Data Service Server Admin’s Manual

V1.0.3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date | Version | Author | Reviewer | Description |
| 2019-03-19 | 1.0.0 | Zach Chih | Zach Chih | First version released |
| 2019-05-28 | 1.0.1 | Zach Chih | Zach Chih | Access Methods/Knowledge |
| 2019-06-13 | 1.0.2 | Zach Chih | Zach Chih | Helm/Kubectl |
| 2019-06-21 | 1.0.3 | Zach Chih | Zach Chih | Kubernetes Dashboard |
|  |  |  |  |  |
|  |  |  |  |  |

Revision History

**Table of Contents**

[1 Preface 5](#_Toc12374096)

[2 Introduction 6](#_Toc12374097)

[3 Architecture 7](#_Toc12374098)

[4 Components 8](#_Toc12374099)

[5 Network Topology 9](#_Toc12374100)

[6 Starting Points 10](#_Toc12374101)

[6.1 Get Started from your PC 11](#_Toc12374102)

[6.1.1 Installing kubectl 12](#_Toc12374103)

[6.1.2 Logging in Data Service Server as Admin 12](#_Toc12374104)

[6.1.3 Setting the Configuration for Authentication 12](#_Toc12374105)

[6.2 Get Started from Data Service Server 15](#_Toc12374106)

[7 Accessing Web Services 15](#_Toc12374107)

[7.1 Through IP w/ Port 15](#_Toc12374108)

[7.1.1 Viewing the Nginx Resources 16](#_Toc12374109)

[7.1.2 Accessing Web Services 17](#_Toc12374110)

[7.2 Through DNS 18](#_Toc12374111)

[7.2.1 Viewing the DNS Resources 18](#_Toc12374112)

[7.2.2 Setting the DNS 19](#_Toc12374113)

[7.2.2.1 Windows 19](#_Toc12374114)

[7.2.2.2 Linux 25](#_Toc12374115)

[7.2.3 Viewing the Endpoints (Ingress) 26](#_Toc12374116)

[7.2.4 Accessing Web Services 26](#_Toc12374117)

[7.3 Summary: Access Methods 27](#_Toc12374118)

[7.3.1 EdgeSense 28](#_Toc12374119)

[7.3.2 DeviceOn 28](#_Toc12374120)

[7.3.3 RabbitMQ 29](#_Toc12374121)

[7.3.4 Minio 29](#_Toc12374122)

[7.3.5 Prometheus 30](#_Toc12374123)

[7.3.6 Grafana 30](#_Toc12374124)

[7.3.7 AlertManager 31](#_Toc12374125)

[7.3.8 ChartMuseum 31](#_Toc12374126)

[8 Kubernetes Dashboard 32](#_Toc12374127)

[8.1 Login 32](#_Toc12374128)

[8.2 Viewing 34](#_Toc12374129)

[After you login the dashboard, you could explore all the resources on the Data Service Server. 34](#_Toc12374130)

[8.3 Logging 34](#_Toc12374131)

[9 Charts 35](#_Toc12374132)

[9.1 Installing Helm 35](#_Toc12374133)

[9.2 Viewing Charts 36](#_Toc12374134)

[10 Disaster Recovery 36](#_Toc12374135)

[10.1 Restoration 37](#_Toc12374136)

[10.1.1 Restoring Applications 37](#_Toc12374137)

[11 Precautions 37](#_Toc12374138)

[11.1 Shutdown/Reboot 37](#_Toc12374139)

[12 Knowledge 38](#_Toc12374140)

[12.1 Basic Knowledge of Resources of K8s 38](#_Toc12374141)

[12.1.1 Pods 38](#_Toc12374142)

[12.1.2 Services 40](#_Toc12374143)

[12.1.3 ReplicaSets 40](#_Toc12374144)

[12.1.4 Deployments 41](#_Toc12374145)

[12.1.5 Secrets 41](#_Toc12374146)

[12.1.6 ConfigMaps 42](#_Toc12374147)

[12.2 Commands of kubectl 43](#_Toc12374148)

[12.2.1 GET 44](#_Toc12374149)

[12.2.2 DESCRIBE 45](#_Toc12374150)

[12.2.3 LOGS 45](#_Toc12374151)

[12.2.4 DELETE 45](#_Toc12374152)

[12.2.5 EDIT 46](#_Toc12374153)

[12.3 Basic Knowledge of Helm 46](#_Toc12374154)

[12.3.1 Purposes 46](#_Toc12374155)

[12.3.2 Components 47](#_Toc12374156)

[12.3.3 Commands 48](#_Toc12374157)

[12.3.3.1 INIT 48](#_Toc12374158)

[12.3.3.2 INSTALL 48](#_Toc12374159)

[12.3.3.3 LIST 49](#_Toc12374160)

[12.3.3.4 DELETE 49](#_Toc12374161)

[12.3.3.5 REPO 49](#_Toc12374162)

[13 FAQ 50](#_Toc12374163)

[13.1 Accessing 50](#_Toc12374164)

[13.1.1 Why can’t I use kubectl to access the resources on Data Service Server? 50](#_Toc12374165)

[13.1.2 Why can’t I access the services by the endpoints in ingresses? 50](#_Toc12374166)

[13.1.3 How should I set the URLs for dashboard and S3 compatible storage in DeviceOn/EdgeSense? 50](#_Toc12374167)

[13.2 Authentication 51](#_Toc12374168)

[13.2.1 Why do I keep getting an error message “error: You must be logged in to the server (Unauthorized)”when I use kubectl? 51](#_Toc12374169)

[13.2.2 Why do I keep get an error message “Unable to connect to the server: dial tcp [::1]:8080: connectex: No connection could be made because the target machine actively refused it.”when I use kubectl? 51](#_Toc12374170)

[13.3 Unhealthy Pods 51](#_Toc12374171)

[13.3.1 Why do I get the statuses of pods which are not *Running* but *Error* or others? 51](#_Toc12374172)

[13.3.2 How can I do to those unhealthy pods? 51](#_Toc12374173)

[13.4 Environment Change 52](#_Toc12374174)

[13.4.1 Could I change the IP address of the Data Service Server? 52](#_Toc12374175)

[14 Appendix 52](#_Toc12374176)

[14.1 Ports 52](#_Toc12374177)

[14.2 Credentials 53](#_Toc12374178)

[15 References 54](#_Toc12374179)

1. Preface

This manual is intended for new users with little or no experience using the Data Service Server. The goal of this document is to give a broad overview of the main functions of Data Service Server and some basic instructions on how to view and access the resources. This document will concentrate on demonstrating interaction with Data Service Server using the kubectl which is a command line tool used to communicate with Kubernetes.

Every effort has been made to ensure that this document is an accurate representation of the functionality of Data Service Server. As with every software application, development continues after the documentation has gone to press so small inconsistencies may occur. We would appreciate any feedback on this manual.

1. Introduction

If you’re wondering what Data Service Server is, why you might want one and what it can do for you, then read on – you’re about to find out.

At the simplest level, Data Service Server is three things: a data streaming, a data collection system and a monitoring system combined. Data streaming is a process in which every sensor data from devices could be upstream and data relating to controlling commands could be downstream. These two ways are the main communication channel by which end users could interact with their target devices. The interaction might be active or passive depending on your scenarios. The data collection process is in the middle of the whole streaming. Data Service Server would collect all the sensor data that users want and provide a user-friendly dashboard for user to manage and monitor the target devices. And Data Service Server has a backend monitoring system which is very powerful. The monitoring system could help users easily get to know the conditions and statuses of the components on the Data Service Server.

1. Architecture

Here’s the architecture of Data Service Server. All of containerized applications are running based on a PaaS, which consists of kubernetes, docker and ceph. Kubernetes (k8s) is an open-source system for automating deployment, scaling, and management of containerized applications. [1] Docker is a computer program that performs operating-system-level virtualization, also known as "containerization". [2] Ceph is a [free-software](https://en.wikipedia.org/wiki/Free-software) [storage](https://en.wikipedia.org/wiki/Computer_data_storage) [platform](https://en.wikipedia.org/wiki/Computing_platform), implements [object storage](https://en.wikipedia.org/wiki/Object_storage) on a single distributed [computer cluster](https://en.wikipedia.org/wiki/Computer_cluster), and provides interfaces for object-, [block-](https://en.wikipedia.org/wiki/Block_storage) and [file](https://en.wikipedia.org/wiki/Computer_file)-level storage. [3] By leveraging the power of these services behind the scenes, Data Service Server could be based on this robust and easily manageable ecosystem to provide preeminent data services for your devices.



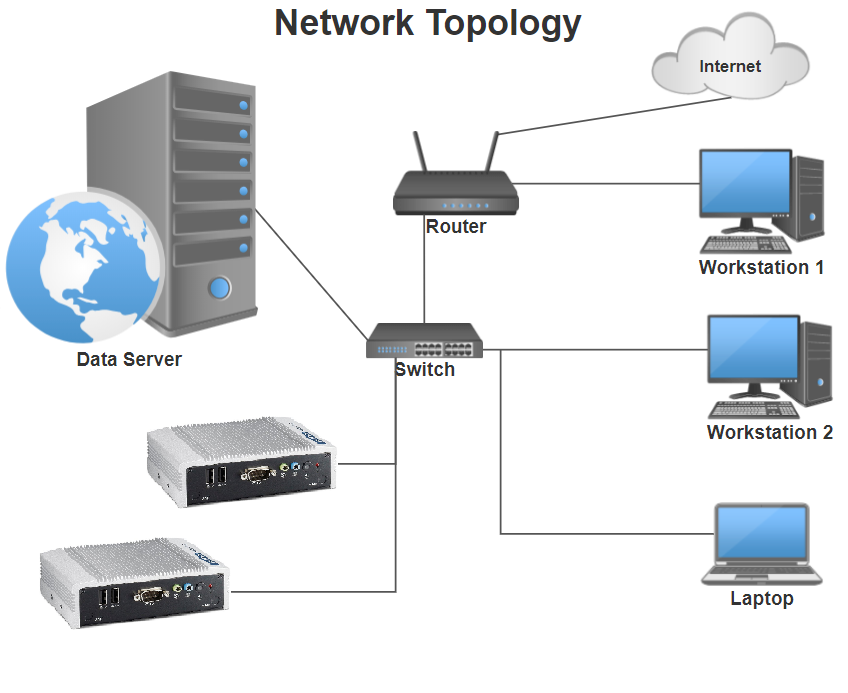
1. Components

Kubernetes supports multiple virtual clusters backed by the same physical cluster. These virtual clusters are called namespaces. Namespaces are intended for use in environments with many users spread across multiple teams, or projects. [4] In Data Service Server, there’re namespaces — default, network and monitoring, etc. The main components are in default namespace. The networking-related components are in network namespace. And the components used to keep an eye on the Data Service Server are in monitoring namespace.

|  |  |
| --- | --- |
| **Namespaces** | **Components** |
| default | EdgeSense/DeviceOn |
| RabbitMQ |
| Postgres |
| Mongo |
| Minio |
| network | Nginx |
| CoreDNS |
| monitoring | Prometheus |
| Grafana |
| Postgres-Exporter |
| RabbitMQ-Exporter |
| **Repository** | ChartMuseum |

1. Network Topology

The figure is an example network topology. Your workstations or laptops might be able to connect to Data Service Server through a router or a switch. And some of the host devices might connect to Data Service Server through the switch. The topology might be different from different users’ working environments. **Just make sure your connections to Data Service Server work well.**



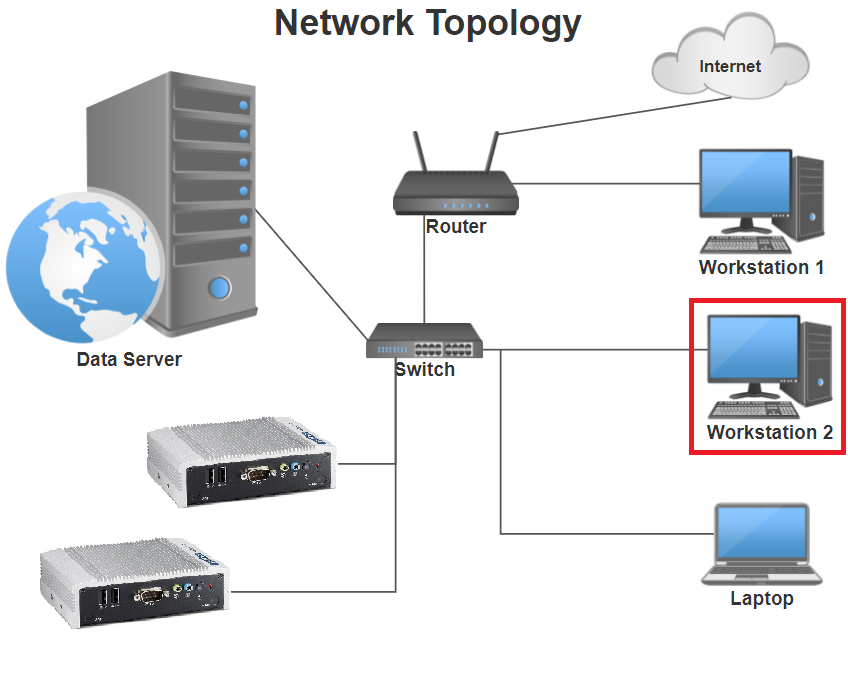
1. Starting Points

Data Service Server has two starting points to access and manage all the resources and applications running on itself. You could get started from your own PC or from Data Service Server.

Get Started from your PC

To have a better picture on what we are going to do, let’s assume you’re using the Workstation 2 as your PC and your connection to Data Service Server works well.

After setting up a good starting point, let’s get started to set an environment to explore the resources on the Data Service Server by a configuration of authentication. Finally, you would know how to access the web services for management and monitoring.



Installing kubectl

Kubectl is a command line interface for running commands against Kubernetes clusters. [5] For details about each command, including all the supported flags and subcommands, see the [kubectl](https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands/) reference documentation. For installation instructions see [installing kubectl](https://kubernetes.io/docs/tasks/kubectl/install/). [6] Installing kubectl in your PC is the first.

Logging in Data Service Server as Admin

To get a configuration and a certificate for authentication in the next section, as a user, you could login by Secure Shell (ssh) with default superuser (root). [7] The default username and password is root/r00tadvant1cH

Setting the Configuration for Authentication

Before exploring the resources on Data Service Server, you should set an environment with a configuration for authentication. There are two files you would need for the configuration. These files are in the following paths on the Data Service Server.

