PowerDNS Cloud Control

Monitoring

Dec 19, 2022

Release 2.2.1

Contents

1 Overview

1.1 Cloud Control Monitoring

Cloud Control Monitoring helps provide insight into Cloud Control deployments. The monitoring stack is built using the following components:

- **Grafana** - Visualisation
- **Prometheus** - Metrics gathering & storage
- **kube-state-metrics (KSM)** - Service that exposes Kubernetes metrics to prometheus
- **Prometheus Adapter** - Exposes Prometheus metrics via Kubernetes Metrics API

The stack can be deployed in its entirety, or partially depending on your existing monitoring infrastructure. When the full stack is deployed, the result will be a namespace containing the following components:
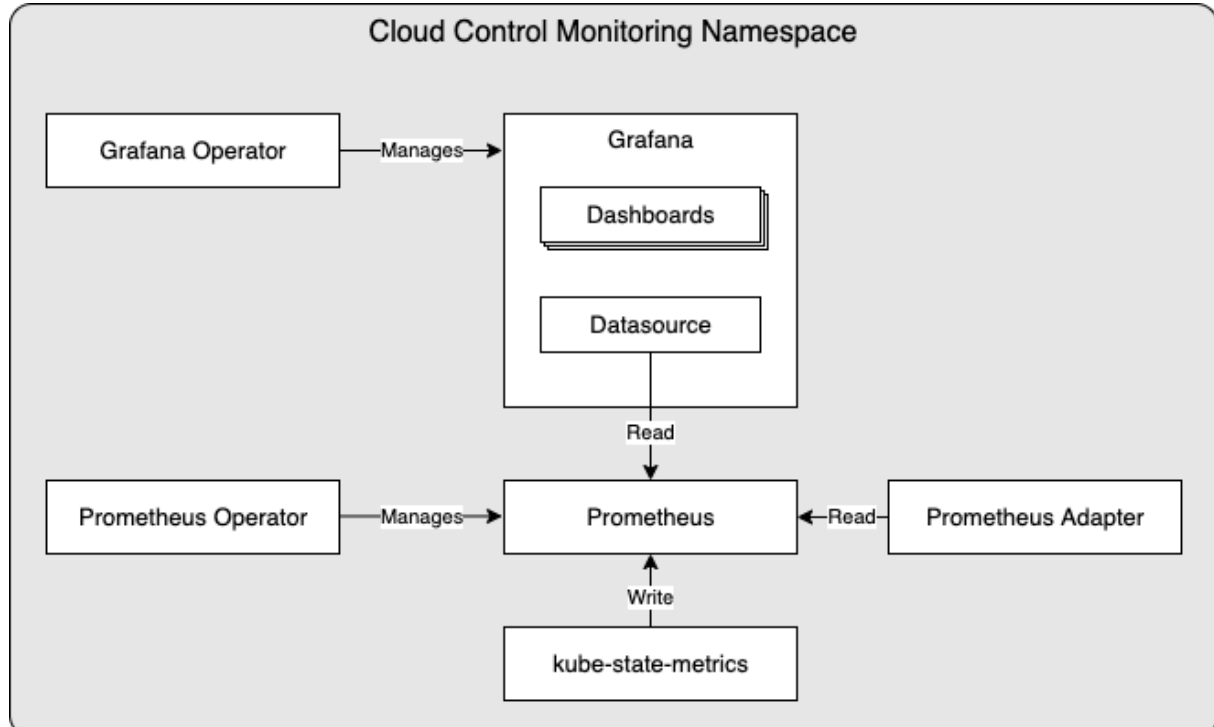
2 Helm Charts

2.1 Charts

The Helm charts which are available to deploy this stack are as follows:

- **monitoring-operators** - Deploy Grafana & Prometheus operators + accompanying CRDs
- **monitoring** - Deploy Prometheus & Grafana (including dashboards + datasource) using above mentioned operators. Also deploys KSM & Prometheus Adapter.

Since the monitoring chart depends on the availability of the operators, the monitoring-operators chart needs to be installed prior to the monitoring chart if you intend to deploy Grafana and/or Prometheus.

2.2 Usage

2.2.1 Install Tools

You will need the following software on the machine from which you want to deploy Cloud Control Monitoring:

- Kubectl (Configured for your target Kubernetes cluster)
- Helm v3 (https://helm.sh/docs/intro/install/)

2.2.2 Download Helm Charts

Cloud Control Monitoring Helm Charts are available on the Open-Xchange registry, located at: registry.open-xchange.com.

