**Data Service Server Monitoring**

**with Prometheus**

**User′s Manual**

**V1.0.2**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date | Version | Author | Reviewer | Description |
| 2019-03-05 | 1.0.0 | Evelyn Tang  Evelyn.Tang@advantech.com.tw |  | First version released |
| 2019-03-07 | 1.0.1 | Evelyn Tang  Evelyn.Tang@advantech.com.tw |  | Paragraph adjustment  Add architecture image |
| 2019-03-08 | 1.0.2 | Evelyn Tang  Evelyn.Tang@advantech.com.tw |  | Change name to “Data Service Server” |
| 2019-05-02 | 1.0.3 | Evelyn Tang  Evelyn.Tang@advantech.com.tw |  | Add MongoDB exporter and PV dashboards. |
| 2019-06-18 | 1.0.4 | Evelyn Tang  Evelyn.Tang@advantech.com.tw |  | Add AlertManager |
|  |  |  |  |  |

**Revision History**

**Table of Contents**

[1 General Introduction 5](#_Toc11761991)

[2 Prometheus Overview 7](#_Toc11761992)

[3 Data visualization 9](#_Toc11761993)

[4 Alert Manager 23](#_Toc11761994)

[FAQ 32](#_Toc11761995)

[Reference 35](#_Toc11761996)

**Table of Figures**

[Figure 1 Architecture of Data Service Server 5](#_Toc11761956)

[Figure 2 Architecture of Prometheus monitoring system [4] 7](#_Toc11761957)

[Figure 3 Grafana login page 9](#_Toc11761958)

[Figure 4 Dashboard overview 11](#_Toc11761959)

[Figure 5 Kubernetes Cluster Overview 12](#_Toc11761960)

[Figure 6 Node Exporter Overview 13](#_Toc11761961)

[Figure 7 Kubernetes Pods 14](#_Toc11761962)

[Figure 8 Kubernetes Persistent Volumes 15](#_Toc11761963)

[Figure 9 RabbitMQ Overview 17](#_Toc11761964)

[Figure 10 PostgresSQL Overview 18](#_Toc11761965)

[Figure 11 MongoDB Overview 19](#_Toc11761966)

[Figure 12 Prometheus Web UI 20](#_Toc11761967)

[Figure 13 Simple query result 21](#_Toc11761968)

[Figure 14 Advanced query result 22](#_Toc11761969)

[Figure 15 Alert rules. 24](#_Toc11761970)

[Figure 16 Alertmanager configmap 26](#_Toc11761971)

[Figure 17 Alertmanager WebUI 27](#_Toc11761972)

[Figure 18 Alerts page in Alertmanager Web UI 29](#_Toc11761973)

[Figure 19 Alerting email 29](#_Toc11761974)

1. **General Introduction**
   1. **Data Service Server Architecture**



Figure 1 Architecture of Data Service Server

Data Service Server runs Kubernetes for automating deployment, scaling and management of containerized applications. [1] As illustrated in Figure 1, collecting data from an environment composed of so many moving parts is complex and Prometheus is the best monitoring and alerting tool for Kubernetes and Docker.

Prometheus was built specifically to monitor applications and microservices running in containers at scale and is native to containerized environments. Originally developed at [Soundcloud](https://developers.soundcloud.com/blog/prometheus-monitoring-at-soundcloud), pioneer in the adoption of cloud technology. [2] In 2016, Prometheus project became the second hosted project of the Cloud Native Computing Foundation (CNCF) after Kubernetes.

* 1. **Prometheus and Kubernetes**

****

Prometheus is a pull-based monitoring system, which means that central Prometheus servers discover and pull metrics from your services. The discovery and pull system fits well with a dynamic, cloud native environment such as Kubernetes, where Prometheus integrates well with Kubernetes to discover and enumerate the services you have running.  As you scale up a service, Prometheus automatically starts pulling metrics from the extra replicas.  Similarly as nodes fail and pods are restarted, Prometheus automatically notices and scrapes them. [3] The architecture of Prometheus will be briefly introduced in next section.

1. **Prometheus Overview**

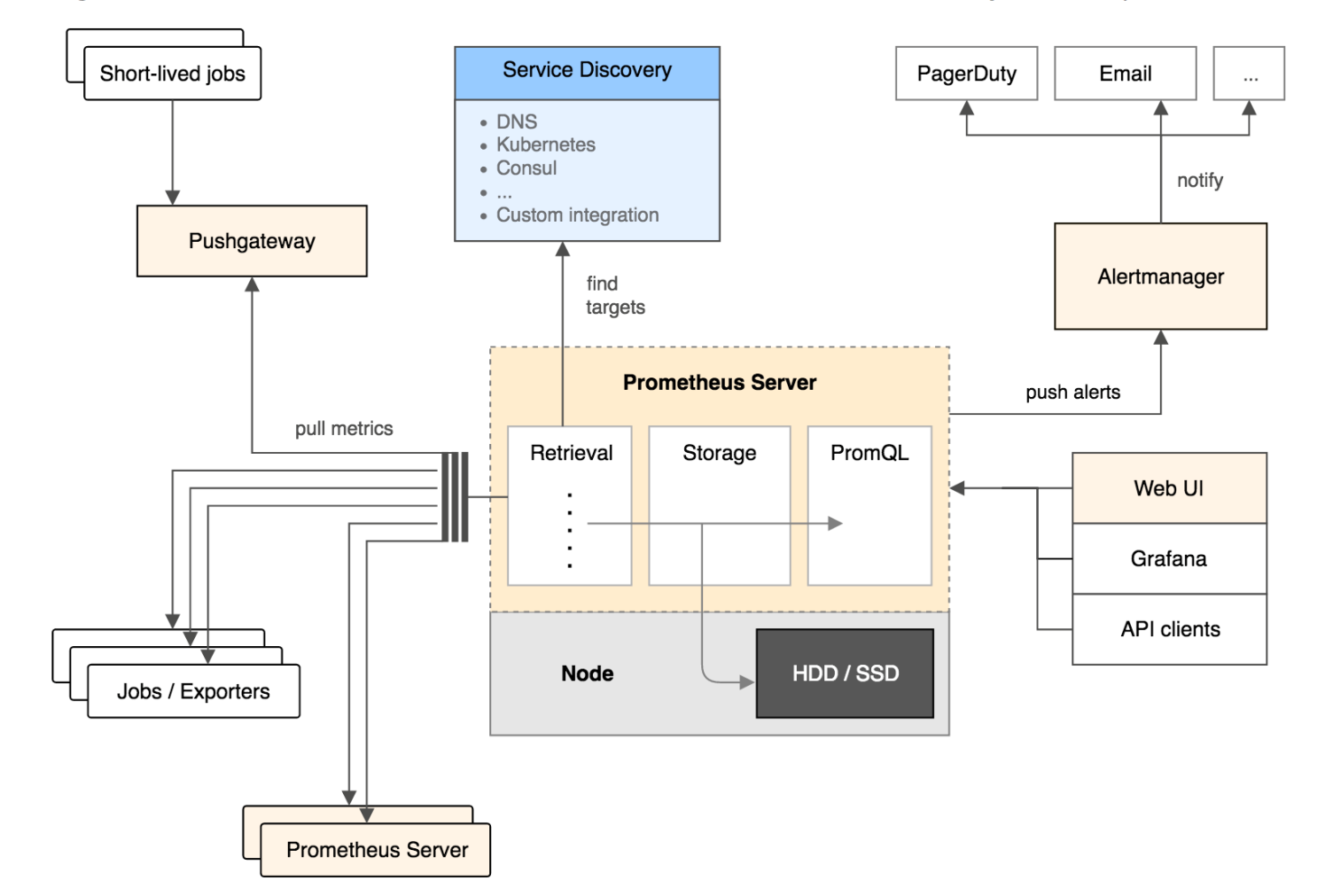
****

Figure 2 Architecture of Prometheus monitoring system [4]

As shown in Figure 2, Prometheus consists of several components working together. Prometheus server pulls in metrics at regular intervals through HTTP endpoints which provide information about hosts and various applications. These endpoints may be provided by the monitored application itself, or through an “exporter”. The endpoint URL is usually /metrics. [5]

Since short-lived jobs not exist long enough to be scraped, they can instead push their metrics to a Pushgateway. Pushgateway then exposes these metrics to Prometheus server. [6] Last but not least, Prometheus Server also exposes its own metrics and monitors its own metrics. [7]

After Prometheus server scrapes metrics from instrumented jobs, either directly or via an intermediary push gateway for short-lived jobs, it stores time-series data locally. User can use PromQL to query this data or send alerts to the Alertmanager, which will convert them into pages, emails, and other notifications. [8]

Prometheus provides a web interface (Web UI) to run queries. Other applications also can run queries through the HTTP API to retrieve and work with the data; we apply Grafana to visualize the data.

Next section will cover two parts: Grafana dashboards and Prometheus WebUI. It is recommended that users completely read this document before evaluation since you will understand which data visualization method meets your needs.

1. **Data visualization**
   1. **Grafana Dashboards**
      1. **Introduction**

Grafana is the leading graph and dashboard builder for visualizing time series infrastructure and application metrics. Therefore, we apply Grafana as data visualization tool.

* + 1. **Imported Dashboards**

First of all, go to your Grafana main view and login admin account.

|  |
| --- |
| We take [http://portal.grafana.example.com](http://portal.grafana.example.com/) for example as Figure 3.  Please go to your Grafana domain server. |

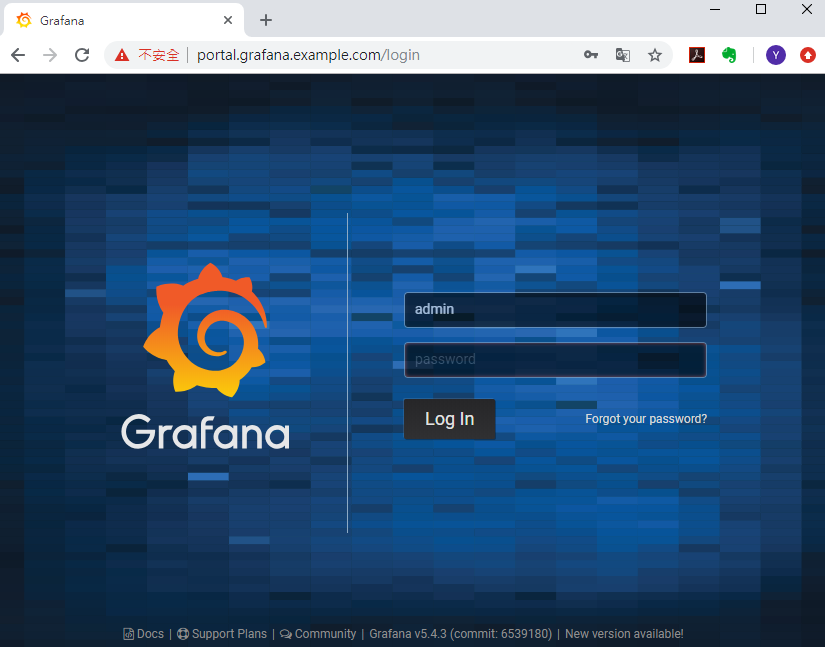
****

Figure 3 Grafana login page

Please access admin account as below.