* config — /root/config — the configuration file for authentication
* ca.pem — /root/ca — the certificate for accessing the Data Service Server

Here’s the content in config:

clusters:

- cluster:

server: <server-ip>

name: cluster.local

contexts:

- context:

cluster: cluster.local

user: root

name: root-ctx

current-context: root-ctx

kind: Config

preferences: {}

users:

- name: root

user:

password: r00tadvant1cH

username: root

First, copy the config file and ca.crt from Data Service Server into your computer and put them in the director **$HOME/.kube**. Then use the command below to set the certificate for the configuration.

After the command, you would see the content of config like this:

$ kubectl config set-cluster cluster.local --certificate-authority=<ca-path>

apiVersion: v1

clusters:

- cluster:

certificate-authority: <ca-path>

server: <server-ip>

name: cluster.local

contexts:

- context:

cluster: cluster.local

user: root

name: root-ctx

current-context: root-ctx

kind: Config

preferences: {}

users:

- name: root

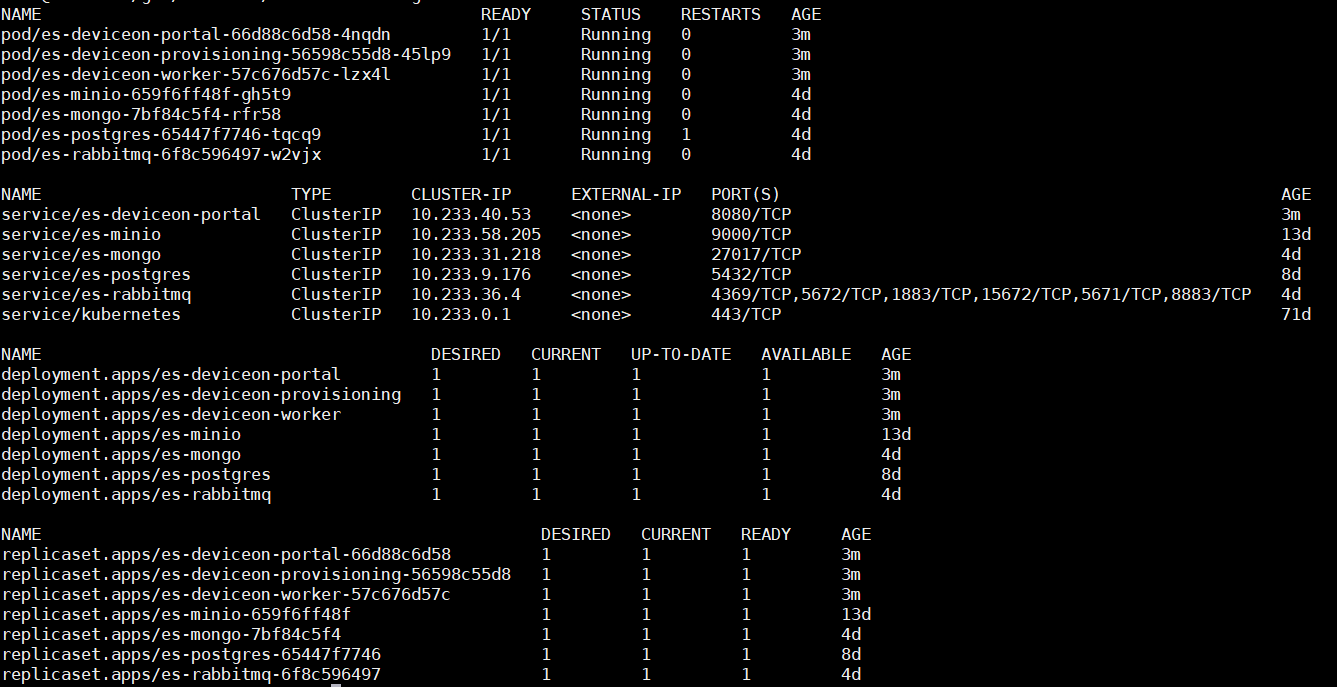
user:

password: r00tadvant1cH

username: root

Then use the command below to check your configuration:

$ kubectl get all



If you could successfully get the resources on the Data Service Server, it means you already pass the authentication.

Get Started from Data Service Server

As a user, you could log in Data Service Server by remote desktop with default user (root). The default username and password is root/r00tadvant1cH

On the Data Service Server, you could skip section 6.1.1/6.1.2/6.1.3 because the kubectl and the configuration for authentication are pre-settled.

1. Accessing Web Services

Data Service Server provides two ways for you to access the web services like EdgeSense, DeviceOn and Grafana, etc. You could access the services directly through IP w/ Port or through DNS.

Through IP w/ Port

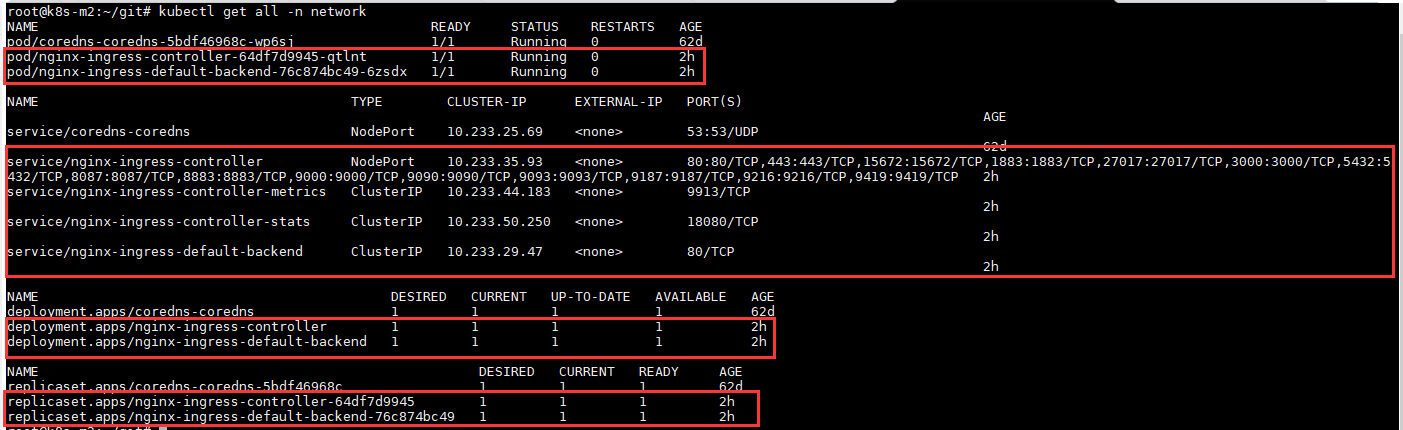
You could access the resources directly through IP with Port. This section takes DeviceOn as the example. The other resources could be accessed by the same way.

Viewing the Nginx Resources

Nginx [engine x] is an HTTP and reverse proxy server, a mail proxy server, and a generic TCP/UDP proxy server. [8] An Ingress Controller is a daemon, deployed as a Kubernetes Pod, that watches the apiserver's /ingresses endpoint for updates to the [Ingress resource](https://kubernetes.io/docs/concepts/services-networking/ingress/). [9] Its job is to satisfy requests for Ingresses. The section is to view the resources of nginx-ingress controller.

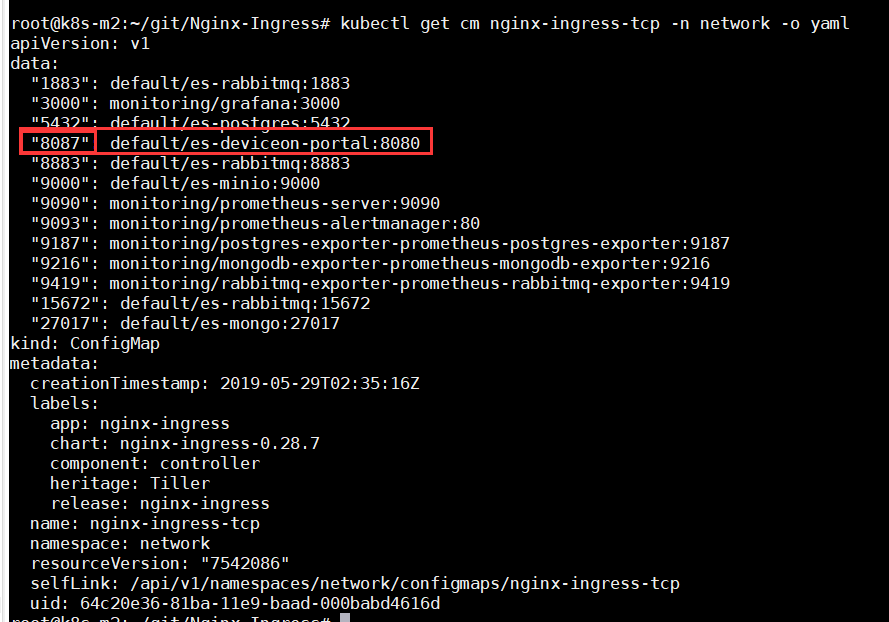
The command below would show the resources related to nginx in the namespace network and check the statuses are running:

$ kubectl get all –n network



Then use the command below to check the exposed port of DeviceOn:

$ kubectl get cm nginx-ingress-tcp –n network –o yaml

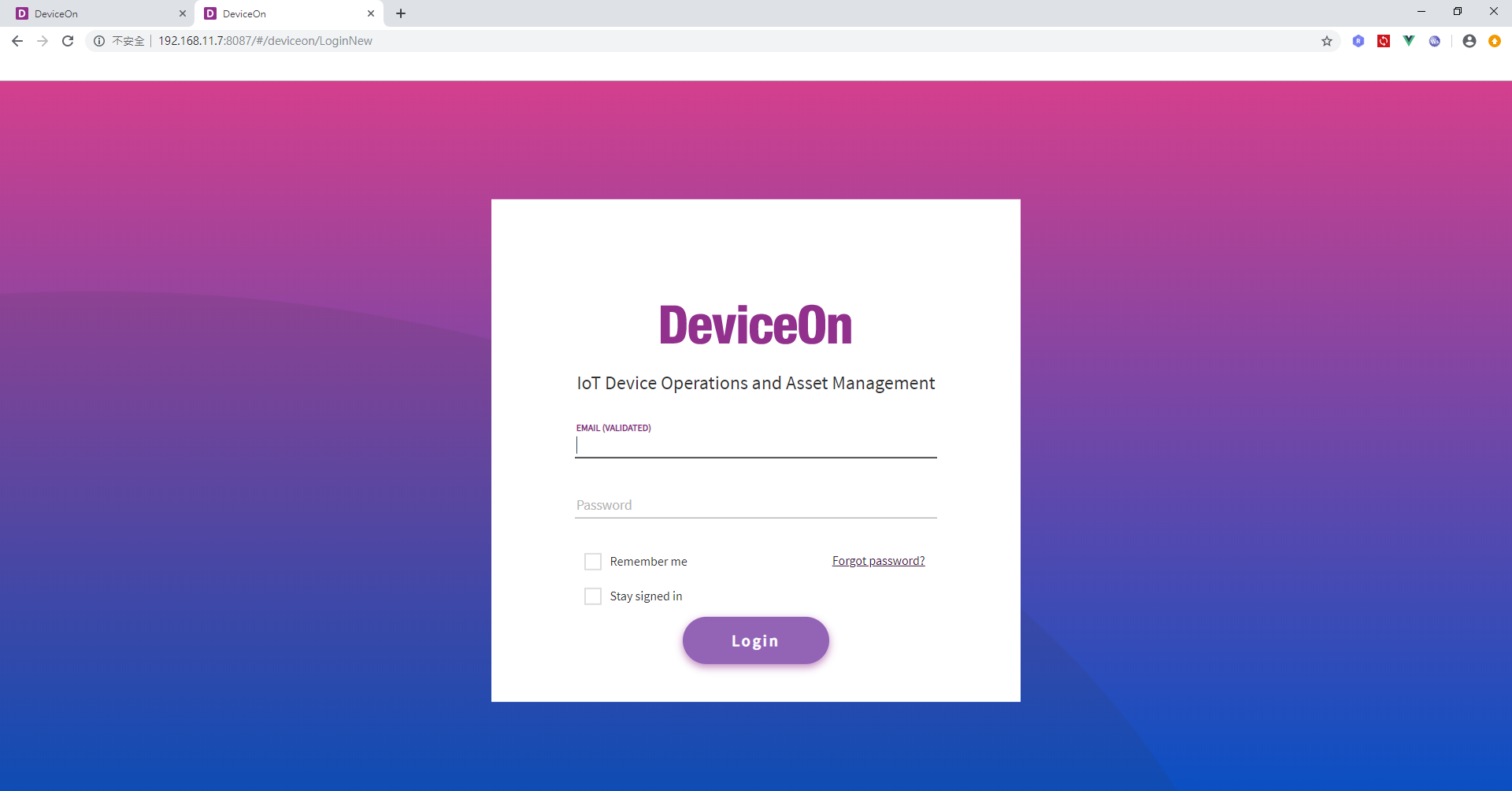


We could see the exposed port for DeviceOn is 8087. (8080 on the right is internal port) You should use exposed port on the left, instead.

Accessing Web Services

Open the browser to access DeviceOn.

IP Access — **http://<DataServer-IP>:8087**



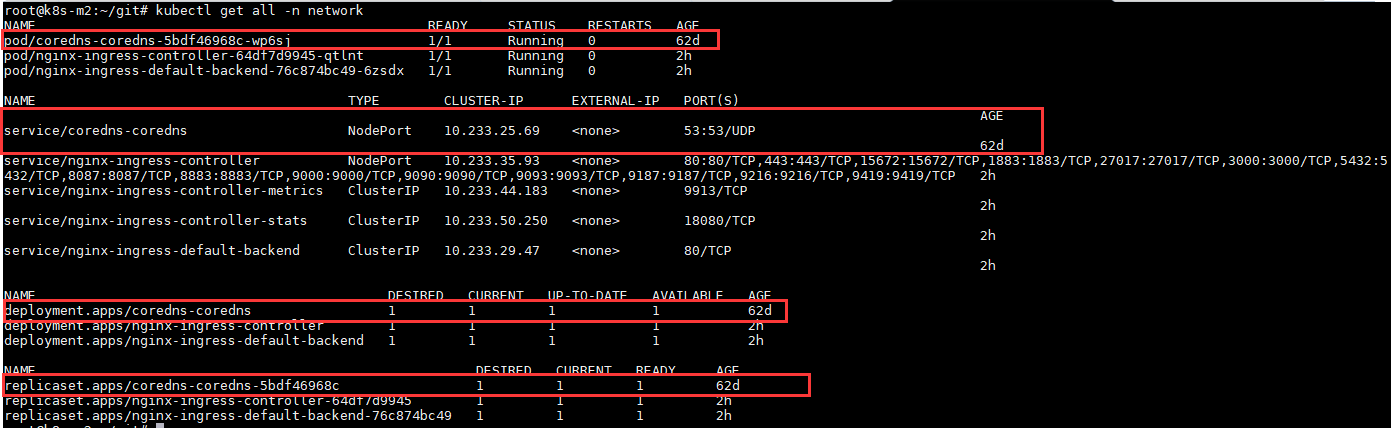
Through DNS

The other way to access the resources is through DNS. This section takes DeviceOn as the example. Still, make sure nginx is healthy because it’s the starting point for any access methods. The other resources could be accessed by the same way.