There are several methods for obtaining Helm Charts using Helm's CLI, in this chapter we are using a method that copies the chart locally to your filesystem prior to using it. Any Helm-supported method will work, but you will need to adjust the commands in this guide accordingly if you wish to utilise a different method.

First step will be to make Helm aware of the Cloud Control repository (replace username & password with your OX registry credentials):

```
helm repo add cloudcontrol https://registry.open-xchange.com/chartrepo/cloudcontrol \
--username=REGISTRY_USERNAME_HERE --password=REGISTRY_PASSWORD_HERE
```

Once the repository has been added you can pull the Cloud Control Monitoring Helm Charts. To pull the monitoring Helm Charts and export them to your current working directory use the following commands:

```
# The release we're working with
CCTAG=2.2.1

# Ensure repo data is up-to-date
helm repo update

# Pull the Helm Chart & unpack
helm pull cloudcontrol/monitoring-operators -d . --version=$CCTAG --untar
helm pull cloudcontrol/monitoring -d . --version=$CCTAG --untar
```

2.2.3 Deploying Cloud Control Monitoring

To deploy the monitoring stack without any customization you can use the following steps:

```
# The namespace
CC_MON_NAMESPACE=ccmon
HELM_RELEASE=ccmon

# Deploy the monitoring operators & CRDs
helm install $HELM_RELEASE-operators ./monitoring-operators --namespace $CC_MON_NAMESPACE \
--create-namespace
```

3

```
# Deploy the monitoring stack
helm install $HELM_RELEASE ./monitoring --namespace $CC_MON_NAMESPACE
```

Note: you can remove `--create-namespace` if you have an existing namespace to deploy into

2.2.4 Accessing Grafana

You can use kubectl's `port-forwarding` to quickly access the Grafana service:

```
# The namespace
CC_MON_NAMESPACE=ccmon

kubectl --namespace=$CC_MON_NAMESPACE port-forward svc/grafana 3000:grafana
```

You can now visit Grafana at: http://localhost:3000/

When prompted for a username/password, you can login using the username configured in 'grafana.admin.username' and based on the 'grafana.admin.password' setting a static or dynamically generated password. To customize this behaviour, you can modify the following block in the helm values:

```
grafana:
  # UI Access
  admin:
    # Grafana admin credentials
    username: admin
    # password: some_password
```

If no password is specified (as in the example above), a random password will be generated and stored in Secret: grafana-credentials

For a more permanent method of accessing Grafana, refer to the *Configuration* chapter to configure an Ingress object.

2.2.5 Accessing Prometheus

You can use kubectl's `port-forwarding` to quickly access the Prometheus service:

```
# The namespace
CC_MON_NAMESPACE=ccmon

kubectl --namespace=$CC_MON_NAMESPACE port-forward svc/prometheus 9090:web
```

You can now visit Grafana at: http://localhost:9090/

For a more permanent method of accessing Prometheus, refer to the *Configuration* chapter to configure an Ingress object.

2.3 Configuration

2.3.1 Enable/Disable components

The helm charts have 'enabled' flags available for all major components. The following sections describe how you can use these to enable/disable components.

Monitoring Operators

This chart installs the following operators & accompanying CRDs:

- Grafana Operator

- Prometheus Operator

By default all components are installed, this can be controlled using the 'enabled' settings in the Helm values:

```
# Grafana Operator
grafana-operator:
    # Enable the Grafana Operator deployment
    enabled: true
```

```
# Prometheus Operator
prometheus-operator:
    # Enable the Prometheus Operator deployment
    enabled: true
```

Setting these to 'false' will stop the operator (& CRDs) from being deployed.

Monitoring

This chart installs the following components:

- Grafana

- Prometheus

- Prometheus Adapter

- kube-state-metrics (KSM)

By default all components are installed, this can be controlled using the 'enabled' settings in the Helm values:

```
# Grafana
grafana:
    # Enable the Grafana deployment
    enabled: true
```

```
# Prometheus
prometheus:
    # Enable the Prometheus deployment
    enabled: true
```

```
# Prometheus Adapter
prometheus-adapter:
    # Enable the Prometheus Adapter deployment
    enabled: true
```

```
# kube-state-metrics
kube-state-metrics:
    # Enable the Kube State Metrics deployment
    enabled: true
```

Setting these to 'false' will stop the component from being deployed.

