VoltDB Kubernetes Administrator's Guide

Abstract

This book explains how to create and manage VoltDB database clusters using Kubernetes.

V10.0

VoltDB Kubernetes Administrator's Guide

V10.0

This document was generated on September 04, 2020.

Table of Contents

List of Figures

List of Tables

Preface

This book describes using Kubernetes and associated products to create and manage VoltDB databases and the clusters that host them. It is intended for database administrators and operators, responsible for the ongoing management and maintenance of database infrastructure in a containerized environment.

This book is *not* a tutorial on Kubernetes or VoltDB. Please see Section 2, “Related Documents” below for documents that can help you familiarize yourself with these topics.

1. Structure of This Book

This book is divided into 3 chapters and 1 appendix:

- Chapter 1, *Introduction*
- Chapter 2, *Configuring the VoltDB Database Cluster*
- Chapter 3, *Managing VoltDB Databases in Kubernetes*
- Appendix A, *VoltDB Helm Properties*

2. Related Documents

This book assumes a working knowledge of Kubernetes, VoltDB, and the other technologies used in a containerized environment (specifically Docker and Helm). For information on developing and managing VoltDB databases, please see the manuals *Using VoltDB* and *VoltDB Administrator's Guide*. For new users, see the *VoltDB Tutorial*. For introductory information on the other products, please see their respective websites for appropriate documentation:

- Docker
- Helm
- Kubernetes

Finally, this book and all other documentation associated with VoltDB can be found on the web at http://docs.voltdb.com/.

Chapter 1. Introduction

Kubernetes is an environment for hosting virtualized applications and services run in containers. It is designed to automate the management of distributed applications, with a particular focus on microservices. VoltDB is not a microservice — there is coordination between the nodes of a VoltDB cluster that requires additional attention. So although it is possible to spin up a generic set of Kubernetes "pods" to run a VoltDB database, additional infrastructure is necessary to realize the full potential of Kubernetes and VoltDB working together.

VoltDB Enterprise Edition now provides additional services to simplify, automate, and unlock the power of running VoltDB within Kubernetes environments. There are six key components to the VoltDB Kubernetes offering, three available as open-source applications for establishing the necessary hosting environment and three provided by VoltDB to Enterprise customers. The three open-source products required to run VoltDB in a Kubernetes environment are:

- **Kubernetes** itself
- **Docker**, for managing the container images
- **Helm**, for automating the creation and administration of VoltDB in Kubernetes

In addition to these base requirements, VoltDB provides the following three custom components:

- **Pre-packaged docker image** for running VoltDB cluster nodes
- **The VoltDB Operator**, a separate utility (and docker image) for orchestrating the startup and management of VoltDB clusters in Kubernetes
- **Helm charts** for initializing and communicating with Kubernetes, the VoltDB Operator and its associated VoltDB cluster

The remainder of this chapter provides an overview of how these components work together to support running virtualized VoltDB clusters in a Kubernetes environment, the requirements for the host and client systems, and instructions for preparing the host environment prior to running VoltDB. Subsequent chapters provide details on configuring and starting your VoltDB cluster as well as common administrative tasks such as:

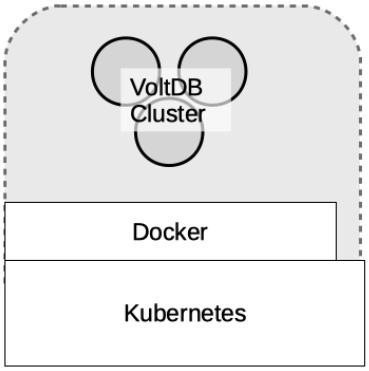
- Starting, stopping, and modifying the VoltDB cluster
- Managing the database schema and configuration

Finally, an appendix provides a full list of the Helm properties for configuring and controlling your VoltDB clusters.

1.1. Overview: Running VoltDB in Kubernetes

Kubernetes lets you create clusters of virtual machines, on which you run "pods". Each pod acts as a separate virtualized system or container. The containers are pre-defined collections of system and application components needed to run an application or service. Kubernetes provides the virtual machines, Docker defines the containers, and Kubernetes takes responsibility for starting and stopping the appropriate number of pods that your application needs.

So the basic architecture for running VoltDB is a VoltDB database running on multiple instances of a Docker container inside a Kubernetes cluster.

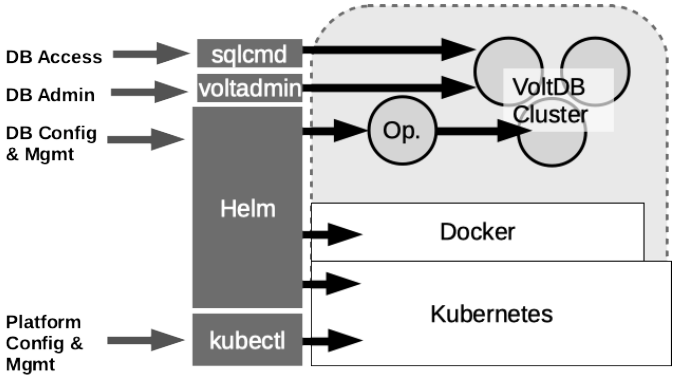


However, out of the box, VoltDB and Kubernetes do not "talk together" and so there is no agreement on when pods are started and stopped and whether a VoltDB node is active or not. To solve this problem, VoltDB provides an additional service, the VoltDB Operator that manages the interactions between the VoltDB cluster and the Kubernetes infrastructure. The Operator takes responsibility for initializing and starting the VoltDB server instances as appropriate, monitoring their health, and coordinating changes to the configuration.