Viewing the DNS Resources

The DNS Sever in Data Service Server is CoreDNS. Use the command below to view the resources related to coredns in namespace network and check the statuses are running:

$ kubectl get all –n network

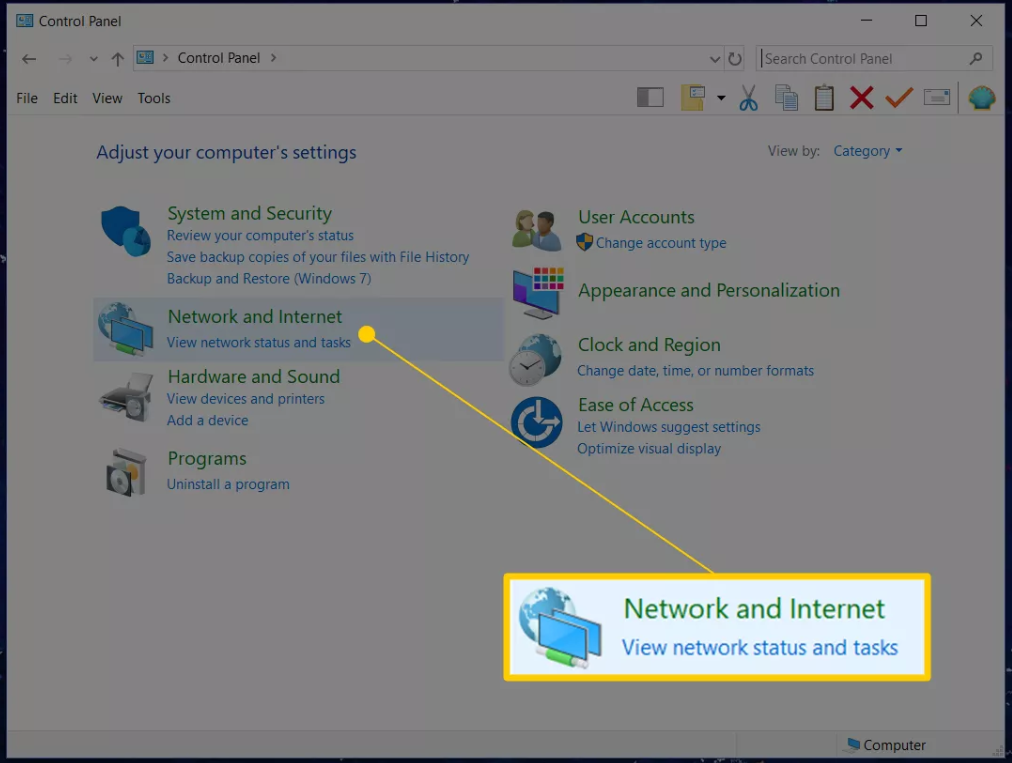


Setting the DNS

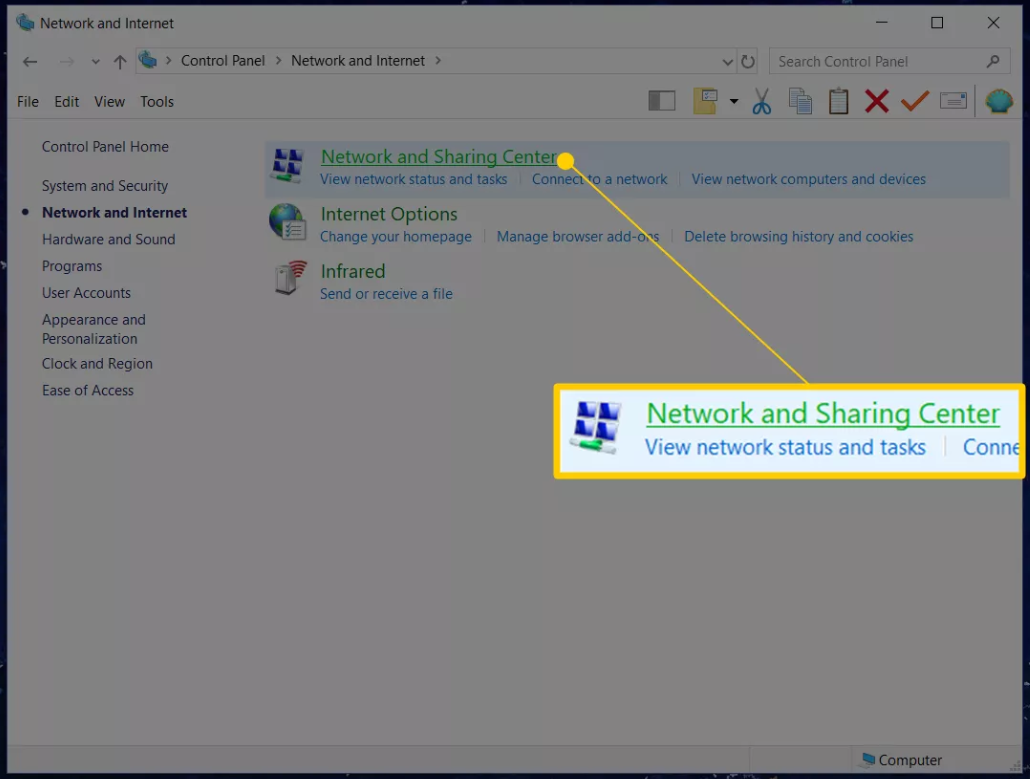
Before you try to connect to these services by these endpoints, you should set your DNS IP to the IP Data Service Server locates at. This is because the URL translation is being handled by the CoreDNS which the Data Service Server provides. The steps of setting the DNS Server are as follows. [10]

* + - 1. Windows

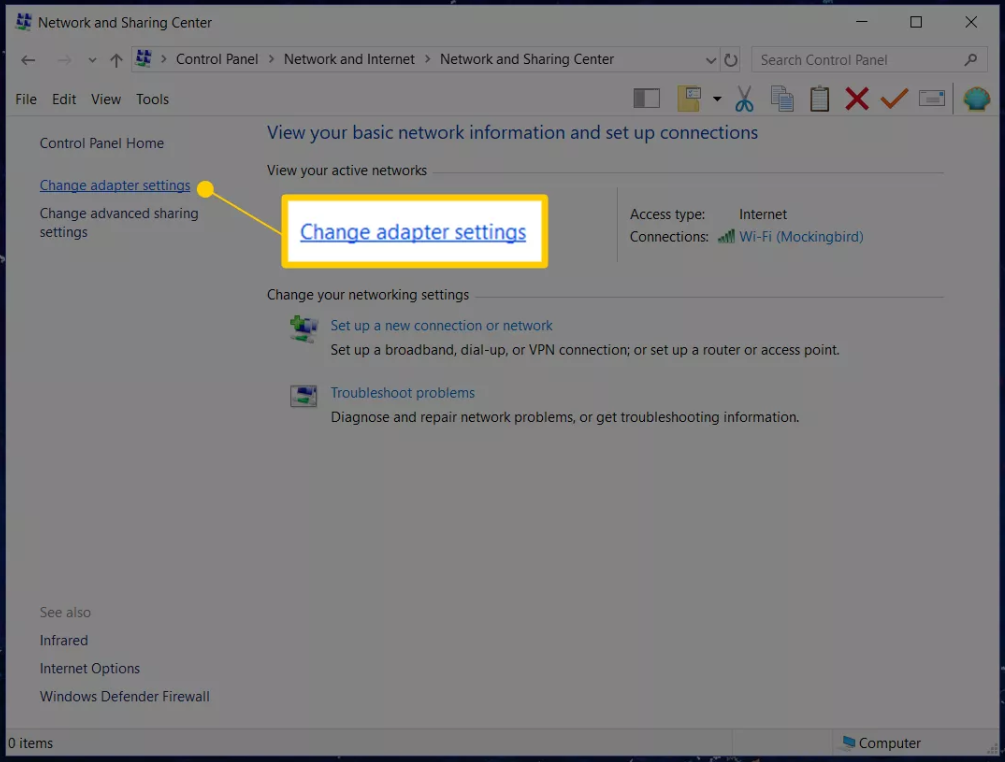
1. [Open Control Panel](https://www.lifewire.com/how-to-open-control-panel-2625782).
2. Once in [Control Panel](https://www.lifewire.com/control-panel-2625841), touch or click **Network and Internet**.



1. In the Network and Internet window that's now open, click or touch**Network and Sharing Center** to open that [applet](https://www.lifewire.com/control-panel-applet-2625831).



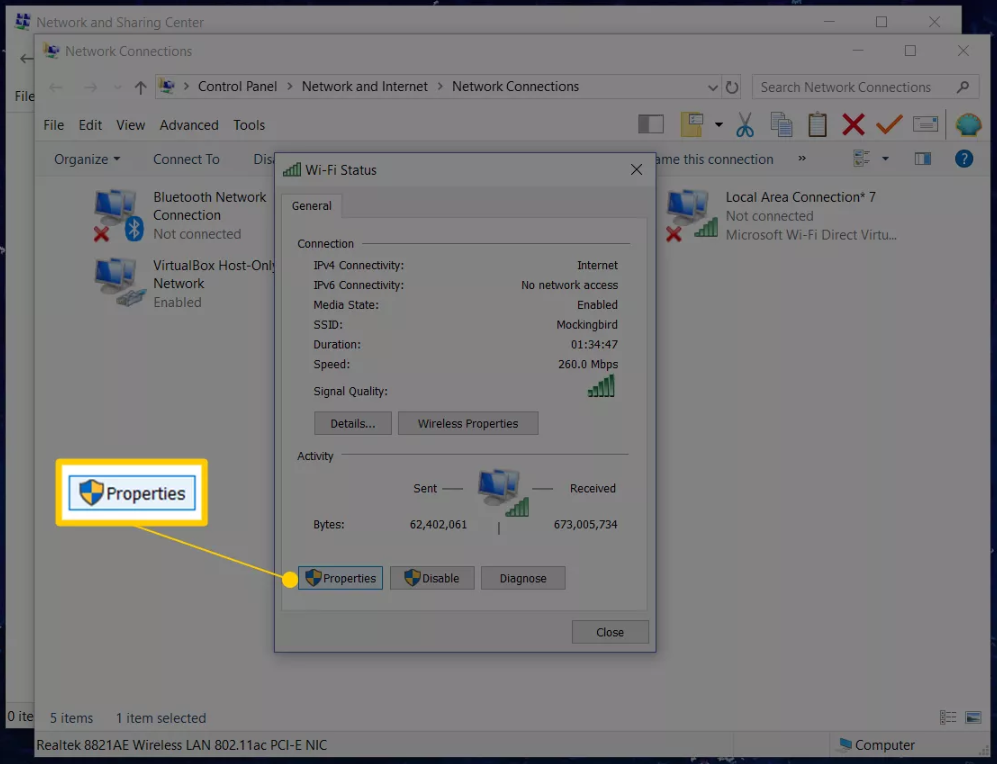
1. Now that the Network and Sharing Center window is open, click or touch **Change adapter settings**, located in the left margin.



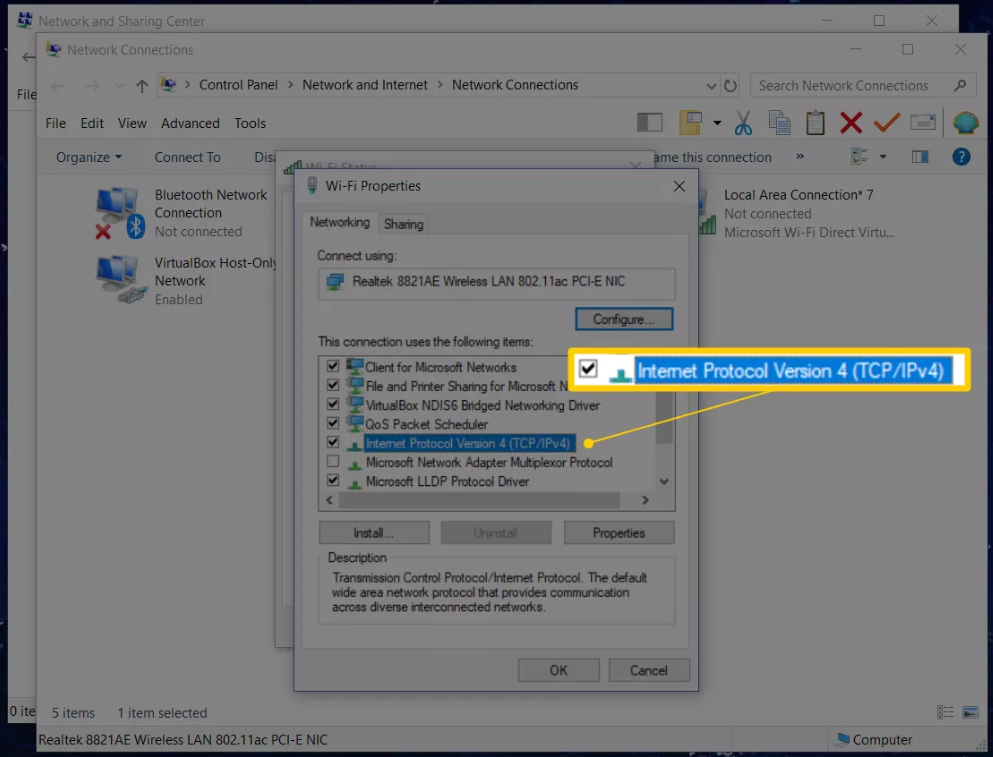
1. From this new Network Connections screen, locate the network connection that you want to change the DNS servers for.



