Administrator's Guide

Abstract

This books explains how to create and manage Volt Active Data databases and the clusters that run them.

V13

Administrator's Guide

V13

This document was generated on July 22, 2024.

Table of Contents

List of Tables

Preface

This book explains how to manage Volt Active Data 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.

1. Structure of This Book

This book is divided into 7 chapters and 3 appendices:

- Chapter 1, *Managing Volt Active Data Databases*
- Chapter 2, *Preparing the Servers*
- Chapter 3, *Starting and Stopping the Database*
- Chapter 4, *Maintenance and Upgrades*
- Chapter 5, *Monitoring VoltDB Databases*
- Chapter 6, *Logging and Analyzing Activity in a VoltDB Database*
- Chapter 7, *What to Do When Problems Arise*
- Appendix A, *Server Configuration Options*
- Appendix B, *Volt Active Data Metrics*
- Appendix C, *Snapshot Utilities*

2. Related Documents

This book does not describe how to design or develop Volt Active Data databases. For a complete description of the development process for VoltDB and all of its features, please see the accompanying manual *Using VoltDB*. For new users, see the *VoltDB Tutorial*. These and other books describing Volt Active Data are available on the web from http://docs.voltactivedata.com/.

Chapter 1. Managing Volt Active Data Databases

VoltDB is a distributed, in-memory database designed from the ground up to maximize throughput performance on commodity servers. The VoltDB architecture provides many advantages over traditional database products while avoiding the pitfalls of NoSQL solutions:

- By partitioning the data and stored procedures, VoltDB can process multiple queries in parallel without sacrificing the consistency or durability of an ACID-compliant database.
- By managing all data in memory with a single thread for each partition, VoltDB avoids overhead such as record locking, latching, and device-contention inherent in traditional disk-based databases.
- VoltDB databases can scale up to meet new capacity or performance requirements simply by adding more nodes to the cluster.
- Partitioning is automated, based on the schema, so there is no need to manually shard or repartition the data when scaling up as with many NoSQL solutions.
- Finally, VoltDB Enterprise Edition provides features to ensure durability and high availability through command logging, locally replicating partitions (K-safety), and wide-area database replication.

Each of these features is described, in detail, in the *Using VoltDB* manual. This book explains how to use these and other features to manage and maintain a VoltDB database cluster from a database administrator's perspective.

1.1. Getting Started

Before you set up VoltDB for use in a production environment, you need to make four decisions:

- **What database features to use** — Which features you want to use are defined in the configuration file and set with the **voltdb init** command.
- **Physical structure of the cluster** — The number and addresses of the nodes in the cluster, which you specify when you start the cluster with the **voltdb start** command.
- **Logical structure of the database** — The logical structure of the database tables and views, otherwise known as the *schema*, is defined in standard SQL statements and can be applied to the database using the **sqlcmd** command line utility.
- **Stored procedures** — The schema declares stored procedures. The procedures themselves execute transactions against the data and are written as Java classes. You load the stored procedures as JAR files using the **sqlcmd** command line utility.

To initialize a VoltDB database cluster, you need a configuration file. The *configuration file* lets you enable and configure various database options including availability, durability, and security. The configuration file also defines certain attributes of the database on the current server, in particular the paths for disk-based files created by the database such as command logs and snapshots. All nodes in the cluster must specify the same cluster configuration file when they initialize the database root directory with the **voltdb init** command.

When you actually start the database cluster, using the **voltdb start** command, you declare the size the cluster by specifying the number of nodes in the cluster and one or more of the nodes as potential hosts. VoltDB selects one of the specified nodes as the "leader" to coordinate startup.

When using the VoltDB Enterprise Edition, you will also need a license file, often called `license.xml`. VoltDB automatically looks for the license file in the user's current working directory, the home directory, or the `voltdb/` subfolder where VoltDB is installed. If you keep the license file in a different directory or under a different name, you can use to `--license` argument on the **voltdb init** command to specify the license file location.

Finally, to prepare the database for a specific application, you will need the database schema, including the DDL statements that describe the database's logical structure, and a JAR file containing the stored procedure class files. In general, the database schema and stored procedures are produced as part of the database development process, which is described in the *Using VoltDB* manual.

This book assumes the schema and stored procedures have already been created. The configuration file, on the other hand, defines the run-time configuration of the cluster. Establishing the correct settings for the configuration file and physically managing the database cluster is the duty of the administrators who are responsible for maintaining database operations. This book is written for those individuals and covers the standard procedures associated with database administration.

1.2. Understanding the VoltDB Utilities

VoltDB provides several command line utilities, each with a different function. Familiarizing yourself with these utilities and their uses can make managing VoltDB databases easier. The three primary command line tools for creating, managing, and testing VoltDB databases are:

voltdb	Starts the VoltDB database process. The **voltdb** command can also collect log files for analyzing possible system errors (see Section 7.3, “Collecting the Log Files” for details). The **voltdb** command runs locally and does not require a running database.
voltadmin	Issues administrative commands to a running VoltDB database. You can use **voltadmin** to save and restore snapshots, pause and resume admin mode, and to shutdown the database, among other tasks. The **voltadmin** command can be run remotely, performs cluster-wide operations and requires a running database to connect to.
sqlcmd	Lets you issue SQL queries and invoke stored procedures interactively. The **sqlcmd** command is handy for testing database access without having to write a client application. The **sqlcmd** command can be run remotely and requires a running database to connect to.

In addition to the preceding general-purpose tools, VoltDB provides several other tools for specific tasks:

csvloader, jdbcloader, and kafkaloader	These utilities load data from external sources into an existing VoltDB database. They let you load data from CSV or text-based data files, JDBC data sources, or Apache Kafka streams. These commands can be run remotely and require a running database to connect to.
snapshotconvert	Converts native snapshot files to csv or tabbed text files. The **snapshotconvert** command is useful when exporting a snapshot in native format to text files for import into another data utility. (This utility is provided for legacy purposes. It is now possible to write snapshots directly to CSV format without post-processing, which is the recommended approach.)

	The **snapshotconvert** command runs locally and does not require a running database.
snapshotverify	Verifies that a set of native snapshot files are complete and valid. The **snapshotverify** command runs locally and does not require a running database.

Finally, VoltDB includes a browser-based management console — VoltDB Management Center — for monitoring databases in real time. See Section 5.1.1, “Volt Management Center” for more information about using the Management Center.

1.3. Management Tasks

Database administration responsibilities fall into five main categories, as described in Table 1.1, “Database Management Tasks”. The following chapters are organized by category and explain how to perform each task for a VoltDB database.

Table 1.1. Database Management Tasks

Preparing the Servers	Before starting the database, you must make sure that the server hardware and software is properly configured. This chapter provides a checklist of tasks to perform before starting VoltDB.
Basic Database Operations	The basic operations of initializing, starting, and stopping the database. This chapter describes the procedures needed to handle these fundamental tasks.
Maintenance and Upgrades	Over time, both the cluster and the database may require maintenance — either planned or emergency. This chapter explains the procedures for performing hardware and software maintenance, as well as standard maintenance, such as backing up the database and upgrading the hardware, the software, and the database schema.
Performance Monitoring	Another important role for many database administrators is monitoring database performance. Monitoring is important for several reasons: • Performance Analysis • Load Balancing • Fault Detection This chapter describes the tools available for monitoring VoltDB databases.
Problem Reporting & Analysis	If an error does occur and part or all of the database cluster fails, it is not only important to get the database up and running again, but to diagnose the cause of the problem and take corrective actions. VoltDB produces a number of log files that can help with problem resolution. This chapter describes the different logs that are available and how to use them to diagnose database issues.

Chapter 2. Preparing the Servers

VoltDB is designed to run on commodity servers, greatly reducing the investment required to operate a high performance database. However, out of the box, these machines are not necessarily configured for optimal performance of a dedicated, clustered application like VoltDB. This is especially true when using cloud-based services. This chapter provides best practices for configuring servers to maximize the performance and stability of your VoltDB installation.

2.1. Server Checklist

The very first step in configuring the servers is making sure you have sufficient memory, computing power, and system resources such as disk space to handle the expected workload. The *VoltDB Planning Guide* provides detailed information on how to size your server requirements.

The next step is to configure the servers and assign appropriate resources for VoltDB tasks. Specific server features that must be configured for VoltDB to perform optimally are:

- Install required software
- Configure memory management
- Turn off TCP Segmentation
- Configure the time synchronization services
- Increase resource limits
- Define network addresses for all nodes in the cluster
- Assign network ports

2.2. Install Required Software

To start, VoltDB requires a recent release of the Linux operating system. The supported operating systems for running production VoltDB databases are:

- Red Hat (RHEL) 8.6 or later, including 9.0 and subsequent releases
- Rocky Linux 8.6 or later, including 9.0 and subsequent releases
- Ubuntu 20.04, and 22.04

It may be possible to run VoltDB on other versions of Linux and Macintosh OS X 11.0 and later is supported for development purposes. However, the preceding operating system versions are the only fully tested and supported base platforms for running VoltDB in production.

In addition to the base operating system, VoltDB requires the following software at a minimum:

- Java 11, 17, or 21
- Time synchronization services, such as NTP or chrony
- Python 3.9 or later

Oracle Java SDK 11, 17, or 21 is recommended, but OpenJDK 11, 17, or 21 are also supported.

VoltDB works best when the system clocks on all cluster nodes are synchronized to within 100 milliseconds or less. However, the clocks are allowed to differ by up to 200 milliseconds before VoltDB refuses to start. NTP, the Network Time Protocol, or chrony are recommended for achieving the necessary synchronization. NTP is installed and enabled by default on many operating systems. However, the configuration may need adjusting (see Section 2.5, “Configure Time Services” for details) and in cloud instances where hosted servers are run in a virtual environment, a time service may not be installed or enabled by default. Therefore you need to do this manually.

Finally, VoltDB implements its command line interface through Python. Python 3.9 or later is required to use the VoltDB shell commands.

2.3. Configure Memory Management

Because VoltDB is an in-memory database, proper memory management is vital to the effective operation of VoltDB databases. Three important aspects of memory management are:

- Swapping
- Memory Mapping (Transparent Huge Pages)
- Virtual memory

The following sections explain how best to configure these features for optimal performance of VoltDB.

2.3.1. Disable Swapping

Swapping is an operating system feature that optimizes memory usage when running multiple processes by swapping processes in and out of memory. However, any contention for memory, including swapping, will have a very negative impact on VoltDB performance and functionality. You should disable swapping when using VoltDB.

To disable swapping on Linux systems, use the `swapoff` command. Alternately, you can set the kernel parameter `vm.swappiness` to zero.

2.3.2. Disable Transparent Huge Pages

Transparent Huge Pages (THP) are another operating system feature that optimizes memory usage for systems with large amounts of memory. THP changes the memory mapping to use larger physical pages. This can be helpful for general-purpose computing running multiple processes. However, for memory-intensive applications such as VoltDB, THP can actually negatively impact performance.

Therefore, it is important to disable Transparent Huge Pages on servers running VoltDB. The following commands, run as root or from another privileged account, disable THP:

```
$ echo never >/sys/kernel/mm/transparent_hugepage/enabled
$ echo never >/sys/kernel/mm/transparent_hugepage/defrag
```

Or:

```
$ echo madvise >/sys/kernel/mm/transparent_hugepage/enabled
$ echo madvise >/sys/kernel/mm/transparent_hugepage/defrag
```

For RHEL systems (including CentOS), replace "transparent_hugepage" with "redhat_transparent_hugepage".

Note, however, that these commands disable THP only while the server is running. Once the server reboots, the default setting will return. Therefore, we recommend you disable THP permanently as part of the startup process. For example, you can add the following commands to a server startup script (such as /etc/rc.local):

```
#!/bin/bash
for f in /sys/kernel/mm/*transparent_hugepage/enabled; do
    if test -f $f; then echo never > $f; fi
done
for f in /sys/kernel/mm/*transparent_hugepage/defrag; do
    if test -f $f; then echo never > $f; fi
done
```

THP are enabled by default in Ubuntu 14.04 and later as well as RHEL 6.x and 7.x. To see if they are enabled on your current system, use either of the following pair of commands:

```
$ cat /sys/kernel/mm/transparent_hugepage/enabled
$ cat /sys/kernel/mm/transparent_hugepage/defrag

$ cat /sys/kernel/mm/redhat_transparent_hugepage/enabled
$ cat /sys/kernel/mm/redhat_transparent_hugepage/defrag
```

If THP is disabled, the output from the preceding commands should be either “always madvise [never]” or “always [madvise] never”.

2.3.3. Enable Virtual Memory Mapping and Overcommit

Although swapping is bad for memory-intensive applications like VoltDB, the server does make use of virtual memory (VM) and there are settings that can help VoltDB make effective use of that memory. First, it is a good idea to enable VM overcommit. This avoids VoltDB encountering unnecessary limits when managing virtual memory. This is done on Linux by setting the system parameter `vm.overcommit_memory` to a value of "1".

```
$ sysctl -w vm.overcommit_memory=1
```

Second, for large memory systems, it is also a good idea to increase the VM memory mapping limit. So for servers with 64 Gigabytes or more of memory, the recommendation is to increase VM memory map count to 1048576. You do this on Linux with the system parameter `max_map_count`. For example:

```
$ sysctl -w vm.max_map_count=1048576
```

Remember that for both overcommit and the memory map count, the parameters are only active while the system is running and will be reset to the default on reboot. So be sure to add your new settings to the file `/etc/sysctl.conf` to ensure they are in effect when the system is restarted.

2.4. Turn off TCP Segmentation

Under certain conditions, the use of TCP segmentation offload (TSO) and generic receive offload (GRO) can cause nodes to randomly drop out of a cluster. These settings let the system to batch network packets, producing unnecessary latency and interfering with the necessary communication between VoltDB cluster nodes. The symptoms of this problem are that nodes timeout — that is, the rest of the cluster thinks they

have failed — although the node is still running and no other network issues (such as a network partition) are the cause.

Disabling TSO and GRO is recommended for any VoltDB clusters that experience such instability. The commands to disable offloading are the following, where N is replaced by the number of the ethernet card:

```
ethtool -K ethN tso off
ethtool -K ethN gro off
```

Note that these commands disable offloading temporarily. You must issue these commands every time the node reboots or, preferably, put them in a startup configuration file.

It is also a good idea to check that TCP_RETRIES2 has not been altered. Setting TCP_RETRIES2 too low (below 8) can cause similar unpredictable timeouts. See the description of the VoltDB heartbeat timeout setting in Section A.3.7, “Heartbeat” for details.

2.5. Configure Time Services

To orchestrate activities between the cluster nodes, VoltDB relies on the system clocks being synchronized. Many functions within VoltDB — such as cluster start up, nodes rejoining, and schema updates among others — are sensitive to variations in the time values between nodes in the cluster. Therefore, it is important to keep the clocks synchronized within the cluster. Specifically:

- The server clocks in the cluster must be synchronized to within 200 milliseconds of each other when the cluster starts. (Ideally, skew between nodes should be kept under 10 milliseconds.)
- Time must not move backwards

The easiest way to achieve these goals is to install and configure a time service such as NTP (Network Time Protocol) or chrony to use a common time host server for synchronizing the servers. NTP is often installed by default but may require additional configuration to achieve acceptable synchronization. Specifically, listing only one time server (and the same one for all nodes in the cluster) ensures minimal skew between servers. You can even establish your own time server to facilitate this. All nodes in the cluster should also list each other as peers. For example, the following NTP configuration file uses a local time server (myntpsvr) and establishes all nodes in the cluster as peers:

```
server myntpsvr burst iburst minpoll 4 maxpoll 4

peer voltsvr1 burst iburst minpoll 4 maxpoll 4
peer voltsvr2 burst iburst minpoll 4 maxpoll 4
peer voltsvr3 burst iburst minpoll 4 maxpoll 4

server 127.127.0.1
```

See the chapter on Configuring NTP in the *Guide to Performance and Customization* for an example of configuring a time service for optimal performance when running VoltDB.

2.6. Increase Resource Limits

There are several resource limits managed by the operating system where per-user default values are optimized for time-sharing systems but can be too restrictive for dedicated applications like VoltDB. In particular, although VoltDB is an in-memory database, process threads require large numbers of file descriptors, to the point where the file descriptor limit can interfere with VoltDB operations.

It is recommended that you increase the process and file descriptor limits for the process starting the VoltDB server. You can do this with the **ulimit** shell command prior to starting VoltDB. The recommended minimum limits for processes and file descriptors are 8192 and 16384, respectively. Note that these are top limits, so on dedicated servers there are no drawbacks to setting these values even higher. For example, the following commands set the limits to 10,000 and 40,000 before starting the server:

```
$ ulimit -u 10000
$ ulimit -n 40000
$ voltdb start
```

To set the limits permanently, you can set the limits as part of the system initialization. See your operation system documentation on **ulimit** and **init.d** for more information.

2.7. Configure the Network

It is also important to ensure that the network is configured correctly so all of the nodes in the VoltDB cluster recognize each other. If the DNS server does not contain entries for all of the servers in the cluster, an alternative is to add entries in the /etc/hosts file locally for each server in the cluster. For example:

```
12.24.48.101   voltsvr1
12.24.48.102   voltsvr2
12.24.48.103   voltsvr3
12.24.48.104   voltsvr4
12.24.48.105   voltsvr5
```

2.8. Assign Network Ports

VoltDB uses a number of network ports for functions such as internal communications, client connections, rejoin, database replication, and so on. For these features to perform properly, the ports must be open and available. Review the following list of ports to ensure they are open and available (that is, not currently in use).

Function	**Default Port Number**
Client Port	21212
Admin Port	21211
Web Interface Port (httpd)	8080
Internal Server Port	3021
Replication Port	5555
Zookeeper port	7181

Alternately, you can reassign the port numbers that VoltDB uses. See Section A.5, “Network Ports” for a description of the ports and how to reassign them.

2.9. Eliminating Server Process Latency

The preceding sections explain how to configure your servers and network to maximize the performance of VoltDB. The goal is to avoid server functions, such as swapping or Java garbage collection, from disrupting the proper operation of the VoltDB process.

Any latency in the scheduling of VoltDB threads can impact the performance of your database. These delays result in corresponding latency in the database transactions themselves. But equally important, prolonged latency can interrupt intra-cluster communication as well, to the point where the cluster may incorrectly assume a node has failed and drop it as a member. If server latency causes a node not to respond to network messages beyond the heartbeat timeout setting, the rest of the cluster will drop the node as a "dead host".

Therefore, in addition to the configuration settings described earlier in this chapter, the following are some known causes of latency you should watch out for:

- **Other applications** — Clearly, running other applications on the same servers as VoltDB can result in unpredictable resource conflicts for memory, CPU, and disk access. Running VoltDB on dedicated servers is always recommended for production environments.