To further simplify the process, VoltDB uses the open-source management product Helm to integrate Kubernetes, Docker, and VoltDB under a single interface. Helm uses "charts" to define complex management operations, such as configuring and starting the Kubernetes pods with the appropriate Docker images and then initializing and starting VoltDB on those pods. Simply by "installing" the appropriate Helm chart you can instantiate and run a VoltDB database cluster within Kubernetes using a single command.

Once the database is running, you can use standard VoltDB command line utilities to interact with and manage the database contents, such as modifying the schema or initiating manual snapshots. However, you will continue to use Helm to manage the server process and cluster on which the database runs, for activities such as stopping and starting the database. Figure 1.1, “Kubernetes/VoltDB Architecture” shows the overall architecture of using VoltDB, the VoltDB Operator, and Helm to automate running a VoltDB database within Kubernetes.

Figure 1.1. Kubernetes/VoltDB Architecture



1.2. Setting Up Your Kubernetes Environment

Before you can run VoltDB in a containerized environment, you must be sure your host systems and client are configured with the right software and permissions to support VoltDB. The following sections outline:

- What products are required on both the host environment and the local client you use to control Kubernetes and VoltDB
- How to configure the host environment and user accounts to run the VoltDB components
- How to configure your local client to control Kubernetes and the Helm charts
- How to set permissions in Kubernetes and Docker to allow access to the VoltDB components

1.2.1. Product Requirements

Before you start, you must make sure you have the correct software products and versions installed on both the host system and your local client. The host environment is the set of servers where Kubernetes is installed, whether they are systems you set up yourself or hosted by a third-party cloud service, such as the Google Cloud Platform or Microsoft Azure. The local client environment is the system, such as a desktop or laptop, you use to access the services.

The following are the software requirements for running VoltDB in Kubernetes.

Host Environment

- Kubernetes V1.16.2 through V1.17.x

Client Environment

- Kubectl V1.16 or later[1]
- Helm V3.1 or later

Optionally, you may want to install VoltDB on the client so you can use the **voltadmin** and **sqlcmd** command utilities to access the database remotely. If not, you can still use Kubectl to ssh into the hosted server instances and run the utilities directly on the Kubernetes pods.

1.2.2. Configuring the Host Environment and Accounts

Once you have the necessary software installed, you must prepare the host environment to run VoltDB. This includes adding the appropriate Docker and chart repositories to Helm and configuring your host account with the permissions necessary to access those repositories.

First, you need accounts on the Kubernetes host environment and on the docker repository where the VoltDB images are stored, https://docker.io. To run the VoltDB Helm charts, your accounts must be set up with the following permissions:

- **Your Kubernetes host account** must have sufficient permissions to allocate persistent volumes and claims and create and manage pods. [anything else?]
- **Your Docker repository account** must have permission to access the VoltDB docker images. Access to the VoltDB docker images is assigned to VoltDB Enterprise customers on a per account basis. Contact VoltDB support for more information.

1.2.3. Configuring the Client

Next you must configure your client environment so you can communicate with and control Kubernetes and the Helm charts. First, install the Kubernetes and Helm command line interfaces, **kubectl** and **helm**. Next, configure the services to access the appropriate remote accounts and repositories.

[1]Kubectl on the client must be within one minor version of Kubernetes in the host environment. For example, if Kubernetes is at version 1.17, Kubectl can be 1.16, 1.17, or 1.18. See the Kubernetes version skew documentation for further information.

The primary setup task for kubectl is creating the appropriate context for accessing the Kubernetes host you will be using. This is usually done as part of the installation or with a Kubconfig file and the **kubectl config** command. Once you have a context defined, you can use the **kubectl cluster-info** command to verify that your client is configured correctly.

For helm, you must add a link to the VoltDB docker repository, using the **helm repo add** command:

```
$ helm repo add voltdb \
     https://voltdb-kubernetes-charts.storage.googleapis.com
```

The first argument to the command ("voltdb") is a short name for referencing the repository in future commands. You can specify whatever name you like. The second argument is the location of the repository itself and must be entered as shown above.

Note

Helm first looks in local folders for charts you specify, then in the repositories. So if the short name you use matches a local directory, they van conflict and cause errors. In that case, you may want to choose a different name, such as "voltkube", to avoid any ambiguity. Then the chart locations you use in Helm commands would be "voltkube/voltdb" rather than "voltdb/voltdb" as shown in the examples.

1.2.4. Granting Kubernetes Access to the Docker Repository

Finally, you need to tell Kubernetes to access the Docker repository using the credentials for your Docker account. There are several ways to do this. You can specify your credentials on the helm command line each time you install a new VoltDB cluster. You can save the credentials in a YAML file with other parameters you pass to helm. Or you can set the credentials in a Kuerbernetes secret using kubectl.

The advantage of using a secret to store the credentials is that you only need to define them once and they are not easily discovered by others, since they are encrypted. To create a Kubernetes secret you use the **kubectl create secret** command, specifying the type of secret (*docker-registry*) and the name of the secret (which must be *dockerio-registry*), plus the individual credential elements as arguments:

```
$ kubectl create secret docker-registry dockerio-registry \
   --docker-username=johndoe          \
   --docker-password='ThisIsASecret'  \
   --docker-email="jdoe@anybody.org
```

Once you add the secret, you do not need to specify them again. If, on the other hand, you prefer to specify the credentials when you issue the helm commands to initialize the VoltDB cluster, you can supply them as the following helm properties using the methods described in Chapter 2, *Configuring the VoltDB Database Cluster*:

- `global.image.credentials.username`
- `global.image.credentials.password`

Chapter 2. Configuring the VoltDB Database Cluster

Helm simplifies the process of starting a VoltDB database cluster within Kubernetes by coordinating all the different components involved, including Kubernetes, Docker, and VoltDB. By using the provided Helm charts, it is possible to start a default VoltDB cluster with a single command:

```
$ helm install mydb voltdb/voltdb \
   --set-file cluster.config.licenseXMLFile=license.xml
```

The name *mydb* specifies a name for the release you create, *voltdb/voltdb* specifies the Helm chart to install, and the `--set-file` argument specifies a new value for a property to customize the installation. In this case, `--set-file` specifies the location of the VoltDB license needed to start the database. The license is the only property you must specify; all other properties have default values that are used if not explicitly changed.