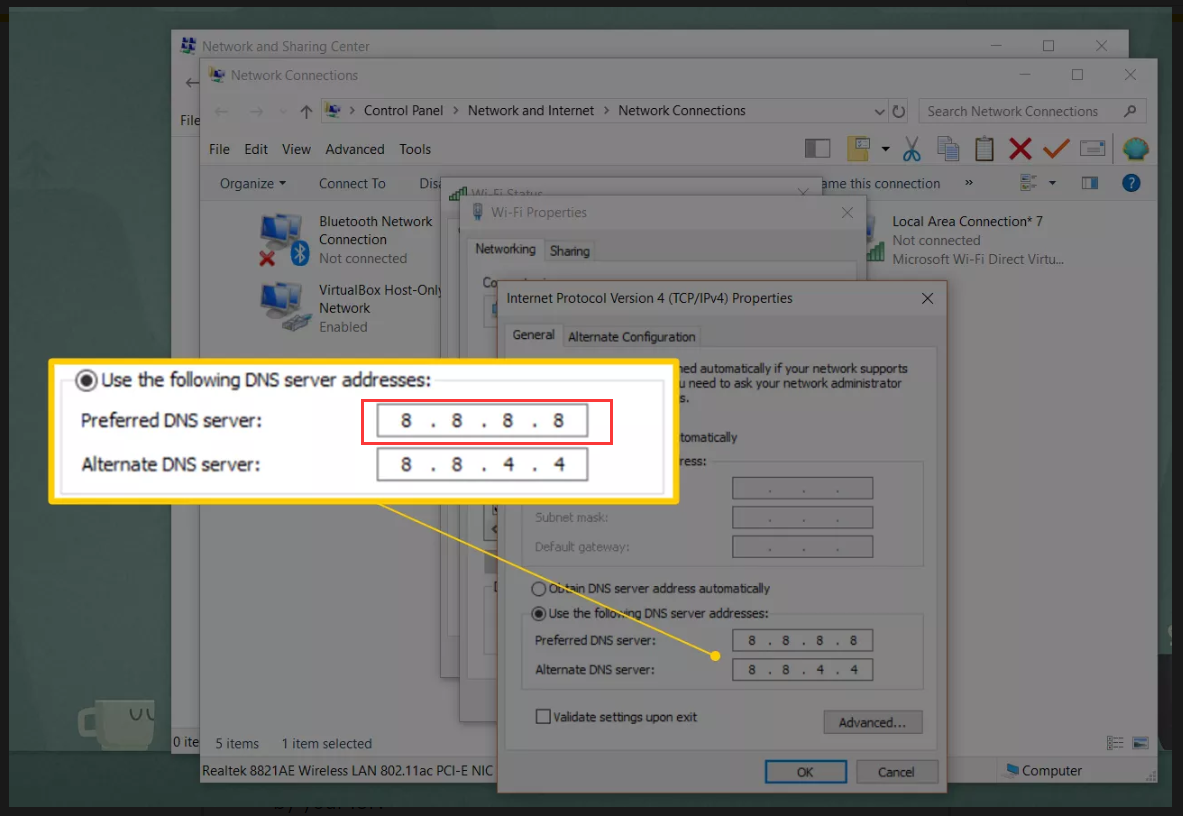
1. Open the network connection you want to change the DNS servers for by double-clicking or double-tapping on its icon.
2. Tap or click **Properties** on the connection's Status window that's now open.



1. On the connection's Properties window that appeared, scroll down in the This connection uses the following items: list and click or tap**Internet Protocol Version 4 (TCP/IPv4)** or **Internet Protocol (TCP/IP)**to select the IPv4 option to change the IPv4 DNS server settings.



1. Tap or click **Properties** below the list.
2. Choose the **Use the following DNS server addresses:** radio button at the bottom of the Internet Protocol Properties window.
3. In the spaces provided, enter the IP address for the **CoreDNS server**.



1. Tap or click **OK**.

The DNS server change takes place immediately. You can now close any Properties, Status, Network Connections, or Control Panel windows that are open.

1. Then use $ nslookup <URL> to check if the URL translation is proper or not.
   * + 1. Linux

In linux, you should just use $ sudo vi /etc/resolv.conf to add “options rotate”and“nameserver <DataServer-IP>”to the first and the second line then save the file. Then you could use $ nslookup <URL> to check if the URL translation is proper or not.

Viewing the Endpoints (Ingress)

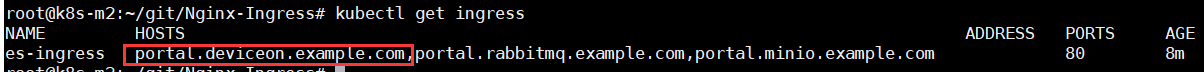
After viewing the resources related to networking in namespace network, what you must want to know is what the endpoints of web services are. Before that, you should know what ingress is.

Ingress, added in Kubernetes v1.1, exposes HTTP and HTTPS routes from outside the cluster to [services](https://kubernetes.io/docs/concepts/services-networking/service/) within the cluster. Traffic routing is controlled by rules defined on the Ingress resource. [11]

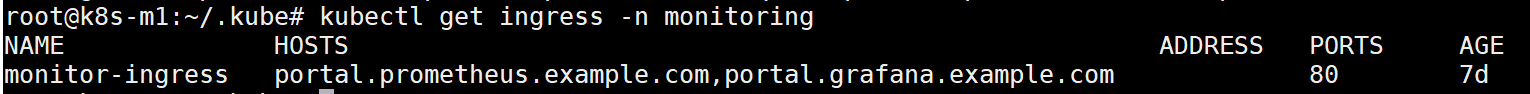
Now, you could view the endpoints of services routing by the ingresses. There’re two ingresses, one in namespace default, the other in namespace network.

Type the command:

$ kubectl get ingress



$ kubectl get ingress –n monitoring

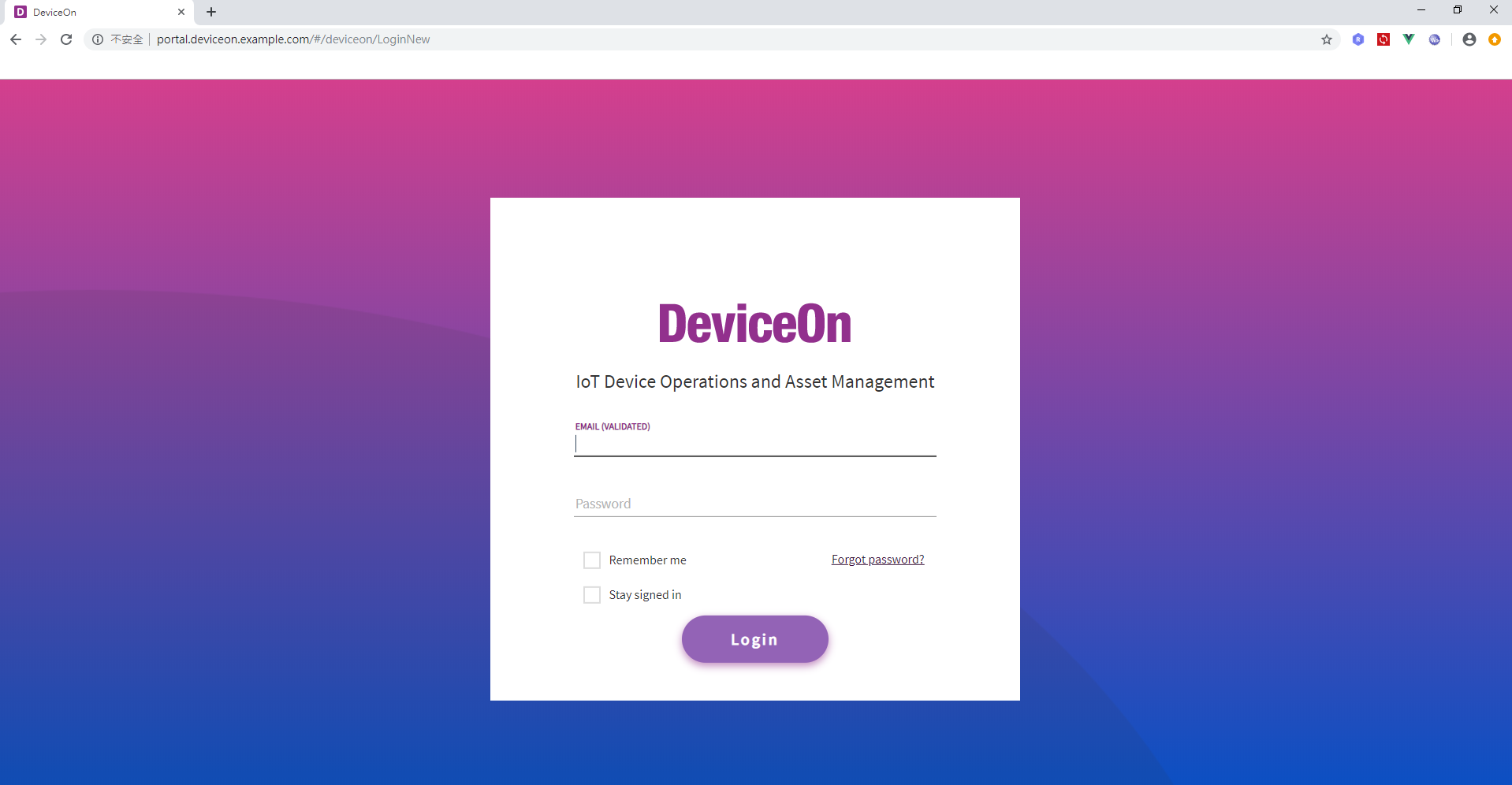


The examples here are the endpoints based on domain of .example.com. You might see the domain different from this one. The URL for DeviceOn is portal.deviceon.example.com in this example.

Accessing Web Services

Open the browser to access DeviceOn by the endpoint.

Endpoint — <http://portal.deviceon.example.com>



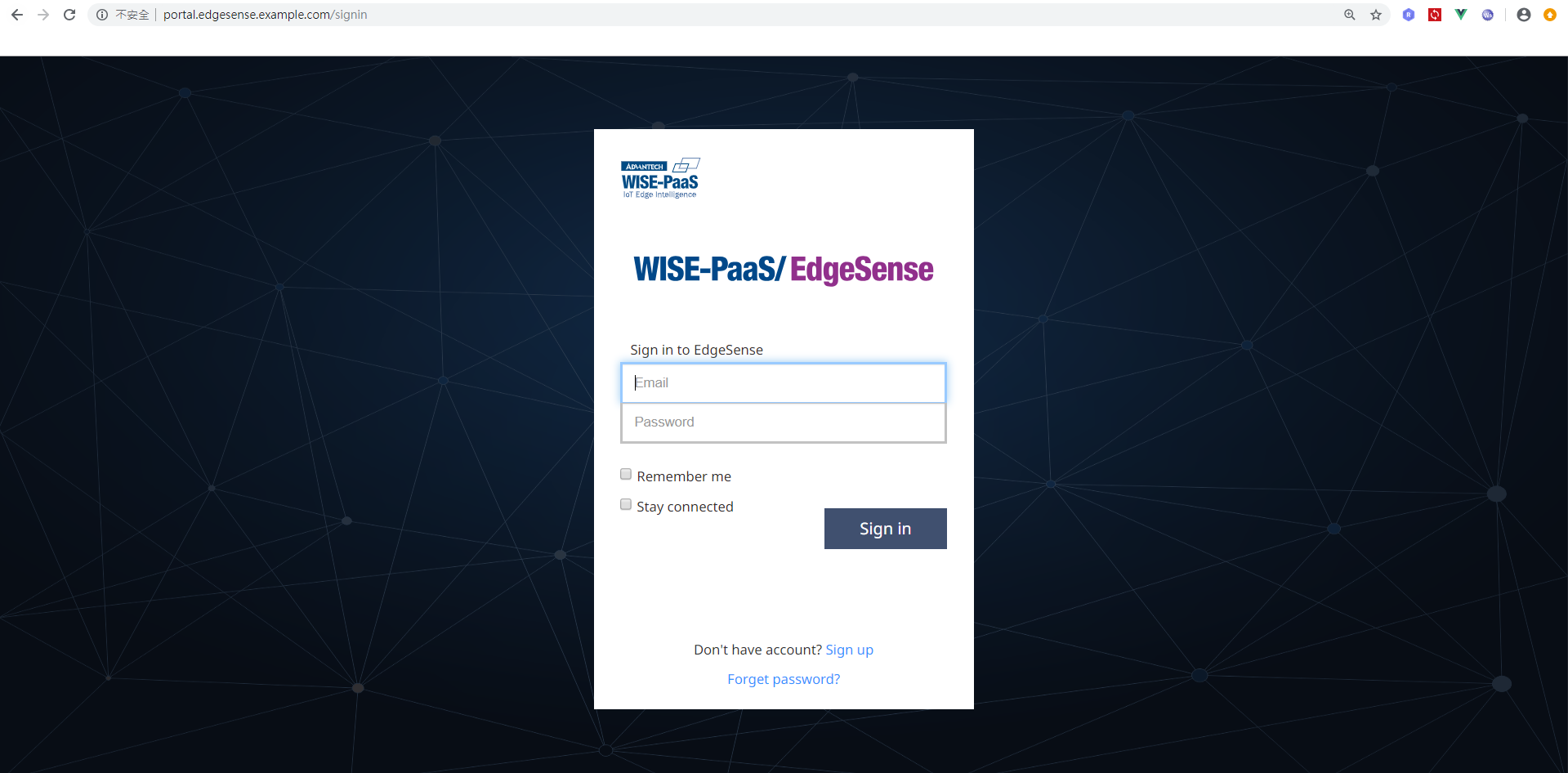
Summary: Access Methods

Here’re the summary for both access methods: IP and DNS. After having the ports or endpoints of the services, you could easily access them by typing the ports and endpoints in the browser. For endpoints, the examples here are based on domain of .example.com. In your case, the domain might be different. You should get your endpoints by the previous steps. To get more info about these services, you could refer to the manuals for EdgeSense, DeviceOn and monitoring system. Please make sure what applications your Data Service Server includes.