User Name: **admin**

Password: **@dvant1cH**

You can get username and password of admin account by issuing the following kubectl command.

|  |
| --- |
| * Get Username:   # kubectl get secret --namespace monitoring grafana -o jsonpath="{.data.admin-user}" | base64 --decode ; echo   * Get Password:   # kubectl get secret --namespace monitoring grafana -o jsonpath="{.data.admin-password}" | base64 --decode ; echo |

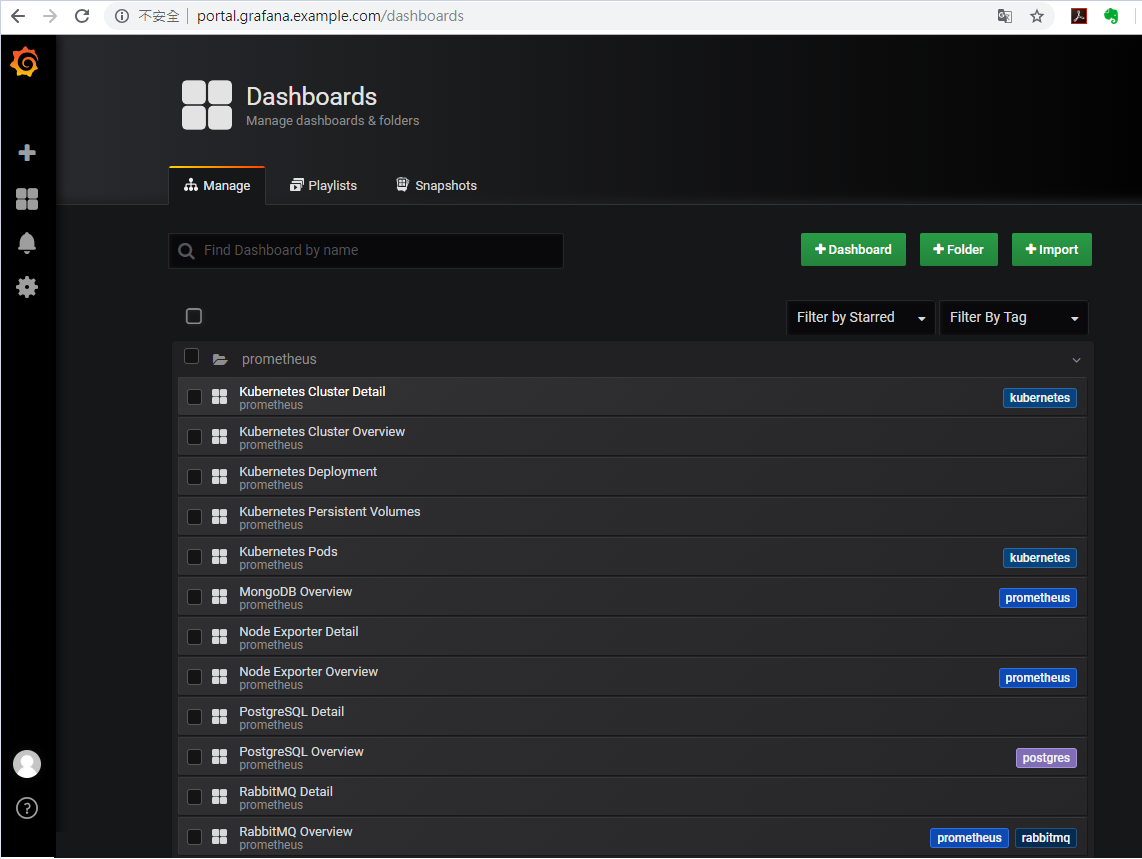


Figure 4 Dashboard overview

Generally speaking, there are several Kubernetes metrics worthy to monitor. As shown in Figure 4, adequate and detailed dashboards have been imported.

* Kubernetes Cluster Overview

For cluster monitoring, the objective is to monitor the health of the entire Kubernetes cluster. As an administrator, we are interested in discovering if all the nodes in the cluster are working properly and at what capacity, how many applications are running on each node, and the resource utilization of the entire cluster. [9]The overview of cluster is shown in Figure 5 below.

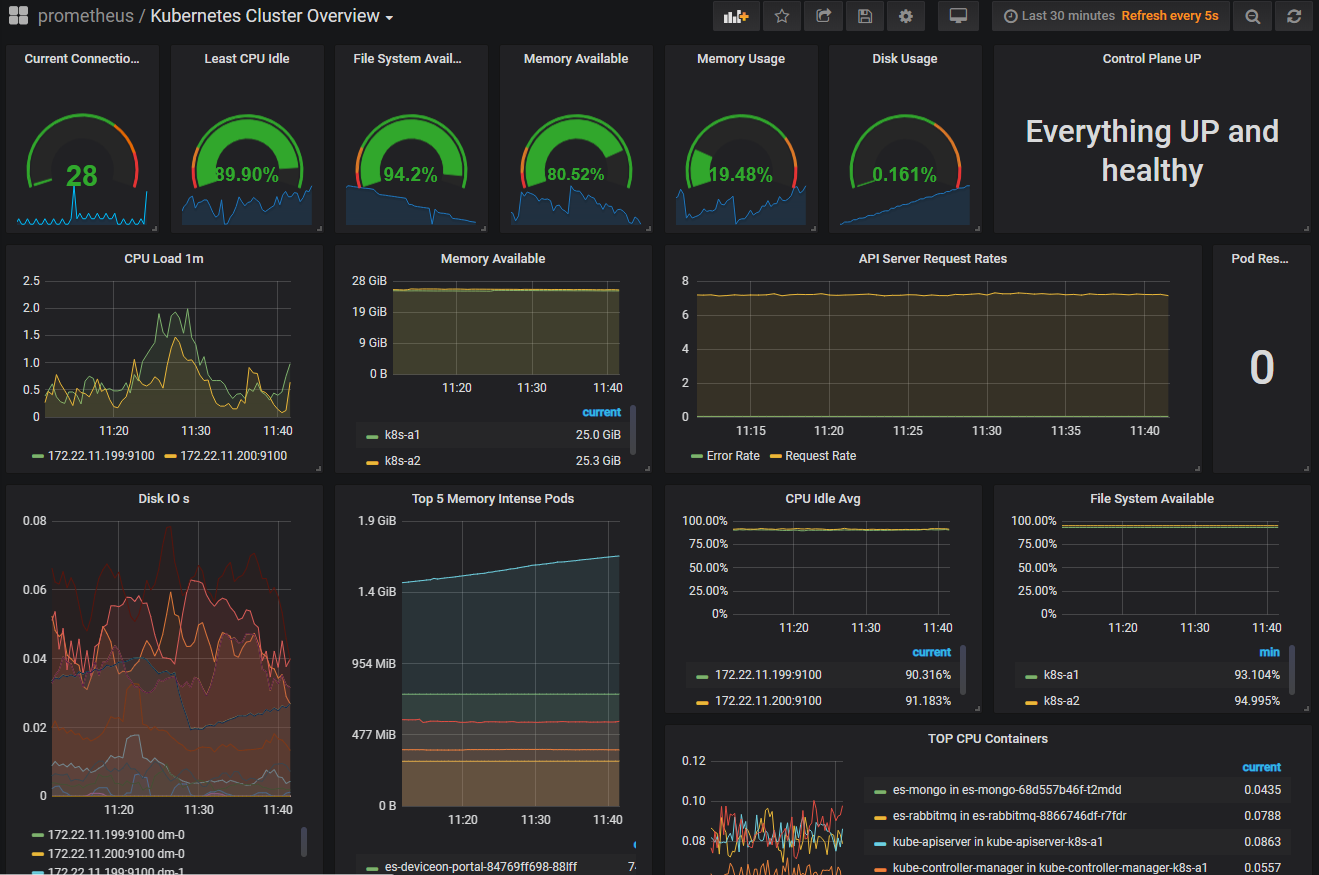


Figure 5 Kubernetes Cluster Overview

There are many measurable metrics worthy of mention, all related to node resource utilization, network bandwidth, disk utilization, CPU, and memory utilization are examples of this. You can navigate to dashboard “**Kubernetes Cluster Detail”** for more cluster information in details.

* Node Exporter Overview

One of the most widely used exporters is the NodeExporter. When NodeExporter runs on a host, it will collect data on I/O, memory, disk and CPU pressure and expose them for scraping. The overview of NodeExporter is as below Figure 6. [10] Also, we have another dashboard “**Node Exporter Detail**”, it shows individual details of memory, network and other system metrics.