- **Frequent snapshots** — Initiating snapshots consumes resources. Especially on a database under heavy load, this can result in latency spikes. Although it is possible to run both automated snapshots and command logging (which performs its own snapshots), they are redundant and can cause unnecessary delays. Also, when using command logging on a busy database, consider increasing the size of the command log segments if snapshots are occurring too frequently.

- **I/O contention** — Contention for disk resources can interfere with the effective processing of VoltDB durability features. This can be avoided by allocating separate devices for individual disk-based activity. For example, wherever possible locate command logs and snapshots on separate devices.

- **JVM statistics collection** — Enabling Java Virtual Machine (JVM) statistics can produce erratic latency issues for memory-intensive applications like VoltDB. Disabling JVM stats is strongly recommended when running VoltDB servers. You can disable JVM stats by issuing the following command before starting the VoltDB process:

  ```
  export VOLTDB_OPTS='-XX:+PerfDisableSharedMem'
  ```

  Alternately, you can write the JVM stats to an in-memory virtual disk, such as `/tmpfs`.

- **Hardware power saving options** — Beware of hardware options that attempt to conserve energy by putting "idle" processes or resources into a reduced or sleep state. Resuming quiesced resources takes time and the requesting process is blocked for that period. Make sure power saving options are disabled for the resources you need (such as CPUs and disks).

Although not specific to server resources, perhaps the most common cause of latency is queries that require **a sequential scan of extremely large tables** of data. Any query that must read through every record in a table will perform badly in proportion to the size of the table. Be sure to review the execution plans for key transactions to ensure indexes are used as expected and add indexes to avoid sequential scans wherever possible.

Chapter 3. Starting and Stopping the Database

The fundamental operations for database administration are starting and stopping the database. But before you start the database, you need to decide what database features you want to enable and how they should work. These features include attributes such as the amount of replication you want to use to increase availability in case of server failure and what level of durability is required for those cases where the database itself stops. These and other settings are defined in the configuration file, which you specify on the command line when you initialize the root directory for the database on each server.

This chapter explains how to configure the cluster's physical structure and features in the configuration file and how to initialize the root directory and start and stop the database.

3.1. Configuring the Cluster and Database

You specify the cluster configuration and what features to use in the configuration file, which is an XML file that you can create and edit manually. In the simplest case, the configuration file specifies how many partitions to create on each server, and what level of availability (K-safety) to use. For example:

```
<?xml version="1.0"?>
<deployment>
  <cluster sitesperhost="12"
           kfactor="1"
  />
</deployment>
```

- The `sitesperhost` attribute specifies the number of partitions (or "sites") to create on each server. Set to eight by default, it is possible to optimize the number of sites per host in relation to the number of processors per machine. The optimal number is best determined by performance testing against the expected workload. See the chapter on "Benchmarking" in the *VoltDB Planning Guide* for details.

- The `kfactor` attribute specifies the K-safety value to use. The higher the K-safety value, the more node failures the cluster can withstand without affecting database availability. However, increasing the K-safety value increases the number of copies of each unique partition. High availability is a trade-off between replication to protect against node failure and the number of unique partitions, therefore throughput performance. See the chapter on availability in the *Using VoltDB* manual for more information on determining an optimal K-safety value.

In addition to the sites per host and K-safety, you can use the configuration file to enable and configure specific database features such as export, command logging, and so on. The following table summarizes some of the key features that are settable in the configuration file.

Table 3.1. Selecting Database Features in the Configuration File

Feature	Example

Command Logging — Command logging provides durability by logging transactions to disk so they can be replayed during a recovery. You can configure the type of command logging (synchronous or asynchronous), the	`<commandlog enabled="true"`
	`synchronous="false">`
	`<frequency time="300"`
	`transactions="1000"/>`
	`</commandlog>`

Feature	Example
log file size, and the frequency of the logs (in terms of milliseconds or number of transactions).	

Snapshots — Automatic snapshot provide another form of durability by creating snapshots of the database contents, that can be restored later. You can configure the frequency of the snapshots, the unique file prefix, and how many snapshots are kept at any given time.	`<snapshot enabled="true"`
	`frequency="30m"`
	`prefix="mydb"`
	`retain="3" />`

Export — Export allows you to write selected records from the database to one or more external targets, which can be files, another database, or another service. VoltDB provides different export connectors for each protocol. You can configure the type of export for each stream as well as other properties, which are specific to the connector type. For example, the file connector requires a specific type (or format) for the files and a unique identifier called a "nonce".	`<export>`
	`<configuration target="dblog" type="file">`
	`<property name="type">csv</property>`
	`<property name="nonce">dblog</property>`
	`</configuration>`
	`</export>`

Security & Accounts — Security lets you protect your database against unwanted access by requiring all connections authenticate against known usernames and passwords. In the deployment file you can define the user accounts and passwords and what role or roles each user fulfills. Roles define what permissions the account has. Roles are defined in the database schema.	`<security enabled="true"/>`
	`<users>`
	`<user name="admin"`
	`password="superman"`
	`roles="administrator"/>`
	`<user name="mitty"`
	`password="thurber"`
	`roles="user,writer"/>`
	`</users>`

File Paths — Paths define where VoltDB writes any files or other disc-based content. You can configure specific paths for each type of service, such as snapshots, command logs, export overflow, etc.	`<paths>`
	`<exportoverflow path="/tmp/overflow" />`
	`<snapshots path="/opt/archive" />`
	`</paths>`

3.2. Initializing the Database Root Directory

Once you create the configuration file, you are ready to initialize the database root directory, using the **voltdb init** command. You issue this command on each node of the cluster, specifying the location for the root directory, the configuration file, license, and schema and stored procedure class files. There are defaults for each argument. But if you do specify the configuration, license, schema or classes you must specify the same values on every node of the cluster. For example:

```
$ voltdb init --dir=~/database         \ ❶
              --config=deployment.xml  \ ❷
              --license=~/license.xml  \ ❸
              --schema=myschema.sql    \ ❹
              --classes=myprocs.jar      ❺
```

On the command line, you can specify up to five arguments:

❶ The location where the root directory will be created
❷ The configuration file, which enables and sets attributes for specific VoltDB features
❸ The license file (when using the VoltDB Enterprise Edition)
❹ One or more SQL DDL files
❺ One or more JAR files containing stored procedure classes

When you initialize the root directory, VoltDB:

1. Creates the root directory (voltdbroot) as a subfolder of the specified parent directory

2. Saves the configuration and license, plus any schema and class files to preload, in the new root directory

Note that you only need to initialize the root directory once. Once the root directory is initialized, you can start and stop the database as needed. VoltDB uses the root directory to manage the current configuration options and backups of the data — if those features are selected — in command logs and snapshots. If you do not specify a license on the command line, VoltDB looks for a license in the current working directory, your home directory, or in the directory where the VoltDB software is installed and copies it into the root directory if it finds one.

If the root directory already exists or has been initialized before, you cannot re-initialize the directory unless you include the --force argument. This is to protect you against accidentally deleting data from a previous database session.

Important

Volt uses the root directory to store files vital to the operation and recovery of the database in case of failure. Many product features rely on information stored within the root directory and its subfolders. Do *not* manually add new or modify existing files within the directory structure. You can, judiciously, delete files as a maintenance activity (such as old log files or snapshots archived in numbered subfolders if and when you reinitialize an existing root directory). But even these activities are best handled automatically by setting the appropriate retention properties for each feature in the database configuration.

3.3. Starting the Database

Once you initialize the root directory, you are ready to start the database using the **voltdb start** command. You issue this command, specifying the location of the root directory, the number of servers required, and one or more server addresses to use as "host" to manage the initial formation of the cluster. You issue the same command on every node in the cluster. For example:

```
$ voltdb start --dir=~/database     \   ❶
               --count=5            \   ❷
               --host=svr1,svr2         ❸
```

On the command line, you specify four arguments:

❶ The location of the root directory
❷ The number of servers in the cluster
❸ One or more nodes from the cluster to use as the "host", to coordinate the initial startup of the cluster

You must specify the same number of servers and hosts (listed in exactly the same order) on all nodes of the cluster. You can, optionally, specify *all* nodes of the cluster in the --host argument. In which case, you can leave off the --count argument and VoltDB assumes the number of hosts is the total number of servers.

When you start the database, all nodes select one of the servers from the host list as the "host". The host then:

1. Waits until the necessary number of servers (as specified by the count) are connected

2. Creates the network mesh between the servers

3. Verifies that the configuration options match for all nodes

At this point, the cluster is fully initialized and the "host" ends its special role and becomes a peer to all the other nodes. If the database was run before and command logs or automated snapshots exist, the cluster now recovers the data from the previous session. All nodes in the cluster then write an informational message to the console verifying that the database is ready:

```
Server completed initialization.
```

3.4. Loading the Database Definition

Stored procedures are compiled into classes and then packaged into a JAR file, as described in the section on installing stored procedures in the *Using VoltDB* manual. To fully load the database definition you will need one or more JAR files of stored procedure classes and a text file containing the data definition language (DDL) statements that declare the database schema.

Responsibility for loading the database schema and stored procedures varies from company to company. In some cases, operators and administrators are only responsible for initiating the database; developers may load and modify the schema themselves. In other cases, the administrators are responsible for both starting the cluster and loading the correct database schema as well.

If the schema and stored procedures are predefined, you can include them when you initialize the database root directory and VoltDB will preload them when the database starts for the first time. Otherwise, you can load the schema and class files using the **sqlcmd** utlity after the database starts. The following sections describe each approach.

3.4.1. Preloading the Schema and Classes When You Initialize the Database

If the database schema is predefined, you can include it when you initialize the database root directory, using the `--schema` and `--classes` arguments to the **voltdb init** command. The `--schema` flag lets you specify one or more text files containing SQL DDL statements and the `--classes` flag lets you specify one or more JAR files containing the classes associated with any stored procedures you want to declare.

Note that DDL statements and Java classes can be order-dependent. For example, a stored procedure definition can depend on the existence of a table definition to define its partitioning column. VoltDB loads any classes before loading the schema file. However, you should be sure to specify the individual schema files or JAR files in the order you want them loaded.

Also, you must specify the same files, in the same order, when initializing all nodes of the cluster. For example:

```
$ voltdb init --dir=~/db \
              --schema=tables.sql,streams.sql,procs.sql    \
              --classes=globalprocs.jar,myprocs.jar
```

3.4.2. Loading the Schema and Classes After the Database Starts

If you are responsible for defining the correct schema once the database is running, or modifying an existing schema, you can do this using the **sqlcmd** utility. The following example assumes the schema is contained in two files: `storedprocs.jar` and `dbschema.sql`. Once the database cluster has started, you can start the **sqlcmd** utility and load the files at the sqlcmd prompt using the sqlcmd **load classes** and **file** directives:

```
$ sqlcmd
1> load classes storedprocs.jar;
2> file dbschema.sql;
```

Note that when loading the schema, you should always load the stored procedures first, so the class files are available for any CREATE PROCEDURE statements within the schema.

3.5. Stopping the Database

How you choose to stop a VoltDB depends on what features you have enabled. If you are using command logging (which is enabled by default in the VoltDB Enterprise Edition), it is a good idea to perform an orderly shutdown when stopping the database to ensure that all active client queries have a chance to complete and return their results (and no new queries start) before the shutdown occurs.

To perform an orderly shutdown you can use the **voltadmin shutdown** command:

```
$ voltadmin shutdown
```

As with all **voltadmin** commands, you can use them remotely by specifying one of the cluster servers on the command line:

```
$ voltadmin shutdown --host=voltsvr2
```

If security is enabled, you will also need to specify a username and password for a user with admin permissions:

```
$ voltadmin shutdown --host=voltsvr2 -u root -p Suda51
```

If you are not using command logging, you want to make sure you perform a snapshot before shutting down. You can do this manually using the **voltadmin save** command. Or you can simply add the **--save** argument to the **voltadmin shutdown** command:

```
$ voltadmin shutdown --save
```

The most recent snapshot saved to the database snapshots directory (by the **voltadmin save** command to the default location, automated snapshots, or **voltadmin shutdown --save**) will automatically be restored by the next **voltdb start** command.

3.6. Restarting the Database

Restarting a VoltDB database is done the same way as starting the database for the first time, except there is no need to initialize the root directory. You simply issue the same **voltdb start** command you did when you started it for the first time. For example:

```
$ voltdb start --dir=~/database     \
```

```
                --count=5             \
                --host=svr1,svr2
```

If you are using command logging, or you created a snapshot in the default snapshots directory, VoltDB automatically reinstates the data once the cluster is established. After the schema is loaded and all data is restored, the database enables client access.

3.7. Starting and Stopping Individual Servers

When using K-safety, it is possible for one or more nodes in a cluster to stop without stopping the database itself. (See the chapter on availability in the Using VoltDB manual for a complete description of K-safety.) If a server stops — either intentionally or accidentally — you can start the server and have it rejoin the cluster using the same **voltdb start** command used to start the cluster. For example:

```
$ voltdb start --dir=~/database     \
               --count=5            \
               --host=svr1,svr2
```

The start command will check to see if the cluster is still running, based on the list of servers in the `--host` argument. If so, the server will rejoin the cluster.

Note that if there are multiple servers listed in the `--host` argument, the server can rejoin even if it is one of the listed hosts. If you only list one host and that is the server that stopped, you will need to list a different server in the `--host` argument — any server that is still an active member of the running cluster. (This is why listing multiple nodes in the `--host` argument is beneficial: you can use exactly the same start command in multiple situations.)

If you want to stop a single node in a K-safe cluster — for example, to perform maintenance on the hardware — you can do this using the **voltadmin stop** command. The **voltadmin stop** command stops a single node, as long as the cluster has enough K-safety to remain viable after the nodes stops. (If not, the **stop** command is rejected.) For example to stop svr2, you can issue the following command:

```
$ voltadmin stop --host=svr1 svr2
```

Note that the stop command does not have to issued on the server that is being stopped. You can issue the command on any active server in the cluster. See Chapter 4, *Maintenance and Upgrades* for more information about performing maintenance tasks.

Chapter 4. Maintenance and Upgrades

Once the database is running, it is the administrator's role to keep it running. This chapter explains how to perform common maintenance and upgrade tasks, including:

- Database backups
- Schema and stored procedure updates
- System and hardware upgrades
- VoltDB software upgrades
- License updates

4.1. Backing Up the Database

It is a common safety precaution to backup all data associated with computer systems and store copies off-site in case of system failure or other unexpected events. Backups are usually done on a scheduled basis (every day, every week, or whatever period is deemed sufficient).

VoltDB provides several options for backing up the database contents. The easiest option is to save a native snapshot then backup the resulting snapshot files to removable media for archiving. The advantage of this approach is that native snapshots contain both a complete copy of the data and the schema. So in case of failure the snapshot can be restored to the current or another cluster using a single **voltadmin restore** command.

The key thing to remember when using native snapshots for backup is that each server saves its portion of the database locally. So you must fetch the snapshot files for all of the servers to ensure you have a complete set of files. The following example performs a manual snapshot on a five node cluster then uses scp to remotely copy the files from each server to a single location for archiving.

```
$ voltadmin save --blocking --host=voltsvr3 \
            /tmp/voltdb backup
$ scp -l 100 'voltsvr1:/tmp/voltdb/backup*' /tmp/archive/
$ scp -l 100 'voltsvr2:/tmp/voltdb/backup*' /tmp/archive/
$ scp -l 100 'voltsvr3:/tmp/voltdb/backup*' /tmp/archive/
$ scp -l 100 'voltsvr4:/tmp/voltdb/backup*' /tmp/archive/
$ scp -l 100 'voltsvr5:/tmp/voltdb/backup*' /tmp/archive/
```

Note that if you are using automated snapshots or command logging (which also creates snapshots), you can use the automated snapshots as the source of the backup. However, the automated snapshots use a programmatically generated file prefix, so your backup script will need some additional intelligence to identify the most recent snapshot and its prefix.

The preceding example also uses the scp limit flag (`-l 100`) to constrain the bandwidth used by the copy command to 100kbits/second. Use of the -l flag is recommended to avoid the copy operation blocking the VoltDB server process and impacting database performance.

Finally, if you wish to backup the data in a non-proprietary format, you can use the **voltadmin save --format=csv** command to create a snapshot of the data as comma-separated value (CSV) formatted text files. The advantage is that the resulting files are usable by more systems than just VoltDB. The disadvantage is that the CSV files only contain the data, not the schema. These files cannot be read directly into VoltDB,

like a native snapshot can. Instead, you will need to initialize and start a new database, load the schema, then use the **csvloader** utility to load individual files into each table to restore the database completely.

4.2. Updating the Database Schema

As an application evolves, the database schema often needs changing. This is particularly true during the early stages of development and testing but also happens periodically with established applications, as the database is tuned for performance or adjusted to meet new requirements. In the case of VoltDB, these updates may involve changes to the table definitions, to the indexes, or to the stored procedures. The following sections explain how to:

- Perform live schema updates
- Change unique indexes and partitioning using save and restore

4.2.1. Performing Live Schema Updates

There are two ways to update the database schema for a VoltDB database: live updates and save/restore updates. For most updates, you can update the schema while the database is running. To perform this type of live update, you use the DDL CREATE, ALTER, and DROP statements to modify the schema interactively as described in the section on modifying the schema in the *Using VoltDB* manual.

You can make any changes you want to the schema as long as the tables you are modifying do not contain any data. The only limitations on performing live schema changes are that you cannot:

- Add or broaden unique constraints (such as indexes or primary keys) on tables with existing data
- Reduce the datatype size of columns on tables with existing data (for example, changing the datatype from INTEGER to TINYINT)

These limitations are in place to guarantee that the schema change will succeed without any pre-existing data violating the constraint. If you know that the data in the database does not violate the new constraints you can make these changes using the **save** and **restore** commands, as described in the following section.

4.2.2. Performing Updates Using Save and Restore

If you need to add unique indexes or reduce columns to database tables with existing data, you must use the **voltadmin save** and **restore** commands to perform the schema update. This requires shutting down and restarting the database to allow VoltDB to validate the existing data against the new constraints.

To perform a schema update using save and restore, use the following steps:

1. Create a new schema file containing the updated DDL statements.
2. Pause the database (**voltadmin pause**).
3. Save a snapshot of the database contents to an specific location (**voltadmin save --blocking {path} {file-prefix}**).
4. Shutdown the database (**voltadmin shutdown**).
5. Re-initialize and restart the database starting in admin mode (**voltdb init --force** and **voltdb start --pause**).
6. Load the stored procedures and new schema (using the sqlcmd **LOAD CLASSES** and **FILE** directives)

7. Restore the snapshot created in Step #3 (**voltadmin restore {path} {file-prefix}**).

8. Return the database to normal operations (**voltadmin resume**).

For example:

```
$ # Issue once
$ voltadmin pause
$ voltadmin save --blocking /opt/archive/  mydb
$ voltadmin shutdown

$ # Issue next two commands on all servers
$ voltdb init  --dir=~/mydb --config=deployment.xml --force
$ voltdb start --dir=~/mydb --host=svr1,svr2 --count=5

$ # Issue only once
$ sqlcmd
1> load classes storedprocs.jar;
2> file newschema.sql;

3> exit
$ voltadmin restore /opt/archive  mydb
$ voltadmin resume
```

The key point to remember when adding new constraints is that there is the possibility that the restore operation will fail if existing records violate the new constraint. This is why it is important to make sure your database contents are compatible with the new schema before performing the update.