EdgeSense

Endpoint — <http://portal.edgesense.example.com>

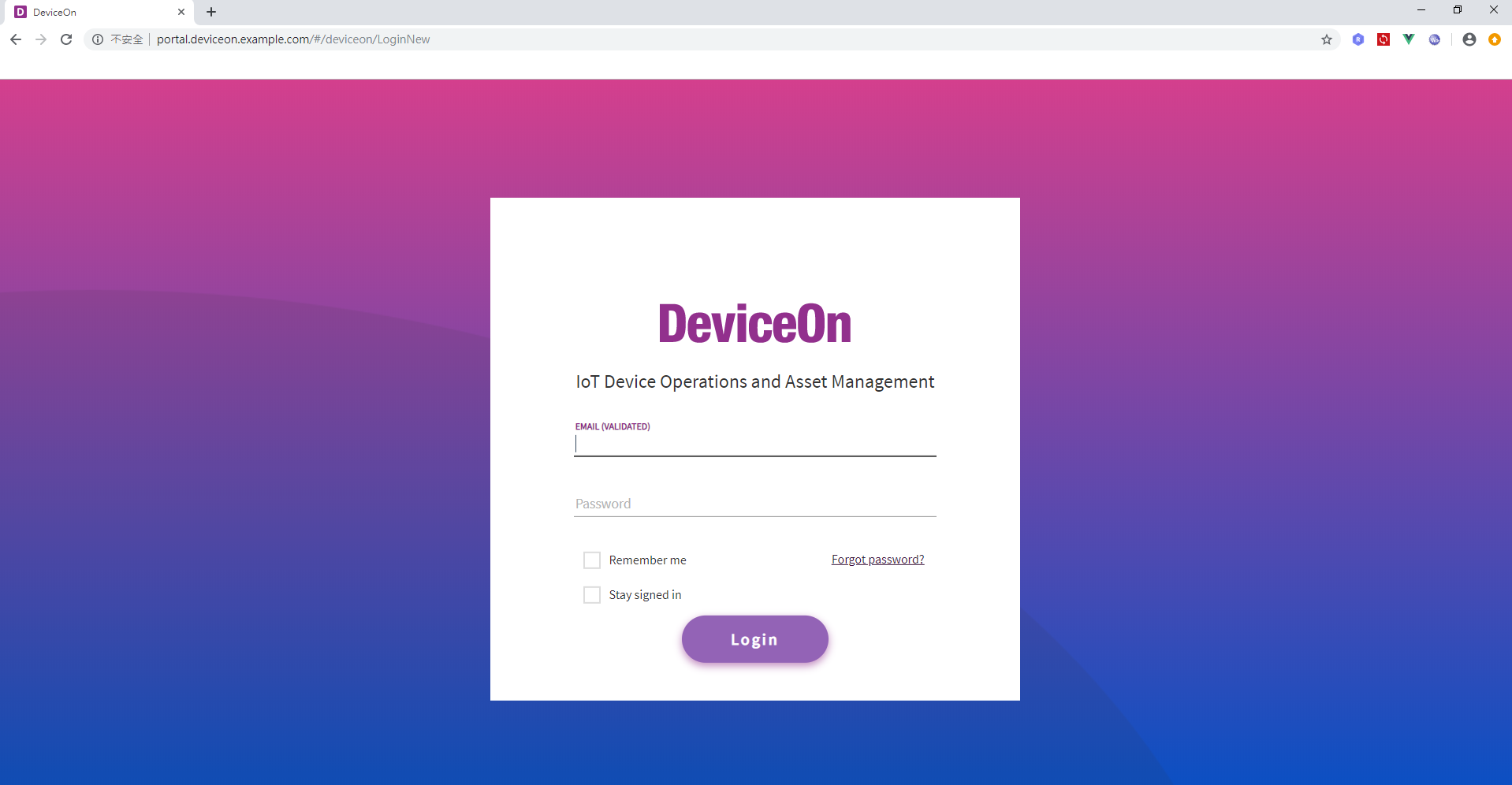
IP Access — **http://<DataServer-IP>:8087**



DeviceOn

Endpoint — <http://portal.deviceon.example.com>

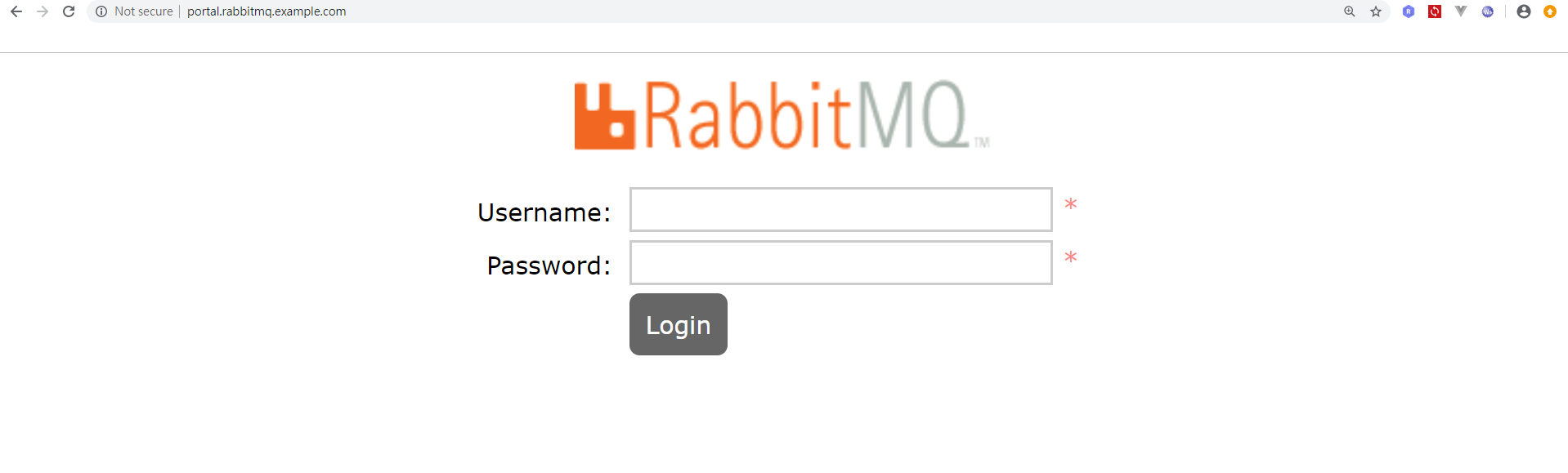
IP Access — **http://<DataServer-IP>:8087**



RabbitMQ

Endpoint — <http://portal.rabbitmq.example.com>

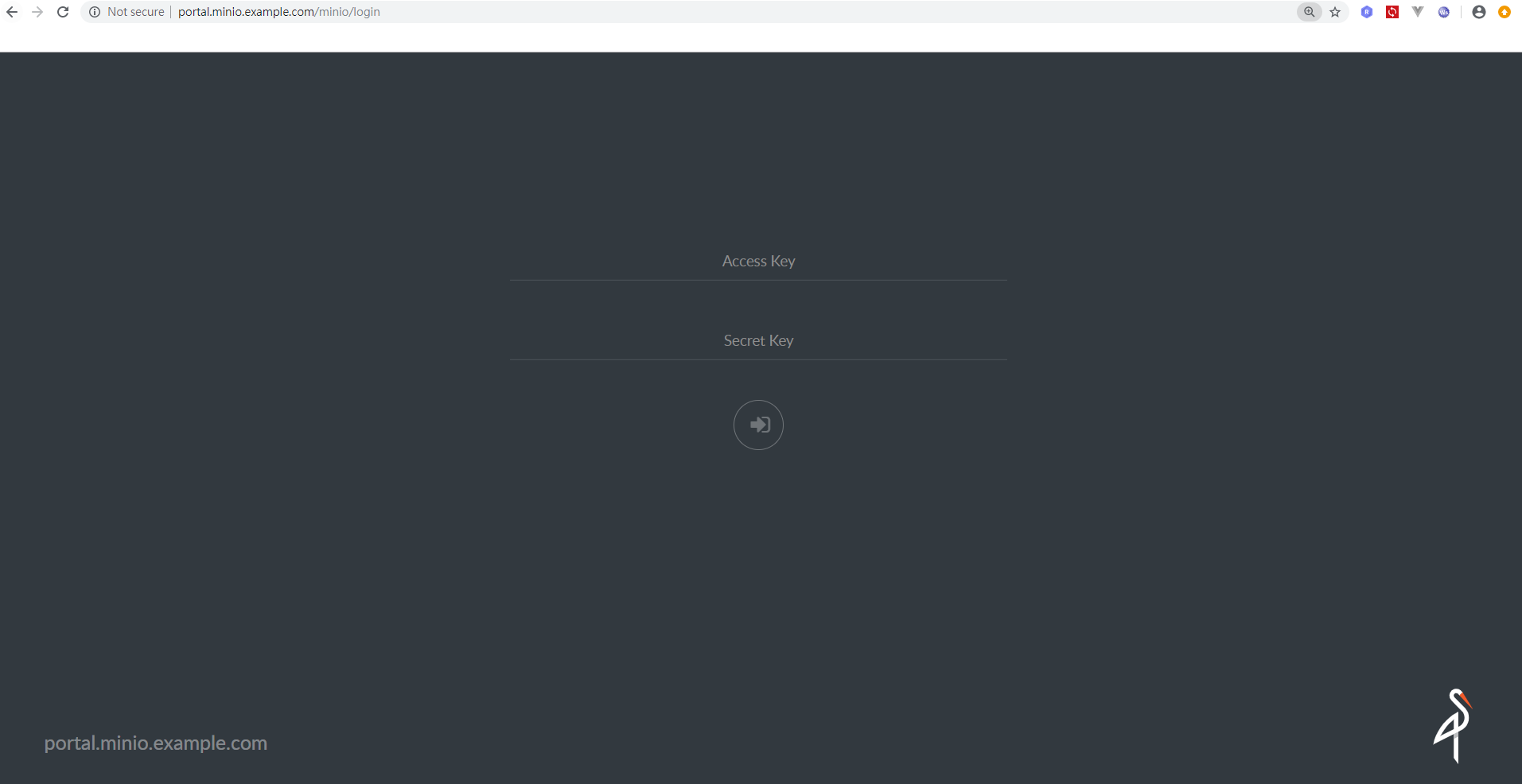
IP Access — **http://<DataServer-IP>:15672**



Minio

Endpoint — <http://portal.minio.example.com>

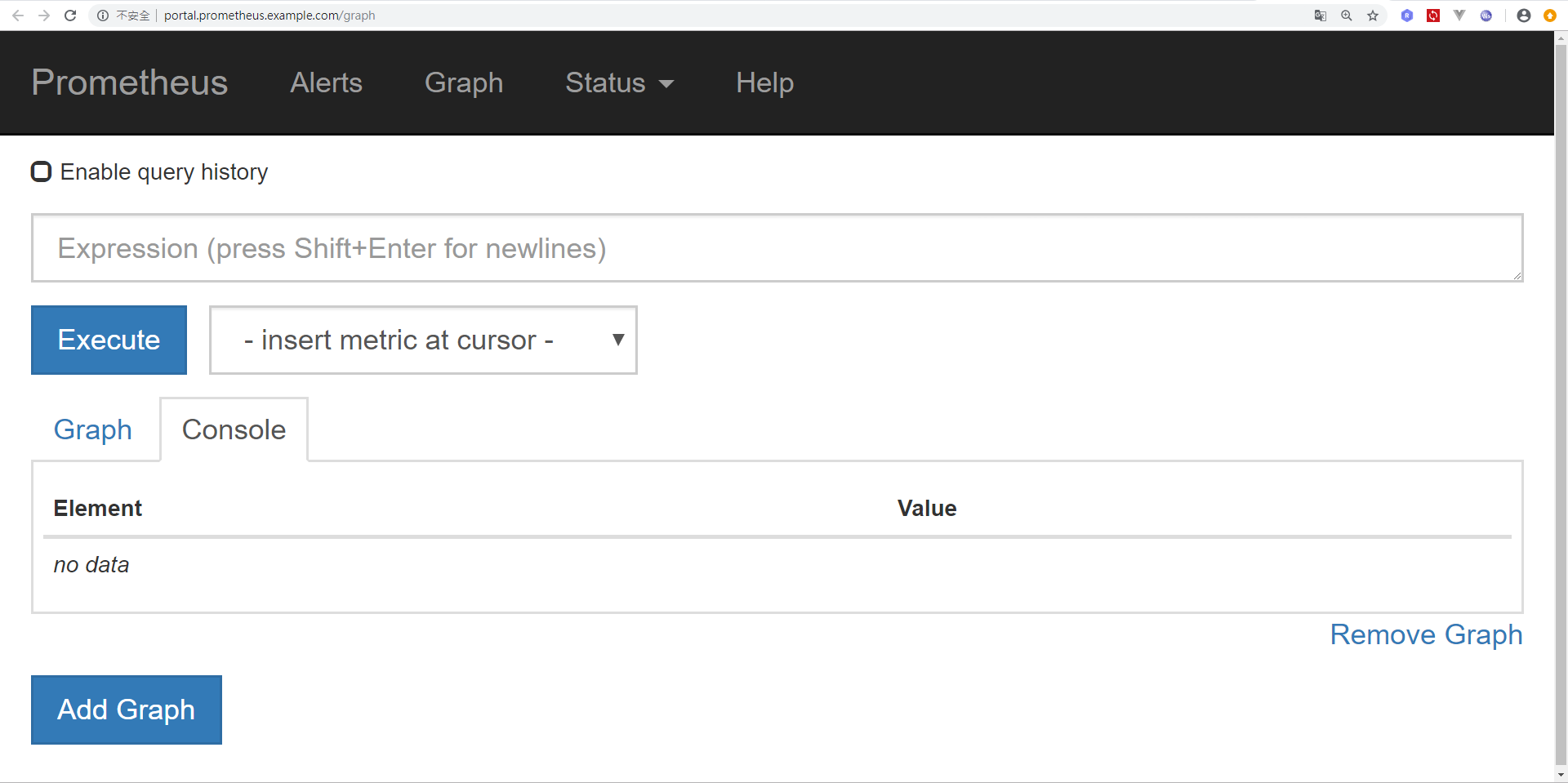
IP Access — **http://<DataServer-IP>:9000**