Figure 6 Node Exporter Overview

* Kubernetes Pods

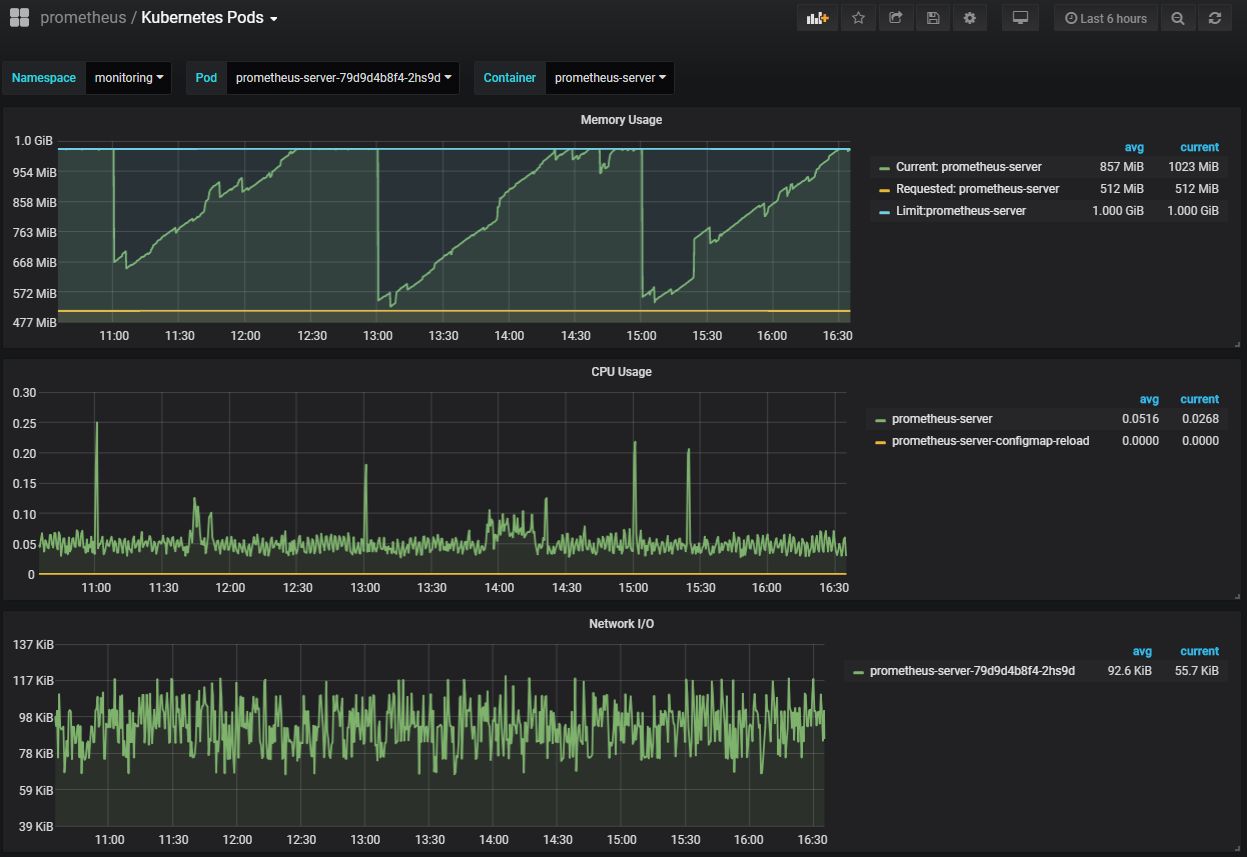


Figure 7 Kubernetes Pods

The act of monitoring a pod can also extend to its containers. You can monitor individual container if there is more than one container in a pod. Container metrics query running container information like CPU, network, and memory usage. As Figure 7 shows above, maximum limitation is set to compare with memory usage.

* Kubernetes Persistent Volumes

A persistent volume (PV) is a piece of storage in the cluster. PVs are volume plugins like Volumes, but have a lifecycle independent of any individual pod that uses the PV.

A persistent volume claim (PVC) is similar to a pod. Pods consume node resources and PVCs consume PV resources. Claims can request specific size and access modes. [11]

Data Service Server implements RBD (Ceph Block Device) as plugins. The disk and inode usage of individual PVCs are shown in Figure 8.

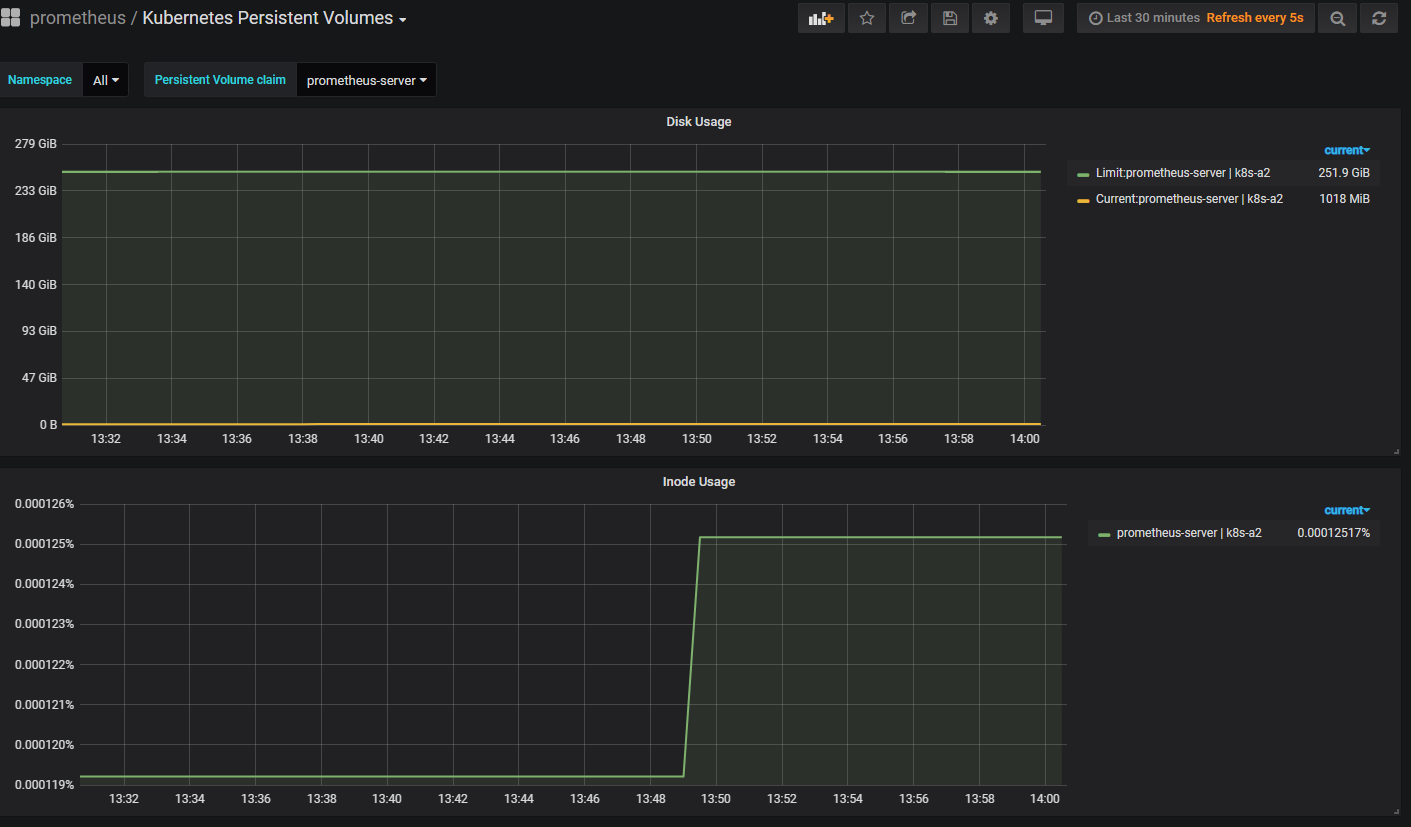


Figure 8 Kubernetes Persistent Volumes

* Other Kubernetes dashboards

“**Kubernetes Deployment**”: Kubernetes deployment is an abstraction layer for the pods. The main purpose of the deployment is to maintain the resources declared in the deployment configuration to be its desired state. [11] User can monitor the status of every deployment in every namespace.

“**Kubernetes StatefulSet**”: Manages the deployment and scaling of a set of [Pods](https://kubernetes.io/docs/reference/glossary/?all=true#term-pod), and provides guarantees about the ordering and uniqueness of these Pods. [12] The status overview of StatefulSet will show in this dashboard.

* Third-Party Exporter

Some other services are not natively integrated, but can be easily adapted using an exporter. An exporter is a service that collects service status and translates to Prometheus metrics ready to be scraped. [13] We have three exporters in Data Service Server, they are RabbitMQ, PostgreSQL and MongoDB respectively.

* + RabbitMQ Overview

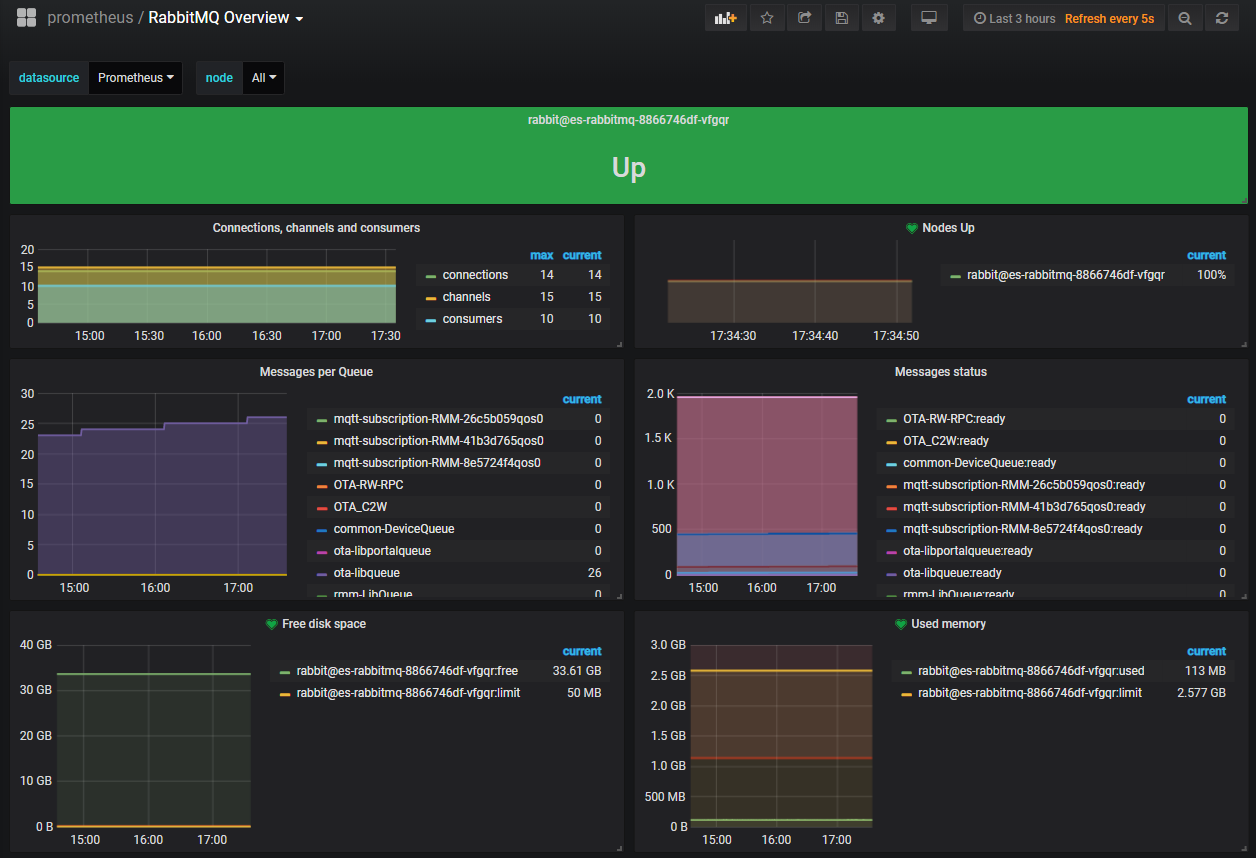


Figure 9 RabbitMQ Overview

RabbitMQ is the most widely deployed open source message broker, it supports multiple messaging protocols, message queuing, delivery acknowledgment, flexible routing to queues, multiple exchange type. [14] RabbitMQ exporter is applied to provide a starting point for monitoring RabbitMQ metrics as shown in Figure 8. Prometheus is configured 60 seconds interval to collect RabbitMQ status.