4.3. Upgrading the Cluster

Sometimes you need to update or reconfigure the server infrastructure on which the VoltDB database is running. Server upgrades are one example. A *server upgrade* is when you need to fix or replace hardware, update the operating system, or otherwise modify the underlying system.

Server upgrades usually require stopping the VoltDB database process on the specific server being serviced. However, if your database cluster uses K-safety for enhanced availability, it is possible to complete server upgrades without any database downtime by performing a *rolling hardware upgrade*, where each server is upgraded in turn using the **voltadmin stop** and **start** commands.

Another type of upgrade is when you want to reconfigure the cluster as a whole. Reasons for reconfiguring the cluster are because you want to add or remove servers from the cluster or you need to modify the number of partitions per server that VoltDB uses.

Adding and removing servers from the cluster can happen without stopping the database. This is called *elastic scaling*. Changing the K-Safety factor or number of sites per host requires restarting the cluster during a *maintenance window*.

The following sections describe five methods of cluster upgrade:

- Performing server upgrades
- Performing rolling upgrades on K-safe clusters
- Adding servers to a running cluster through elastic scaling
- Removing servers from a running cluster through elastic scaling

- Reconfiguring the cluster with a maintenance window

4.3.1. Performing Server Upgrades

If you need to upgrade or replace the hardware or software (such as the operating system) of the individual servers, this can be done without taking down the database as a whole. As long as the server is running with a K-safety value of one or more, it is possible to take a server out of the cluster without stopping the database. You can then fix the server hardware, upgrade software (other than VoltDB), even replace the server entirely with a new server, then bring the server back into the cluster.

To perform a server upgrade:

1. Stop the VoltDB server process on the server using the **voltadmin stop** command. As long as the cluster is K-safe, the rest of the cluster will continue running.

2. Perform the necessary upgrades.

3. Have the server rejoin the cluster using the **voltdb start** command.

The **start** command starts the database process on the server, contacts the database cluster, then copies the necessary partition content from other cluster nodes so the server can then participate as a full member of the cluster, While the server is rejoining, the other database servers remain accessible and actively process queries from client applications.

When rejoining a cluster you can use the same **start** command used when starting the cluster as a whole. If, however, you need to replace the server (say, for example, in the case of a disk failure), you will also need to initialize a root directory for the database process on the new machine. You do this using the current configuration file for the cluster. For example:

```
$ voltdb init --dir=~/database --config=deployment.xml
$ voltdb start --dir=~/database --host=svr1,svr2
```

If no changes have been made, you can use the same configuration file used to initialize the other servers. If you have used **voltadmin update** to change the configuration or changed settings using the Volt Management Center (VMC), you can download a copy of the latest configuration from VMC.

If the cluster is not K-safe — that is, the K-safety value is 0 — then you must follow the instructions in Section 4.3.5, “Reconfiguring the Cluster During a Maintenance Window” to upgrade the servers.

4.3.2. Performing Rolling Hardware Upgrades on K-Safe Clusters

If you need to upgrade all of the servers in a K-safe cluster (for example, if you are upgrading the operating system), you can perform a rolling hardware upgrade by stopping, upgrading, then rejoining each server one at a time. Using this process the entire cluster can be upgraded without suffering any downtime of the database. Just be sure to wait until the rejoining server has become a full member of the cluster before removing and upgrading the next server in the rotation. Specifically, wait until the following message appears in the log or on the console for the rejoining server:

```
Node rejoin completed.
```

Alternately, you can attempt to connect to the server remotely — for example, using the **sqlcmd** command line utility. If your connection is rejected, the rejoin has not finished. If you successfully connect to the client port of the rejoining node, you know the rejoin is complete:

```
$ sqlcmd --servers=myserver
SQL Command :: myserver:21212
1>
```

Note

You *cannot* update the VoltDB software itself using the rolling hardware upgrade process, only the operating system, hardware, or other software. See Section 4.4, "Upgrading Existing VoltDB Installations" for information about minimizing downtime during a VoltDB software upgrade.

4.3.3. Adding Servers to a Running Cluster with Elastic Scaling

If you want to add servers to a VoltDB cluster — usually to increase performance and/or capacity — you can do this without having to restart the database. You add servers to the cluster using the **voltdb start** command with the `--add` flag. Note, as always, you must initialize a root directory before issuing the **start** command. For example:

```
$ voltdb init  --dir=~/database --config=deployment.xml
$ voltdb start --dir=~/database --host=svr1,svr2 --add
```

The `--add` flag specifies that if the cluster full — that is, all of the specified number of servers are currently active in the cluster — the joining node can be added to elastically expand the cluster. You must elastically add a full complement of servers to match the K-safety value (K+1) before the servers can participate as active members of the cluster. For example, if the K-safety value is 2, you must add 3 servers before they actually become part of the cluster and the cluster rebalances its partitions.

When you add servers to a VoltDB database, the cluster performs the following actions:

1. The new servers are added to the cluster configuration and sent copies of the schema, stored procedures, and deployment file.

2. Once sufficient servers are added, copies of all replicated tables and their share of the partitioned tables are sent to the new servers.

3. As the data is rebalanced, the new servers begin processing transactions for the partition content they have received.

4. Once rebalancing is complete, the new servers are full members of the cluster.

If the cluster is *not* at its full complement of servers when you issue a **voltdb start --add** command, the added server will join the cluster as a replacement for a missing node rather than extending the cluster. Once the cluster is back to its full complement of nodes, the next **voltdb start --add** command will extend the cluster.

4.3.4. Removing Servers from a Running Cluster with Elastic Scaling

Just as you can add nodes to a running cluster to add capacity, you can remove nodes from a running cluster to reduce capacity. Obviously, you want to make sure that the smaller cluster has sufficient resources, such as memory, for your data and workload. If you are using K-safety, you also need to be sure the current cluster is large enough to remove nodes and still meet the requirements for your specific K-safety setting.

To remove nodes from a running cluster, you use the **voltadmin resize** command. The first step is to verify that the cluster has enough nodes to reduce in size. You do this with the **voltadmin resize --test** command:

```
$ voltadmin resize --test
```

The **voltadmin resize --test** command checks the cluster to make sure there are enough nodes to still be operational after the reduction and it reports which nodes will be removed as a result of the operation. The number of nodes that will be removed is calculated as the smallest number that allows the cluster to maintain K-safety. Without K-Safety, that is one node. With K-Safety, that is at least K+1, but possibly more depending on the cluster configuration. The remaining node count and configuration must satisfy the requirement that the number of nodes and the total number of partitions are both divisible by K+1.

Once you are ready to start reducing the cluster size, issue the **voltadmin resize** command without any arguments:

```
$ voltadmin resize
```

This command verifies that the cluster can be resized, reports which nodes will be removed, asks you to confirm that you want to begin, and then starts the resize operation. Because resizing the cluster involves reorganizing and rebalancing the partitions, it can take a significant amount of time, depending on the size of the database and the ongoing workload. You can track the progress of the resize operation using the **voltadmin status** command. You can also adjust the priority between rebalancing the partitions and ongoing client transactions by setting the duration and throughput of the rebalance operation. See the section on "Configuring How VoltDB Rebalances Nodes During Elastic Scaling" in the *Using VoltDB* manual for details.

Note that once resizing starts, you cannot cancel the operation. So be certain you want to reduce the size of the cluster before beginning. If for any reason the resize operation fails unexpectedly, you can use the **voltadmin resize --retry** command to restart the cluster reduction.

4.3.5. Reconfiguring the Cluster During a Maintenance Window

If you want to modify the cluster configuration, such as the number of sites per host or K-Safety factor, you need to restart the database cluster as a whole. You can also choose to add or remove nodes from the cluster during this operation. Stopping the database temporarily to reconfigure the cluster is known as a *maintenance window*.

The steps for reconfiguring the cluster with a maintenance window are:

1. Place the database in admin mode (**voltadmin pause**).
2. Perform a manual snapshot of the database (**voltadmin save --blocking**).
3. Shutdown the database (**voltadmin shutdown**).
4. Make the necessary changes to the configuration file.
5. Reinitialize the database root directory on all nodes specifying the edited configuration file (**voltdb init --force**).
6. Start the new database in admin mode (**voltdb start --pause)**
7. Restore the snapshot created in Step #2 (**voltadmin restore**).

8. Return the database to normal operations (**voltadmin resume**).

4.4. Upgrading Existing VoltDB Installations

As new versions of VoltDB become available, you will want to upgrade the VoltDB software on your database cluster. The simplest approach for upgrading recent versions of VoltDB — V6.8 or later — is to perform an orderly shutdown saving a final snapshot, upgrade the software on all servers, then re-start the database. (If you are upgrading from earlier versions of the software, you can still upgrade using a snapshot. But you will need to perform the save and restore operations manually.)

However, upgrading using snapshots involves downtime while the software is being updated. Two alternatives for upgrading VoltDB without downtime are in-service upgrades — upgrading nodes of the cluster one at a time — and using cross data center replication (XDCR) to upgrade clusters.

An in-service upgrade (a separately licensable feature of VoltDB) lets you upgrade a single running cluster by removing individual nodes, upgrading the VoltDB software, then rejoining the node to the cluster. The cluster continues to process transactions throughout the upgrade process, During the upgrade, the cluster operates as the older version software. Once all of the nodes are upgraded, the cluster transitions to the new version.

Using cross data center replication (XDCR), it is possible to use two or more clusters to perform an *online upgrade*, where there is no downtime and the database is accessible throughout the upgrade operation. If two or more clusters are already active participants in an XDCR environment, you can shutdown and upgrade the clusters, one at a time, to perform the upgrade leaving at least one cluster available at all times.

The following sections describe four approaches to upgrading existing VoltDB installations, starting with how to replace the software itself:

- Upgrading the VoltDB Software
- Upgrading VoltDB Using Save and Restore
- Upgrading Older Versions of VoltDB Manually
- Performing an In-Service Upgrade of a Single Cluster
- Performing an Online Upgrade Using Multiple XDCR Clusters

4.4.1. Upgrading the VoltDB Software

Updating the VoltDB software is very simple. However, you must make sure you perform this step at the right stage in the upgrade process, as described in the following sections. The product comes as a .tar.gz file. When the time comes to upgrade the software, you unpack the tar file and move the resulting folder to replace your current installation. For example, if you have the VoltDB software installed as /var/voltdb, the software installation looks like the following, where you delete the previous version and replace it with the new one:

```
$ tar -zxvf voltdb-ent-n.n.n-xxxx.tar.gz -C /var
$ cd /var
$ rm -vr voltdb
$ mv voltdb-ent-n.n.n-xxxx voltdb
```

Remember, when upgrading an existing installation with a running database, you need to upgrade both the software *and* the database itself. Which means you must make sure you perform the update steps in

the correct order. The following sections explain the different options for updating existing installations, including at what stage in the process you should replace the software.

4.4.2. Upgrading VoltDB Using Save and Restore

Upgrading the VoltDB software on a single database cluster is easy. All you need to do is perform an orderly shutdown saving a final snapshot, upgrade the VoltDB software on all servers in the cluster, then restart the database. The steps to perform this procedure are:

1. Shutdown the database and save a final snapshot (**voltadmin shutdown --save**).
2. Upgrade the VoltDB software on all cluster nodes (instructions).
3. Restart the database (**voltdb start**).

This process works for any recent (V6.8 or later) release of VoltDB.

4.4.3. Upgrading Older Versions of VoltDB Manually

To upgrade older versions of VoltDB software (prior to V6.8), you must perform the save and restore operations manually. The steps when upgrading from older versions of VoltDB are:

1. Place the database in admin mode (**voltadmin pause**).
2. Perform a manual snapshot of the database (**voltadmin save --blocking**).
3. Shutdown the database (**voltadmin shutdown**).
4. Upgrade the VoltDB software on all cluster nodes (instructions).
5. Re-initialize the root directory on all nodes (**voltdb init --force**).
6. Start a new database in admin mode (**voltdb start --pause**).
7. Restore the snapshot created in Step #2 (**voltadmin restore**).
8. Return the database to normal operations (**voltadmin resume**).

4.4.4. Performing an In-Service Upgrade of a Single Cluster

Normally, when upgrading the VoltDB software, you must shutdown the cluster (for example, with the **voltadmin shutdown --save** command) and restart the entire cluster using the new software. Downtime can be avoided by performing an *in-service upgrade*. An in-service upgrade allows a K-safe cluster to be upgraded one node at a time, rather than the entire cluster all at once. This means the cluster, and the business processes it supports, remain available throughout the upgrade procedure.

The requirements for performing an in-service upgrade are:

- The cluster has the appropriate license for VoltDB that includes the In-Service Upgrade feature.
- The cluster must be K-safe. That is, the cluster has a K-safety factor of one or more. This is required so individual nodes can be stopped without crashing the cluster.
- The cluster must be running VoltDB V13.1.0 or later.

- The new version falls within the parameters allowed by in-service upgrades, as described in Section 4.4.4.1, “The Scope of In-Service Upgrades”.

To perform an in-service upgrade on bare metal servers, you upgrade the VoltDB software on each node consecutively. Specifically:

1. Stop one of the cluster nodes, using the **voltadmin stop node** command
2. Once the server process stops, replace the VoltDB software with the new version.
3. Restart the node using the **voltdb start** command, specifying one or more of the other nodes in the cluster as hosts.
4. Once the rejoin process is finished and the cluster is complete, repeat the process for the next node until all nodes are upgraded.

During the upgrade process, you can determine which nodes have been updated using the @SystemInformation system procedure with the OVERVIEW selector and looking for the VERSION keyword. For example, in the following command output, the first column is the host ID and the last column is the currently installed software version for that host. Once all hosts report using the upgraded software version, the upgrade is complete.

```
$ echo "exec @SystemInformation overview" | sqlcmd | grep VERSION
       2 VERSION                   13.1.2
       1 VERSION                   13.1.2
       0 VERSION                   13.1.3
```

Until the upgrade process is complete, all nodes in the cluster maintain the functionality of the lower version, even for those nodes that have already upgraded to the higher version software. Once the upgrade is complete and all nodes are running on the newer version, the cluster switches to operating with the higher version functionality. In other words, if the new software contains any new function or behavior, that feature will not be accessible until the entire in-service upgrade process is complete.

If the upgrade fails for any reason, or you choose to stop the upgrade midway, you can revert to the original version by reversing the process: removing a node that has been upgraded, replace the VoltDB software with the original version, rejoin the node and repeat for all nodes that were upgraded. Once the upgrade process is complete, the in-service upgrade is over. At which point, you can longer return to the previous version through an in-service upgrade and must perform a full cluster restart to downgrade.

4.4.4.1. The Scope of In-Service Upgrades

There are limits to which software versions can use in-service upgrades. The following table describes the rules for which releases can be upgraded with an in-service upgrade and which releases cannot.

✔ Patch Releases	You can upgrade between any two *patch releases*. That is, any two releases where only the third and final number of the version identifier changes. For example, upgrading from 13.1.1 to 13.1.4.
✔ Minor Releases	You can also use in-service upgrades to upgrade between two consecutive *minor releases*. That is where the second number in the version identifier differ. For example, you can upgrade from V13.2.0 to V13.3.0. You can also upgrade between any patch releases within those minor releases. For example, upgrading from V13.2.3 to V13.3.0. You *cannot* use in-service upgrades to upgrade more than one minor version at a time. In other words, you can upgrade from V13.2.0 to V13.3.0 but you cannot

perform an in-service upgrade from V13.2.0 to V13.4.0. To transition across multiple minor releases your options are to perform consecutive in-service upgrades (for example, from V13.2.0 to V13.3.0, then from V13.3.0 to V13.4.0) or to perform a regular upgrade where all cluster nodes are upgrading at one time.

✗ Major Releases — You *cannot* use in-service upgrades between major versions of VoltDB. That is, where the first number in the version identifier is different. For example, you must perform a full cluster upgrade when migrating from V13.x.x to V14.0.0 or later.

4.4.5. Performing an Online Upgrade Using Multiple XDCR Clusters

It is also possible to upgrade the VoltDB software using cross data center replication (XDCR), by simply shutting down, upgrading, and then re-initalizing each cluster, one at a time. This process requires no downtime, assuming your client applications are already designed to switch between the active clusters.

Use of XDCR for upgrading the VoltDB software is easiest if you are already using XDCR because it does not require any additional hardware or reconfiguration. The following instructions assume that is the case. Of course, you could also create a new cluster and establish XDCR replication between the old and new clusters just for the purpose of upgrading VoltDB. The steps for the upgrade outlined in the following sections are the same. But first you must establish the cross data center replication between the two (or more) clusters. See the chapter on Database Replication in the *Using VoltDB* manual for instructions on completing this initial step.

Once you have two clusters actively replicating data with XCDCR (let's call them clusters A and B), the steps for upgrading the VoltDB software on the clusters is as follows:

1. Pause and shutdown cluster A (**voltadmin pause --wait** and **shutdown**).
2. Clear the DR state on cluster B (**voltadmin dr reset**).
3. Update the VoltDB software on cluster A.
4. Start a new database instance on A, making sure to use the old deployment file so the XDCR connections are configured properly (**voltdb init --force** and **voltdb start**).
5. Load the schema on Cluster A so replication starts.
6. Once the two clusters are synchronized, repeat steps 1 through 4 for cluster B.

Note that since you are upgrading the software, you must create a new instance after the upgrade (step #3). When upgrading the software, you cannot recover the database using just the **voltdb start** command; you must use **voltdb init --force** first to create a new instance and then reload the existing data from the running cluster B.

Also, be sure all data has been copied to the upgraded cluster A after step #4 and before proceeding to upgrade the second cluster. You can do this by checking the @Statistics system procedure selector DRCONSUMER on cluster A. Once the DRCONSUMER statistics `State` column changes to "RECEIVE", you know the two clusters are properly synchronized and you can proceed to step #5.

4.4.5.1. Falling Back to a Previous Version

In extreme cases, you may decide after performing the upgrade that you do not want to use the latest version of VoltDB. If this happens, it is possible to fall back to the previous version of VoltDB.

To "downgrade" from a new version back to the previous version, follow the steps outlined in Section 4.4.5, “Performing an Online Upgrade Using Multiple XDCR Clusters” except rather than upgrading to the new version in Step #2, reinstall the older version of VoltDB. This process is valid as long as you *have not modified the schema or deployment to use any new or changed features introduced in the new version*.

4.4.6. Downgrading, or Falling Back to a Previous VoltDB Version

The section describing the upgrade process for active XDCR explains how to fall back to the previous version of VoltDB in case of emergency. This section explains how to fall back, or downgrade, when using the standard save and restore process described in Section 4.4.2, “Upgrading VoltDB Using Save and Restore”.