2.3.2 Ingresses & Loadbalancers

The following components have configuration options to add an Ingress and/or Loadbalancer to expose them outside of the cluster:

- Grafana

- Prometheus

To configure an Ingress and/or Loadbalancer, you can override the 'ingress' and 'service' configuration under the 'prometheus' & 'grafana' sections in the helm values. The following sections describe how you can use these to configure them.

Ingress

By default, the ingress is disabled:

```
# Ingress configuration
ingress:
  # Enable the Ingress
  enabled: false
```

To create an ingress which only serves HTTP, set 'enabled' to true and add the hosts on which you want the ingress to listen. Depending on your Ingress Controller, add the needed annotations.

Example which exposes Prometheus on 'http://prometheus.example.com' using the NGINX Ingress Controller:

```
ingress:
  enabled: true
  annotations:
    kubernetes.io/ingress.class: "nginx"
  hosts:
    - prometheus.example.com
```

To create an ingress which serves HTTPS (and has an HTTP->HTTPS redirect), provide additional annotations and a 'tls' configuration block.

Example which exposes Prometheus on 'https://prometheus.example.com' via the NGINX Ingress Controller and a pre-existing certificate in a secret named 'prometheus-cert':

```
ingress:
  enabled: true
  annotations:
    kubernetes.io/ingress.class: "nginx"
  hosts:
    - prometheus.example.com
  tls:
    - secretName: prometheus-cert
      hosts:
      - prometheus.example.com
```

Example which exposes Prometheus on 'https://prometheus.example.com' via the NGINX Ingress Controller and an on-demand certificate provisioned by 'cert-manager' in a secret named 'prometheus-cert' (note the additional annotation):

```
ingress:
  enabled: true
  annotations:
    kubernetes.io/ingress.class: "nginx"
    cert-manager.io/cluster-issuer: ca-issuer
  hosts:
    - prometheus.example.com
  tls:
    - secretName: prometheus-cert
      hosts:
      - prometheus.example.com
```

Loadbalancer

By default, services of type 'ClusterIP' are created. To expose the service using a loadbalancer, set the type to 'LoadBalancer' and add the necessary additional configuration, based on your LoadBalancer provider. Example Grafana service configuration in a cluster with MetalLB:

```
# Grafana Service
service:
  type: LoadBalancer
  annotations:
    metallb.universe.tf/address-pool: name_of_pool
  loadBalancerIP: 12.34.56.78 # Omit this to have a random IP assigned from the pool
  ports:
    grafana-http:
      port: 3000
```

For a NodePort service, set the type to 'NodePort' and if desirable, specify the nodePort:

```
# Grafana Service
service:
  type: NodePort
  ports:
    grafana-http:
      port: 3000
      nodePort: 30003
```

2.3.3 Private Registries

All images referenced by the monitoring & monitoring-operators charts are available on public registries. If you intend to run the monitoring stack on a kubernetes cluster which makes use of a local/private registry, you can use one or more of the following settings in your helm values to configure that registry:

```
# Monitoring - global overrides
global:
  # Override image-related settings for this chart and all subcharts
  image:
    # Override registry for all images
    registry: "myregistry.local:8085"

    # Override repository for all images
    repository: "myrepository"

    # Override pullPolicy for all images
    pullPolicy: "IfNotPresent"

  # Add imagePullSecrets for this chart and all subcharts
  imagePullSecrets:
    myIPSSecret:
      registry: registry.example.com:5000
      username: regUsername
      password: regPassword
      email: admin@registry.example.com

  # Reference existing Image Pull Secrets for this chart and all subcharts
  imagePullSecretsList:
    - "my-existing-IPS-secret"
```

Each setting explained:

global.image.registry

All images will be attempted to be pulled from this registry (format: host:port)

global.image.repository

All images will be attempted to be pulled from this repository, on above configured registry

global.image.pullPolicy

This pull policy will be specified for each image

global.imagePullSecrets

For each entry a Secret will be created and assigned to each Pod

global.imagePullSecretsList

Each pre-existing Secret referenced in this list (by name) will be assigned to each Pod

If you have a need to override the above settings for specific images, you can find the corresponding 'image:' configuration blocks in the values file.

3 Prometheus Adapter

3.1 Adapter

Included in the monitoring stack is the Prometheus Adapter (`https://github.com/kubernetes-sigs/prometheus-adapter`). This component is used to expose metrics via the Kubernetes API, based on data stored in Prometheus. Exposing these metrics via the Kubernetes API enables the use of Horizontal Pod Autoscalers in CloudControl deployments.