* + PostgresSQL Overview

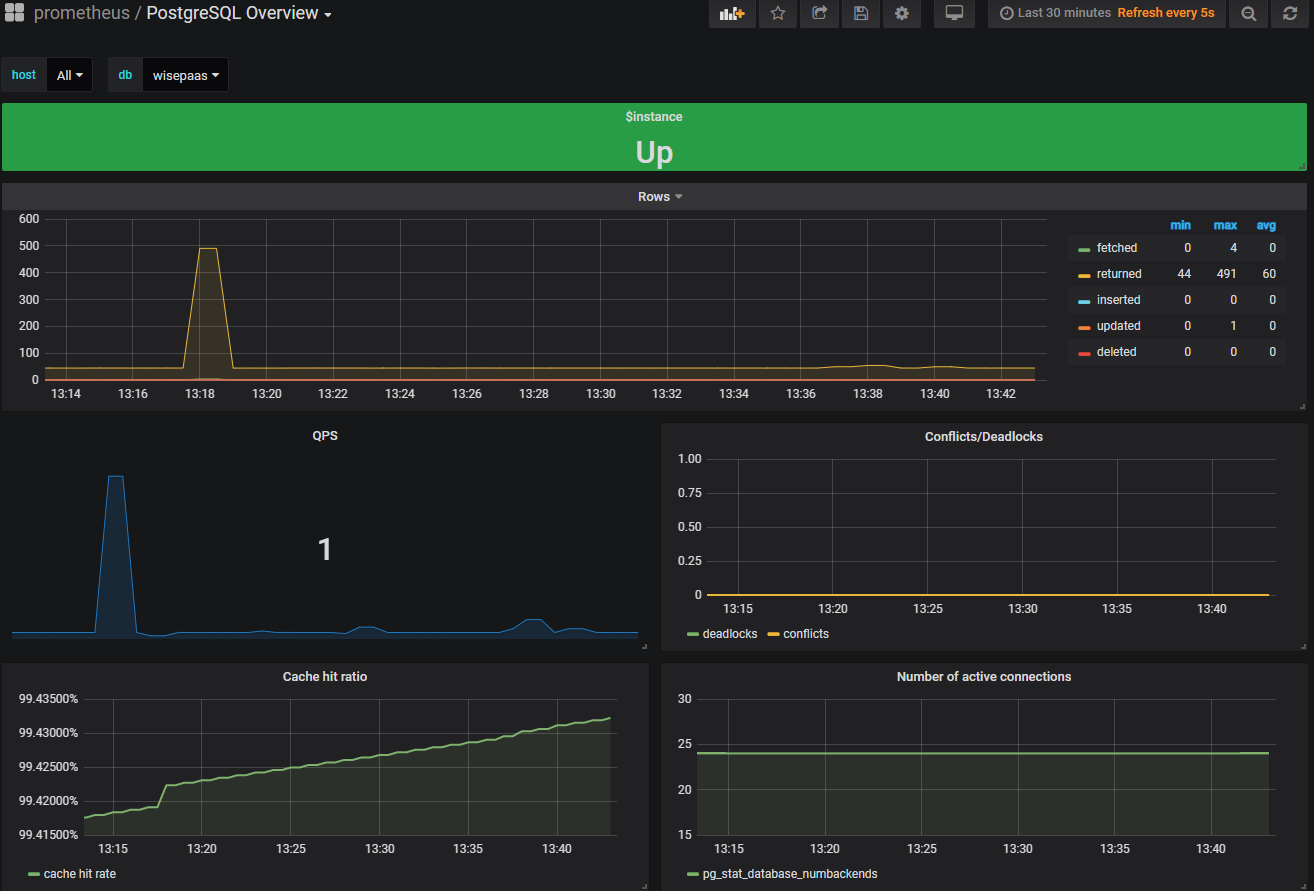


Figure 10 PostgresSQL Overview

PostgreSQL comes with [many features](https://www.postgresql.org/about/featurematrix/) aimed to help developers build applications, administrators to protect data integrity and build fault-tolerant environments, and help you manage your data no matter how big or small the dataset. [15] [16] The user account and connected device info of Data Service Server are stored in PostgresSQL, as shown in Figure 9, you can monitor real-time status of PostgresSQL.

* + MongoDB Overview

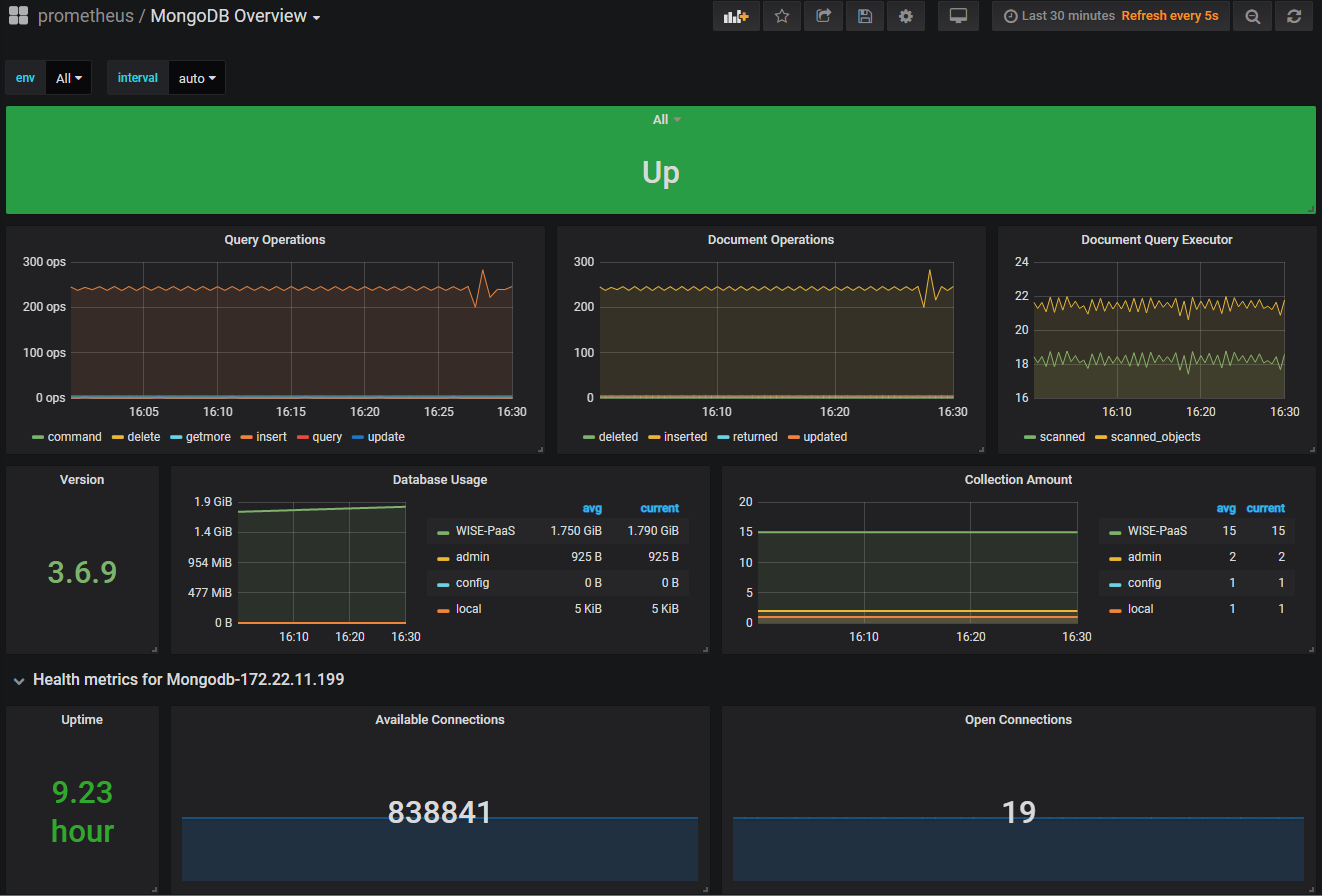


Figure 11 MongoDB Overview

MongoDB is the most popular open-source NoSQL(Not Only SQL) database, it is a document database designed for ease of development and scaling. MongoDB provides high performance data persistence and horizontal scalability. [17]

All the reported data from connected devices in Data Service Server is stored in MongoDB. User can check if data network works normally from MongoDB Overview as shown in Figure 10.

* 1. **Prometheus Web UI**
     1. **Introduction**

Prometheus provides a functional query language called PromQL (Prometheus Query Language) that lets user select and aggregate time series data in real time. The result of an expression can either be shown as a graph, viewed as tabular data in Prometheus's expression browser. It will be a good choice if you need to query specific metrics using PromQL.