The following process works *if* you are reverting between two recent versions of VoltDB *and* you do not use any new features between the upgrade and the downgrade. There are no guarantees an attempt to downgrade will succeed if the two software versions are more than one major version apart or if you utilize a new feature from the higher version software prior to downgrading.

With those caveats, the most reliable way to fall back to a previous VoltDB version is:

1. Extract the database schema and stored procedure classes
2. Pause the database, save a snapshot, and shutdown
3. Re-install the previous version of VoltDB
4. Initialize a new database root directory, using the extracted schema and classes
5. Start the new database instance (in pause mode) using the older version of VoltDB
6. Manually restore the data from the snapshot created in Step #2
7. Resume normal operations

This process ensures that only the schema, stored procedures, and data are returned to the older version of the software, and new software features will not impact your restore process. For example:

```
$ voltdb get schema -D ~/db/new --output=/tmp/mydb.sql
$ voltdb get classes -D ~/db/new --output=/tmp/mydb.jar
$ voltadmin pause
$ voltadmin save /tmp mydata
$ voltadmin shutdown

[ downgrade VoltDB software . . . ]

$ voltdb init -f -D ~/db/old --schema=/tmp/mydb.sql --classes=/top/mydb.jar
$ voltdb start -D ~/db/old --pause &
$ voltadmin restore /tmp mydata
$ voltadmin resume
```

4.5. Updating the VoltDB Software License

The VoltDB Enterprise Edition is licensed software. Once the license expires, you will not be able to restart your database cluster without a new license. So it is a good idea to update the license before it expires to avoid any interruption to your service.

You can use the **voltadmin show license** command to see information about your current license, including the expiration date. You can then use the **voltadmin license** command to replace the current license with a new license file.

```
$ voltadmin license newlicense.xml
INFO: The license is updated successfully.
 . . .
```

When you issue the **show license** command, VoltDB verifies that the license file is valid and the terms of the license are sufficient to support the current database configuration. Once verified, the license is applied to all nodes of the cluster and information about the new license is displayed.

If a node fails to get updated (for example, if a node fails during the license update), you will need to update that node independently when bringing it back into the cluster. You can do this by including the new license file on the command line when you restart the node. For example:

```
$ voltdb start -D ~/mydb --license newlicense.xml
Initializing VoltDB...
```

Chapter 5. Monitoring VoltDB Databases

Monitoring is an important aspect of systems administration. This is true of both databases and the infrastructure they run on. The goals for database monitoring include ensuring the database meets its expected performance target as well as identifying and resolving any unexpected changes or infrastructure events (such as server failure or network outage) that can impact the database. This chapter explains:

- How to monitor overall database health and performance using VoltDB
- How to automatically pause the database when resource limits are exceeded
- How to integrate VoltDB monitoring with Prometheus

5.1. Monitoring Overall Database Activity

VoltDB provides several tools for monitoring overall database activity. The following sections describe the three primary monitoring tools within VoltDB:

- Volt Management Center
- System Procedures
- SNMP Alerts

5.1.1. Volt Management Center

```
http://voltserver:8080/
```

The Volt Management Center provides a graphical display of key aspects of database performance, including throughput, memory usage, query latency, and partition usage. To use the Management Center, connect to one of the cluster nodes using a web browser, specifying the HTTP port (8080 by default) as shown in the example URL above. The Management Center shows graphs for cluster throughput and latency as well as CPU and memory usage for the current server. You can also use the Management Center to examine the database schema and to issue ad hoc SQL queries.

5.1.2. System Procedures

VoltDB provides callable system procedures that return detailed information about the usage and performance of the database. In particular, the @Statistics system procedure provides a wide variety of information depending on the selector keyword you give it. Some selectors that are particularly useful for monitoring include the following:

- **MEMORY** — Provides statistics about memory usage for each node in the cluster. Information includes the resident set size (RSS) for the server process, the Java heap size, heap usage, available heap memory, and more. This selector provides the type of information displayed by the Process Memory Report, except that it returns information for all nodes of the cluster in a single call.
- **PROCEDUREPROFILE** — Summarizes the performance of individual stored procedures. Information includes the minimum, maximum, and average execution time as well as the number of invocations, failures, and so on. The information is summarized from across the cluster as whole. This selector returns information similar to the latency graph in Volt Management Center.
- **TABLE** — Provides information about the size, in number of tuples and amount of memory consumed, for each table in the database. The information is segmented by server and partition, so you can use

it to report the total size of the database contents or to evaluate the relative distribution of data across the servers in the cluster.

When using the @Statistics system procedure with the PROCEDUREPROFILE selector for monitoring, it is a good idea to set the second parameter of the call to "1" so each call returns information since the last call. In other words, statistics for the interval since the last call. Otherwise, if the second parameter is "0", the procedure returns information since the database started and the aggregate results for minimum, maximum, and average execution time will have little meaning.

When calling @Statistics with the MEMORY or TABLE selectors, you can set the second parameter to "0" since the results are always a snapshot of the memory usage and table volume at the time of the call. For example, the following Python script uses @Statistics with the MEMORY and PROCEDUREPROFILE selectors to check for memory usage and latency exceeding certain limits. Note that the call to @Statistics uses a second parameter of 1 for the PROCEDUREPROFILE call and a parameter value of 0 for the MEMORY call.

```
import sys
from voltdbclient import *

nano = 1000000000.0
memorytrigger = 4 * (1024*1024)     # 4gbytes
avglatencytrigger = .01 * nano      # 10 milliseconds
maxlatencytrigger = 2 * nano        # 2 seconds

server = "localhost"
if (len(sys.argv) > 1): server = sys.argv[1]

client = FastSerializer(server, 21212)
stats = VoltProcedure( client, "@Statistics",
   [ FastSerializer.VOLTTYPE_STRING,
     FastSerializer.VOLTTYPE_INTEGER ] )

# Check memory
response = stats.call([ "memory", 0 ])
for t in response.tables:
   for row in t.tuples:
      print 'RSS for node ' + row[2] + "=" + str(row[3])
      if (row[3] > memorytrigger):
         print "WARNING: memory usage exceeds limit."

# Check latency
response = stats.call([ "procedureprofile", 1 ])
avglatency = 0
maxlatency = 0
for t in response.tables:
   for row in t.tuples:
      if (avglatency < row[4]): avglatency = row[4]
      if (maxlatency < row[6]): maxlatency = row[6]
print 'Average latency= ' + str(avglatency)
print 'Maximum latency= ' + str(maxlatency)
if (avglatency > avglatencytrigger):
   print "WARNING: Average latency exceeds limit."
if (maxlatency > maxlatencytrigger):
   print "WARNING: Maximum latency exceeds limit."
```

```
client.close()
```

The @Statistics system procedure is the the source for many of the monitoring options discussed in this chapter. Two other system procedures, @SystemCatalog and @SystemInformation, provide general information about the database schema and cluster configuration respectively and can be used in monitoring as well.

The system procedures are useful for monitoring because they let you customize your reporting to whatever level of detail you wish. The other advantage is that you can automate the monitoring through scripts or client applications that call the system procedures. The downside, of course, is that you must design and create such scripts yourself. As an alternative for custom monitoring, you can consider integrating VoltDB with existing third party monitoring applications, as described in Section 5.3, “Integrating VoltDB with Prometheus”. You can also set the database to automatically pause if certain system resources run low, as described in the next section.

5.1.3. SNMP Alerts

In addition to monitoring database activity on a "as needed" basis, you can enable VoltDB to proactively send Simple Network Management Protocol (SNMP) alerts whenever important events occur within the cluster. SNMP is a standard for how SNMP agents send messages (known as "traps") to management servers or "management stations".

SNMP is a lightweight protocol. SNMP traps are sent as UDP broadcast messages in a standard format that is readable by SNMP management stations. Since they are broadcast messages, the sending agent does not wait for a confirmation or response. And it does not matter, to the sender, whether there is a management server listening to receive the message or not. You can use any SNMP-compliant management server to receive and take action based on the traps.

When you enable SNMP in the deployment file, VoltDB operates as an SNMP agent sending traps whenever management changes occur in the cluster. You enable SNMP with the `<snmp>` element in the deployment file. You configure how and where VoltDB sends SNMP traps using one or more of the attributes listed in Table 5.1, “SNMP Configuration Attributes”.

Table 5.1. SNMP Configuration Attributes

Attribute	Default Value	Description
target	(none)	Specifies the IP address or host name of the SNMP management station where traps will be sent in the form *{IP-or-host-name}[:port-number]*. If you do not specify a port number, the default is 162. The `target` attribute is required.
community	public	Specifies the name of the "community" the VoltDB agent belongs to.
username	(none)	Specifies the username for SNMP V3 authentication. If you do not specify a username, VoltDB sends traps in SNMP V2c format. If you specify a username, VoltDB uses SNMP V3 and the following attributes let you configure the authentication mechanisms used.
authprotocol	SHA (SNMP V3 only)	Specifies the authentication protocol for SNMP V3. Allowable options are: • SHA • MD5 • NoAuth

Attribute	Default Value	Description
authkey	voltdbauthkey (SNMP V3 only)	Specifies the authentication key for SNMP V3 when the protocol is other than NoAuth.
privacyprotocol	AES (SNMP V3 only)	Specifies the privacy protocol for SNMP V3. Allowable options are: • AES • DES • NoPriv • 3DES* • AES192* • AES256*
privacykey	voltdbprivacykey (SNMP V3 only)	Specifies the privacy key for SNMP V3 when the privacy protocol is other than NoPriv.

*Use of 3DES, AES192, or AES256 privacy requires the Java Cryptography Extension (JCE) be installed on the system. The JCE is specific to the version of Java you are running. See the the Java web site for details.

SNMP is enabled by default when you include the `<snmp>` element in the deployment file. Alternately, you can explicitly enable and disable SNMP using the `enabled={true|false}` attribute to the element. For example, the following deployment file entry enables SNMP alerts, sending traps to mgtsvr.mycompany.com using SNMP V3 with the username "voltdb":

```
<snmp enabled="true"
      target="mgtsvr.mycompany.com"
      username="voltdb"
/>
```

Once SNMP is enabled, VoltDB sends alerts for the events listed in Table 5.2, “SNMP Events”.

Table 5.2. SNMP Events

Name	Severity	Description
crash	FATAL	When a server or cluster crashes.
clusterPaused	INFO	When the cluster pauses and enters admin mode.
clusterResume	INFO	When the cluster exits admin mode and resumes normal operation.
hostDown	ERROR	When a server shuts down or is recognized as having left the cluster.
hostUp	INFO	When a server joins the cluster.
streamBlocked	WARN	When an export stream is blocked due to data missing from the export queue and all cluster nodes are running.
statisticsTrigger	WARN	When certain operational states are compromised. Specifically: • When a K-safe cluster loses one or more nodes • When using database replication, the connection to the remote cluster is broken
resourceTrigger	WARN	When certain resource limits are exceeded. Specifically • Memory usage • Disk usage

Name	Severity	Description
		See Section 5.2, “Setting the Database to Read-Only Mode When System Resources Run Low” for more information about configuring SNMP alerts for resources.
resourceClear	INFO	When resource limits return to levels below the trigger value.

For the latest details about each event trap, see the VoltDB SNMP Management Information Base (MIB), which is installed with the VoltDB server software in the file `/tools/snmp/VOLTDB-MIB` in the installation directory.

5.2. Setting the Database to Read-Only Mode When System Resources Run Low

VoltDB, like all software, uses system resources to perform its tasks. First and foremost, as an in-memory database, VoltDB relies on having sufficient memory available for storing the data and processing queries. However, it also makes use of disk resources for snapshots and caching data for other features, such as export and database replication.

If system resources run low, one or more nodes may fail impacting availability, or worse, causing a service interruption. The best solution for this situation is to plan ahead and provision sufficient resources for your needs. The goal of the *VoltDB Planning Guide* is to help you do this.

However, even with the best planning, unexpected conditions can result in resource shortages or overuse. In these situations, you want the database to protect itself against all-out failure.

You can do this by setting resource limits in the VoltDB deployment file. System resource limits are set within the `<systemsettings>` and `<resourcemonitor>` elements. For example:

```
<systemsettings>
  <resourcemonitor frequency="30">
     <memorylimit size="70%" alert="60%"/>
     <disklimit>
        <feature name="snapshots" size="75%" alert="60%"/>
        <feature name="droverflow" size="60%"/>
     </disklimit>
  </resourcemonitor>
</systemsettings>
```

The deployment file lets you set limits on two types of system resources:

- Memory Usage
- Disk Usage

For each resource type you can set the maximum size and, optionally, the level at which an alert is sent if SNMP is enabled. In all cases, the allowable amount of the resource to be used can be specified as either a value representing a number of gigabytes or a percentage of the total available. If the limit set by the `alert` attribute is exceeded and SNMP is enabled, an SNMP alert is sent. If the limit set by the `size` attribute is exceeded, the database will be "paused", putting it into read-only mode to avoid using any further resources or possibly failing when the resource becomes exhausted. When the database pauses, an error message is written to the log file (and the console) reporting the event. This allows you as the system administrator to correct the situation by reducing memory usage or deleting unnecessary files. Once sufficient resources are freed up, you can return the database to normal operation using the **voltadmin resume** command.

The resource limits are checked every 60 seconds by default. However, you can adjust how frequently they are checked — to accommodate the relative stability or volatility of your resource usage — using the `frequency` attribute of the `<resourcemonitor>` tag. In the preceding example, the frequency has been reduced to 30 seconds.

Of course, the ideal is to catch excessive resource use *before* the database is forced into read-only mode. Use of SNMP and system monitors such as Nagios and New Relic to generate alerts at limits lower than the VoltDB resource monitor are strongly recommended. And you can integrate other VoltDB monitoring with these monitoring utilities as described in Section 5.3, “Integrating VoltDB with Prometheus”. But the resource monitor `size` limit is provided as a last resort to ensure the database does not completely exhaust resources and crash before the issue can be addressed.

The following sections describe how to set limits for the individual resource types.

5.2.1. Monitoring Memory Usage

You specify a memory limit in the deployment file using the `<memorylimit>` element and specifying the maximum allowable resident set size (RSS) for the VoltDB process in the `size` attribute. You can express the limit as a fixed number of gigabytes or as a percentage of total available memory. Use a percent sign to specify a percentage.

In addition to pausing the database, you can specify that it runs a full compaction of table data to recover whatever unused space is available due to fragmentation. This is the equivalent of running the **voltadmin defrag --full** command manually. By setting the `compact` attribute to true, when the memory limit is exceeded, the database will pause, defragment all table data on the affected node, and if enough space is recovered to bring memory usage down under the limit, the database will automatically resume normal operation. See the chapter on "Understanding Memory Usage" in the *Volt Performance and Customization* guide for more information about memory compaction.

For example, the following setting will cause the VoltDB database to go into read-only mode and perform a full compaction if the RSS size exceeds 10 gigabytes on any of the cluster nodes.

```
<systemsettings>
  <resourcemonitor>
     <memorylimit size="10" compact="true"/>
  </resourcemonitor>
</systemsettings>
```

Whereas the following example sets the limit at 70% of total available memory but does not automatically compact memory used for table data.

```
<systemsettings>
  <resourcemonitor>
     <memorylimit size="70%"/>
  </resourcemonitor>
</systemsettings>
```

You can also set a trigger value for SNMP alerts — assuming SNMP is enabled — using the `alert` attribute. For instance, the following example sets the SNMP trigger value to 60%.

```
<systemsettings>
  <resourcemonitor>
     <memorylimit size="70%" alert="60%"/>
  </resourcemonitor>
</systemsettings>
```

If you do not specify a limit in the deployment file, VoltDB automatically sets a maximum size limit of 80% and an SNMP alert level of 70% by default.

5.2.2. Monitoring Disk Usage

You specify disk usage limits in the deployment file using the `<disklimit>` element. Within the `<disklimit>` element, you use the `<feature>` element to identify the limit for a device based on the VoltDB feature that utilizes it. For example, to set a limit on the amount of space used on the device where automatic snapshots are stored, you identify the feature as "snapshots" and specify the limit as a number of gigabytes or as a percentage of total space on the disk. The following deployment file entry sets the disk limit for snapshots at 200 gigabytes and the limit for command logs at 70% of the total available space:

```
<systemsettings>
  <resourcemonitor>
     <disklimit>
        <feature name="snapshots" size="200"/>
        <feature name="commandlog" size="70%"/>
     </disklimit>
  </resourcemonitor>
</systemsettings>
```

You can also set a trigger value for SNMP alerts — assuming SNMP is enabled — using the `alert` attribute. For instance, the following example sets the SNMP trigger value to 150 gigabytes for the snapshots disk and 60% for the commandlog disk.

```
<systemsettings>
  <resourcemonitor>
     <disklimit>
        <feature name="snapshots" size="200" alert="150"/>
        <feature name="commandlog" size="70%" alert="60%"/>
     </disklimit>
  </resourcemonitor>
</systemsettings>
```

Note that you specify the device based on the feature that uses it. However, the limits applies to *all* data on that device, not just the space used by that feature. If you specify limits for two features that use the same device, the lower of the two limits will be applied. So, in the previous example, if snapshots and command logs both use a device with 250 gigabytes of total space, the database will be set to read-only mode if the total amount of used space exceeds the command logs limit of 70%, or 175 gigabytes.

It is also important to note that there are *no* default resource limits or alerts for disks. If you do not explicitly specify a disk limit, there is no protection against running out of disk space. Similarly, unless you explicitly set an SNMP alert level, no alerts will be sent for the associated device.

You can identify disk limits and alerts for any of the following VoltDB features, using the specified keywords:

- Automated snapshots (snapshots)
- Command logs (commandlog)
- Command log snapshots (commandlogsnapshot)
- Database replication overflow (droverflow)
- Export overflow (exportoverflow)

5.3. Integrating VoltDB with Prometheus

If you use Prometheus to monitor your systems and services, you can enable the collection and reporting of Prometheus-compliant metrics on the database cluster. You enable Prometheus metrics in the configuration file when initializing the database by adding the `<metrics>` element to the Volt configuration file:

```
<deployment>
   <cluster kfactor="1"/>
   <metrics enabled="true"/>
</deployment>
```

Next, add the Volt cluster nodes as targets in the Prometheus configuration. Since each node reports its own data, be sure to include all of the nodes as scraping targets. For example:

```
global:
  scrape_interval:     15s
  evaluation_interval: 15s

scrape_configs:
  - job_name: volt
    static_configs:
      - targets: ['voltsvr1:11781','voltsvr2:11781','voltsvr3:11781']
```

Once metrics are enabled, each Volt server reports server-specific information through the Prometheus endpoint (/metrics) on the metrics port, which defaults to 11781. You can specify an alternate port and/or network interface using the **--metrics** qualifier on the **voltdb start** command.