Prometheus

Endpoint — <http://portal.prometheus.example.com>

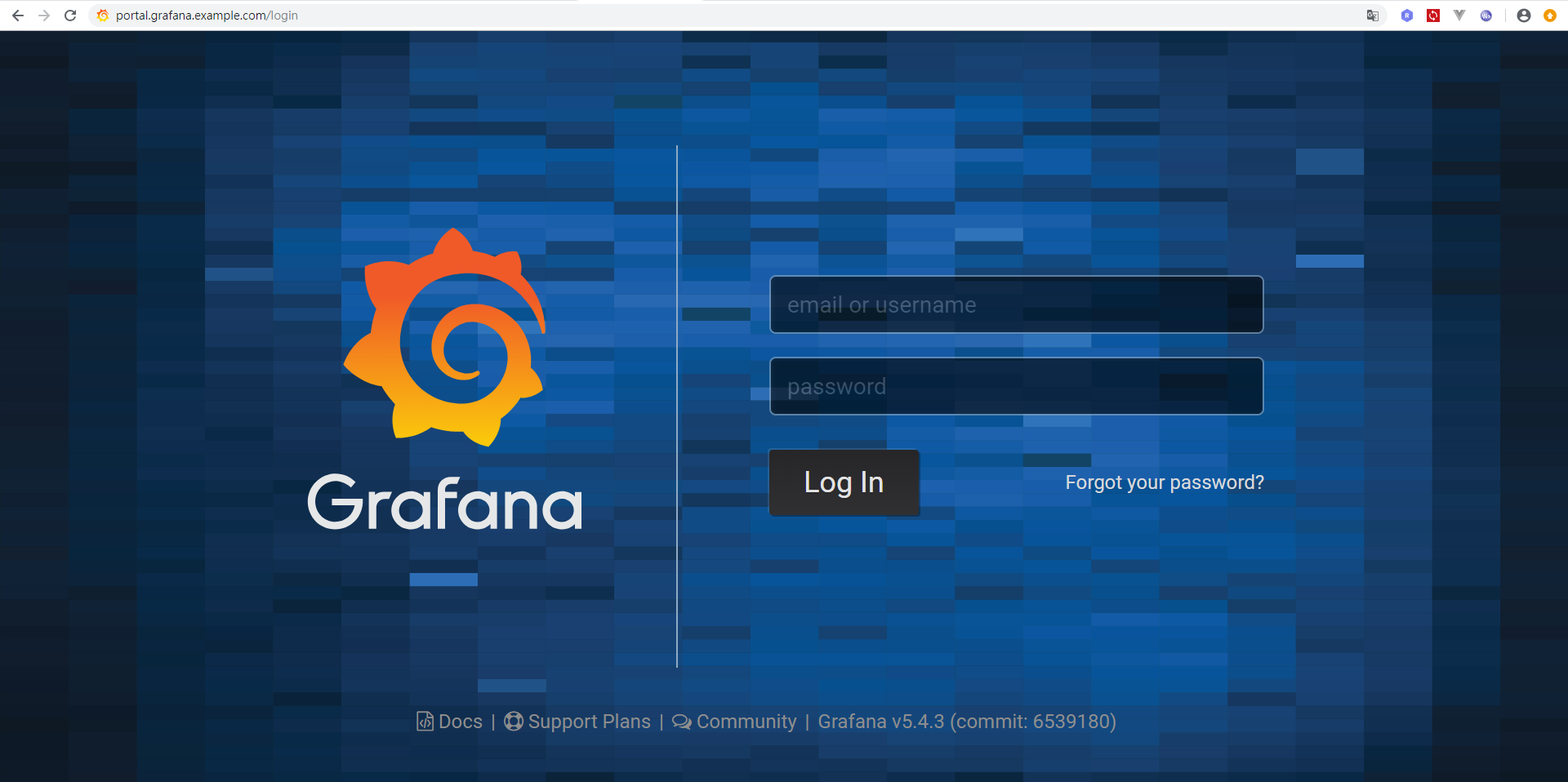
IP Access — **http://<DataServer-IP>:9090**



Grafana

Endpoint — <http://portal.grafana.example.com>

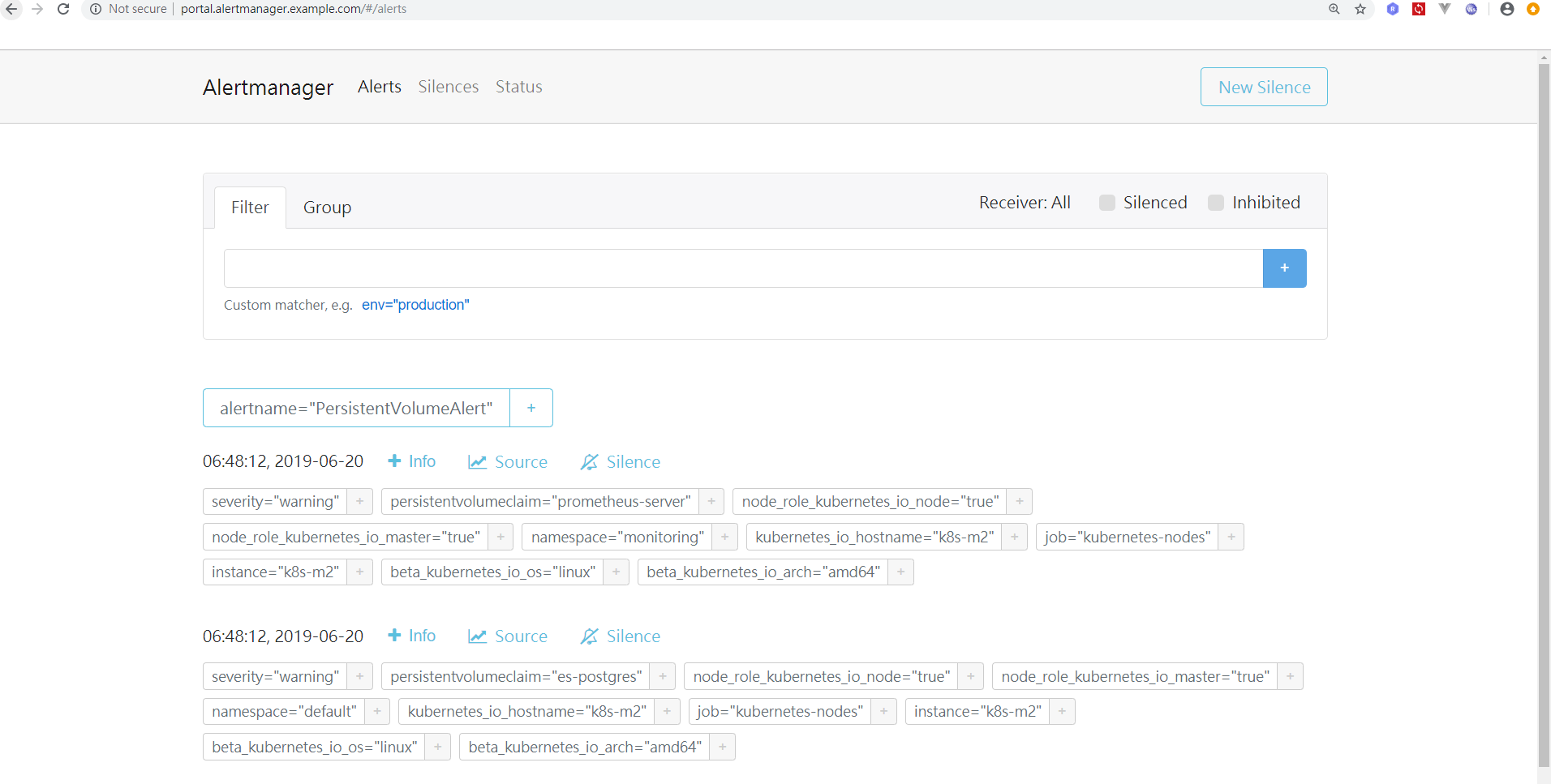
IP Access — **http://<DataServer-IP>:3000**



AlertManager

Endpoint — <http://portal.alertmanager.example.com>

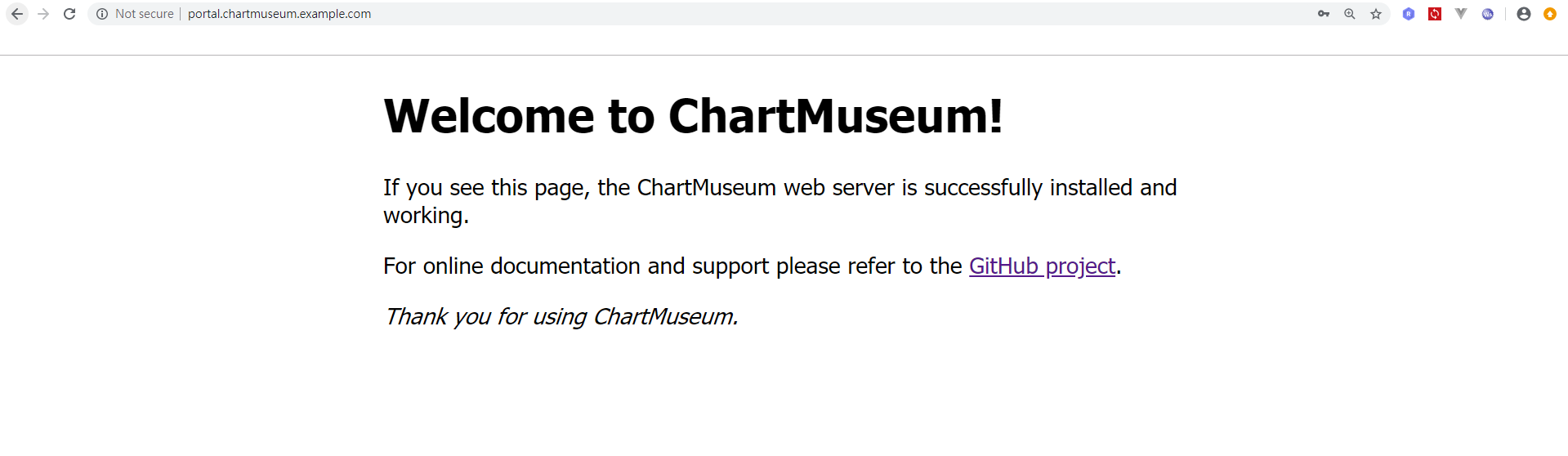
IP Access — **http://<DataServer-IP>:9093**



ChartMuseum

Endpoint — <http://portal.chartmuseum.example.com>

IP Access — **http://<DataServer-IP>:8081**



1. Kubernetes Dashboard

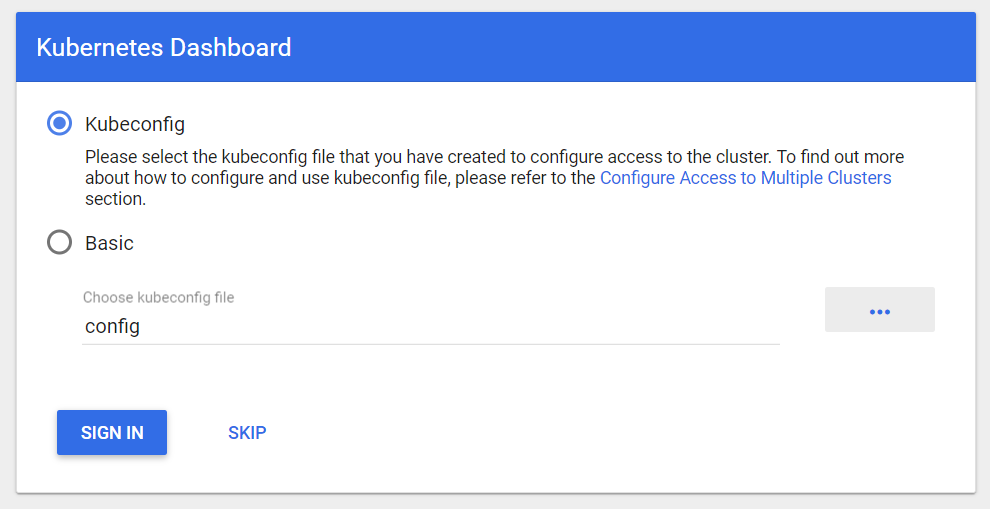
Kubernetes Dashboard is an overview for management of the whole cluster. You could view all the resources on the Data Serivce Server.

Login

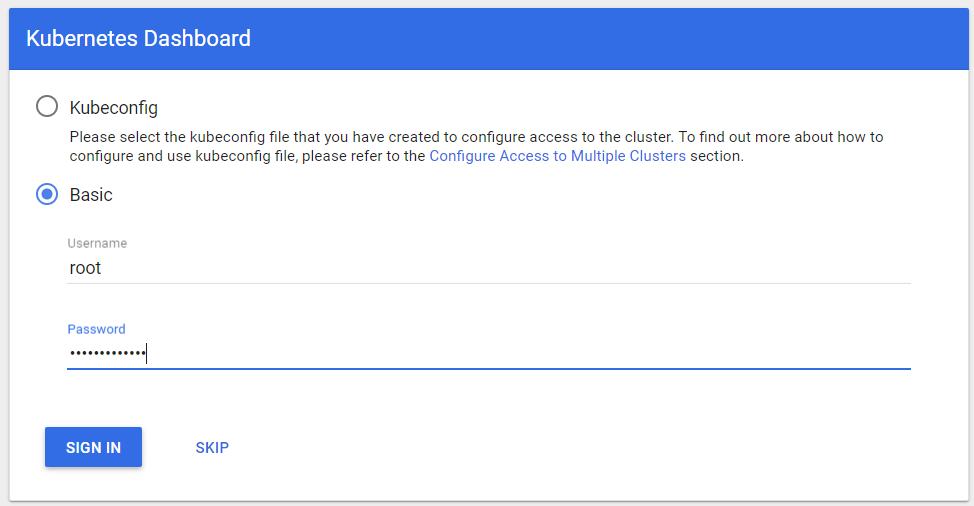
To access the dashboard, use the URL as below:

**https://<DataServer-IP>:6443/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/#!/login**

Then you could use Kubeconfig to choose the config file in **$HOME/.kube** to login.

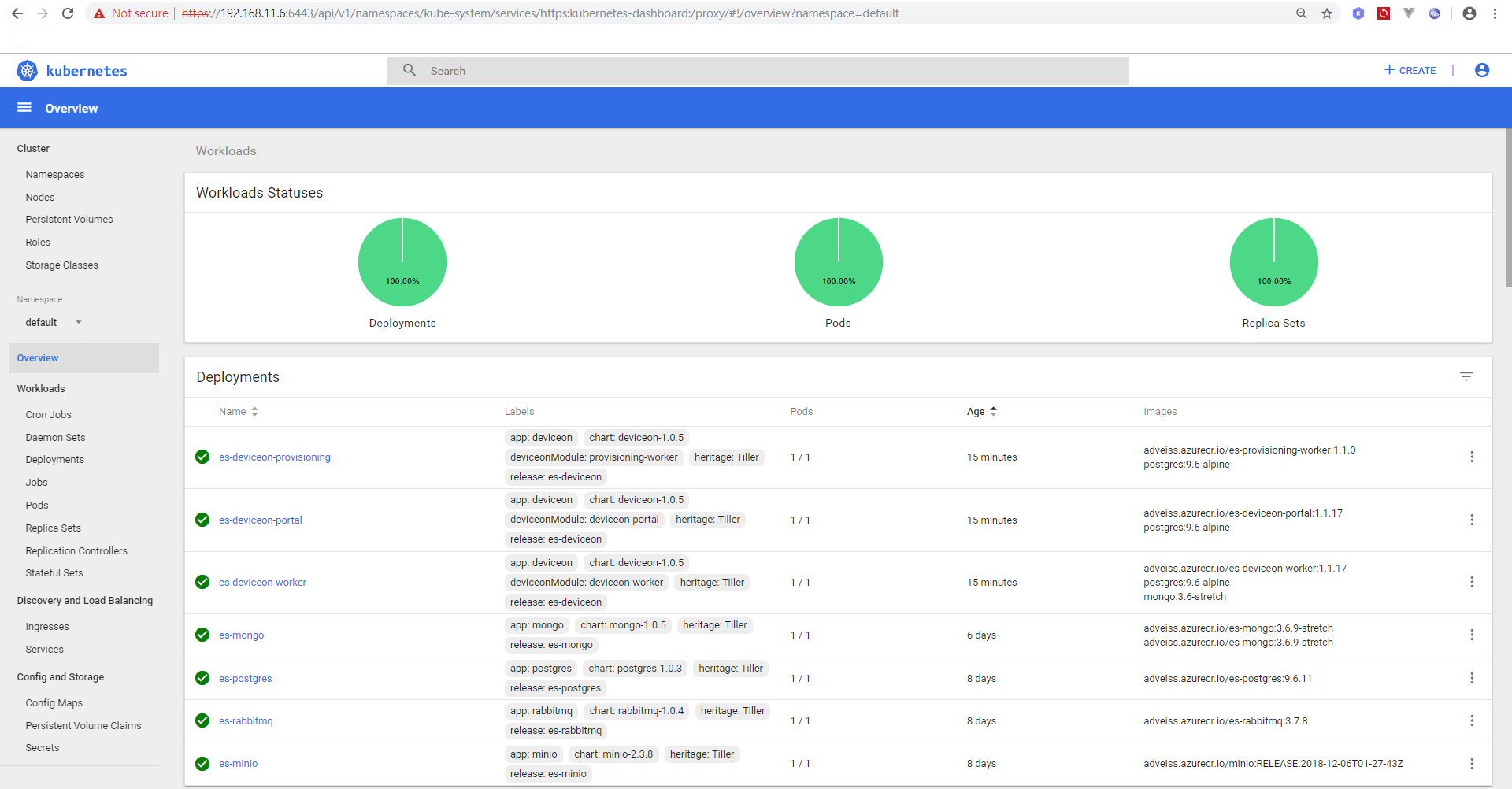


Or you could choose Basic method and type in username/password to login: **root/r00tadvant1cH**