* + 1. **PromQL**

At first, go to your Prometheus Web UI with no username and password.

|  |
| --- |
| We take <http://portal.prometheus.example.com> for example as Figure 10.  Please go to your Prometheus WebUI domain server. |

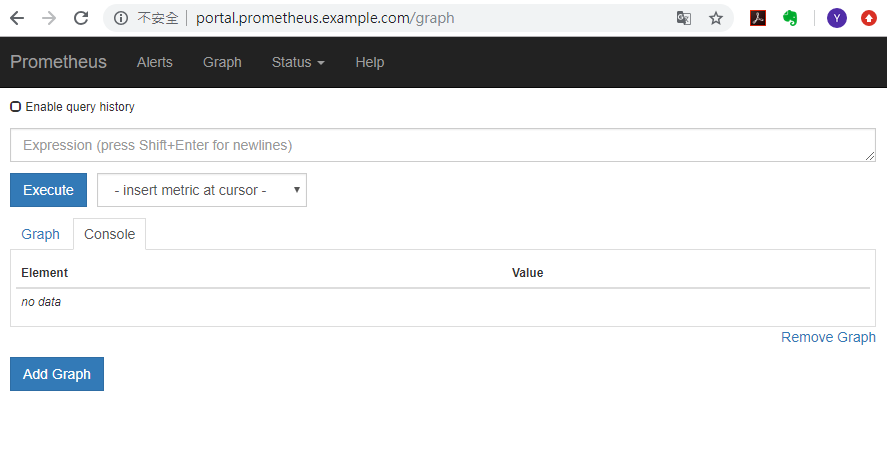


Figure 12 Prometheus Web UI

Take container\_memory\_usage\_bytes for quick example, it selects all time-series that have the container\_memory\_usage\_bytes metric name. Figure 11 below shows the query results with graph mode.

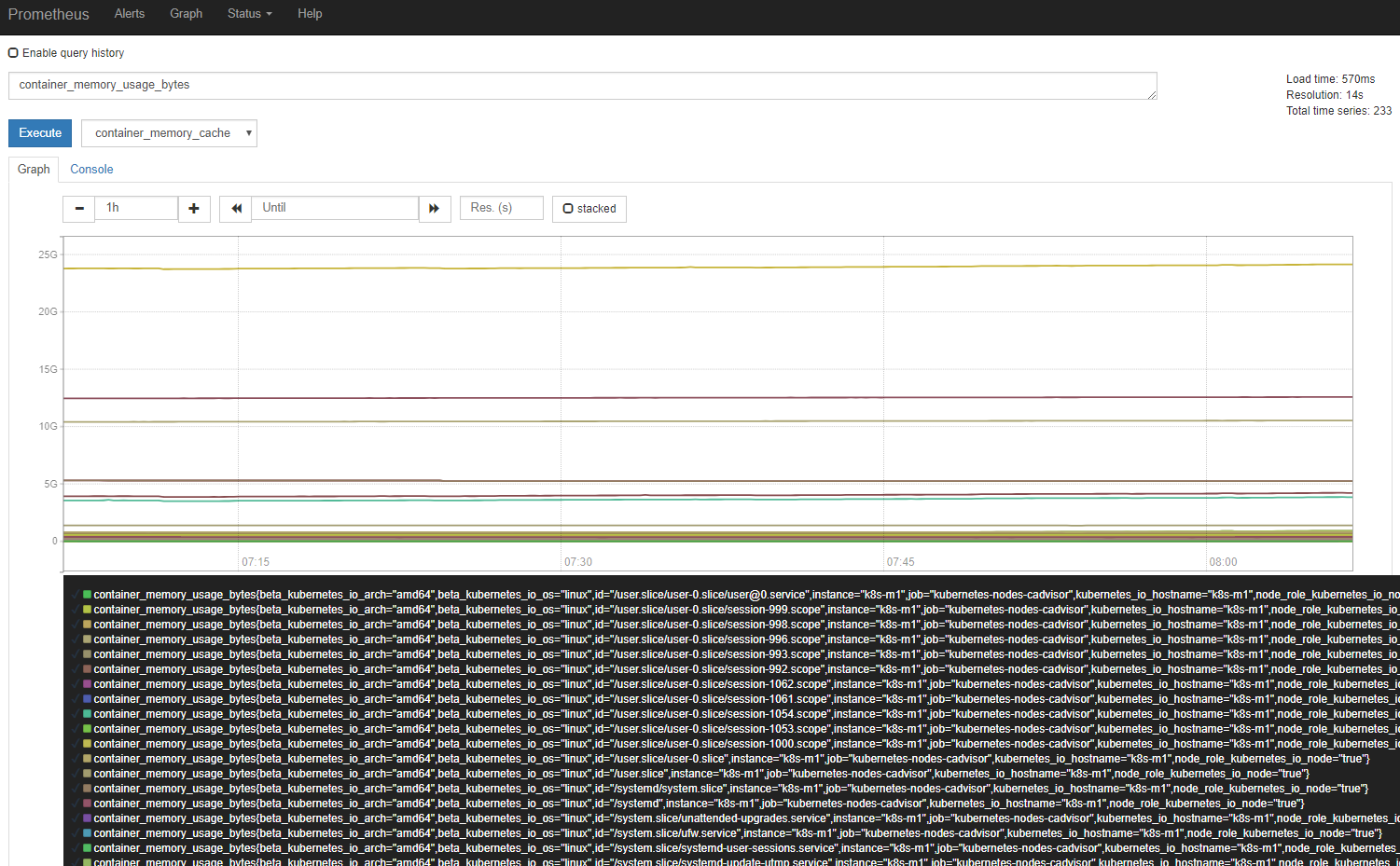


Figure 13 Simple query result

The above results contain all the container\_memory\_usage\_bytes metrics, the data is large. If you want to query more specific data in advanced, append a set of labels to match in curly braces ({}). Take container\_memory\_usage\_bytes for example, append one term in PromQL as below.

|  |
| --- |
| * container\_memory\_usage\_bytes{namespace="default"} |

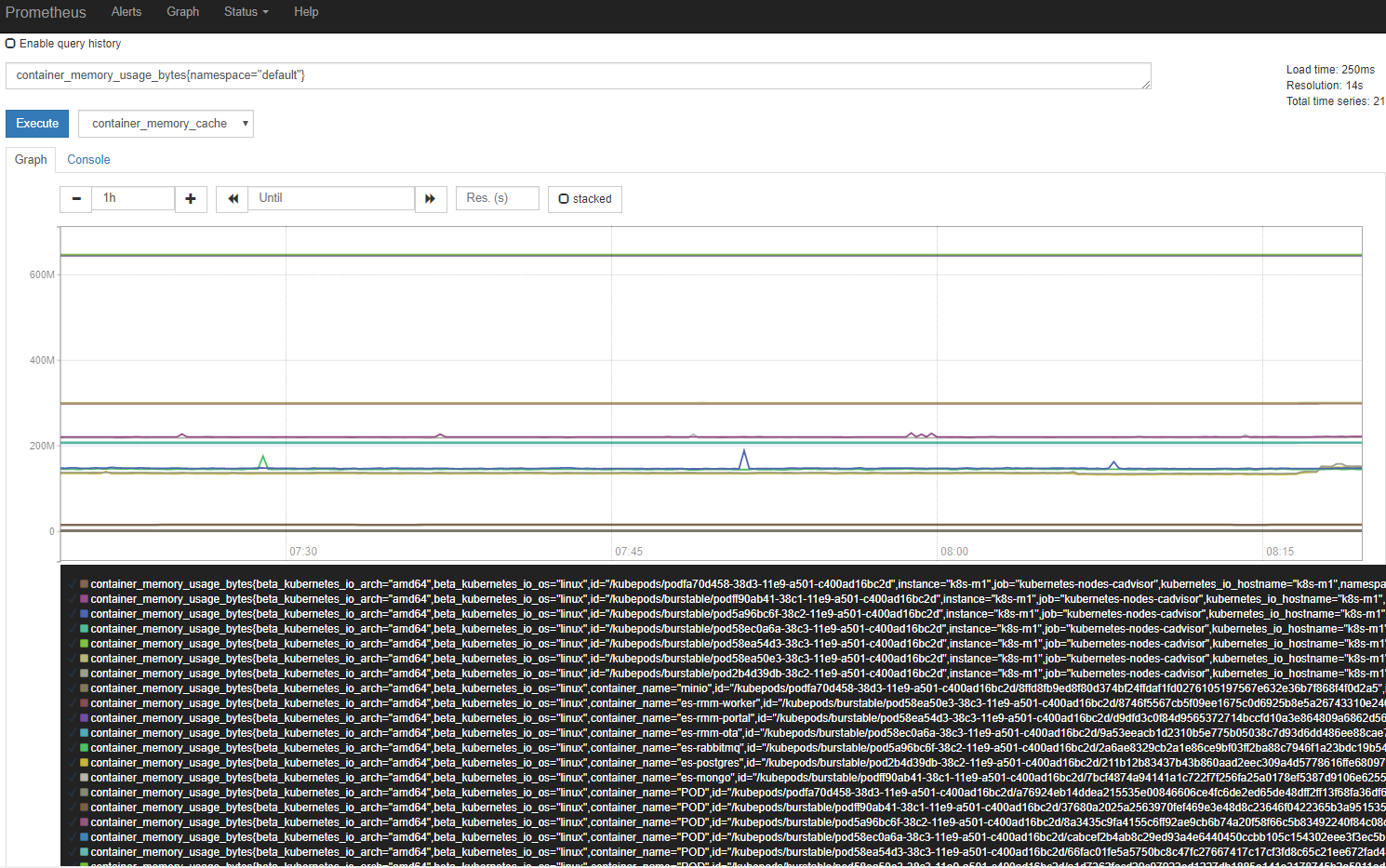


Figure 14 Advanced query result

The result shows in Figure 12 that container\_memory\_usage\_bytes metrics with namespace “default”. Therefore, user can query any specific metric if in need. Please refer to the document for more detail of PromQL. <https://prometheus.io/docs/prometheus/latest/querying/basics/>

1. **Alert Manager**

The Alertmanager handles alerts sent by client applications such as the Prometheus server. When an alert reaches its threshold, it is forwarded to Alertmanager that acts as a crossroad. Depending on its internal rules, it can forward those alerts further to various destinations like Slack, email, and PagerDuty (only to name a few). [19]

* 1. **Rules**

We already define all the necessary alerting rules in Prometheus. Alerting rules define alert conditions based on Prometheus expression language expressions and to send notifications about firing alerts to an external service.