However, a default cluster of three nodes and no schema or configuration is not particularly useful. So VoltDB provides Helm properties to let you customize every aspect of the database and cluster configuration, including:

- Cluster configuration, including size of the cluster, available resources, and so on
- Network configuration, including the assignment of ports and external mappings
- Database initialization options, including administration username and password, schema, and class files
- Database configuration, including the settings normally found in the XML configuration file on non-Kubernetes installations

The following sections explain how to use those properties to make some of the most common customizations to your database. Appendix A, *VoltDB Helm Properties* provides a full list of the properties, including a brief description and the default value for each.

2.1. Using Helm Properties

First, it is useful to understand the different ways you can specify properties on the Helm command line. The following discussion is not intended as a complete description of Helm; only a summary to give you an idea of what they do and when to use them.

Helm offers three different ways to specify properties:

--set
: The `--set` flag lets you specify individual property values on the command line. You can use `--set` multiple times or separate multiple property/value pairs with commas. For example, the following two commands are equivalent:

```
$ helm install mydb voltdb/voltdb \
   --set cluster.serviceSpec.clientPort=22222 \
   --set cluster.serviceSpec.adminPort=33333
$ helm install mydb voltdb/voltdb \
   --set cluster.serviceSpec.clientPort=22222,\
```

```
cluster.serviceSpec.adminPort=33333
```

The --set flag is useful for setting a few parameters that change frequently or for overriding parameters set earlier in the command line (such as in a YAML file).

--set-file
: The `--set-file` flag lets you specify the contents of a file as the value for a property. For example, the following command sets the contents of the file `license.xml` as the license for starting the VoltDB cluster:

```
$ helm install mydb voltdb/voltdb \
   --set-file cluster.config.licenseXMLFile=license.xml
```

As with --set, You can use `--set-file` multiple times or separate multiple property/file pairs with commas. The `--set-file` flag is useful for setting parameters where the value is too complicated to set directly on the command line. For example, the contents of the VoltDB license file.

--values, -f
: The --values flag lets you specify a file that contains multiple property definitions in YAML format. Whereas properties set on the command line with `--set` use dot notation to separate the property hierarchy, YAML puts each level of the hierarchy on a separate line, with indentation and followed by a colon. For example, the following YAML file (and `--values` flag set the same two properties show in the `--set` example above:

```
$ cat ports.yaml
cluster:
   serviceSpec:
      clientPort: 22222
      adminPort: 33333
$ helm install mydb voltdb/voltdb \
   --values ports.yaml
```

YAML files are extremely useful for setting multiple properties with values that do not change frequently. You can also use them to group properties (such as port settings or security) that work together to configure aspects of the database environment.

You can use any of the preceding techniques for specifying properties for the VoltDB Helm charts. In fact, you can use each method multiple times on the command line and mixed in any order. For example, the following example uses `--values` to set the database configuration and ports, `--set-file` to identify the license, and `--set` to specify the number of nodes requested:

```
$ helm install mydb voltdb/voltdb                          \
   --values dbconf.xml,dbports.xml                         \
   --set-file cluster.config.licenseXMLFile=license.xml \
   --set cluster.config.replicas=5
```

2.2. Configuring the Cluster and Database

The two major differences between creating a VoltDB database cluster in Kubernetes and starting a cluster using traditional servers are:

- With Helm there is a single command (install) that performs both the initialization and the startup of the database.

- You specify the database configuration with properties rather than as an XML file.

In fact, all of the configuration — including the configuration of the virtual servers (or pods), the server processes, and the database — is accomplished using Helm properties. The following sections provide examples of some of the most common configuration settings when using Kubernetes. Appendix A, *VoltDB Helm Properties* gives a full list of all of the properties that are available for customization.

2.2.1. Configuring the Cluster

Many of the configuration options that are performed through hardware configuration, system commands or environment variables on traditional server platforms are now available through Helm properties. Most of these settings are listed in Section A.3, “Kubernetes Cluster Startup Options”.

Hardware Settings

Hardware settings, such as the number of processors and memory size, are defined as Kubernetes image resources through the Helm `cluster.clusterSpec.resources` property. Under `resources`, you can specify any of the YAML properties Kubernetes expects when configuring pods within a container. For example:

```
cluster:
  clusterSpec:
    resources:
      requests:
        cpu: 500m
        memory: 1000Mi
      limits:
        cpu: 500m
        memory: 1000Mi
```

System Settings

System settings that control process limits that are normally defined through environment variables can be set with the `cluster.clusterSpec.env` properties. For example, the following YAML increases the Java maximum heap size and disables the collection of JVM statistics:

```
cluster:
  clusterSpec:
    env:
      VOLTDB_HEAPMAX: 3072
      VOLTDB_OPTS: -XX+PerfDisableSharedMem
```

One system setting that is *not* configurable through Kubernetes or Helm is whether the base platform has Transparent Huge Pages (THP) enabled or not. This is dependent of the memory management settings on the actual base hardware on which Kubernetes is hosted. Having THP enabled can cause problems with memory-intensive applications like VoltDB and it is strongly recommended that THP be disabled before starting your cluster. (See the section on Transparent Huge Pages in the *VoltDB Administrator's Guide* for an explanation of why this is an issue.)