Viewing

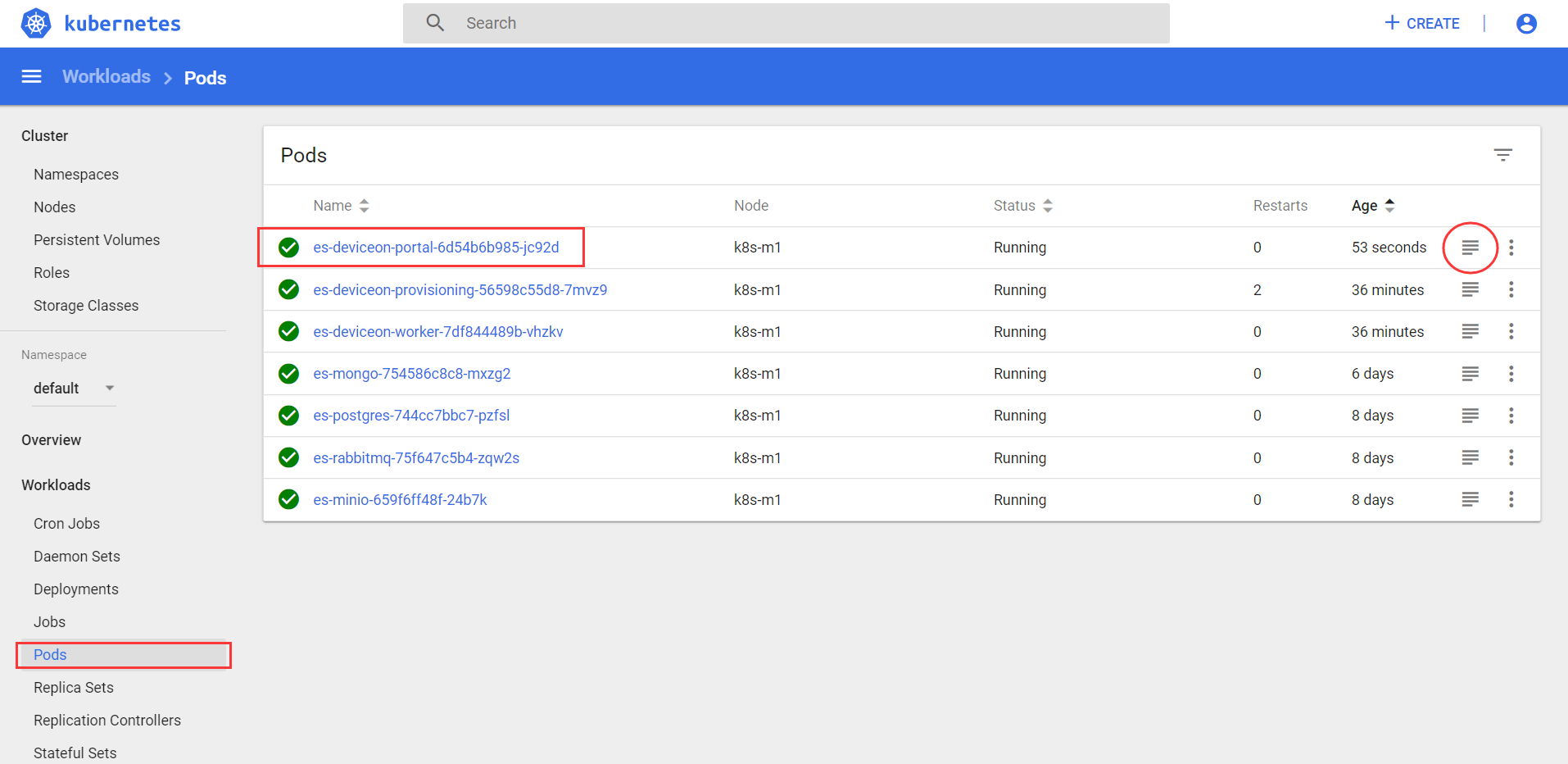
After you login the dashboard, you could explore all the resources on the Data Service Server.

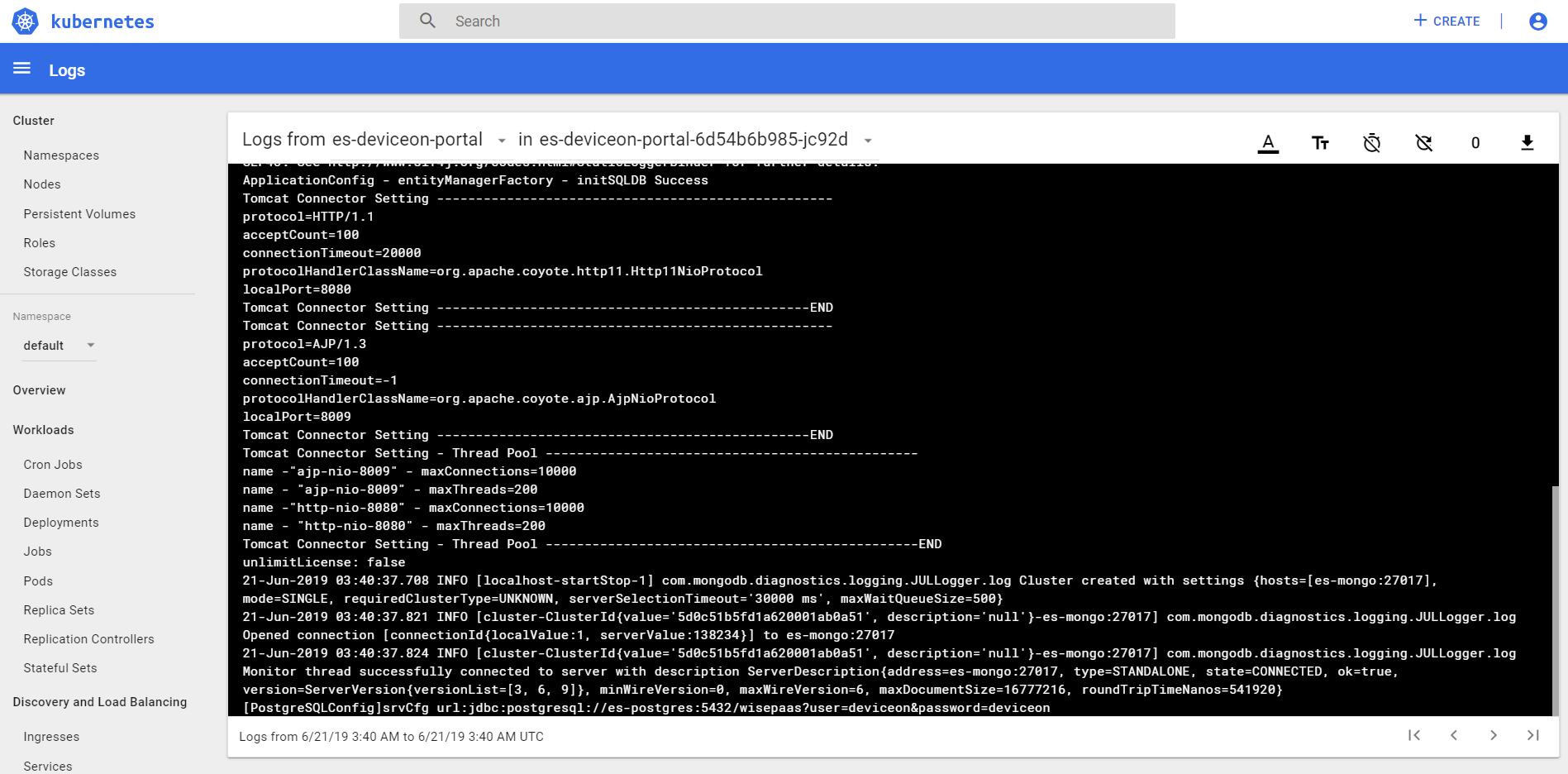


Logging

You could go to the Pods page and select the pod you want to get the logs.

The following takes deviceon portal as the example.





1. Charts

Charts are packages of pre-configured Kubernetes resources. [12] Data Service Server takes all applications as charts by using helm, which is a package manager for kubernetes. The chapter would show you how to view the charts on the Data Service Server. Before diving deeper into it, you should make sure the authentication parts in section 6.1.1/6.1.2/6.1.3 are all complete because you need to be authenticated to do your management.

Installing Helm

Helm was installed in Data Service Server by default. If you want manage the resources from your own PC then you should continue or you could skip this section. Helm is a tool for managing Kubernetes charts. Charts are packages of pre-configured Kubernetes resources. [12] Helm uses a packaging format called charts. A chart is a collection of files that describe a related set of Kubernetes resources. A single chart might be used to deploy something simple, like a memcached pod, or something complex, like a full web app stack with HTTP servers, databases, caches, and so on. [13] Data Service Server takes all applications as charts by using helm. Please follow the instructions to install helm. [14]

Use Helm to: [12]

* Find and use popular software packaged as Helm charts to run in Kubernetes
* Share your own applications as Helm charts
* Create reproducible builds of your Kubernetes applications
* Intelligently manage your Kubernetes manifest files
* Manage releases of Helm packages

Viewing Charts

By using the command below to check the installed charts:

# list all of the releases

$ helm list

1. Disaster Recovery

Running services for a long period of time, unfortunately, you might encounter some disasters. We all like to live in a perfect world. However, things might not be on the right way as we expect they should be so we should be always prepared. Data Service Server has some ways for you to restore your applications when disasters come up.

Restoration

Restoring Applications

If some applications are not running properly, you could restore them from a directory (/etc/dss/charts) or from chartmuseum, the private chart repository. Before you start to restore an abnormal application, please use the uninstall scripts in each directories. Let’s take deviceon as the abnormal example for our restoration.

If the applications belonging to deviceon are not working properly, which might be worker, portal or provisioning, you could use the commands below to restore them.

***# Delete a chart of DeviceOn***

$ cd /etc/dss/charts/DeviceOn

$ ./uninstall.sh

***# Install a chart of DeviceOn from the directory***

$ cd /etc/dss/charts/DeviceOn

$ ./install.sh

***# Install a chart of DeviceOn from private chart repo***

$ cd /etc/dss/charts/DeviceOn

$ ./installFromRepo.sh

1. Precautions

Shutdown/Reboot

Since suddenly shutdown might cause Prometheus server or ceph crashes, user needs to execute shutdown utility before shutdown and reboot. Although the crash rarely happens, executing shutdown utility is to play it safe.

1. Execute shutdown script by the following commands.

***# Execute the shutdown script to make sure promethus & ceph are closed normally***

$ cd /root/scripts/shutdown-bootup

$ sh –x k8s-shutdown.sh

***WARNING: Please execute the script before you shutdown or reboot the Data Service Server. Otherwise, take the responsibility of any possible outcome.***

1. Knowledge

Basic Knowledge of Resources of K8s

Before moving on, we need to have some basic knowledge of the resources of k8s. Here’re some main resources you are going to see.

Pods

A Pod is the basic building block of Kubernetes–the smallest and simplest unit in the Kubernetes object model that you create or deploy. A Pod represents a running process on your cluster. [15]

A Pod encapsulates an application container (or, in some cases, multiple containers), storage resources, a unique network IP, and options that govern how the container(s) should run. A Pod represents a unit of deployment: a single instance of an application in Kubernetes, which might consist of either a single container or a small number of containers that are tightly coupled and that share resources. [15]

To specify a memory limit, include **resources:limits**

In this exercise, you create a Pod that has one Container. The Container has a memory request of 100 MiB and a memory limit of 200 MiB. Here’s the configuration file for the Pod: [15]

**apiVersion: v1**

**kind: Pod**

**metadata:**

**name: memory-demo**

**namespace: mem-example**

**spec:**

**containers:**

**- name: memory-demo-ctr**

**image: polinux/stress**

**resources:**

**limits:**

**memory: "200Mi"**

**requests:**

**memory: "100Mi"**

**command: ["stress"]**

**args: ["--vm", "1", "--vm-bytes", "150M", "--vm-hang", "1"]**

Services

A Kubernetes Service is an abstraction which defines a logical set of Pods and a policy by which to access them - sometimes called a micro-service. [16] By using Kubernetes Service, you could define the traffic routes to the backend services on your pods.

For example, suppose you have a set of Pods that each expose port 9376 and carry a label "app=MyApp".