The Prometheus data format is a readily accessible text format and can be used equally well by other reporting applications. Applications can either send HTTP requests to the metrics endpoint like Prometheus or use the @Metrics system procedure, which returns the same data formatted in a sequence of VoltTable structures. Appendix B, *Volt Active Data Metrics* lists the metrics values reported VoltDB.

Once Prometheus is scraping the Volt metrics, you can use tools such as Grafana to combine, analyze, and present the information in meaningful ways. There are example Grafana dashboards in the Volt Github repository (https://github.com/VoltDB/volt-monitoring) demonstrating some of the visualizations that are possible.

Chapter 6. Logging and Analyzing Activity in a VoltDB Database

VoltDB uses Log4J as an open source logging service to provide access to information about database events. In actuality, the library used is Reload4j, which is a drop-in replacement for Log4J that corrects known security vulnerabilities in the original library while maintaining all of the same package names. Consequently, the commands, examples, and following documentation continue to refer to the service itself as "Log4J".

By default, when using the VoltDB shell commands, the console display is limited to warnings, errors, and messages concerning the status of the current process. A more complete listing of messages (of severity INFO and above) is written to log files in the subfolder `/log`, relative to the database root directory.

The advantages of using Log4J are:

- Logging is compiled into the code and can be enabled and configured at run-time.
- Log4J provides flexibility in configuring what events are logged, where, and the format of the output.
- By using an open source logging service with standardized output, there are a number of different applications, such as Chainsaw, available for filtering and presenting the results.

Logging is important because it can help you understand the performance characteristics of your application, check for abnormal events, and ensure that the application is working as expected.

Of course, any additional processing and I/O will have an incremental impact on the overall database performance. To counteract any negative impact, Log4J gives you the ability to customize the logging to support only those events and servers you are interested in. In addition, when logging is not enabled, there is no impact to VoltDB performance. With VoltDB, you can even change the logging profile on the fly without having to shutdown or restart the database.

The following sections describe how to enable and customize logging of VoltDB using Log4J. This chapter is **not** intended as a tutorial or complete documentation of the Log4J logging service. For general information about Log4J, see the Log4J web site at http://wiki.apache.org/logging-log4j/.

6.1. Introduction to Logging

Logging is the process of writing information about application events to a log file, console, or other destination. Log4J uses XML files to define the configuration of logging, including three key attributes:

- **Where** events are logged. The destinations are referred to as *appenders* in Log4J (because events are appended to the destinations in sequential order).
- **What** events are logged. VoltDB defines named classes of events (referred to as *loggers*) that can be enabled as well as the severity of the events to report.
- **How** the logging messages are formatted (known as the *layout*),

6.2. Creating the Logging Configuration File

VoltDB ships with a default Log4J configuration file, voltdb/log4j.xml, in the installation directory. The VoltDB shell commands use this file to configure logging and it is recommended for new application

development. This default Log4J file lists all of the VoltDB-specific logging categories and can be used as a template for any modifications you wish to make. Or you can create a new file from scratch.

The following is an example of a Log4J configuration file:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">

<appender name="Async" class="org.apache.log4j.AsyncAppender">
  <param name="Blocking" value="true" />
  <appender-ref ref="Console" />
  <appender-ref ref="File" />
</appender>

<appender name="Console" class="org.apache.log4j.ConsoleAppender">
  <param name="Target" value="System.out" />
  <layout class="org.apache.log4j.TTCCLayout" />
</appender>

<appender name="File" class="org.apache.log4j.FileAppender">
  <param name="File" value="/tmp/voltdb.log" />
  <param name="Append" value="true" />
  <layout class="org.apache.log4j.TTCCLayout" />
</appender>

<logger name="AUTH">
<!-- Print all VoltDB authentication messages -->
  <level value="trace" />
</logger>

<root>
      <priority value="debug" />
      <appender-ref ref="Async" />
</root>
</log4j:configuration>
```

The preceding configuration file defines three destinations, or appenders, called *Async*, *Console*, and *File*. The appenders define the type of output (whether to the console, to a file, or somewhere else), the location (such as the file name), as well as the layout of the messages sent to the appender. See the log4J documentation for more information about layout.

Note that the appender Async is a superset of Console and File. So any messages sent to Async are routed to both Console and File. This is important because for logging of VoltDB, you should always use an asynchronous appender as the primary target to avoid the processing of the logging messages from blocking other execution threads.

More importantly, you should not use any appenders that are susceptible to extended delays, blockages, or slow throughput, This is particularly true for network-based appenders such as SocketAppender and third-party log infrastructures including logstash and JMS. If there is any prolonged delay in writing to the appenders, messages can end up being held in memory causing performance degradation and, ultimately, generating out of memory errors or forcing the database into read-only mode.

The configuration file also defines a root class. The root class is the default logger and all loggers inherit the root definition. So, in this case, any messages of severity "debug" or higher are sent to the Async appender.

Note

This example is for demonstration purposes only. Normally, do *not* set the severity to either "debug" or "trace" for production systems unless instructed to by VoltDB Support. Trace and debug logging generate a significant number of messages that can negatively impact performance. They contain internal information for debugging purposes and provide no additional value otherwise.

Finally, the configuration file defines a logger specifically for VoltDB authentication messages. The logger identifies the class of messages to log (in this case "AUTH"), as well as the severity ("trace"). VoltDB defines several different classes of messages you can log. Table 6.1, “VoltDB Components for Logging” lists the loggers you can invoke.

Table 6.1. VoltDB Components for Logging

Logger	Description
ADHOC	Execution of ad hoc queries
AUTH	Authentication and authorization of clients
COMPILER	Interpretation of SQL in ad hoc queries
CONSOLE	Informational messages intended for display on the console
DR	Database replication sending data
DRAGENT	Database replication receiving data
EXPORT	Exporting data
GC	Java garbage collection
HOST	Host specific events
IMPORT	Importing data
ELASTIC	Elastic addition of nodes to the cluster
LOADER	Bulk loading of data (including as part of import)
NETWORK	Network events related to the database cluster
REJOIN	Node recovery and rejoin
SNAPSHOT	Snapshot activity
SQL	Execution of SQL statements
TM	Transaction management
TOPICS	Streaming data in topics

6.3. Changing the Timezone of Log Messages

By default all VoltDB logging is reported in GMT (Greenwich Mean Time). If you want the logging to be reported using a different timezone, you can use extensions to the Log4J service to achieve this.

To change the timezone of log messages:

1. Download the extras kit from the Apache Extras for Apache Log4J website, http://logging.apache.org/log4j/extras/.

2. Unpack the kit and place the included JAR file in the `/lib/extension` folder of the VoltDB installation directory.

3. Update your Log4J configuration file to enable the Log4J extras and specify the desired timezone for logging for each appender.

You enable the Log4J extras by specifying `EnhancedPatternLayout` as the layout class for the appenders you wish to change. You then identify the desired timezone as part of the layout pattern. For example, the following XML fragment changes the timezone of messages written to the file appender to GMT minus four hours:

```
<appender name="file" class="org.apache.log4j.DailyMaxRollingFileAppender">
  <param name="file" value="log/volt.log"/>
  <param name="DatePattern" value="'.'yyyy-MM-dd" />
  <layout class="org.apache.log4j.EnhancedPatternLayout">
     <param name="ConversionPattern"
            value="%d{ISO8601}{GMT-4}   %-5p [%t] %c: %m%n"/>
  </layout>
</appender>
```

You can use any valid ISO-8601 timezone specification, including named timezones, such as EST.

6.4. Managing VoltDB Log Files

VoltDB uses a rolling log appender that "rolls" the files, periodically saving the old log files and creating a new file for subsequent messages. By default, the log files are rolled daily.

VoltDB also automatically "prunes" older log files to help conserve disk space on the server. The appender specifies the maximum number of files to keep, keeping 30 by default.

You can customize your log configuration to specify a different rolling period and/or a different number of files to keep. For example, the following Log4J configuration rolls the log files twice a day and keeps 14 files, or a week's worth of logs:

```
<!-- file appender captures all loggers messages. -->
<appender name="file" class="org.apache.log4j.DailyMaxRollingFileAppender">
  <param name="file" value="log/volt.log"/>
  <param name="MaxBackupIndex" value="14"/>
  <param name="DatePattern" value="'.'yyyy-MM-dd-a" />
  <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="%d   %-5p [%t] %c: %m%n"/>
  </layout>
</appender>
```

6.5. Enabling Your Custom Log Configuration When Starting VoltDB

Once you create your Log4J configuration file, you specify which configuration file to use by defining the variable LOG4J_CONFIG_PATH before starting the VoltDB database. For example:

```
$ LOG4J_CONFIG_PATH="$HOME/MyLog4jConfig.xml"
$ voltdb start -H svr1,svr2
```

6.6. Changing the Configuration on the Fly

Once the database has started, you can still start or reconfigure the logging without having to stop and restart the database. By calling the system procedure @UpdateLogging you can pass the configuration

XML to the servers as a text string. For any appenders defined in the new updated configuration, the existing appender is removed and the new configuration applied. Other existing appenders (those not mentioned in the updated configuration XML) remain unchanged.

Chapter 7. What to Do When Problems Arise

As with any high performance application, events related to the database process, the operating system, and the network environment can impact how well or poorly VoltDB performs. When faced with performance issues, or outright failures, the most important task is identifying and resolving the root cause. VoltDB and the server produce a number of log files and other artifacts that can help you in the diagnosis. This chapter explains:

- Where to look for log files and other information about the VoltDB server process
- What to do when recovery fails
- How to collect the log files and other system information when reporting a problem to VoltDB

7.1. Where to Look for Answers

The first place to look when an unrecognized problem occurs with your VoltDB database is the console where the database process was started. VoltDB echoes key messages and errors to the console. For example, if a server becomes unreachable, the other servers in the cluster will report an error indicating which node has failed. Assuming the cluster is K-safe, the remaining nodes will then re-establish a quorum and continue, logging this event to the console as well.

However, not all messages are echoed on the console.[1] A more complete record of errors, warnings, and informational messages is written to a log file, `log/volt.log`, inside the voltdbroot directory. So, for example, if you start the database using the command **voltdb start --dir=~/db**, the log file is `~/db/voltdbroot/log/volt.log`.) The `volt.log` file can be extremely helpful for identifying unexpected but non-fatal events that occurred earlier and may identify the cause of the current issue.

If VoltDB encounters a fatal error and exits, shutting down the database process, it also attempts to write out a crash file in the current working directory. The crash file name has the prefix "voltdb_crash" followed by a timestamp identifying when the file is created. Again, this file can be useful in diagnosing exactly what caused the crash, since it includes the last error message, a brief profile of the server and a dump of the Java threads running in the server process before it crashed.

To summarize, when looking for information to help analyze system problems, three places to look are:

1. The console where the server process was started.
2. The log file in `log/volt.log`
3. The crash file named `voltdb_crash{timestamp}.txt` in the server process's working directory

7.2. Handling Errors When Restoring a Database

After determining what caused the problem, the next step is often to get the database up and running again as soon as possible. When using snapshots or command logs, this is done using the **voltdb start** command described in Section 3.6, “Restarting the Database”. However, in unusual cases, the restart itself may fail.

[1]Note that you can change which messages are echoed to the console and which are logged by modifying the Log4j configuration file. See the chapter on logging in the *Using VoltDB* manual for details.

There are several situations where an attempt to recover a database — either from a snapshot or command logs — may fail. For example, restoring data from a snapshot to a schema where a unique index has been added can result in a constraint violation. In this case, the restore operation continues but any records that caused a constraint violation are saved to a CSV file.

Or when recovering command logs, the log may contain a transaction that originally succeeded but fails and raises an exception during playback. In this situation, VoltDB issues a fatal error and stops the database to avoid corrupting the contents.

Although protecting you from an incomplete recovery is the appropriate default behavior, there may be cases where you want to recover as much data as possible, with full knowledge that the resulting data set does *not* match the original. VoltDB provides two processes for performing partial recoveries in case of failure:

- Logging constraint violations during snapshot restore
- Performing command log recovery in safe mode

The following sections describe these procedures.

Warning

It is critically important to recognize that the techniques described in this section *do not* produce a complete copy of the original database or resolve the underlying problem that caused the initial recovery to fail. These techniques should never be attempted without careful consideration and full knowledge and acceptance of the risks associated with partial data recovery.

7.2.1. Logging Constraint Violations

There are several situations that can cause a snapshot restore to fail because of constraint violations. Rather than have the operation fail as a whole, VoltDB continues with the restore process and logs the constraint violations to a file instead. This way you can review the tuples that were excluded and decide whether to ignore or replace their content manually after the restore completes.

By default, the constraint violations are logged to one or more files (one per table) in the same directory as the snapshot files. In a cluster, each node logs the violations that occur on that node. If you know there are going to constraint violations and want to save the logged constraints to a different location, you can use a special JSON form of the @SnapshotRestore system procedure. You specify the path of the log files in a JSON attribute, `duplicatePaths`. For example, the following commands perform a restore of snapshot files in the directory `/var/voltdb/snapshots/` with the unique identifier `myDB`. The restore operation logs constraint violations to the directory `/var/voltdb/logs`.

```
$ sqlcmd
1> exec @SnapshotRestore '{ "path":"/var/voltdb/snapshots/",
                            "nonce":"myDB",
                            "duplicatesPath":"/var/voltdb/logs/" }';
2> exit
```

Constraint violations are logged as needed, one file per table, to CSV files with the name `{table}-duplicates-{timestamp}.csv`.

7.2.2. Safe Mode Recovery

On rare occasions, recovering a database from command logs may fail. This can happen, for example, if a stored procedure introduces non-deterministic content. If a recovery fails, the specific error is known.

However, there is no way for VoltDB to know the root cause or how to continue. Therefore, the recovery fails and the database stops.

When this happens, VoltDB logs the last successful transaction before the recovery failed. You can then ask VoltDB to restart up to but not including the failing transaction by performing a recovery in *safe mode*.

You request safe mode by adding the **--safemode** switch to the **voltdb start** command, like so:

```
$ voltdb start --safemode --dir=~/mydb
```

When VoltDB recovers from command logs in safe mode it enables two distinct behaviors:

- Snapshots are restored, logging any constraint violations
- Command logs are replayed up to the last valid transaction

This means that if you are recovering using an automated snapshot (rather than command logs), you can recover some data even if there are constraint violations during the snapshot restore. Also, when recovering from command logs, VoltDB will ignore constraint violations in the command log snapshot and replay all transactions that succeeded in the previous attempt.

It is important to note that to successfully use safe mode with command logs, you must perform a regular recovery operation first — and have it fail — so that VoltDB can determine the last valid transaction. Also, if the snapshot and the command logs contain both constraint violations and failed transactions, you may need to run recovery in safe mode twice to recover as much data as possible. Once to complete restoration of the snapshot, then a second time to recover the command logs up to a point before the failed transaction.

7.3. Collecting the Log Files

VoltDB includes a utility that collects all of the pertinent logs for a given server. The log collector retrieves the necessary system and process files from the server and saves them in a single compressed archive file. For customers requesting support from VoltDB, your support contact will often provide instructions on how and when to use the log collector and where to submit the files.

Note that the database does not need to be running to use the log collector. It can find and collect the log files based solely on the location of the VoltDB root directory where the database was run.

To collect the log files, use the **voltdb collect** command with the same directory specification you would use to initialize or start the database:

```
$ voltdb collect --prefix=mylogs -D /home/db
```

When you run the command you must specify the location of the root directory for the database with the `--dir` or `-D` flag. Otherwise, the default is the current working directory. The archive file that the **collect** command generates is also created in your current working directory unless you use the `--output` flag to specify an alternate location and filename.

The **collect** command has optional arguments that let you control what data is collected and the name of the resulting archive file. In the preceding example the `--prefix` flag specifies the prefix for the archive file name. The `--skip-heap-dump` flag excludes the heap dump, which can be significantly larger than any other collection artifact, from the resulting archive. For example:

```
$ voltdb collect --dir=/home/db \
     --prefix=mylogs \
     --skip-heap-dump
```

Note that the **voltdb collect** command collects log files for the current system only. To collect logs for all servers in a cluster, you will need to issue the **voltdb collect** command locally on each server separately. See the **voltdb collect** documentation in the *Using VoltDB* manual for details.

Appendix A. Server Configuration Options

There are a number of system, process, and application options that can impact the performance or behavior of your VoltDB database. You control these options when initializing and/or starting VoltDB. The configuration options fall into five main categories:

- Server configuration
- Process configuration
- Database configuration
- Path configuration
- Network ports used by the database cluster

This appendix describes each of the configuration options, how to set them, and their impact on the resulting VoltDB database and application environment.

A.1. Server Configuration Options

VoltDB provides mechanisms for setting a number of options. However, it also relies on the base operating system and network infrastructure for many of its core functions. There are operating system configuration options that you can adjust to to maximize your performance and reliability, including:

- Network configuration
- Time configuration

A.1.1. Network Configuration (DNS)

VoltDB creates a network mesh among the database cluster nodes. To do that, all nodes must be able to resolve the IP address and hostnames of the other server nodes. Make sure all nodes of the cluster have valid DNS entries or entries in the local hosts files.

For servers that have two or more network interfaces — and consequently two or more IP addresses — it is possible to assign different functions to each interface. VoltDB defines two sets of ports:

- External ports, including the client and admin ports. These are the ports used by external applications to connect to and communicate with the database.
- Internal ports, including all other ports. These are the ports used by the database nodes to communicate among themselves. These include the internal port, the zookeeper port, and so on. (See Section A.5, “Network Ports” for a complete listing of ports.)

You can specify which network interface the server expects to use for each set of ports by specifying the internal and external interface when starting the database. For example:

```
$ voltdb start --dir=~/mydb        \
  --externalinterface=10.11.169.10 \
  --internalinterface=10.12.171.14
```

Note that the default setting for the internal and external interface can be overridden for a specific port by including the interface and a colon before the port number when specifying a port on the command line. See Section A.5, “Network Ports” for details on setting specific ports.

A.1.2. Time Configuration

Keeping VoltDB cluster nodes in close synchronization is important for the ongoing performance of your database. At a minimum, use of a time service such as NTP or chrony to synchronize time across the cluster is recommended. If the time difference between nodes is too large (greater than 200 milliseconds) VoltDB refuses to start. It is also important to avoid having nodes adjust time backwards, or VoltDB will pause while it waits for time to "catch up" to its previous setting.

A.2. Process Configuration Options

In addition to system settings, there are configuration options pertaining to the VoltDB server process itself that can impact performance. Runtime configuration options are set as command line options when starting the VoltDB server process.

The key process configuration for VoltDB is the Java maximum heap size. It is also possible to specify which garbage collector to use and to pass other arguments to the Java Virtual Machine directly.

A.2.1. Maximum Heap Size (VOLTDB_HEAPMAX)

The heap size is a parameter associated with the Java runtime environment. Certain portions of the VoltDB server software use the Java heap. In particular, the part of the server that receives and responds to stored procedure requests uses the Java heap.

Depending upon how many transactions your application executes a second, you may need additional heap space. The higher the throughput, the larger the maximum heap needed to avoid running out of memory.

In general, a maximum heap size of two gigabytes (2048) is recommended. For production use, a more accurate measurement of the needed heap size can be calculated from the size of the schema (number of tables), number of sites per host, and what durability and availability features are in use. See the *VoltDB Planning Guide* for details.

It is important to remember that the heap size is not directly related to data storage capacity. Increasing the maximum heap size does not provide additional data storage space. In fact, quite the opposite. Needlessly increasing the maximum heap size reduces the amount of memory available for storage.

To set the maximum heap size when starting VoltDB, define the environment variable VOLTDB_HEAPMAX as an integer value (in megabytes) before issuing the **voltdb start** command. For example, the following commands start VoltDB with a 3 gigabyte heap size (the default is 2 gigabytes):