If you are not managing the Kubernetes environment yourself or cannot get your provider to modify their environment, you will need to override VoltDB's warning about THP on startup by setting the cluster.clusterSpec.additionalArgs property to include the VoltDB start argument to disable the check for THP. For example:

```
cluster:
  clusterSpec:
    additionalArgs: "--ignore=thp"
```

2.2.2. Configuring the Database

In addition to configuring the environment VoltDB runs in, there are many different characteristics of the database itself you can control. These include mapping network interfaces and ports, selecting and configuring database features, and identifying the database schema, class files, and security settings.

The network settings are defined through the `cluster.serviceSpec` properties, where you can choose the individual ports and choose whether to expose them through the networking service (`cluster.serviceSpec.type`) you can also select. For example, the following YAML file disables exposure of the admin port and assigns the externalized client port to 31313:

```
cluster:
  serviceSpec:
    type: NodePort
      adminPortEnabled: false
      clientPortEnabled: true
      clientNodePort: 31313
```

The majority of the database configuration options for VoltDB are traditionally defined in an XML configuration file. When using Kubernetes, these options are declared using YAML and Helm properties.

In general, the Helm properties follow the same structure as the XML configuration, beginninging with "cluster.config". So, for example, where the number of sites per host is defined in XML as :

```
<deployment>
   <cluster sitesperhost="{n}"/>
</deployment>
```

It is defined in Kubernetes as:

```
cluster:
  config:
    deployment:
      cluster:
        sitesperhost: {n}
```

The following sections give examples of defining common database configurations options using both XML and YAML. See Section A.6, “VoltDB Database Configuration Options” for a complete list of the Helm properties available for configuring the database.

2.2.2.1. Command Logging

Command logging provides durability of the database content across failures. You can control the level of durability as well as the length of time required to recover the database by configuring the type of command logging and size of the logs themselves. In Kubernetes this is done with the `cluster.config.deployment.commandlog` properties. The following examples show the equivalent configuration in both XML and YAML:

XML Configuration File	YAML Configuration File

`<commandlog enabled="true"`
`synchronous="true"`
`logsize="3072">`
`<frequency time="300"`
`transactions="1000"/>`
	`cluster:`
	`config:`
	`deployment:`
	`commandlog:`
	`enabled: true`

XML Configuration File	YAML Configuration File

`</commandlog>`	`synchronous: true`
	`logsize: 3072`
	`frequency:`
	`  transactions 1000`

2.2.2.2. Export

Export simplifies the integration of the VoltDB database with external databases and systems. You use the export configuration to define external "targets" the database can write to. In Kubernetes you define export targets using the `cluster.config.deployment.export.configurations` property. Note that the `configurations` property can accept multiple configuration definitions. In YAML, you specify a list by prefixing each list element with a hyphen, even if there is only one element. The following examples show the equivalent configuration in both XML and YAML for configuring a file export connector:

XML Configuration File	YAML Configuration File

`<export>`
`  <configuration`
`  target="eventlog"`
`  type="file">`
`  <property`
`  name="type">csv</property>`
`  <property`
`  name="nonce">eventlog</property>`
`  </configuration>`
`</export>`
	`cluster:`
	`  config:`
	`  deployment:`
	`  export:`
	`  configurations:`
	`  - target: eventlog`
	`  type: file`
	`  properties:`
	`  type: csv`
	`  nonce: eventlog`

2.2.2.3. Security and User Accounts

There are a number of options for securing a VoltDB database, including basic usernames and passwords in addition to industry network solutions such as Kerberos and SSL. Basic security is enabled in the configuration with the `cluster.config.deployment.security.enabled` property. You must also use the property and its children to define the actual usernames, passwords, and assigned roles. Again, the `users` property expects a list of sub-elements so you must prefix each set of properties with a hyphen.

Finally, if you do enable basic security, you must also tell the VoltDB operator which account to use when accessing the database. To do that, you define the `cluster.config.auth` properties, as shown below, which must specify an account with the built-in *administrator* role. The following examples show the equivalent configurations in both XML and YAML, including the assignment of an account to the VoltDB Operator:

XML Configuration File	YAML Configuration File

`<security enabled="true"/>`
`<users>`
`  <user name="admin"`
`  password="superman"`
`  roles="administrator"/>`
`  <user name="mitty"`
`  password="thurber"`
`  roles="user"/>`
`</users>`
	`cluster:`
	`  config:`
	`  deployment:`
	`  security:`
	`  enabled: true`
	`  users:`
	`  - name: admin`
	`  password: superman`
	`  roles: administrator`

XML Configuration File	YAML Configuration File

	<pre>- name: mitty
	  password: thurber
	  roles: user
	auth:
	  username: admin
	  password: superman</pre>

Chapter 3. Managing VoltDB Databases in Kubernetes

When running VoltDB in Kubernetes, you are implicitly managing two separate technologies: the database cluster, that consists of "nodes" and the server processes that run on them and the collection of Kubernetes "pods" the database cluster runs on. There is a one-to-one relationship between VoltDB nodes and Kubernetes pods and it is important that these two technologies stay in sync.

The good news is that if, for example, a database server goes down, Kubernetes recognizes that the corresponding pod is not "live" and spins up a replacement. On the other hand, if you *intentionally* stop the database without telling Kubernetes, it insists on trying to recreate it.

Fortunately, you do not have to worry about this — because the VoltDB Operator and Helm charts manage the synchronization of VoltDB and Kubernetes for you. But it does mean you must use Helm and the Operator to perform operations that affect Kubernetes, and not use the equivalent **voltadmin** command. Specifically, you must use Helm, or kubectl, for operations that modify the cluster, including:

- Starting the database
- Stopping the database
- Resizing the database
- Pausing and resuming the database

Whereas, operations that affect the internal structure or content of the database can be performed using the **voltadmin** or **sqlcmd** utilities. These operations include, among others:

- Modifying the schema or runtime configuration
- Saving or restoring snapshots
- Resetting DR connections

The following sections describe how to manage the cluster with Helm and the database contents with voltadmin and sqlcmd.