**apiVersion: v1**

**kind: Service**

**metadata:**

**name: my-service**

**spec:**

**selector:**

**app: MyApp**

**ports:**

**- protocol: TCP**

**port: 80**

**targetPort: 9376**

ReplicaSets

A ReplicaSet’s purpose is to maintain a stable set of replica Pods running at any given time. As such, it is often used to guarantee the availability of a specified number of identical Pods. It ensures that a specified number of pod replicas are running at any given time. However, a Deployment is a higher-level concept that manages ReplicaSets and provides declarative updates to Pods along with a lot of other useful features. [17]

Deployments

A Deployment controller provides declarative updates for [Pods](https://kubernetes.io/docs/concepts/workloads/pods/pod/) and [ReplicaSets](https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/). [18]

The following is an example of a Deployment. It creates a ReplicaSet to bring up three nginx Pods: [18]

**apiVersion: apps/v1**

**kind: Deployment**

**metadata:**

**name: nginx-deployment**

**labels:**

**app: nginx**

**spec:**

**replicas: 3**

**selector:**

**matchLabels:**

**app: nginx**

**template:**

**metadata:**

**labels:**

**app: nginx**

**spec:**

**containers:**

**- name: nginx**

**image: nginx:1.7.9**

**ports:**

**- containerPort: 80**

Secrets

Kubernetes secret objects let you store and manage sensitive information, such as passwords, OAuth tokens, and ssh keys. Putting this information in a secret is safer and more flexible than putting it verbatim in a Pod definition or in a container image. [19]

A Secret is an object that contains a small amount of sensitive data such as a password, a token, or a key. Such information might otherwise be put in a Pod specification or in an image; putting it in a Secret object allows for more control over how it is used, and reduces the risk of accidental exposure. [19]

For example, to store two strings in a Secret using the data field, convert them to base64 as follows: [19]

**echo -n 'admin' | base64**

**YWRtaW4=**

**echo -n '1f2d1e2e67df' | base64**

**MWYyZDFlMmU2N2Rm**

**apiVersion: v1**

**kind: Secret**

**metadata:**

**name: mysecret**

**type: Opaque**

**data:**

**username: YWRtaW4=**

**password: MWYyZDFlMmU2N2Rm**

ConfigMaps

ConfigMaps allow you to decouple configuration artifacts from image content to keep containerized applications portable. [20]

The ConfigMap API resource holds key-value pairs of configuration data that can be consumed in pods or used to store configuration data for system components such as controllers. ConfigMap is similar to Secrets, but designed to more conveniently support working with strings that do not contain sensitive information. [21]

kind: ConfigMap

apiVersion: v1

metadata:

creationTimestamp: 2016-02-8T19:14:8Z

name: example-config

namespace: default

data:

example.property.1: hello

example.property.2: world

example.property.file: |-

property.1=value-1

property.2=value-2

property.3=value-3

Commands of kubectl

This section would introduce some basic functions of kubectl. There’re more commands you could find out. [5]

**$ kubectl [command] [TYPE] [NAME] [flags]**

where **command**, **TYPE**, **NAME**, and **flags** are:

**command** — Specifies the operation that you want to perform on one or more resources, for example **create**, **get**, **describe**, **delete**.

**TYPE** — Specifies the [resource type](https://kubernetes.io/docs/reference/kubectl/overview/#resource-types). Resource types are case-insensitive and you can specify the singular, plural, or abbreviated forms. For example, the following commands produce the same output:

**NAME** — Specifies the name of the resource. Names are case-sensitive. If the name is omitted, details for all resources are displayed, for example **$ kubectl get pods**.

However, the authorized actions for the viewer in the config only include the following.

**$ kubectl get pod pod1**

**$ kubectl get pods pod1**

**$ kubectl get po pod1**

GET

**get** — List one or more resources. For more info, visit the website. [5]

***# List all resources.***

$ kubectl get all –n <namespace>

***# List all pods in plain-text output format.***

**$ kubectl get pods**

***# List all pods in plain-text output format and include additional information (such as node name).***

**$ kubectl get pods -o wide**

***# Get details of a resource and output it as a yaml***

**$ kubectl get <resource> <name> -o yaml**

**$ kubectl get pod <name> -o yaml**

If no specified namespace, the namespace default is in.

DESCRIBE

**describe** — Display detailed state of one or more resources, including the uninitialized ones by default. For more info, visit the website. [5]

***# Display the details of the node with name <node-name>.***

$ kubectl describe nodes <node-name>

***# Display the details of the pod with name <pod-name>.***

**$ kubectl describe pods/<pod-name>**

***# Display the details of all the pods that are managed by the replication controller named <rc-name>.***

***# Remember: Any pods that are created by the replication controller get prefixed with the name of the replication controller.***

**$ kubectl describe pods <rc-name>**

***# Describe all pods, not including uninitialized ones***

**$ kubectl describe pods --include-uninitialized=false**

LOGS

**logs** — Print the logs for a container in a pod. For more info, visit the website. [5]

***# Return a snapshot of the logs from pod <pod-name>.***

**$ kubectl logs <pod-name>**

***# Start streaming the logs from pod <pod-name>. This is similar to the 'tail -f' Linux command.***

**$ kubectl logs -f <pod-name>**

DELETE

**delete** — Delete resources either from a file, stdin, or specifying label selectors, names, resource selectors, or resources. [5]

***# Delete a pod with name <pod-name>.***

**$ kubectl delete pod/<pod-name>**

***# Delete a pod with name <pod-name> in <namespace>.***

**$ kubectl delete pod/<pod-name> -n <namespace>**

***WARNING: This action should be carefully taken because it’s not reversible. Take the responsibility of your own action.***

EDIT

**edit** — Edit and update the definition of one or more resources on the server by using the default editor. [5]

***# Edit a service with name <service-name>.***

**$ kubectl edit service/<service-name>**

***# Edit a deployment with name <deployment-name>.***

**$ kubectl edit deployment.apps/<deployment-name>**

***WARNING: This action should be carefully taken. Take the responsibility of your own action.***

Basic Knowledge of Helm

Purposes

Helm is a tool for managing Kubernetes packages called *charts*. Helm can do the following: [22]

* Create new charts from scratch
* Package charts into chart archive (tgz) files
* Interact with chart repositories where charts are stored
* Install and uninstall charts into an existing Kubernetes cluster
* Manage the release cycle of charts that have been installed with Helm

For Helm, there are three important concepts:

1. The *chart* is a bundle of information necessary to create an instance of a Kubernetes application.
2. The *config* contains configuration information that can be merged into a packaged chart to create a releasable object.
3. A *release* is a running instance of a *chart*, combined with a specific *config*.

Components

Helm has two major components: [22]

**The Helm Client** is a command-line client for end users. The client is responsible for the following domains:

* Local chart development
* Managing repositories
* Interacting with the Tiller server
  + Sending charts to be installed
  + Asking for information about releases
  + Requesting upgrading or uninstalling of existing releases

**The Tiller Server** is an in-cluster server that interacts with the Helm client, and interfaces with the Kubernetes API server. The server is responsible for the following:

* Listening for incoming requests from the Helm client
* Combining a chart and configuration to build a release
* Installing charts into Kubernetes, and then tracking the subsequent release
* Upgrading and uninstalling charts by interacting with Kubernetes

In a nutshell, the client is responsible for managing charts, and the server is responsible for managing releases.

Commands

Here’re some the useful commands you might use to control your charts. For more info, you could find them on the website of helm. [23]

* + - 1. INIT

This will install Tiller to your running Kubernetes cluster. It will also set up any necessary local configuration.

# install Tiller to your running Kubernetes cluster

**$ helm init**

* + - 1. INSTALL

This command installs a chart archive. The install argument must be a chart reference, a path to a packaged chart, a path to an unpacked chart directory or a URL.

There are five different ways you can express the chart you want to install:

1. By chart reference: helm install stable/mariadb
2. By path to a packaged chart: helm install ./nginx-1.2.3.tgz
3. By path to an unpacked chart directory: helm install ./nginx
4. By absolute URL: helm install <https://example.com/charts/nginx-1.2.3.tgz>
5. By chart reference and repo url: helm install –repo <https://example.com/charts/> nginx

# install a chart archive

**$ helm install [CHART] [flags]**

* + - 1. LIST

This command lists all of the releases.

# list all of the releases

**$ helm list**

* + - 1. DELETE

This command takes a release name, and then deletes the release from Kubernetes. It removes all of the resources associated with the last release of the chart.

# delete a chart

**$ helm delete [NAME] --purge**

***WARNING: This action should be carefully taken because it’s not reversible. Take the responsibility of your own action.***

* + - 1. REPO

This command consists of multiple subcommands to interact with chart repositories.

It can be used to add, remove and list chart repositories.

# add repo

$ helm repo add [flags] [NAME] [URL]

# list repo

$ helm repo list

# update repo

$ helm repo update

# remove repo

$ helm repo remove [NAME]

1. FAQ

Accessing

Why can’t I use kubectl to access the resources on Data Service Server?

First, you should make sure the Data Service Server is accessible in your network by using ping or whatever tools. If it’s accessible, the problem might be the configuration for authentication. (13.2)

Why can’t I access the services by the endpoints in ingresses?

First, you should make sure the Data Service Server is accessible in your network by using ping or whatever tools. Second, use the commands kubectl get, describe or logs (12.2) to check whether the networking services in namespace network are healthy or not. If the networking services are healthy, go check health condition of the pod of the service you want to access.

How should I set the URLs for dashboard and S3 compatible storage in DeviceOn/EdgeSense?

You could set the IP with ports for both of them. **<DataServer-IP>:3000** for dashboard and **<DataServer-IP>:9000** for S3 compatible storage. You shouldn’t set the URLs in ingresses because they are serving for the public not for internal applications.

Authentication

Why do I keep getting an error message “error: You must be logged in to the server (Unauthorized)”when I use kubectl?

The error message might be from the wrong configuration of username/password in the config. The username/password for the viewer is **viewer/viewer.** You can find the instruction in **Setting the Configuration for Authentication** (6.1.2).

Why do I keep get an error message “Unable to connect to the server: dial tcp [::1]:8080: connectex: No connection could be made because the target machine actively refused it.”when I use kubectl?

The error message might be from the wrong path or wrong configuration of the config. You can find the instruction in **Setting the Configuration for Authentication** (6.1.2).

Unhealthy Pods

Why do I get the statuses of pods which are not *Running* but *Error* or others?

The healthy status for a pod is *Running.* If you get a status other than *Running*, in which case it might be *Error* or whatever, it might means the pod is unhealthy. An unhealthy pod could make your services unavailable or not work normally.

How can I do to those unhealthy pods?

Normally, the kubernetes would restart unhealthy pods to try to make them healthy again. However, if you keep getting unhealthy statuses, please try to make an approach to the supports who have more authorized actions which might come in handy to solve your problems.

Environment Change

Could I change the IP address of the Data Service Server?

The answer to this is No. The IP setting is one of the core parts for most of the servers and Data Service Server is no exception. Changing the IP address might cause many unexpected problems. If you do need to change the IP address, please contact the technical support.

1. Appendix

Ports

Here’s the list of ports of all applications. You could access these applications by using **<DataServer-IP>:<Port>**

|  |  |
| --- | --- |
| **Applications** | **Port** |
| **EdgeSensePortal/DeviceOnPortal** | 8087 |
| **RabbitMQ** | 15672 (Portal) |
| 1883 (MQTT) |
| 8883 (MQTT\_SSL) |
| **Postgres** | 5432 |
| **Mongo** | 27017 |
| **Minio** | 9000 |
| **Prometheus** | 9090 |
| **AlertManager** | 9093 |
| **Grafana** | 3000 |
| **ChartMuseum** | 8081 |

Credentials

Here’re the lists of credentials. You could access the applications by any tools as your want.

|  |  |  |
| --- | --- | --- |
| **Applications** | **Username** | **Password** |
| **OS (Superuser)** | root | r00tadvant1cH |
| **Kubernetes Dashboard** | root | r00tadvant1cH |
| **EdgeSensePortal/DerviceOnPortal** | root@advantech.com.tw | root |
| **RabbitMQ** | admin | advant1cH |
| **Postgres** | postgres | advant1cH |
| **Mongo** | admin | advant1cH |
| **Minio** | admin | advant1cH |
| **Grafana** | admin | advant1cH |
| **ChartMuseum** | admin | advant1cH |

1. References

|  |  |
| --- | --- |
| [1] | kubernetes, "kubernetes," [Online]. Available: https://kubernetes.io/. |
| [2] | docker, "docker," [Online]. Available: https://en.wikipedia.org/wiki/Docker\_(software). |
| [3] | ceph, "ceph," [Online]. Available: https://en.wikipedia.org/wiki/Ceph\_(software). |
| [4] | kubernetes, "namespaces," [Online]. Available: https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/. |
| [5] | kubernetes, "kubectl," [Online]. Available: https://kubernetes.io/docs/reference/kubectl/overview/. |
| [6] | kubernetes, "install kubectl," [Online]. Available: https://kubernetes.io/docs/tasks/tools/install-kubectl/. |
| [7] | wiki, "ssh," [Online]. Available: https://en.wikipedia.org/wiki/Secure\_Shell. |
| [8] | nginx, "nginx," [Online]. Available: https://nginx.org/en/. |
| [9] | kubernetes, "nginx-ingress-controller," [Online]. Available: https://github.com/kubernetes/ingress-nginx. |
| [10] | T. Fisher, "How to Change DNS Servers in Windows," [Online]. Available: https://www.lifewire.com/how-to-change-dns-servers-in-windows-2626242. |
| [11] | kubernetes, "ingress," [Online]. Available: https://kubernetes.io/docs/concepts/services-networking/ingress/. |
| [12] | github, "helm," [Online]. Available: https://github.com/helm/helm. |
| [13] | helm, "charts," [Online]. Available: https://helm.sh/docs/developing\_charts/#charts. |
| [14] | helm, "helm," [Online]. Available: https://helm.sh/docs/using\_helm/#installing-helm. |
| [15] | kubernetes, "pods," [Online]. Available: https://kubernetes.io/docs/concepts/workloads/pods/. |
| [16] | kubernetes, "services," [Online]. Available: https://kubernetes.io/docs/concepts/services-networking/service/. |
| [17] | kubernetes, "replicaset," [Online]. Available: https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/. |
| [18] | kubernetes, "deployments," [Online]. Available: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/. |
| [19] | kubernetes, "secrets," [Online]. Available: https://kubernetes.io/docs/concepts/configuration/secret/. |
| [20] | kubernetes, "configmaps," [Online]. Available: https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-configmap/#before-you-begin. |
| [21] | u. kubernetes, "configmaps," [Online]. Available: https://unofficial-kubernetes.readthedocs.io/en/latest/tasks/configure-pod-container/configmap/. |
| [22] | helm, "architecture," [Online]. Available: https://helm.sh/docs/architecture/. |
| [23] | helm, "commands," [Online]. Available: https://helm.sh/docs/helm/#helm. |
| [24] | helm, "chart\_template," [Online]. Available: https://helm.sh/docs/chart\_template\_guide/#getting-started-with-a-chart-template. |
| [25] | wiki, "docker," [Online]. Available: https://en.wikipedia.org/wiki/Docker\_(software). |
| [26] | docker, "curriculum," [Online]. Available: https://docker-curriculum.com/. |
| [27] | chartmuseum, "push," [Online]. Available: https://github.com/chartmuseum/helm-push. |