1. Please navigate to Prometheus WebUI as described in Chapter 3.2.

|  |
| --- |
| We take <http://portal.prometheus.example.com> for example.  Please go to your Prometheus WebUI domain server. |

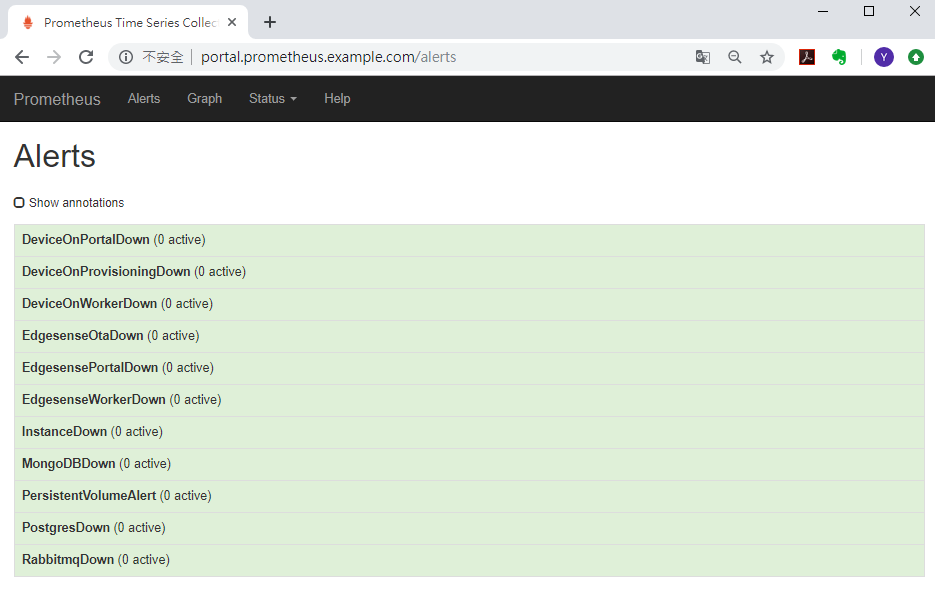
1. Navigate to “Alerts” label; you will see all the alerting rules we defined. You might check the expressions by clicking on each alert name. We already make sure that if there’s any instance works abnormally, alertmanager will send notification to user at the first time. 

Figure 15 Alert rules.

* 1. **Configuration**

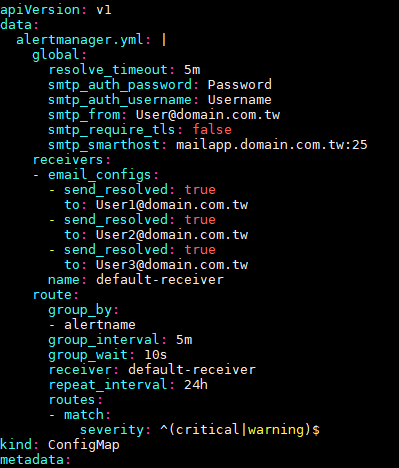
Alertmanager sends notifications to receivers which defined in configuration file. To receive alerting notifications, user needs to modify smtp server and receiver emails in configuration by issuing “kubectl” commands. Alertmanager can reload its configuration at runtime. We set alerting interval to1 day, in other words; the same alerting only send notification once in one day to avoid frequently mailing.

1. Issue command to edit configmap of alertmanager in namespace “monitoring”.

|  |
| --- |
| # kubectl edit -n=monitoring configmaps prometheus-alertmanager |

1. Configmap is shown as Fig 16. Please modify the **first** part to your smtp server host and login information. **Smtp\_from** means the alerting notification sender. **Smtp\_require\_tls** is default to false; you don’t have to change it.

**Second** part is receiver’s email; please add receivers in this part. If you have more receivers, please feel free to append it.



1

1

2

1

Figure 16 Alertmanager configmap

After saving all the modification, alertmanager will reload the configurations immediately. We can check the latest status on Alertmanager WebUI.

* 1. **Alertmanager Web UI**

1. Please go to your Alertmanager Web UI with no username and password.

|  |
| --- |
| We take [**http://portal.prometheus.example.com**](http://portal.prometheus.example.com) for example as Figure 17.  Please go to your Alertmanager WebUI domain server. |

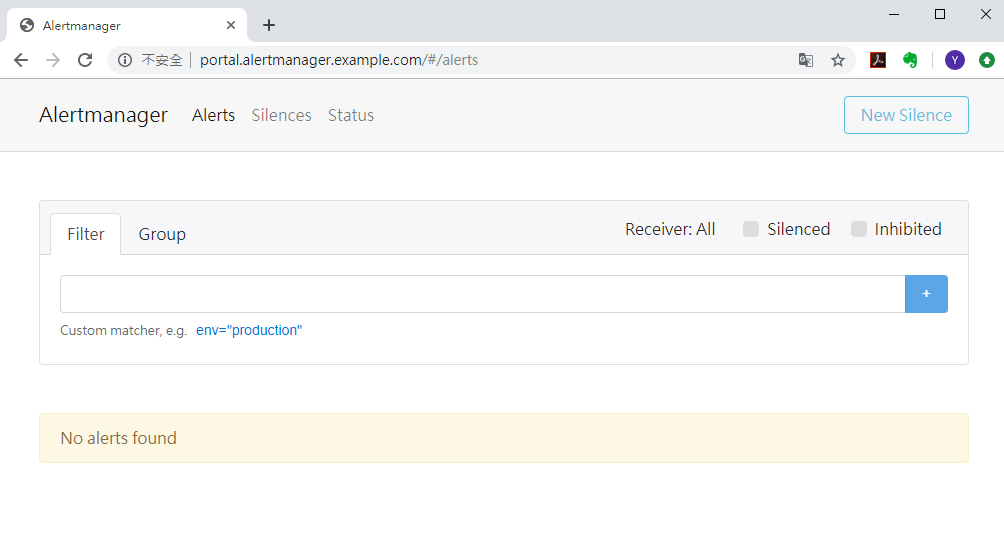
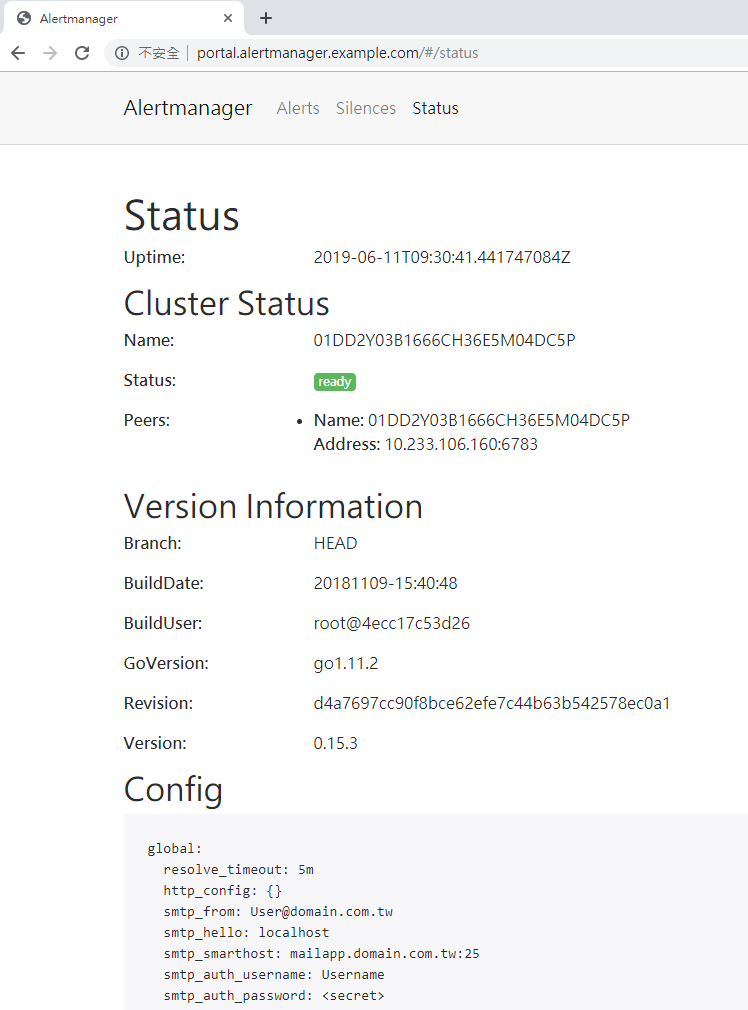


Figure 17 Alertmanager WebUI

1. Navigate to “Status” label; you will see the configuration you just saved. If smtp server works fine, receivers will receive alerting email when alert occurs. Please make sure that smtp server works fine.



* + 1. **Alerts**

Let’s forward to “Alerts” label, if there are alerts sent from Alertmanager, the alerts will be listed in this page as Figure 18. Here we take the alert called “DeviceOnPortalDown” for example.

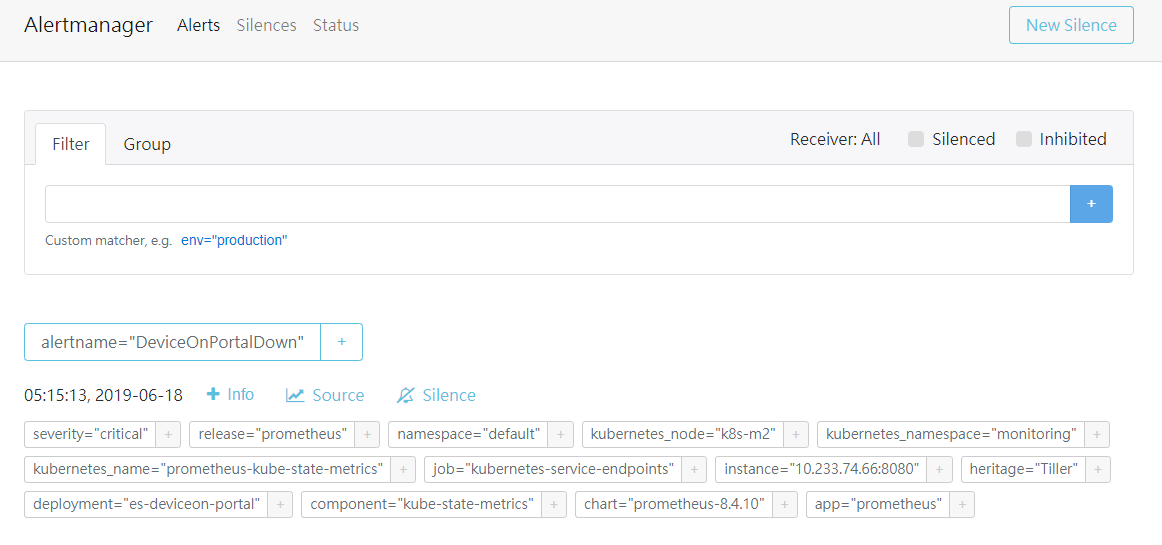


Figure 18 Alerts page in Alertmanager Web UI

In the same time, receiver will receive the alerting mail of “DeviceOnPortalDown”.