3.1. Managing the Cluster with Helm

The key to managing VoltDB clusters in Kubernetes is to let the Helm charts do the work for you. As described in Chapter 2, *Configuring the VoltDB Database Cluster* you can customize every aspect of the database and the cluster using Helm properties and the configuration can be a simple or as complex as you choose. But once you have determined the configuration options you want to use, actually initializing and starting the database cluster is a single command, **helm install**. For example:

```
$ helm install mydb voltdb/voltdb                          \
   --values myconfig.yaml                                  \
   --set-file cluster.config.licenseXMLFile=license.xml  \
   --set cluster.clusterSpec.replicas=5
```

Once the cluster is running (what Helm calls a "release"), you can adjust the cluster to stop it, restart it, or resize it, by "upgrading" the release chart, specifying the new value for the number of nodes you want. You upgrade the release using much the same command, except rather than repeating the configuration,

you can use the `--reuse-values` flag. So, for example, to stop the cluster, you simply set the number of replicas to zero, reusing all other parameters:

```
$ helm upgrade mydb voltdb/voltdb       \
   --reuse-values                       \
   --set cluster.clusterSpec.replicas=0
```

To restart the cluster after you stop it, you reset the replica count to five, or whatever you set it to when you initially defined and started it:

```
$ helm upgrade mydb voltdb/voltdb       \
   --reuse-values                       \
   --set cluster.clusterSpec.replicas=5
```

Finally to resize the cluster, by either adding nodes or removing nodes, you simply upgrade the release specifying the new number of nodes you want. Of course, the new value must meet the requirements for elastically expanding or contracting the cluster, as set out in the discussion of adding and removing nodes from the cluster in the *VoltDB Administrator's Guide*. So, for example, to increase the cluster size by two nodes, you can set the replica count to seven:

```
$ helm upgrade mydb voltdb/voltdb       \
   --reuse-values                       \
   --set cluster.clusterSpec.replicas=7
```

One caveat to using the **helm upgrade** command is that it not only upgrades the release, it checks to see if there is a new version of the original chart (in this example, *voltdb/voltdb*) and upgrades that too. Problems could occur if there are changes to the original chart between when you first start the cluster and when you need to stop or resize it.

The public charts are not changed very frequently. But if your database is in production for an extended period of time it could be an issue. Fortunately, there is a solution. To avoid any unexpected changes, you can tell Helm to use a specific version of the chart — the version you started with.

First, use the **helm list** command to list all of the releases (that is, database instances) you have installed. In the listing it will include both the name and version of the chart in use. For example:

```
$ helm list
NAME  NAMESPACE  REVISION  UPDATED                 STATUS    CHART          APP VERSION
mydb  default    1         2020-08-12 12:45:30     deployed  voltdb-1.0.0   10.0.0
```

You can then specify the specific chart version when your upgrade the release, thereby avoiding any unexpected side effects:

```
$ helm upgrade mydb voltdb/voltdb       \
   --reuse-values                       \
   --set cluster.clusterSpec.replicas=7 \
   --version=1.0.0
```

3.2. Managing the Database Contents

Once the VoltDB database starts, you are ready to manage the database contents. Using Kubernetes does not change *how* you manage the database content. However, it does require a few extra steps to ensure you have access to the database. In Kubernetes, the IP addresses are not known until the pods start. More importantly, the network ports on the pods are not, by default, accessible outside of the Kubernetes virtual cluster itself.

Therefore, to use VoltDB utilities such as voltadmin and sqlcmd remotely, you must first make your database servers accessible. Kubernetes provides several different approaches to doing this, including port forwarding, node ports, load balancers, external port maps, as well as additional services such as Consul to help manage the association of the internal networks with external addresses. Which approach you choose and how you set them up depends on your application needs. However, once the ports are made accessible you can use the arguments to the VoltDB utilities such as **voltadmin --host** to access the database remotely.

One alternative to establishing persistent network mapping is to access the individual pods interactively from your local system. You can do this using the kubectl command line utility.

First you need to identify the pods you want to access. In kubectl, you can do this with the **kubectl get pods** command. You will notice that, by default, the VoltDB helm chart names the pods based on the Helm release name, the text string "-voltdb-cluster-", and an incremental number starting at 0. In other words, if you create a cluster with three nodes using the `helm install mydb voltdb/voltdb` command, the three pods will be named:

mydb-voltdb-cluster-0
mydb-voltdb-cluster-1
mydb-voltdb-cluster-2

You can then access the pods, individually, using the kubectl exec command, specifying the pod you want to access and the command you want to run. For example, to run sqlcmd on the first pod, use the following command:

```
$ kubectl exec -it mydb-voltdb-cluster-0 -- sqlcmd
SQL Command :: localhost:21212
1>
```

You can even execute a local batch file of sqlcmd commands remotely by piping the file into the utility. For example:

```
$ cat schema.sql
 CREATE TABLE HELLOWORLD (
    HELLO VARCHAR(15), WORLD VARCHAR(15),
    DIALECT VARCHAR(15) NOT NULL
 );
 PARTITION TABLE HELLOWORLD ON COLUMN DIALECT;
$ kubectl exec -it mydb-voltdb-cluster-0 -- sqlcmd < schema.sql
Command succeeded.
Command succeeded.
$
```

Appendix A. VoltDB Helm Properties

You communicate with the VoltDB Operator, and Kubernetes itself, through the Helm charts that VoltDB provides. You can also specify additional Helm properties that customize what the Helm charts do. The properties are hierarchical in nature and can be specified on the Helm command line either as one or more YAML files or as individual arguments. For example, you can specify multiple properties in a YAML file then reference the file as part of your command using the `--values` or `-f` argument, like so:

```
$ helm install mydb voltdb/voltdb --values myoptions.yaml
```