3.2 Predefined Metrics

By default a few basic metrics are exposed via the adapter, which are configured in the helm values under the section 'prometheus-adapter.rules.custom'. The format that these metrics must follow is defined here: https://github.com/kubernetes-sigs/prometheus-adapter/blob/master/docs/config.md

One of the default values available is the amount of queries received by dnsdist, defined as follows:

```
- seriesQuery: 'dnsdist_queries'
  resources:
    overrides:
      namespace: {resource: "namespace"}
      pod: {resource: "pod"}
  name:
    matches: "^(.*)"
    as: "${1}_per_second"
  metricsQuery: 'sum(rate(<<.Series>>{<<.LabelMatchers>>}[1m])) by (<<.GroupBy>>)'
```

Based on the 'counter' type metric 'dnsdist_queries' in Prometheus, the rate (ie: the increase of the metric over a period of time) at which it increases per second is calculated, giving the amount of queries handled per second over the interval. This makes a metric available via the Kubernetes API named "dnsdist_queries_per_second"

Another example focuses on a 'guage' type metric, exposing the average latency reported for a dnsdist instance:

```
- seriesQuery: '{__name__=~"^dnsdist_latency_avg.*$"}'
  resources:
    overrides:
      namespace: {resource: "namespace"}
      pod: {resource: "pod"}
  name:
    matches: ""
    as: ""
  metricsQuery: 'avg(<<.Series>>{<<.LabelMatchers>>}) by (<<.GroupBy>>)'
```

Since there are several guages that match the query expression, this leads to the following metrics being available via the Kubernetes API:

- dnsdist_latency_avg100

- dnsdist_latency_avg1000

- dnsdist_latency_avg10000

- dnsdist_latency_avg1000000

To see all the metrics exposed via the Prometheus Adapter, you can use kubectl:

```
# Note: If you have other sources for metrics available on your cluster this might be a␣
↪long list
kubectl get --raw /apis/custom.metrics.k8s.io/v1beta1/ | jq
```

3.3 Horizontal Pod Autoscaler

These metrics are made available to allow for using Kubernetes' Horizontal Pod Autoscaler to be used to automatically scale up & down deployments. Since the Horizontal Pod Autoscaling functionality is part of the actual deployment of dnsdist & Recursor, you can find more about this part of the configuration in the CloudControl deployment documentation.
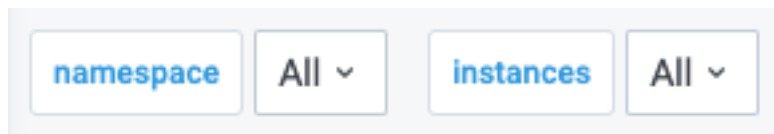
4 Grafana Dashboards

4.1 Dashboards

When the 'monitoring' chart is used to deploy the stack, several monitoring dashboards will be provisioned automatically. Currently, this includes the following dashboards:

- **PowerDNS dnsdist** Detailed insight into running dnsdist instances

- **PowerDNS Recursor Overview** High-level overview of running Recursor instances

- **PowerDNS Recursor details** Detailed insight into running Recursor instances

All dashboards have selectors available, which you can use to view details regarding specific instances or instances within a namespace. These selectors are located top-left of each dashboard and look as follows:

4.2 Download

If you opt to utilize an existing Grafana installation, you can download the dashboards from the the Open-Xchange registry, located at: registry.open-xchange.com

The dashboards are stored inside an OCI artifact, so you will have to use a compatible client to obtain them. Recommended client to use for this is: ORAS (https://oras.land/)

With a CLI such as oras available, you can download the dashboards artifact via:

```
oras pull registry.open-xchange.com/cloudcontrol/monitoring-dashboards:2.2.1
```

The result will be a tar archive which contains all the dashboards in JSON format.

5 Component Reference

5.1 Operators

The following operators are utilized in the monitoring stack:

Prometheus Operator: https://github.com/prometheus-operator/prometheus-operator

Grafana Operator: https://github.com/grafana-operator/grafana-operator

5.2 Components

The following components are utilized in the monitoring stack:

Prometheus: https://prometheus.io/

Grafana: https://grafana.com/oss/grafana/

Prometheus Adapter: https://github.com/kubernetes-sigs/prometheus-adapter

kube-state-metrics: https://github.com/kubernetes/kube-state-metrics