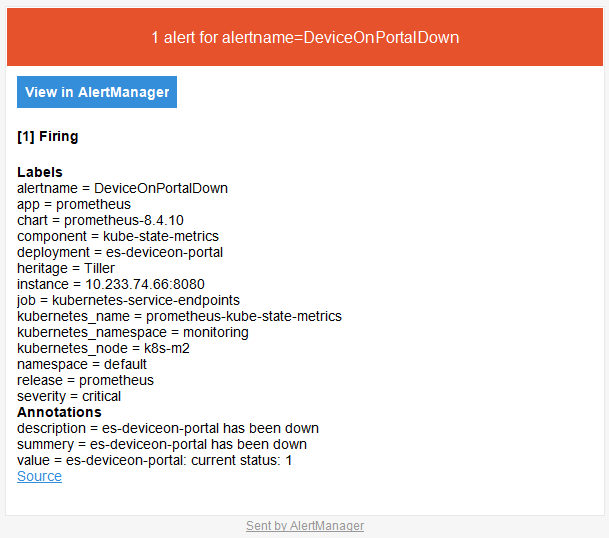
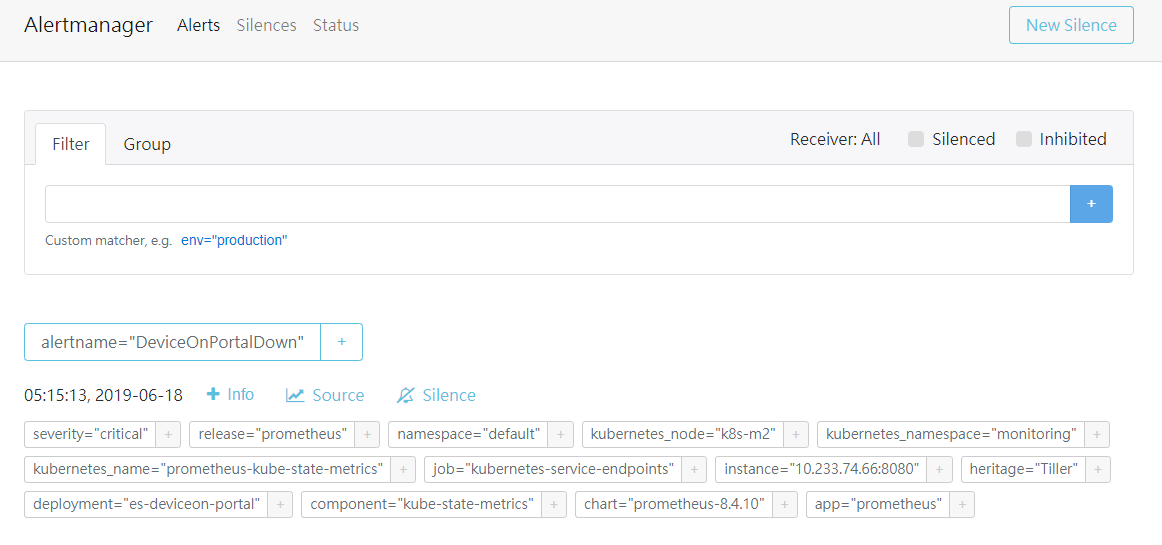


Figure 19 Alerting email

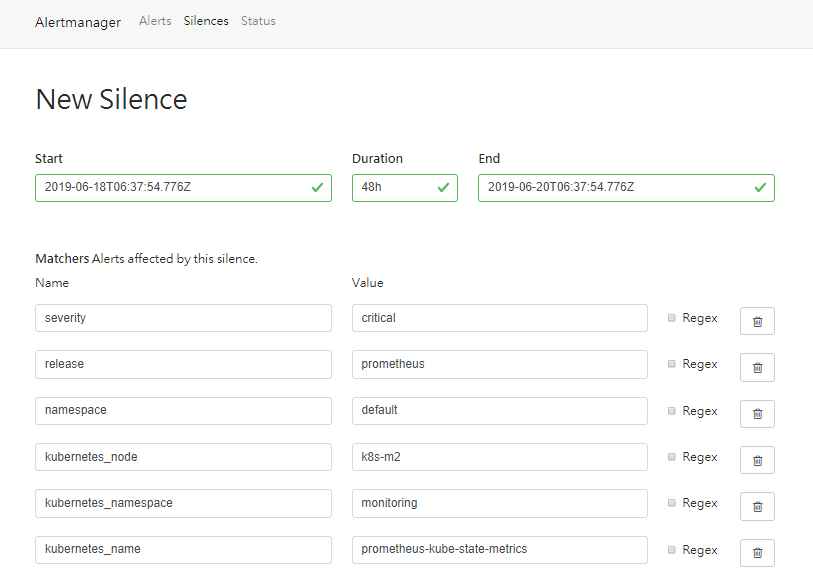
* + 1. **Silence**

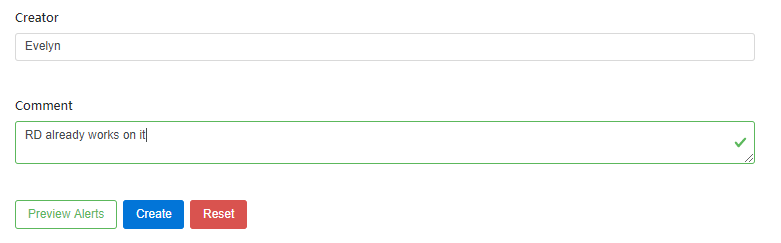
Alertmanager offers silence function. User can determine specific alert be silent for an interval not to send alerts.

1. We also take “DeviceOnPortalDown” for example. Click on “Silence”.

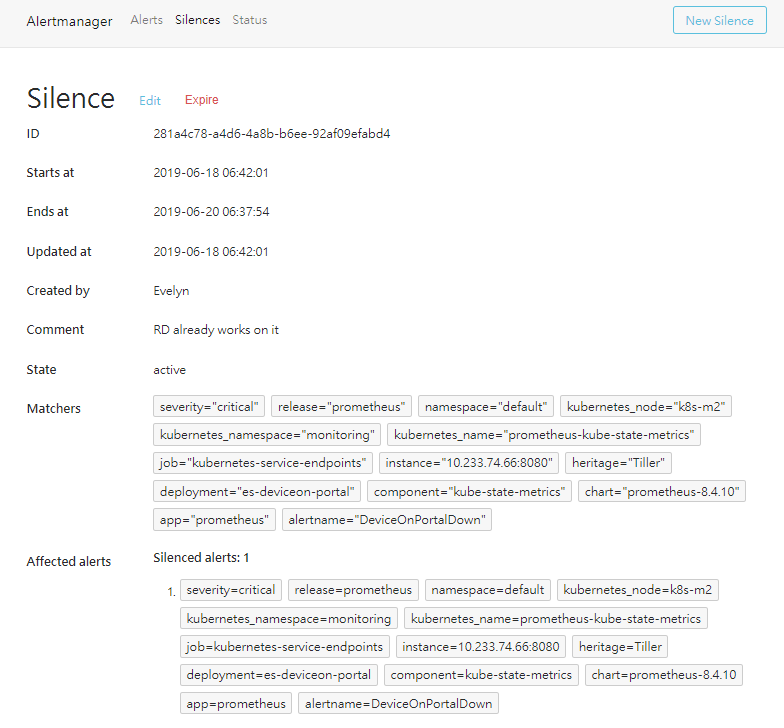


1. After setting the duration and comment, click on “Create”.





1. User will not receive this alert in the following 48 hours.



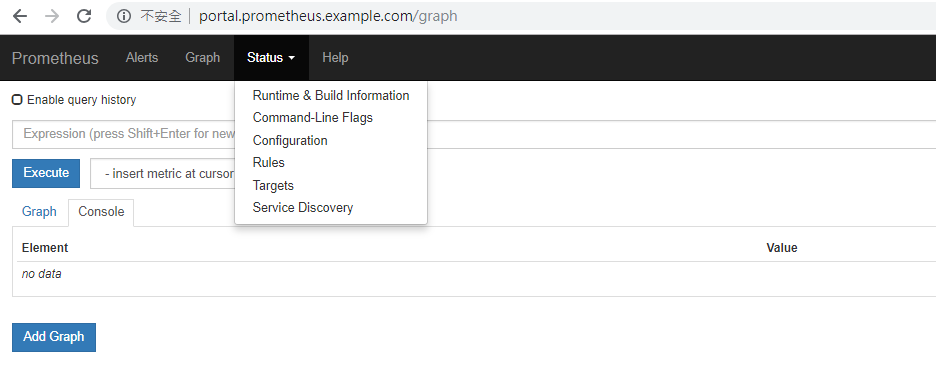
Note:

1. User can edit silence anytime, user can make it expire or extend the duration.
2. You won’t receive alerting if silence is set, please use it carefully.