Or you can specify the properties individually in dot notation on the command line using the --set flag, like so:

```
$ helm install mydb voltdb/voltdb \
     --set cluster.clusterSpec.replicas=5 \
     --set cluster.config.deployment.cluster.kfactor=2 \
     --set cluster.config.deployment.cluster.sitesperhost=12
```

In YAML, you specify each element of the property on a separate line, following each parent element with a colon, indenting each level appropriately, and following the last element with the value of the property . On the command line you specify the property with the elements separated by periods and the value following an equals sign. So in the preceding example, the matching YAML file for the command line properties would look like this:

```
cluster:
  clusterSpec:
     replicas: 5
  config:
     deployment:
        cluster:
           kfactor: 2
           sitesperhost: 12
```

Many of the properties have default values; the following tables specify the default values where applicable. You do not need to specify values for all of the properties. In fact, you can start a generic VoltDB database specifying only the license file. Otherwise, you need only specify those properties you want to customize.

Finally, the properties are processed in order and can be overriden. So if you specify different values for the same property in two YAML files and as a command line argument, the latter YAML file setting overrides the first and the command line option overrides them both.

A.1. How to Use the Properties

The following sections detail all of the pertinent Helm properties that you can specify when creating or modifying the VoltDB Operator and its associated cluster. The properties are divided into categories and each category identified by the root elements common to all properties in that category:

- Top-Level Kubernetes Options
- Kubernetes Cluster Startup Options
- Network Options
- VoltDB Database Startup Options

- VoltDB Database Configuration Options

For the sake of brevity and readability, the properties in the tables are listed by only the unique elements of the property after the root. However, when specifying a property in YAML or on the command line, you must specify all elements of the full property name, including both the root and the unique elements.

A.2. Top-Level Kubernetes Options

The following properties affect how Helm interacts with the Kubernetes infrastructure.

Table A.1. Top-Level Options

Parameter	Description	Default
cluster.enabled	Create VoltDB Cluster	true
cluster.serviceAccount.create	If true, create & use service account for VoltDB cluster node containers	true
cluster.serviceAccount.name	If not set and create is true, a name is generated using the fullname template	""

A.3. Kubernetes Cluster Startup Options

The following properties affect the size and structure of the Kubernetes cluster that gets started, as well as the startup attributes of the VoltDB cluster running on those pods.

Table A.2. Options Starting with cluster.clusterSpec...

Parameter	Description	Default
.replicas	Pod (VoltDB Node) replica count, scaling to 0 will shutdown the cluster gracefully	3
.maxPodUnavailable	Maximum pods unavailable in Pod Disruption Budget	kfactor
.maintenanceMode	VoltDB Cluster maintenance mode (pause all nodes)	false
.takeSnapshotOnShutdown	Takes a snapshot when cluster is shutdown by scaling to 0. Valid options are NoCommandLogging, Always, Never. Defaults to NoCommandLogging (commandlog enabled=false only) if not specified.	""
.initForce	Always init --force on VoltDB node start/restart. WARNING: This will destroy VoltDB data on PVCs except snapshots.	false
.deletePVC	Delete and cleanup generated PVCs when VoltDBCluster is	false

Parameter	Description	Default
	deleted, requires finalizers to be enabled (on by default)	
.stoppedNodes	User-specified list of stopped nodes based on StatefulSet # (e.g. [2, 3])	[ ]
.additionalXDCRReadiness	Add additional readiness checks using XDCR to ensure both clusters are healthy (WARNING: May cause app downtime)	false
.persistentVolume.size	Persistent Volume size per Pod (VoltDB Node)	1Gi
.persistentVolume .storageClassName	Storage Class name to use, otherwise use default	""
.ssl.certificateFile	PEM encoded certificate chain used by the VoltDB operator when SSL/TLS is enabled	""
.ssl.insecure	If true, skip certificate verification by the VoltDB operator when SSL/TLS is enabled	false
.storageConfigs	Optional storage configs for provisioning additional persistent volume claims automatically	[ ]
.additionalVolumes	Additional list of volumes that can be mounted by node containers	[ ]
.additionalVolumeMounts	Pod volumes to mount into the container's filesystem, cannot be modified once set	[ ]
.image.registry	Image registry	docker.io
.image.repository	Image repository	voltdb/voltdb-enterprise
.image.tag	Image tag	10.0.0
.image.pullPolicy	Image pull policy	Always
.additionalStartArgs	Additional VoltDB start command args for the pod container	[ ]
.priorityClassName	Pod priority defined by an existing PriorityClass	""
.additionalAnnotations	Additional custom Pod annotations	{ }
.additionalLabels	Additional custom Pod labels	{ }
.resources	CPU/Memory resource requests/ limits	{ }
.nodeSelector	Node labels for pod assignment	{ }
.tolerations	Pod tolerations for Node assignment	[ ]

