers or more.

Another approach would be to use an additional monitoring application such has Dynatrace or Splunk or a bespoke application built from a RabbitMQ framework to investigate deeper to a per-object level, see **Figure 10**.

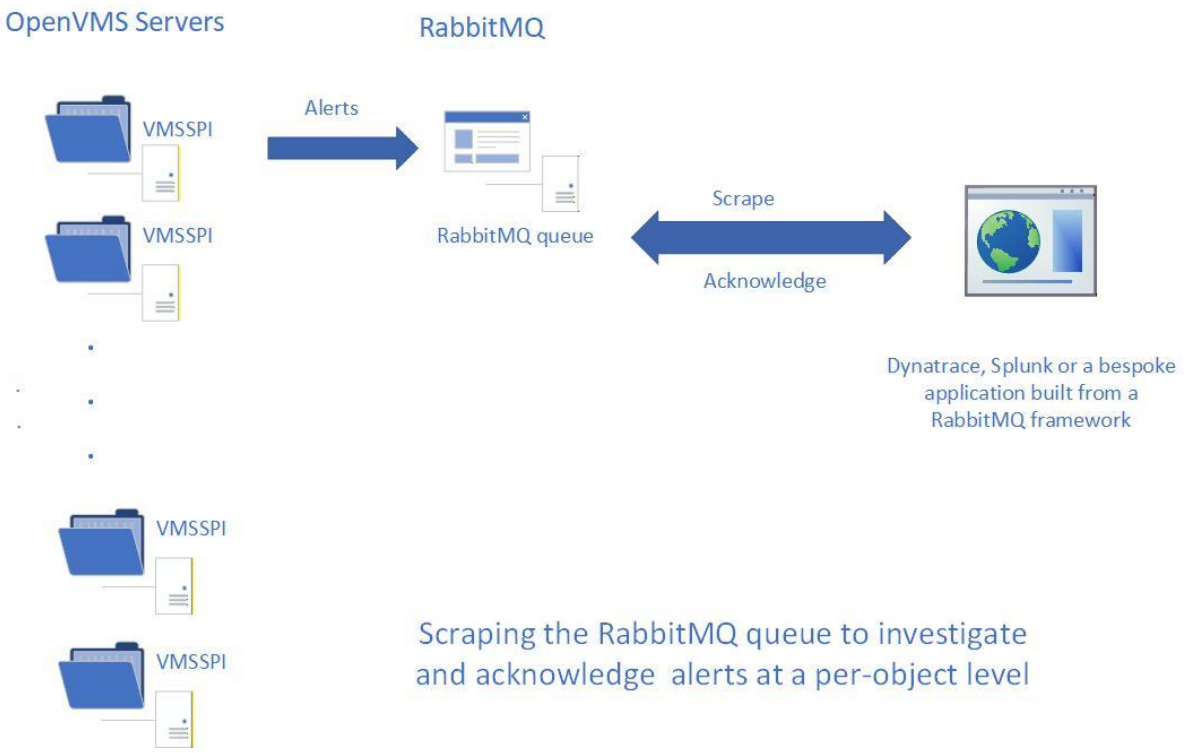

Figure 10

RabbitMQ provides several frameworks for many of the popular development languages. With some development effort you can add a purpose-built application which can pull the alerts from the RabbitMQ queue and allow them to be investigated and then acknowledged once resolved so the alert is removed from the queue, see **Figure 11**.

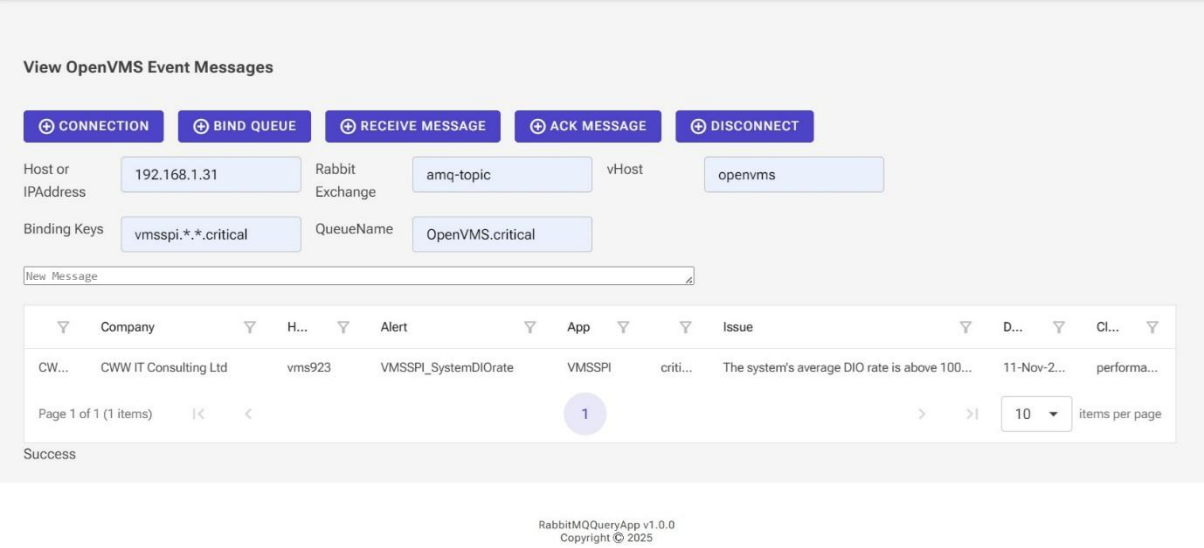

Figure 11