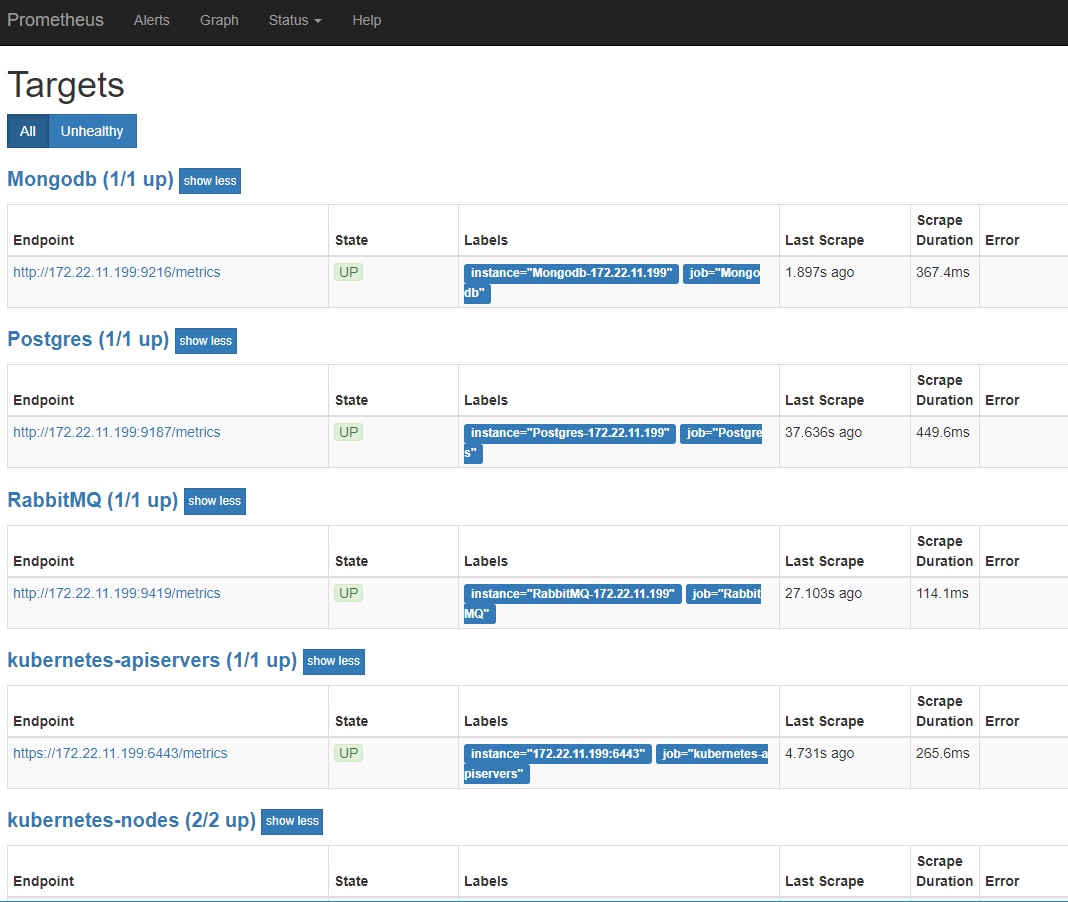
**FAQ**

* How to simply check if all the targets are under monitored?

1. Go to your Prometheus Web UI and navigate to “Targets” page.

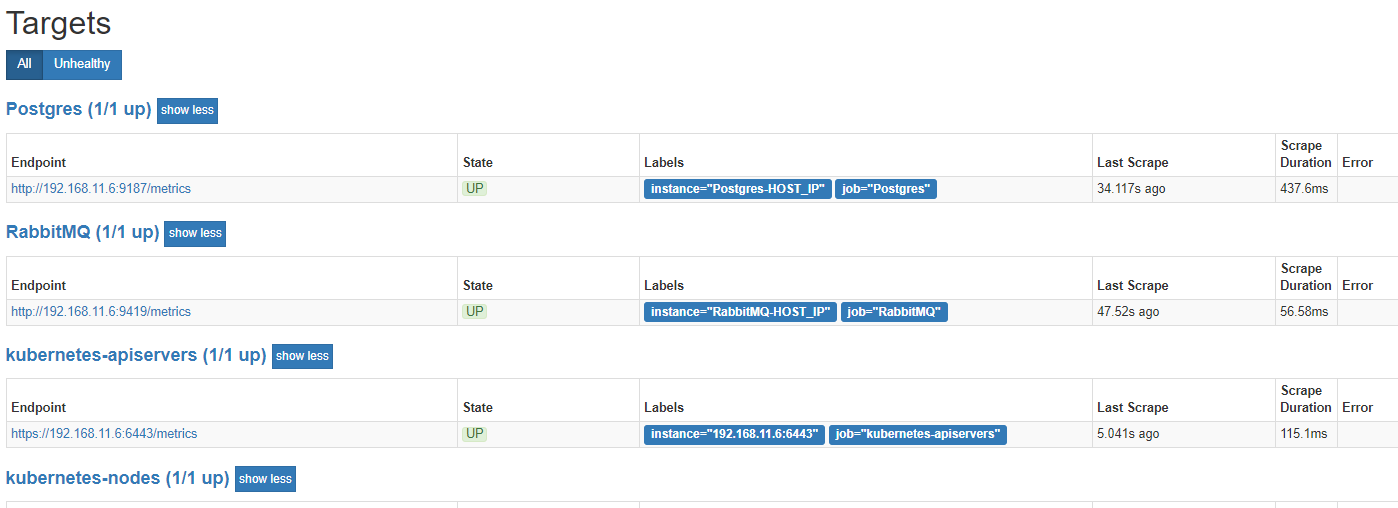


1. All the monitored targets will be listed as below, the targets will show “up” if their status are healthy.



* As mentioned above, if there is a target with “down” status and error message, how to check the error in detail?

Take “Postgres” for example and assume “Postgres” is down.



down

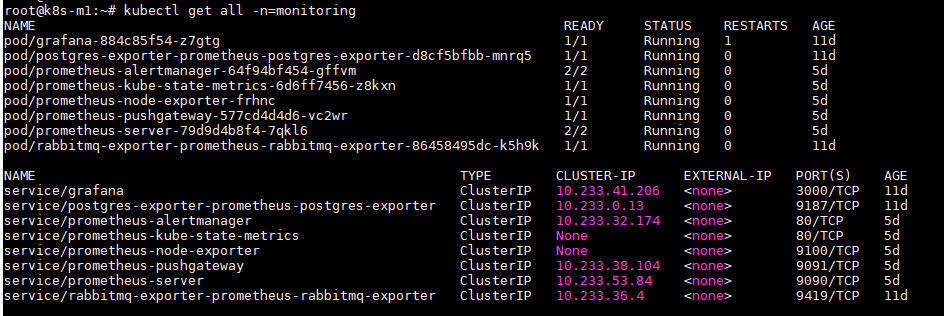
down

Please use command kubectl to get more information.

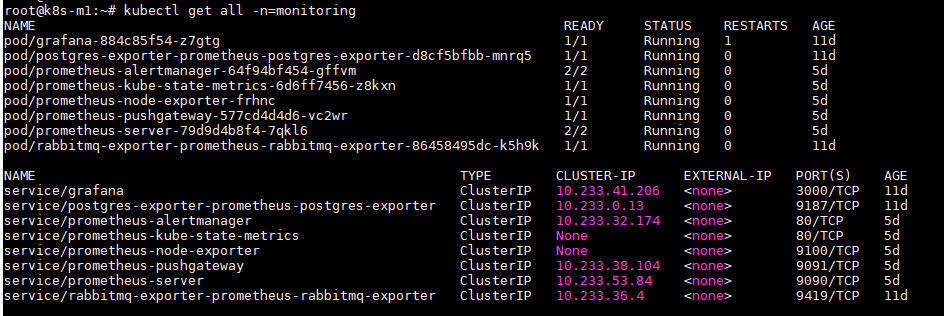
1. Issue command to get all resources in namespace “monitoring”.

|  |
| --- |
| # kubectl get all -n=monitoring |

All the resources in “monitoring” will show as below.



1. We need to get more information from “Postgres” Pod.



|  |
| --- |
| # kubectl logs -n=monitoring ${POD\_NAME}  Please fill in your pod name here and you will see the logs of this pod.  i.e. kubectl logs -n=monitoring postgres-exporter-prometheus-postgres-exporter-d8cf5bfbb-mnrq5 |

1. By the two steps above, if you can correct the error successfully, congrats!

But if you still have no ideas, please issue command below.

|  |
| --- |
| # kubectl describe pod -n=monitoring ${POD\_NAME}  Please fill in your pod name here.  i.e. kubectl describe pod -n=monitoring postgres-exporter-prometheus-postgres-exporter-d8cf5bfbb-mnrq5 |

Please copy and paste all the information from “kubectl logs” and “kubectl describe” commands and contact Advantech support team. Thanks!

**Reference**

|  |  |
| --- | --- |
| [1] | "Kubernetes," [Online]. Available: https://kubernetes.io/. |
| [2] | Weaveworks, "Monitoring Kubernetes with Prometheus," [Online]. Available: https://www.weave.works/technologies/monitoring-kubernetes-with-prometheus/. |
| [3] | Tom, "Prometheus and Kubernetes: A Perfect Match," [Online]. Available: https://www.weave.works/blog/prometheus-kubernetes-perfect-match/. |
| [4] | 殷. 蔡. 吴莉, "Prometheus 入门与实践," [Online]. Available: https://www.ibm.com/developerworks/cn/cloud/library/cl-lo-prometheus-getting-started-and-practice/index.html. |
| [5] | S. BISWAS. [Online]. Available: https://www.booleanworld.com/install-use-prometheus-monitoring/. |
| [6] | "Github of Prometheus/pushgateway," [Online]. Available: https://github.com/prometheus/pushgateway. |
| [7] | K. Aziz. [Online]. Available: http://weblogs.com.pk/khurram/archive/2018/04/16/prometheus.aspx. |
| [8] | B. Brazil, Prometheus: Up & Running, O'Reilly Media, Inc., 2018. |
| [9] | D. Berman, "Kubernetes Monitoring: Best Practices, Methods, and Existing Solutions," [Online]. Available: https://logz.io/blog/kubernetes-monitoring/. |
| [10] | A. Ellis, "Monitor your applications with Prometheus," [Online]. Available: https://blog.alexellis.io/prometheus-monitoring/. |
| [11] | [Online]. Available: https://kubernetes.io/docs/concepts/storage/persistent-volumes/. |
| [12] | devopscube, "Kubernetes Deployment Tutorial For Beginners," [Online]. Available: https://devopscube.com/kubernetes-deployment-tutorial/. |
| [13] | "Kubernetes concepts: StatefulSets," [Online]. Available: https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/. |
| [14] | M. Burillo, "Kubernetes Monitoring with Prometheus -The ultimate guide," [Online]. Available: https://sysdig.com/blog/kubernetes-monitoring-prometheus/. |
| [15] | Kubedex, "Prometheus-Rabbitmq-Exporter," [Online]. Available: https://kubedex.com/resource/prometheus-rabbitmq-exporter/. |
| [16] | "Github Docs of Postgres exporter," [Online]. Available: https://docs.gitlab.com/ee/administration/monitoring/prometheus/postgres\_exporter.html. |
| [17] | P. org, "About PostgreSQL," [Online]. Available: https://www.postgresql.org/about/. |
| [18] | "mongoDB," [Online]. Available: https://docs.mongodb.com/manual/introduction/. |
| [19] | [Online]. Available: https://prometheus.io/docs/alerting/alertmanager/. |