Parameter	Description	Default
.affinity	Node affinity	{ }
.podSecurityContext	Pod security context	{"runAsNonRoot":true, "runAsUser":1001, "fsGroup":1001}
.securityContext	Container security context. WARNING: Changing user or group ID may prevent VoltDB from operating.	{"privileged":false, "runAsNonRoot":true, "runAsUser":1001, "runAsGroup":1001, "readOnlyRootFilesystem":true}
.podTerminationGracePeriodSeconds	Duration in seconds the Pod needs to terminate gracefully. Defaults to 30 seconds if not specified.	30
.livenessProbe.enabled	Enable/disable livenessProbe	true
.livenessProbe .initialDelaySeconds	Delay before liveness probe is initiated	20
.livenessProbe.periodSeconds	How often to perform the probe	10
.livenessProbe.timeoutSeconds	When the probe times out	1
.livenessProbe.failureThreshold	Minimum consecutive failures for the probe	20
.livenessProbe.successThreshold	Minimum consecutive successes for the probe	1
.readinessProbe.enabled	Enable/disable readinessProbe	true
.readinessProbe .initialDelaySeconds	Delay before readiness probe is initiated	20
.readinessProbe.periodSeconds	How often to perform the probe	60
.readinessProbe.timeoutSeconds	When the probe times out	1
.readinessProbe.failureThreshold	Minimum consecutive failures for the probe	3
.readinessProbe .successThreshold	Minimum consecutive successes for the probe	1
.startupProbe.enabled	Enable/disable startupProbe, feature flag must also be enabled at a cluster level (enabled by default in 1.18)	true
.startupProbe .initialDelaySeconds	Delay before startup probe is initiated	40
.startupProbe.periodSeconds	How often to perform the probe	10
.startupProbe.timeoutSeconds	When the probe times out	1
.startupProbe.failureThreshold	Minimum consecutive failures for the probe	20
.startupProbe.successThreshold	Minimum consecutive successes for the probe	1

Parameter	Description	Default
.env.VOLTDB_OPTS	VoltDB cluster additional java runtime options (VOLTDB_OPTS)	""
.env.VOLTDB_HEAPMAX	VoltDB cluster heap size (VOLTDB_HEAPMAX)	""
.env .VOLTDB_K8S_LOG_CONFIG	VoltDB log4jcfg file path	""
.customEnv	Key-value map of additional envvars to set in all VoltDB node containers	{ }

A.4. Network Options

The following properties specify what ports to use and the port-mapping protocol.

Table A.3. Options Starting with cluster.serviceSpec...

Parameter	Description	Default
.type	VoltDB service type (options ClusterIP, NodePort, and LoadBalancer)	ClusterIP
.vmcPort	VoltDB Management Center web interface Service port	8080
.vmcNodePort	Port to expose VoltDB Management Center service on each node, type NodePort only	31080
.vmcSecurePort	VoltDB Management Center secure web interface Service port	8443
.vmcSecureNodePort	Port to expose VoltDB Management Center secure service on each node, type NodePort only	31443
.replicationPort	VoltDB replication exposed Service port	5555
.replicationNodePort	Port to expose VoltDB replication service on each node, type NodePort only	31555
.adminPortEnabled	Enable exposing admin port with the VoltDB Service	true
.adminPort	VoltDB Admin exposed Service port	21211
.adminNodePort	Port to expose VoltDB Admin service on each node, type NodePort only	31211
.clientPortEnabled	Enable exposing client port with the VoltDB Service	true

Parameter	Description	Default
.clientPort	VoltDB Client exposed service port	21212
.clientNodePort	Port to expose VoltDB Client service on each node, type NodePort only	31212
.loadBalancerIP	VoltDB Load Balancer IP	""
.loadBalancerSourceRanges	VoltDB Load Balancer Source Ranges	[ ]
.externalIPs	List of IP addresses at which the VoltDB service is available	[ ]

A.5. VoltDB Database Startup Options

The following properties affect how Helm interacts with the VoltDB cluster adn specific initialization options, such as the initial schema and procedure classes.

Table A.4. Options Starting with cluster.config...

Parameter	Description	Default
.auth.username	User added for operator VoltDB API communication when hash security is enabled	voltdb-operator
.auth.password	Password added for operator VoltDB API communication when hash security is enabled	""
.schemas	List of optional schema files containing data defintion statements	[ ]
.classes	List of optional jar files container stored procedures	[ ]
.licenseXMLFile	VoltDB Enterprise license.xml	{ }
.log4jcfgFile	Custom Log4j configuration file	{ }

A.6. VoltDB Database Configuration Options

The following properties define the VoltDB database configuration.

Table A.5. Options Starting with cluster.config.deployment...

Parameter	Description	Default
.cluster.kfactor	K-factor to use for database durability and data safety replication	1
.cluster.sitesperhost	SitesPerHost for VoltDB Cluster	8
.heartbeat.timeout	Internal VoltDB cluster verification of presence of other nodes (seconds)	90

Parameter	Description	Default
.partitiondetection.enabled	Controls detection of network partitioning	true
.commandlog.enabled	Command logging for database durability (recommended)	true
.commandlog.logsize	Command logging allocated disk space (MB)	1024
.commandlog.synchronous	Transactions do not complete until logged to disk	false
.commandlog.frequency.time	How often the command log is written, by time (milliseconds)	200
.commandlog.frequency .transactions	How often the commang log is written, by transaction command	2147483647
.dr.id	Unique cluster id, 0-127	0
.dr.role	Role for this cluster, one of: master, replica, xdcr, none	master
.dr.connection.enabled	Specifies whether disaster recovery is enabled	false
.dr.connection.source	If role is replica or xdcr: list of host names or IP addresses of remote node(s)	""
.dr.connection.preferredSource	Cluster ID of preferred source	""
.dr.connection.ssl	Certificate file for DR consumer (replica or xdcr mode)	""
.export.configurations	List of export configurations	[ ]
.import.configurations	List of import configurations	[ ]
.httpd.enabled	Determines if HTTP API daemon is enabled	true
.httpd.jsonapi.enabled	Determines if jSON over HTTP API is enabled	true
.paths.commandlog.path	Directory path for command log	/pvc/voltdb/CLUSTER_NAME /command_log
.paths.commandlogsnapshot.path	Directory path for command log snapshot	/pvc/voltdb/CLUSTER_NAME /command_log_snapshot
.paths.droverflow.path	Directory path for disaster recovery overflow	/pvc/voltdb/CLUSTER_NAME /dr_overflow
.paths.exportcursor.path	Directory path for export cursors	/pvc/voltdb/CLUSTER_NAME /export_cursor
.paths.exportoverflow.path	Directory path for export overflow	/pvc/voltdb/CLUSTER_NAME /export_overflow
.paths.largequeryswap.path	Directory path for large query swapping	/pvc/voltdb/CLUSTER_NAME /large_query_swap
.paths.snapshots.path	Directory path for snapshots. If not default, path must not be a read-only root folder of	/pvc/voltdb/CLUSTER_NAME /snapshots

Parameter	Description	Default
	mounted storage (as init --force will rename snapshot folder). Use a subfolder (such as snapshots) when possible.	
.security.enabled	Controls whether user-based authentication and authorization are used	false
.security.provider	Allows use of external Kerberos provider; one of: hash, kerberos	hash
.snapshot.enabled	Enable/disable periodic automatic snapshots	true
.snapshot.frequency	Frequency of automatic snapshots (in s,m,h)	24h
.snapshot.prefix	Unique prefix for snapshot files	AUTOSNAP
.snapshot.retain	Number of snapshots to retain	2
.snmp.enabled	Enables or disables use of SNMP	false
.snmp.target	Host name or IP address, and optional port (default 162), for SNMP server	""
.snmp.authkey	SNMPv3 authentication key if protocol is not NoAuth	voltdbauthkey
.snmp.authprotocol	SNMPv3 authentication protocol. One of: SHA, MD5, NoAuth	SHA
.snmp.community	Name of SNMP community	public
.snmp.privacykey	SNMPv3 privacy key if protocol is not NoPriv	voltdbprivacykey
.snmp.privacyprotocol	SNMPv3 privacy protocol. One of: AES, DES, 3DES, AES192, AES256, NoPriv	AES
.snmp.username	Username for SNMPv3 authentication; else SNMPv2c is used	""
.ssl.enabled	Enables TLS/SSL security for the HTTP port (default 8080, 8443)	false
.ssl.external	Extends TLS/SSL security to all external ports (default admin 21211, client 21212)	false
.ssl.internal	Extends TLS/SSL security to the internal port (default 3021)	false
.ssl.dr	Extends TLS/SSL security to the DR port (5555)	false
.ssl.keystore.file	Keystore file to mount at the keystore path	""

Parameter	Description	Default
.ssl.keystore.path	File-path to the VoltDB keystore location	/etc/voltdb/ssl/keystore.jks
.ssl.keystore.password	Password for VoltDB keystore	""
.ssl.truststore.file	Truststore file to mount at the truststore path	""
.ssl.truststore.path	File-path to the VoltDB truststore location	/etc/voltdb/ssl/cacerts.jks
.ssl.truststore.password	Password for VoltDB truststore	""
.systemsettings.elastic.duration	Target value for the length of time each rebalance transaction will take (milliseconds)	50
.systemsettings.elastic.throughput	Target value for rate of data processing by rebalance transactions (MB)	2
.systemsettings.flushinterval .minimum	Interval between checking for need to flush (milliseconds)	1000
.systemsettings.flushinterval.dr .interval	Interval for flushing DR data (milliseconds)	1000
.systemsettings.flushinterval .export.interval	Interval for flushing export data (milliseconds)	4000
.systemsettings.procedure.loginfo	Threshold for long-running task detection (milliseconds)	10000
.systemsettings.query.timeout	Timeout on SQL queries (milliseconds)	10000
.systemsettings.resourcemonitor .frequency	Resource Monitor interval between resource checks (seconds)	60
.systemsettings.resourcemonitor .memorylimit.size	Limit on memory use (in GB or as percentage)	80.00%
.systemsettings.resourcemonitor .memorylimit.alert	Alert level for memory use (in GB or as percentage)	70.00%
.systemsettings.resourcemonitor .disklimit.commandlog.size	Resource Monitor disk limit on disk use (in GB or percentage, empty is unlimited)	""
.systemsettings.resourcemonitor .disklimit.commandlog.alert	Resource Monitor alert level for disk use (in GB or as percentage, empty is unlimited)	""
.systemsettings.resourcemonitor .disklimit.commandlogsnapshot .size	Resource Monitor disk limit on disk use (in GB or percentage, empty is unlimited)	""
.systemsettings.resourcemonitor .disklimit.commandlogsnapshot .alert	Resource Monitor alert level for disk use (in GB or as percentage, empty is unlimited)	""

Parameter	Description	Default
.systemsettings.resourcemonitor .disklimit.droverflow.size	Resource Monitor disk limit on disk use (in GB or percentage, empty is unlimited)	""
.systemsettings.resourcemonitor .disklimit.droverflow.alert	Resource Monitor alert level for disk use (in GB or as percentage, empty is unlimited)	""
.systemsettings.resourcemonitor .disklimit.exportoverflow.size	Resource Monitor disk limit on disk use (in GB or percentage, empty is unlimited)	""
.systemsettings.resourcemonitor .disklimit.exportoverflow.alert	Resource Monitor alert level for disk use (in GB or as percentage, empty is unlimited)	""
.systemsettings.resourcemonitor .disklimit.snapshots.size	Resource Monitor disk limit on disk use (in GB or percentage, empty is unlimited)	""
.systemsettings.resourcemonitor .disklimit.snapshots.alert	Resource Monitor alert level for disk use (in GB or as percentage, empty is unlimited)	""
.systemsettings.resourcemonitor .disklimit.topicsdata.size	Resource Monitor disk limit on disk use (in GB or percentage, empty is unlimited)	""
.systemsettings.resourcemonitor .disklimit.topicsdata.alert	Resource Monitor alert level for disk use (in GB or as percentage, empty is unlimited)	""
.systemsettings.snapshot.priority	Priority for snapshot work	6
.systemsettings.temptables .maxsize	Limit the size of temporary database tables (MB)	100
.users	Define a list of VoltDB users to be added to the deployment	