```
$ export VOLTDB_HEAPMAX="3072"
$ voltdb start --dir=~/mydb -H serverA
```

A.2.2. Garbage Collector (VOLTDB_GC_OPTS)

The Java garbage collector (GC) intermittently frees up unused memory. Different garbage collectors use different algorithms for choosing when and how to do garbage collection. They also can have specific variables to further refine the garbage collection process.

Because Java processing can pause while the collector recycles memory, it can impact your application's latency. By default, for Java versions 8 and 11, VoltDB uses the following settings:

- ConcMarkSweepGC
- ClassUnloadingEnabled

- InitiatingOccupancyFraction=75
- InitiatingOccupancyOnly
- MaxAbortablePrecleanTime=120000
- ParallelRemarkEnabled
- ScavengeBeforeRemark
- WaitDuration=12000

For Java version 17, VoltDB uses the G1GC garbage collector with default settings. You can choose an alternate Java garbage collector by specifying your choice using the standard Java syntax in the VOLTDB_GC_OPTS environment variable before starting the database process. You can include any other GC-related options at the same time. For example:

```
$ export VOLTDB_GC_OPTS="-XX+useG1GC -XX+UseStringDeduplication"
$ voltdb start --dir=~/mydb -H serverA
```

See the Java documentation for your current Java implementation for more information on garbage collection and GC settings.

Warning

VoltDB does not validate the correctness of the arguments you specify using VOLTDB_GC_OPTS or their appropriateness for use with VoltDB. This feature is intended for experienced users only and should be used with extreme caution.

A.2.3. Other Java Runtime Options (VOLTDB_OPTS)

VoltDB sets the Java options — such as heap size and classpath — that directly impact VoltDB. There are a number of other configuration options available in the Java Virtual machine (JVM).

VoltDB provides a mechanism for passing arbitrary options directly to the JVM. If the environment variable VOLTDB_OPTS is defined, its value is passed as arguments to the Java command line. Note that the contents of VOLTDB_OPTS are added to the Java command line on the current server only. In other words, you must define VOLTDB_OPTS on each server to have it take effect for all servers.

Warning

VoltDB does not validate the correctness of the arguments you specify using VOLTDB_OPTS or their appropriateness for use with VoltDB. This feature is intended for experienced users only and should be used with extreme caution.

A.3. Database Configuration Options

Runtime configuration options are set either as part of the configuration file or as command line options when starting the VoltDB server process. These database configuration options are only summarized here. See the *Using VoltDB* manual for a more detailed explanation. The configuration options include:

- Sites per host
- K-Safety
- Network partition detection
- Automated snapshots

- Import and export
- Command logging
- Heartbeat
- Temp table size
- Query timeout
- Flush Interval
- Long-running process warning
- Copying array parameters
- Transaction Prioritization
- Clock skew

A.3.1. Sites per Host

Sites per host specifies the number of unique VoltDB "sites" that are created on each physical database server. The section on "Determining How Many Sites per Host" in the *Using VoltDB* manual explains how to choose a value for sites per host.

You set the value of sites per host using the `sitesperhost` attribute of the `<cluster>` tag in the configuration file.

A.3.2. K-Safety

K-safety defines the level of availability or durability that the database can sustain, by replicating individual partitions to multiple servers. K-safety is described in detail in the "Availability" chapter of the *Using VoltDB* manual.

You specify the level of K-safety that you want in the configuration file using the `kfactor` attribute of the `<cluster>` tag.

A.3.3. Network Partition Detection

Network partition detection protects a VoltDB cluster in environments where the network is susceptible to partial or intermittent failure among the server nodes. Partition detection is described in detail in the "Availability" chapter of the *Using VoltDB* manual.

Use of network partition detection is strongly recommended for production systems and therefore is enabled by default. You can enable or disable network partition detection in the configuration file using the `<partition-detection>` tag.

A.3.4. Automated Snapshots

Automated snapshots provide ongoing protection against possible database failure (due to hardware or software issues) by taking periodic snapshots of the database's contents. Automated snapshots are described in detail in the section on "Scheduling Automated Snapshots" in the *Using VoltDB* manual.

You enable and configure automated snapshots with the `<snapshot>` tag in the configuration file.

Snapshot activity involves both processing and disk I/O and so may have a noticeable impact on performance (in terms of throughput and/or latency) on a very busy database. You can control the priority of snapshots activity using the `<snapshot>` tag within the `<systemsettings>` element of the deployment file. The snapshot priority is an integer value between 0 and 10, with 0 being the highest priority and 10 being the lowest. The closer to 10, the longer snapshots take to complete, but the less they can affect ongoing database work.

Warning

Setting snapshot priority directly as described is deprecated. If transaction prioritization is not enabled, this method continues to work for backwards compatibility. However, the recommended method for setting snapshot priority is to enable transaction prioritization and set the snapshot priority as a child of `<priorities>`, described in Section A.3.13, “Transaction Prioritization”.

Note that snapshot priority affects all snapshot activity, including automated snapshots, manual snapshots, and command logging snapshots.

A.3.5. Import and Export

The import and export functions let you automatically import and/or export selected data between your VoltDB database and another database or distributed service at runtime. These features are described in detail in the chapter on "Importing and Exporting Live Data" in the *Using VoltDB* manual.

You enable and disable import and export using the `<import>` and `<export>` tags in the configuration file.

A.3.6. Command Logging

The command logging function saves a record of each transaction as it is initiated. These logs can then be "replayed" to recreate the database's last known state in case of intentional or accidental shutdown. This feature is described in detail in the chapter on "Command Logging and Recovery" in the *Using VoltDB* manual.

To enable and disable command logging, use the `<commandlog>` tag in the configuration file.

A.3.7. Heartbeat

The database servers use a "heartbeat" to verify the presence of other nodes in the cluster. If a heartbeat is not received within a specified time limit, that server is assumed to be down and the cluster reconfigures itself with the remaining nodes (assuming it is running with K-safety). This time limit is called the "heartbeat timeout" and is specified as a integer number of seconds.

For most situations, the default value for the timeout (90 seconds) is appropriate. However, if your cluster is operating in an environment that is susceptible to network fluctuations or unpredictable latency, you may want to increase the heartbeat timeout period.

You can set an alternate heartbeat timeout using the `<heartbeat>` tag in the configuration file.

Note

Be aware that certain Linux system settings can override the VoltDB heartbeat messages. In particular, lowering the setting for TCP_RETRIES2 may result in the system network timeout

interrupting VoltDB's heartbeat mechanism and causing timeouts sooner than expected. Values lower than 8 for TCP_RETRIES2 are not recommended.

A.3.8. Temp Table Size

VoltDB uses temporary tables to store intermediate table data while processing transactions. The default temp table size is 100 megabytes. This setting is appropriate for most applications. However, extremely complex queries or many updates to large records could cause the temporary space to exceed the maximum size, resulting in the transaction failing with an error.

In these unusual cases, you may need to increase the temp table size. You can specify a different size for the temp tables using the <`systemsettings`> and `<temptables>` tags in the configuration file and specifying the `maxsize` attribute as a whole number of megabytes. For example:

```
<systemsettings>
   <temptables maxsize="200"/>
</systemsettings>
```

Note: since the temp tables are allocated as needed, increasing the maximum size can result in a Java out-of-memory error at runtime if the system is memory-constrained. Modifying the temp table size should be done with caution.

A.3.9. Query Timeout

In general, SQL queries execute extremely quickly. But it is possible, usually by accident, to construct a query that takes an unexpectedly long time to execute. This usually happens when the query is overly complex or accesses extremely large tables without the benefit of an appropriate filter or index.

You have the option to set a query timeout limit cluster-wide, for an interactive session, or per transaction. The query limit sets a limit on the length of time any read-only query (or batch of queries in the case of the voltExecuteSQL() method in a stored procedure) is allowed to run. You specify the timeout limit in milliseconds.

To set a cluster-wide query limit you use the `<systemsettings>` and `<query timeout="{limit}">` tags in the configuration file. To set a limit for an interactive session in the sqlcmd utility, you use the `--query-timeout` flag when invoking sqlcmd. To specify a limit when invoking a specific stored procedure, you use the callProcedureWithTimeout method in place of the callProcedure method.

The cluster-wide limit is set when you initialize the database root directory. By default, the system-wide limit is 10 seconds. You can set a different timeout in the configuration file. Or It can be adjusted using the **voltadmin update** command to modify the configuration settings while the database is running. If security is enabled, any user can set a lower query limit on a per session or per transaction basis. However, the user must have the ADMIN privilege to set a query limit longer than the cluster-wide setting.

The following example configuration file sets a cluster-wide query timeout value of three seconds:

```
<systemsettings>
   <query timeout="3000"/>
</systemsettings>
```

If any query or batch of queries exceeds the query timeout, the query is interrupted and an error returned to the calling application. Note that the limit is applied to read-only ad hoc queries or queries in read-only stored procedures only. In a K-Safe cluster, queries on different copies of a partition may execute at different rates. Consequently the same query may timeout in one copy of the partition but not in another.

To avoid possible non-deterministic changes, VoltDB does not apply the time out limit to any queries or procedures that may modify the database contents.

A.3.10. Flush Interval

VoltDB features that interact with external systems, including database replication (DR) and export, limit their activity to balance I/O latency against potentially competing with ongoing database work. These features trigger I/O based on two factors: batch size and a flush interval. In other words, data is written when enough records are received to match the batch size or, if input is sporadic, data is written when the flush interval is reached to avoid small amounts of data be held indefinitely.

There are two different settings that control how frequently data is flushed from the queues. There is a feature-specific flush setting and a system-wide minimum value. You can set different flush intervals with individual features. For example, you might set the DR flush interval to 500 milliseconds to reduce the latency of database replication, while setting the export flush interval to 4 seconds if export latency is not critical.

The system-wide minimum defines how often flush intervals are checked. So no buffers can be written more frequently than the system-wide minimum. And since the minimum check event and the feature-specific intervals may not line up exactly, actual writes occur at some incremental time after the defined interval. For example, if you set both the minimum and the DR interval at 500 milliseconds, the actual buffer writes might occur anywhere between 500 and 1000ms apart.

You set both the system-wide minimum and feature-specific intervals in the configuration file using the `<systemsettings>` and `<flushinterval>` tags. You set the system-wide minimum in the `minimum` attribute of the `<flushinterval>` tag and you set the feature-specific intervals using the `<dr>` and `<export>` sub-elements. All values are specified in milliseconds. For example:

```
<systemsettings>
   <flushinterval minimum="500">
       <export interval="4000" />
       <dr interval="500" />
   </flushinterval>
</systemsettings>
```

The default system-wide minimum is one second (1000). The default flush intervals for DR and export are one second (1000) and four seconds (4000), respectively.

A.3.11. Long-Running Process Warning

You can avoid runaway read-only queries using the query timeout setting. But you cannot stop read-write procedures or other computational tasks, such as automated snapshots. These processes must run to completion. However, you may want to be notified when a process is blocking an execution queue for an extended period of time.

By default, VoltDB writes an informational message into the log file whenever a task runs for more than ten seconds in any of the execution sites. These tasks may be stored procedures, procedure fragments (in the case of multi-partitioned procedures), or operational tasks such as snapshot creation. You can adjust the limit when these messages are written by specifying a value, in milliseconds in the `loginfo` attribute of the `<procedure>` tag in the configuration file. For example, the following configuration file entry changes the threshold after which a message is written to the log to three seconds:

```
<systemsettings>
   <procedure loginfo="3000"/>
```

```
</systemsettings>
```

Note that in a cluster, the informational message is written only to the log of the server that is hosting the affected queue, not to all server logs.

A.3.12. Copying Array Parameters

You can send mutable datatypes, most notably arrays, as arguments to a VoltDB stored procedure. By default, when this happens on a cluster with K=1 or more, VoltDB makes a copy of the array before using it in a transactional statement, to ensure that the execution of the statement is deterministic. However, copying the contents of the array consumes additional memory, which can add up if procedures are called frequently with large arrays.

The alternative, if the procedures do not modify the contents of the array, is to tell VoltDB not to copy array parameters on K-safe clusters by setting the `copyparameters` attribute of the `<procedure>` element to "false":

```
<systemsettings>
   <procedure copyparameters="false"/>
</systemsettings>
```

Warning

Only disable copying of parameters if you are sure the stored procedures *do not* modify any array parameters. If a stored procedures does modify an array when arrays are not being copied, the transaction can result in non-deterministic behavior, including possible data corruption and/or crashing the database.

A.3.13. Transaction Prioritization

By default, all transactions are treated equally and executed in a first in, first out basis. However, you can enable transaction priorities where individual transactions (or groups of transactions) are given higher or lower priority.

To use transaction priorities, you must enable them in the configuration file by adding `<priorities>` as a child of the `<systemsettings>` element. If the <priorities> element is present, priorities are enabled. Or you can explicitly enable or disable them. For example:

```
<systemsettings>
   <priorities enabled="true"/>
</systemsettings>
```

You can also set a priority for database replication and/or snapshot transactions using corresponding subelements and specifying a priority between 1 and 8 (1 being the highest priority, 8 being the lowest):

```
<systemsettings>
  <priorities enabled="true">
    <dr priority="3"/>
    <snapshot priority="6"/>
</systemsettings>
```

You can adjust the effects of prioritization by setting the `maxwait` attribute on the `<priorities>` element. The `maxwait` attribute specifies the maximum number of milliseconds a task remains in a priority queue before it gets scheduled for execution regardless of its pioritization. This helps avoid high priority

transactions essentially blocking lower priority tasks from getting scheduled. The default wait time is 1000 milliseconds. Setting `maxwait` to zero (0) means that prioritization is always in effect. The following example reduces the maximum wait time to half a second:

```
<systemsettings>
   <priorities enabled="true" maxwait="500" />
</systemsettings>
```

A.3.14. Clock Skew

Certain database operations (such as initiating snapshots) depend on synchronizing the nodes of the cluster based on their system clocks. If the clocks are too far apart, it delays the activities and interrupts normal database operations. Which is why the database checks to make sure the clocks are within a minimal level of variation (100 milliseconds) when it starts.

It is also possible for clocks to "drift" over time. So the servers also check the clock skew periodically to make sure they stay within the allowable range. You can see the latest clock skew calculation using the @Statistics system procedure with the CLOCKSKEW selector. By default, clock skew is checked every hour. You can configure the interval between checks using the `<clockskew>` element under `<systemsettings>` in the database configuration file, specifying the interval as an whole number of minutes. For example, the following configuration sets the clock skew interval to every half hour:

```
<systemsettings>
   <clockskew interval="30"/>
</systemsettings>
```

The interval value can be any positive integer. If you set it to zero (0), clock skew will not be checked once the system starts.

A.4. Path Configuration Options

The running database uses a number of disk locations to store information associated with runtime features, such as export, network partition detection, and snapshots. You can control which paths are used for these disk-based activities. The path configuration options include:

- VoltDB root
- Snapshots path
- Export overflow path
- Command log path
- Command log snapshots path

A.4.1. VoltDB Root

VoltDB defines a root directory for any disk-based activity which is required at runtime. This directory also serves as a root for all other path definitions that take the default or use a relative path specification.

If you do not specify a location for the root directory on the command line, VoltDB uses the current working directory as a default. Normally, you specify the location of the root directory using the `--dir` flag on the **voltdb init** and **voltdb start** commands. The root directory is then the subdirectory `voltdbroot` within

the specified location. (If the subfolder does not exist, VoltDB creates it.) See the section on "Configuring Paths for Runtime Features" in the *Using VoltDB* manual for details.

A.4.2. Snapshots Path

The snapshots path specifies where automated and network partition snapshots are stored. The default snapshots path is the "`snapshots`" subfolder of the VoltDB root directory. You can specify an alternate path for snapshots using the `<snapshots>` child element of the `<paths>` tag in the configuration file.

A.4.3. Export Overflow Path

The export overflow path specifies where overflow data is stored for the export streams. The default export overflow path is the "`export_overflow`" subfolder of the VoltDB root directory. You can specify an alternate path using the `<exportoverflow>` child element of the `<paths>` tag in the configuration file.

See the chapter on "Exporting Live Data" in the *Using VoltDB* manual for more information on export overflow.

A.4.4. Command Log Path

The command log path specifies where the command logs are stored when command logging is enabled. The default command log path is the "`command_log`" subfolder of the VoltDB root directory. However, for production use, it is strongly recommended that the command logs be written to a dedicated device, not the same device used for snapshotting or export overflow. You can specify an alternate path using the `<commandlog>` child element of the `<paths>` tag in the configuration file.

See the chapter on "Command Logging and Recovery" in the *Using VoltDB* manual for more information on command logging.

A.4.5. Command Log Snapshots Path

The command log snapshots path specifies where the snapshots created by command logging are stored. The default path is the "`command_log_snapshot`" subfolder of the VoltDB root directory. (Note that command log snapshots are stored separately from automated snapshots.) You can specify an alternate path using the `<commandlogsnapshot>` child element of the `<paths>` tag in the configuration file.

See the chapter on "Command Logging and Recovery" in the *Using VoltDB* manual for more information on command logging.

A.5. Network Ports

A VoltDB cluster opens network ports to manage its own operation and to provide services to client applications. The network ports are configurable as part of the command that starts the VoltDB database process. You can specify just a port number or the network interface and the port number, separated by a colon.

Table A.1, “VoltDB Port Usage” summarizes the ports that VoltDB uses and their default value. The following sections describe each port in more detail and how to set them. Section A.5.9, “TLS/SSL Encryption (Including HTTPS)” explains how to enable TLS encryption for the web and the programming interface ports, client and admin.

Table A.1. VoltDB Port Usage

Port	Default Value
Client Port	21212
Admin Port	21211
Web Interface Port (httpd)	8080
Web Interface Port (with TSL/SSL enabled)	8443
Internal Server Port	3021
Metrics Port	11781
Replication Port	5555
Topics Port	9092
Zookeeper port	7181

A.5.1. Client Port

The client port is the port VoltDB client applications use to communicate with the database cluster nodes. By default, VoltDB uses port 21212 as the client port. You can change the client port. However, all client applications must then use the specified port when creating connections to the cluster nodes.

To specify a different client port on the command line, use the `--client` flag when starting the VoltDB database. For example, the following command starts the database using port 12345 as the client port:

```
$ voltdb start --dir=~/mydb --client=12345
```

If you change the default client port, all client applications must also connect to the new port. The client interfaces for Java and C++ accept an additional, optional argument to the createConnection method for this purpose. The following examples demonstrate how to connect to an alternate port using the Java and C++ client interfaces.

Java

```
org.voltdb.client.Client voltclient;
voltclient = ClientFactory.createClient();
voltclient.createConnection("myserver",12345);
```

C++

```
boost::shared_ptr<voltdb::Client> client = voltdb::Client::create();
client->createConnection("myserver", 12345);
```

A.5.2. Admin Port

The admin port is similar to the client port, it accepts and processes requests from applications. However, the admin port has the special feature that it continues to accept write requests when the database enters admin, or read-only, mode.

By default, VoltDB uses port 21211 on the default external network interface as the admin port. You can change the port assignment on the command line using the `--admin` flag. For example, the following command sets the admin port to 2222:

```
$ voltdb start --dir=~/mydb --admin=2222
```

A.5.3. Web Interface Port (http)

The web interface port is the port that VoltDB listens to for web-based connections. This port is used for both the JSON programming interface and access to the Volt Management Center.

By default, VoltDB uses port 8080 on the default external network interface as the web port. You can change the port assignment on the command line using the `--http` flag. For example, the following command sets the port to 8888:

```
$ voltdb start --dir=~/mydb --http=8888
```

If you change the port number, be sure to use the new port number when connecting to the cluster using either the Volt Management Center or the JSON interface. For example, the following URL connects to the JSON interface using the reassigned port 8888:

```
http://athena.mycompany.com:8888/api/2.0/?Procedure=@SystemInformation
```

If you do not want to use the http port of the features it supports (the JSON API and Volt Management Center) you can disable the port in the configuration file. For example, for following configuration option disables the default http port:

```
<httpd enabled="false"/>
```

If the port is not enabled, neither the JSON interface nor the Management Center are available from the cluster. By default, the web interface is enabled.

Another aspect of the http port, when it is enabled, is whether the port transmits using http or https. You can enable TLS (Transport Layer Security) encryption on the web interface so that all interaction uses the HTTPS protocol. When TLS is enabled, the default port changes to 8443. See Section A.5.9, “TLS/SSL Encryption (Including HTTPS)” for information on enabling encryption in the configuration file.

A.5.4. Internal Server Port

A VoltDB cluster uses ports to communicate among the cluster nodes. This port is internal to VoltDB and should not be used by other applications.

By default, the internal server port is port 3021 for all nodes in t1he cluster[1]. You can specify an alternate port using the `--internal` flag when starting the VoltDB process. For example, the following command starts the VoltDB process using an internal port of 4000:

```
$ voltdb start --dir=~/mydb --internal=4000
```

A.5.5. Metrics Port

When metrics are enabled, the database uses the metrics port to return statistical data about the state of the database to calling applications, such as Prometheus. By default, the metrics port is 11781. You can specify a different port using the `--metrics` flag when starting the database server. For example:

```
$ voltdb start --dir=~/mydb --metrics=9090
```

[1] In the special circumstance where multiple VoltDB processes are started for one database, all on the same server, the internal server port is incremented from the initial value for each process.

A.5.6. Replication Port

During database replication, producer databases (that is, the master database in passive DR and all clusters in XDCR) use a dedicated port to share data to their consumers. By default, the replication port is port 5555. You can use a different port by specifying a different port number on the **voltdb** command line using the `--replication` flag. For example, the following command changes the replication port:

```
$ voltdb start --dir=~/mydb --replication=6666
```

Note that if you set the replication port on the producer to something other than the default, you must notify the consumers of this change. The replica or other XDCR clusters must specify the port along with the network address or hostname in the `src` attribute of the `<connection>` element when configuring the DR relationship. For example, if the server nyc2 has changed its replication port to 3333, another cluster in the XDCR relationship might have the following configuration:

```
<dr id="1" role="xdcr" >
   <connection source="nyc1,nyc2:3333" />
</dr>
```

Finally, in some cloud environments, such as Kubernetes, remote clusters may not be able to access the producer cluster by its internal network interface. Consumers can specify the location of the producer in the DR configuration using a remapped IP address. But once they initialize contact with the producer, the producer sends a list of IP addresses to use for ongoing replication. By default, these are the internal addresses the producer cluster knows about.

You can tell the producer to advertise a different interface (and port) for this second phase by specifying the alternate interface using the `--drpublic` argument in the **voltdb start** command. If you do not specify a port on the `--drpublic` argument, the internal replication port is used. For example:

```
$ voltdb start --drpublic=some.external.addr
```

A.5.7. Topics Port

When topics are enabled, the database uses the topics port to send and receive data to consumers and producers. By default, the topics port is port 9092. You can specify a different port using the `--topicsport` flag when starting the database server. For example, the following command changes the topics port:

```
$ voltdb start --dir=~/mydb --topicsport=9900
```

In cases where the server's external interface is not directly accessible by outside services and you set up the necessary port forwarding to an alternative public interface for those services to use, you can identify that alternative port to the server using the `--topicspublic` flag. For example:

```
$ voltdb start --dir=~/mydb --topicspublic=myexternalserver:9092
```

A.5.8. Zookeeper Port

VoltDB uses a version of Apache Zookeeper to communicate among supplementary functions that require coordination but are not directly tied to database transactions. Zookeeper provides reliable synchronization for functions such as command logging without interfering with the database's own internal communications.

VoltDB uses a network port bound to the local interface (127.0.0.1) to interact with Zookeeper. By default, 7181 is assigned as the Zookeeper port for VoltDB. You can specify a different port number using the

`--zookeeper` flag when starting the VoltDB process. It is also possible to specify a different network interface, like with other ports. However, accepting the default for the zookeeper network interface is recommended where possible. For example:

```
$ voltdb start --dir=~/mydb --zookeeper=2288
```

A.5.9. TLS/SSL Encryption (Including HTTPS)

VoltDB lets you enable Transport Layer Security (TLS) — the recommended upgrade from Secure Socket Layer (SSL) encryption — for all of its externally-facing interfaces: the web port, client port, admin port, and replication (DR) port. When you enable TLS, you automatically enable encryption for the web port. You can then optionally enable encryption for the external ports (client and admin) and/or the replication port.

To enable TLS encryption you need an appropriate certificate. How you configure TLS depends on whether you create a local certificate or receive one from an authorized certificate provider, such as VeriSign, GeoTrust and others. If you use a commercial certificate, you only need to identify the certificate as the key store. If you create your own, you must specify both the key store and the trust store. (See the section on using TLS/SSL for security in the Using VoltDB manual for an example of creating your own certificate.)

You enable TLS encryption in the deployment file using the <ssl> element. Within <ssl> you specify the location and password for the key store and, for locally generated certificates, the trust store in separate elements like so:

```
<ssl>
   <keystore path="/etc/mydb/keystore" password="twiddledee"/>
   <truststore path="/etc/mydb/truststore" password="twiddledum"/>
</ssl>
```

When you enable the <ssl> element in the configuration file, TLS encryption is enabled for the web port and all access to the httpd port and JSON interface must use the HTTPS protocol. When you enable TLS, the default web port changes from 8080 to 8443.

You can explicitly enable or disable TLS encryption by including the `enable` attribute. (For example, if you want to include the key store and trust store in the configuration but not turn on TLS during testing, you can include `enabled="false"`.) You can specify that the client and admin API ports are also TLS encrypted by adding the `external` attribute and setting it to `true`. Similarly, you can enable TLS encryption for the DR port by adding the `dr` attribute. For example, the following configuration sample, explicitly enables TLS for all externally-facing ports:

```
<ssl enabled="true" external="true" dr="true">
   <keystore path="/etc/mydb/keystore" password="twiddledee"/>
   <truststore path="/etc/mydb/truststore" password="twiddledum"/>
</ssl>
```

Note that you *cannot* disable TLS encryption for the web port separately. TLS is always enabled for the web port if you enable encryption for any ports.

Appendix B. Volt Active Data Metrics

Volt Active Data provides metrics in Prometheus format that you can use to track and monitor database activity and status. This appendix provides a list of all the metrics values available from Volt, including a description of their type and purpose. The metrics are grouped according to the particular aspect of the product they report on, including:

- Database Tables and Indexes
- Transactions, Procedures, and the Planner
- Memory and CPU Usage
- Client Connections and I/O
- High Availability and Durability
- Streaming Data
- User-Defined Tasks
- System and Cluster Status

B.1. Database Tables and Indexes

The following table describes the metrics available for monitoring the database content, such as tables and indexes. This information can be used for determining the number, size, and distribution of tables in the database.

Table B.1. Tables and Indexes

Metrics	Type	Description
voltdb_index_entry_count_total	Gauge	The number of index entries in the partition.
voltdb_index_memory_estimate_bytes	Gauge	The estimated amount of memory consumed by the index entries.
voltdb_table_allocated_memory_bytes	Gauge	The total size of memory allocated for storing inline data associated with this table in this partition. For streams, the amount of memory in use to queue export data (both in memory and as export overflow) prior to its being passed to the export target.
voltdb_table_data_memory_bytes	Gauge	The total memory used for storing inline data associated with this table in this partition.
voltdb_table_string_data_memory_bytes	Gauge	The total memory used for storing non-inline variable length data (VARCHAR, VARBINARY, and GEOGRAPHY) associated with this table in this partition.
voltdb_table_tuple_total	Gauge	The number of rows stored for this table in the current partition. For streams, the cumulative total number of rows inserted into the stream.
voltdb_ttl_failed_total	Counter	Total number of times TTL failed to be processed.
voltdb_ttl_last_execution_timestamp_seconds	Gauge	The timestamp when the last round of TTL processing occurred.

Metrics	Type	Description
voltdb_ttl_rows_deleted_total	Counter	The total number of rows expired and deleted by the TTL attribute.
voltdb_ttl_rows_remaining_total	Gauge	The number of expired rows not deleted during the last TTL processing due to batch size limits. If TTL processing is keeping up with the throughput, this value should tend towards zero.

B.2. Transactions, Procedures, and the Planner

The following tables describe the metrics available for measuring the volume, frequency, and performance of transactions, procedures, and the planner used for precessing ad hoc queries.

Table B.2. Transactions and Procedures

Metrics	Type	Description
voltdb_idle_time_pauses_seconds	Histogram	The distribution of the amount of time the execution site had to wait for a new task.
voltdb_initiator_procedure_aborted_total	Counter	The number of times the procedure was aborted.
voltdb_initiator_procedure_failed_total	Counter	The number of times the procedure failed unexpectedly.
voltdb_initiator_procedure_invoked_total	Counter	The number of times the stored procedure has been invoked by this connection on this host node.
voltdb_initiator_procedure_invoked_time_seconds	Histogram	The distribution of length of time it took to execute the stored procedure.
voltdb_procedure_aborted_total	Counter	The number of times the procedure was aborted.
voltdb_procedure_bad_input_total	Counter	The total number of times this procedure was run with a wrong set of arguments (may only happen for NT procedure).
voltdb_procedure_failed_total	Counter	The number of times the procedure failed unexpectedly.
voltdb_procedure_forwarded_total	Counter	-
voltdb_procedure_invoked_total	Counter	The total number of invocations of this procedure at this site.
voltdb_procedure_invoked_time_seconds	Histogram	The length of time it took to execute the stored procedure.
voltdb_procedure_params_size_bytes	Counter	The cumulative size of the parameters passed as input to the procedure.
voltdb_procedure_result_size_bytes	Counter	The total size of the results returned by the procedure.
voltdb_procedure_sampled_total	Counter	Number of invocations of procedures for which all measurements (such as execution time) were captured.
voltdb_procedure_statement_failed_total	Counter	The number of times this procedure statement failed unexpectedly.
voltdb_procedure_statement_invoked_total	Counter	The total number of invocations of this statement as part of given procedure at this site.
voltdb_procedure_statement_invoked_time_seconds	Histogram	The length of time it took to execute the statement.

Metrics	Type	Description
voltdb_procedure_statement_params_size_bytes	Counter	The total size of the parameters passed as input to the statement.
voltdb_procedure_statement_result_size_bytes	Counter	The total size (in bytes) of the results returned by the statement.
voltdb_procedure_timeout_total	Counter	The number of times the procedure timed out.

Table B.3. Planner

Metrics	Type	Description
voltdb_planner_cache1_hits_total	Counter	The number of queries that matched and reused a plan in the level 1 cache.
voltdb_planner_cache1_level_total	Gauge	The number of query plans in the level 1 cache.
voltdb_planner_cache2_hits_total	Counter	The number of queries that matched and reused a plan in the level 2 cache.
voltdb_planner_cache2_level_total	Gauge	The number of query plans in the level 2 cache. Gauge.
voltdb_planner_cache_misses_total	Counter	The number of queries that had no match in the cache and had to be planned from scratch.
voltdb_planner_failures_total	Counter	The number of times planning for an ad hoc query failed.
voltdb_planner_plan_time_seconds	Histogram	The distribution of length of time (with nanoseconds accuracy) it took to complete the planning of an ad hoc query.

B.3. Memory and CPU Usage

The following tables describe the metrics monitoring memory and CPU usage, including memory compaction triggered by Volt and garbage collection triggered by Java.

Table B.4. Memory, Compaction, and Garbage Collection

Metrics	Type	Description
voltdb_compaction_execution_seconds	Histogram	The amount of time it took for compaction to complete.
voltdb_compaction_fragmented_percent	Gauge	The current fragmentation percentage.
voltdb_compaction_invoked_total	Counter	Number of times compaction was performed.
voltdb_compaction_relocated_total	Histogram	The number of tuples relocated during compaction.
voltdb_gc_count_total	Counter	The number of times garbage collection was performed. Tags: • GC_TYPE - (e.g. CMS, G1), example gc_type="G1 Young Generation".
voltdb_gc_time_seconds	Counter	Cumulative run time of garbage collection. Tags: • GC_TYPE - (e.g. CMS, G1), example gc_type="G1 Young Generation".
voltdb_memory_indexmemory_bytes	Gauge	The amount of memory in use for storing database indexes.
voltdb_memory_javamaxheap_bytes	Gauge	The maximum heap size of the Java runtime environment.

Metrics	Type	Description
voltdb_memory_javaused_bytes	Gauge	The amount of memory allocated by Java and in use by VoltDB.
voltdb_memory_nio_total_buffer_count_total	Gauge	An estimate of the number of buffers in the NIO pool.
voltdb_memory_nio_total_size_bytes	Gauge	An estimate of the total capacity of all the buffers in the NIO pool.
voltdb_memory_nio_used_bytes	Gauge	An estimate of the memory that the Java virtual machine is using for the NIO pool which resides outside the regular Java heap.
voltdb_memory_physicalmemory_bytes	Gauge	The total size of physical memory on the server.
voltdb_memory_pooledmemory_total	Gauge	The total size of memory allocated for tasks other than database records, indexes, and strings.
voltdb_memory_rss_bytes	Gauge	The resident set size. That is, the total amount of memory allocated to the VoltDB processes on the server.
voltdb_memory_stringmemory_bytes	Gauge	The amount of memory in use for storing string, binary, and geospatial data that is not stored in-line in the database record.
voltdb_memory_tupleallocated_bytes	Gauge	The amount of memory allocated for the storage of database records (including free space).
voltdb_memory_tuplecount_total	Gauge	The total number of database records in memory.
voltdb_memory_tupledata_bytes	Gauge	The amount of memory in use for storing database records.
voltdb_memory_undo_log_size_bytes	Gauge	-
voltdb_memory_undo_pool_size_bytes	Gauge	The total size of memory allocated for the undo pool - memory used to store information needed to "undo" database changes if a transaction needs to rollback.

Table B.5. CPU

Metrics	Type	Description
voltdb_cpu_load_percent	Gauge	The percentage of CPU used by the database server process. 0-100.

B.4. Client Connections and I/O

The following tables describe the metrics for client connections and the I/O between clients and the cluster.

Table B.6. Connections

Metrics	Type	Description
voltdb_accepted_connections_total	Gauge	The total number of client connections opened since the server started, including connections that are now closed.
voltdb_client_connections_limit_total	Gauge	The maximum number of client connections allowed for the server.
voltdb_client_connections_open_total	Gauge	The number of client connections open on the server.

Metrics	Type	Description
voltdb_dropped_connections_total	Gauge	The total number of connections that were rejected because the connection limit had been reached.

Table B.7. I/O

Metrics	Type	Description
voltdb_io_message_handled_total	Counter	The number of individual messages sent from the client to the host.
voltdb_io_message_written_total	Counter	The number of individual messages sent from the host to the client.
voltdb_io_network_inbound_queue_time_seconds	Histogram	The distribution of the time tasks were waiting in the queue for the execution on the remote node, initiated by this connection.
voltdb_io_network_outstanding_request_bytes_bytes	Gauge	The number of bytes of data sent from the client pending on the host.
voltdb_io_network_procedure_round_trip_time_seconds	Histogram	The distribution of the time taken to receive acknowledgment of the execution of the stored procedures on the leader node, initiated by this connection.
voltdb_io_network_read_bytes	Counter	The number of bytes of data sent from the client to the host.
voltdb_io_network_read_error_total	Counter	Number of times request has failed.
voltdb_io_network_replication_round_trip_time_seconds	Histogram	The distribution of the time it took to receive acknowledgment from the replica.
voltdb_io_network_write_bytes	Counter	The number of bytes of data sent from the host to the client.
voltdb_io_outstanding_messages_total	Gauge	The number of messages on the host queue waiting to be retrieved by the client.
voltdb_io_outstanding_response_messages_total	Gauge	Number of message scheduled to be sent to the client.
voltdb_io_tls_decryption_latency_seconds	Histogram	The distribution of decryption times.
voltdb_io_tls_encryption_latency_seconds	Histogram	The distribution of encryption times.
voltdb_io_tls_messages_decrypted_total	Counter	The number of messages decrypted with TLS.
voltdb_io_tls_messages_encrypted_total	Counter	The number of messages encrypted with TLS.

B.5. High Availability and Durability

The following tables describe the metrics related to Volt features that provide high availability and durability for the database, including snapshots, command logging, and Active(N) cross datacenter replication.

Table B.8. Snapshots

Metrics	Type	Description
voltdb_snapshot_site_summary_info	Metadata	Informational metric. One for every snapshot file in the recent snapshots performed on the cluster. Tags:

Metrics	Type	Description
		• SNAPSHOT_NONCE - The unique identifier for the snapshot. • TABLE_NAME - The name of the database table whose data the file contains. • SNAPSHOT_COLLECTION_ITERATIONS - . • SNAPSHOT_COLLECTION_TIME - The length of time (in seconds) it took to complete the snapshot. • SNAPSHOT_WRITE_TIME - The length of time (in seconds) it took to complete write stage of snapshot operation.
voltdb_snapshot_summary_info	Metadata	Informational metric. One for every snapshot file in the recent snapshots performed on the cluster. Tags: • SNAPSHOT_NONCE - The unique identifier for the snapshot. • SNAPSHOT_TXN_ID - The transaction ID of the snapshot. • SNAPSHOT_TYPE - String value indicating how the snapshot was initiated. Possible values are: "Auto" - an automated snapshot as defined by the configuration file; "Commandlog" - a command log snapshot; "Manual" - a manual snapshot initiated by a user. • SNAPSHOT_PATH - The directory path where the snapshot file resides. • SNAPSHOT_START_TIME - The timestamp when the snapshot began (in milliseconds). • SNAPSHOT_END_TIME - The timestamp when the snapshot was completed (in milliseconds). • SNAPSHOT_BYTES_WRITTEN - Total number of bytes written to the file so far. • SNAPSHOT_PROGRESS - For snapshots currently in progress, the percent complete at the time of the call (0-100). • SNAPSHOT_RESULT - Value indicating whether the writing of the snapshot file was successful ("Success") or not ("Failure").
voltdb_snapshot_table_summary_info	Metadata	Informational metric. One for every snapshot file in the recent snapshots performed on the cluster. Tags: • SNAPSHOT_NONCE - The unique identifier for the snapshot. • SNAPSHOT_TXN_ID - The transaction ID of the snapshot. • TABLE_NAME - The name of the database table whose data the file contains. • TABLE_FILENAME - The file name. • SNAPSHOT_BYTES_WRITTEN - The total size, in bytes, of the file. • SNAPSHOT_RESULT - Value indicating whether the writing of the snapshot file was successful ("Success") or not ("Failure").

Table B.9. Command Logging

Metrics	Type	Description
voltdb_commandlog_fsync_interval_seconds	Gauge	The average interval between the last 10 fsync system calls.
voltdb_commandlog_in_use_segment_count_total	Gauge	The total number of segment files in use for command logging.
voltdb_commandlog_outstanding_bytes_bytes	Gauge	The size, in bytes, of pending command log data. For synchronous logging, this value is always zero.
voltdb_commandlog_outstanding_txns_total	Gauge	The number of transactions that have been initiated for which the log has yet to be written to disk. For synchronous logging, this value is always zero.
voltdb_commandlog_segment_count_total	Gauge	The number of segment files allocated, including currently unused segments.

Table B.10. Active(N) and XDCR

Metrics	Type	Description
voltdb_dr_conflicts_count_total	Counter	The total number of conflicts that have been recorded for this table in this partition.
voltdb_dr_constraint_violation_count_total	Counter	The number of constraint violation conflicts that occurred.
voltdb_dr_consumer_info	Metadata	Tags: • DR_STATE - A text string indicating the current state of replication. Possible values are: • UNINITIALIZED - DR has not begun yet or has stopped • INITIALIZE - DR is enabled and the consumer is attempting to contact the producer • SYNC - DR has started and the consumer is synchronizing by receiving snapshots of existing data from the master • RECEIVE - DR is underway and the consumer is receiving binary logs from the master • DISABLE - DR has been canceled for some reason and the consumer is stopping DR.
voltdb_dr_consumer_bytes_replicated_bytes	Counter	Total number of bytes this consumer received.
voltdb_dr_consumer_partition_info	Metadata	Tags: • IS_COVERED - Boolean value indicating whether this partition is currently connected to and receiving data from a matching partition on the producer cluster. • COVERING_HOST - The host name of the server in the producer cluster that is providing DR data to this partition. If IS_COVERED is "false", this label is empty. • IS_PAUSED - Boolean indicating whether this partition is paused. A partition "pauses" when the schema of the DR tables on the producer change to no longer match the consumer and all binary logs prior to the change have been processed. • CONSUMER_LIMIT_TYPE - The type of limit on the DR queue. The response is either BYTES or BUFFERS.

Metrics	Type	Description
		• LAST_APPLIED_DR_PROTOCOL - The current DR protocol version of binary logs being received and applied for this partition.
voltdb_dr_consumer_partition_available_buffers_total	Gauge	The number of free buffers left in the DR queue.
voltdb_dr_consumer_partition_available_buffer_bytes_bytes	Gauge	The number of free bytes left in the DR queue.
voltdb_dr_consumer_partition_duplicate_buffers_total	Gauge	The number of repeated buffers received after the initial packets were dropped because the queue was full.
voltdb_dr_consumer_partition_ignored_buffers_total	Gauge	The number of buffers received but dropped because the queue was full.
voltdb_dr_consumer_partition_last_applied_timestamp_seconds	Gauge	The timestamp of the last transaction successfully applied to this partition on the consumer.
voltdb_dr_consumer_partition_last_received_timestamp_seconds	Gauge	The timestamp of the last transaction received from the producer.
voltdb_dr_consumer_remote_creation_timestamp_seconds	Gauge	The timestamp when the remote cluster started for the first time.
voltdb_dr_divergence_count_total	Counter	The number of conflicts that may have resulted in divergence between the clusters, which is a subset of the total conflicts.
voltdb_dr_last_conflict_timestamp_seconds	Gauge	The timestamp of the last conflict.
voltdb_dr_missing_row_count_total	Counter	The number of missing row conflicts that occurred.
voltdb_dr_producer_cluster_info	Metadata	Informational metric, presents cluster level metadata. Tags: • DR_STATE - The current state of the DR relationship. Possible values are the following: "Disabled", "Pending", "Active", "Stopped". • LAST_APPLIED_DR_PROTOCOL - The current DR protocol version of binary logs being received and applied for this partition. • SUPPORTED_DR_PROTOCOL - The highest version of DR protocol this cluster is capable of using to send data to consumers.
voltdb_dr_producer_node_info	Metadata	Informational metric, presents node level metadata. Tags: • DR_STATE - The current state of the DR relationship. Possible values are the following: "Disabled", "Pending", "Active", "Stopped". • DR_SYNC_SNAPSHOT_STATE - The current state of the synchronization snapshot that begins replication. During normal operation, this value is "None" indicating either that replication is not active or that transactions are actively being replicated. If a synchronization snapshot is in progress, this value provides additional information about the specific activity underway.
voltdb_dr_producer_node_remote_creation_timestamp_seconds	Gauge	The timestamp (in seconds) when the remote cluster started for the first time.

Metrics	Type	Description
voltdb_dr_producer_node_rows_acked_for_sync_snapshot_total	Gauge	
voltdb_dr_producer_node_rows_in_sync_snapshot_total	Gauge	
voltdb_dr_producer_node_tasks_queue_depth_total	Gauge	The number of DR tasks waiting to be processed.
voltdb_dr_producer_partition_info	Metadata	Informational metric. Tags: • DR_STREAM_TYPE - The type of stream, which can either be "Transactions" or "Snapshot". • DR_LAST_QUEUED_ID - The ID of the last transaction queued for transmission to the consumer. • DR_LAST_ACK_ID - The ID of the last transaction acknowledged by the consumer. • DR_IS_SYNCED - Indicates whether the database is currently being replicated. If replication has not started, or the overflow capacity has been exceeded (that is, replication has failed), the value of ISSYNCED is "false". If replication is currently in progress, the value is "true". • DR_MODE - Indicates whether this particular partition is replicating data to the consumer ("NOrmal") or not ("Paused"). Only one copy of each logical partition actually sends data during replication. So for clusters with a K-safety value greater than zero, not all physical partitions will report "Normal" even when replication is in progress. • DR_CONNECTION_STATUS - Indicates whether the connection to the consumer is operational ("UP") or not ("DOWN"). • CONSUMER_LIMIT_TYPE - The type of limit on the DR queue. The response is either BYTES or BUFFERS. • CURRENT_DR_PROTOCOL - The DR protocol version currently in use when sending data to consumers. • SUPPORTED_DR_PROTOCOL - The highest version of DR protocol this cluster is capable of using to send data to consumers.
voltdb_dr_producer_partition_available_to_send_buffers_total	Gauge	The number of buffers waiting to be sent to the consumer.
voltdb_dr_producer_partition_available_to_send_total_bytes	Gauge	The number of bytes waiting to be sent to the consumer.
voltdb_dr_producer_partition_buffers_waiting_for_ack_total	Gauge	The total number of buffers in this partition waiting for acknowledgement from the consumer.
voltdb_dr_producer_partition_last_ack_timestamp_seconds	Gauge	The total number of bytes currently queued for transmission to the consumer.
voltdb_dr_producer_partition_last_queued_timestamp_seconds	Gauge	The timestamp of the last transaction queued for transmission to the consumer.
voltdb_dr_producer_partition_queued_in_memory_total_bytes	Gauge	The total number of bytes of queued data currently held in memory. If the amount of total bytes is larger than the amount in memory, the remainder is kept in overflow storage on disk.

Metrics	Type	Description
voltdb_dr_producer_partition_ queued_total_bytes	Gauge	The total number of bytes currently queued for transmission to the consumer.
voltdb_dr_producer_partition_ queue_gap_total	Gauge	The number of missing transactions between those already acknowledged by the consumer and the next available for transmission. Under normal operating conditions, this value is zero.
voltdb_dr_producer_partition_ round_trip_time_seconds	Histogram	The distribution of time it took to receive acknowledgement from the consumer.
voltdb_dr_role_info	Metadata	Informational metric. Tags: • DR_ROLE_NAME - The role of the current cluster in a DR relationship. Possible values are NONE, MASTER, REPLICA, and XDCR. • DR_STATE - The current state of the DR relationship. DISABLED, PENDING, ACTIVE, STOPPED. • REMOTE_CLUSTER_ID - The DR ID of the other DR cluster, or -1 if not available (for example, when DR is disabled or communication has not begun). • SUPPORTED_DR_PROTOCOL - The highest version of DR protocol this cluster is capable of using to send data to consumers.
voltdb_dr_row_timestamp_ mismatch_count_total	Counter	The number of timestamp mismatch conflicts that occurred.
voltdb_dr_schema_change_info	Metadata	Informational metric containing metadata. Tags: • SITE_ID - Numeric ID of the execution site on the host node. • TABLE_TYPE - The type of the table. E.g. "PersistentTable" for normal data tables. • TABLE_NAME - The name of the database table for which schema was mismatched. • CLUSTER_ID - The numeric ID of the current cluster. • REMOTE_CLUSTER_ID - The numeric ID of the remote cluster. • DR_SCHEMA_CHANGE_MATCH - A text string of "true" or "false" indicating whether the schema for the table matches on the two clusters.
voltdb_dr_schema_change_tuple_ count_total	Counter	The total number of tuples exchanged for this tuple while the schema did not match.

B.6. Streaming Data

The following tables describe the metrics related to streaming data into and out of Volt, including import, export, and topics.

Table B.11. Import

Metrics	Type	Description
voltdb_importer_failure_total	Counter	The number of import transactions that failed.
voltdb_importer_outstanding_ requests_total	Gauge	The number of records read from the import stream and waiting to be inserted into the database.

Metrics	Type	Description
voltdb_importer_retries_total	Counter	The number of attempts to replay failed transactions.
voltdb_importer_success_total	Counter	The number of import transactions that succeeded.

Table B.12. Export

Metrics	Type	Description
voltdb_export_info	Metadata	Informational metric that contains metadata. Tags: • EXPORT_IS_ACTIVE - "True" if is enabled and not blocked and is master. • EXPORT_STATUS - The current status of the export connection. • "Active" - Queue is currently exporting to the target; • "Blocked" - There is a gap in the queue and export is waiting to see if the missing records become available when a missing node rejoins; • "Dropped" - either the source stream has been dropped from the schema or the export configuration has been removed from the configuration and queue is draining any remaining records.
voltdb_export_last_acked_timestamp_seconds	Gauge	The timestamp when the last tuple was acknowledged as received by the target.
voltdb_export_last_queued_timestamp_seconds	Gauge	The timestamp when the most recent tuple was added to the export queue for this partition.
voltdb_export_latency_seconds	Histogram	The distribution of time between when records are inserted, and then acknowledged by the target.
voltdb_export_queue_gap_total	Gauge	The number of records missing from the queue for the current stream and partition.
voltdb_export_tuple_count_total	Counter	The total number of records queued to the export target since the queue was created.
voltdb_export_tuple_pending_total	Gauge	The number of records still waiting to be written to or acknowledged by the target.

Table B.13. Topics

Metrics	Type	Description
voltdb_topic_info	Metadata	Informational metric with topic metadata. Tags: • TOPIC_STATUS - • "Stable" - The queue is complete; • "Backfilling" - records are missing but are being retrieved from other servers; • "Blocked" - records are missing from all copies of the partition; • "Orphaned" - the queue is no longer being served by this partition, but is saved in case other copies of the queue are blocked or backfilling and need the data. This is a transitional state and the queue is deleted as soon as no other copies need its records. • TOPIC_RETENTION_POLICY - The retention policy for this topic.

Metrics	Type	Description
		• TOPIC_IS_MASTER - "True" or "False" indicating whether the current site is the master for the logical partition.
voltdb_topic_bytes_fetched_bytes	Gauge	The size of data sent to consumers for this partition and topic.
voltdb_topic_bytes_on_disk_bytes	Gauge	The size of data stored on disk for this partition and topic.
voltdb_topic_error_offset	Gauge	If an error occurs while encoding a message for consumers, an error is returned to the consumer, the offset of the message is recoded here.
voltdb_topic_first_offset	Gauge	The value of the first offset currently available in the topic.
voltdb_topic_first_offset_timestamp_seconds	Gauge	The timestamp when the first offset was inserted into the queue.
voltdb_topic_last_offset	Gauge	The value of the last offset in the topic.
voltdb_topic_last_offset_timestamp_seconds	Gauge	The timestamp when the most recent message (the last offset) was inserted into the queue.
voltdb_topic_skipped_rows_total	Gauge	The number of skipped rows that otherwise would cause an error. Only applies if the topic's option consumer.skip.errors is true.

B.7. User-Defined Tasks

The following tables describe the metrics for measuring and monitoring user-defined tasks.

Table B.14. Tasks

Metrics	Type	Description
voltdb_scheduler_action_status_info	Metadata	Informational metric with scheduler metadata. Tags: • PARTITION_ID - The numeric ID for the logical partition running the task procedure. Directed procedures run on each logical partition. Multi-partition procedures run on the multi-partition initiator. • TASK_NAME - The name of the task. • TASK_STATUS - The current status of the task. Possible values include: RUNNING - The task is enabled and running normally; DISABLED - The task is disabled and not running. ERROR - The task returned an error and was stopped due to the ON ERROR STOP attribute. PAUSED - The database is paused or is running on a DR replica, so the task is not currently running but will restart when the database resumes or is promoted. • TASK_ORIGIN - "System" or "User". • TASK_SCOPE - The execution scope of the task, which matches the RUN ON attribute. Possible values are "Database", "Hosts", or "Partitions".
voltdb_scheduler_procedure_execution_time_seconds	Histogram	The distribution of time the task took to execute.
voltdb_scheduler_procedure_failure_total	Counter	The number of times the procedure generated an error when run.Tags: • same as above.

Metrics	Type	Description
voltdb_scheduler_procedure_invocation_total	Counter	The total number of invocations of the task's procedure.
voltdb_scheduler_procedure_wait_time_seconds	Histogram	The distribution of time between when the procedure was scheduled to run and when it was invoked.
voltdb_scheduler_task_execution_time_seconds	Histogram	The distribution of the amount of time taken to schedule an instance of the task.
voltdb_scheduler_task_invocation_total	Counter	The total number of invocations of the task's schedule.
voltdb_scheduler_task_wait_time_seconds	Histogram	The distribution of the difference between when the task was scheduled to run and when the scheduler was invoked.
voltdb_task_priority_queue_depth_total	Gauge	The number of tasks in the queue.
voltdb_task_priority_queue_poll_count_total	Counter	The number of tasks that left the queue.
voltdb_task_priority_queue_wait_time_seconds	Histogram	The distribution of the length of time tasks were waiting in the queue.
voltdb_task_queue_depth_total	Gauge	The number of tasks in the queue.
voltdb_task_queue_poll_count_total	Counter	The number of tasks that left the queue (and started executing) in the past five seconds.
voltdb_task_queue_wait_time_seconds	Histogram	The distribution of time tasks were waiting in the queue.

B.8. System and Cluster Status

The following tables describe the metrics that report the status of the cluster and the individual systems within it, as well as additional metrics not covered by other categories.

Table B.15. System

Metrics	Type	Description
voltdb_clockskew_seconds	Gauge	The number of milliseconds difference between the system clock time of the current host and the remote host.
voltdb_file_descriptors_count_total	Gauge	The number of file descriptors open in the process.
voltdb_file_descriptors_limit_total	Gauge	The maximum number of file descriptors allowed for the process running the server.

Table B.16. Miscellaneous

Metrics	Type	Description
voltdb_balance_partitions_info	Metadata	Reports the status of recent rebalancing operations. Informational metric. Tags: • BALANCE_MOVED_ROWS - The number of rows of data that have been moved. • BALANCE_PERCENTAGE_MOVED - The percentage of the total segments that have already been moved.

Metrics	Type	Description
		• BALANCE_ROWS_PER_SECOND - The average number of rows moved per second. • BALANCE_ESTIMATED_REMAINING - The estimated time remaining until the rebalance is complete. • BALANCE_MEGABYTES_PER_SECOND - The average volume of data moved per second, measured in megabytes. • BALANCE_CALLS_PER_SECOND - The average number of rebalance work units, or transactions, executed per second. • BALANCE_CALLS_LATENCY - The average total time between start and finish of rebalance operations. (TODO: in millis? should be in seconds) • BALANCE_CALLS_TIME - The average execution time for rebalance transactions. (TODO: in millis? should be in seconds) • BALANCE_CALLS_TRANSFER_TIME - The average time spent transferring data during rebalance transactions. (TODO: in millis? should be in seconds)
voltdb_partition_count_total	Gauge	Number of unique partitions in the cluster (not including MP partition).
voltdb_xdcr_readiness_info	Metadata	-

Appendix C. Snapshot Utilities

VoltDB provides two utilities for managing snapshot files. These utilities verify that a native snapshot is complete and usable and convert the snapshot contents to a text representation that can be useful for uploading or reloading data in case of severe errors.

It is possible, as the result of a design flaw or failed program logic, for a database application to become unusable. However, the data is still of value. In such emergency cases, it is desirable to extract the data from the database and possibly reload it. This is the function that save and restore perform within VoltDB.

But there may be cases when you want to use the data created by a VoltDB snapshot elsewhere. The goal of the utilities is to assist in that process. The snapshot utilities are:

- *snapshotconvert* converts a snapshot (or part of a snapshot) into text files, creating one file for each table in the snapshot.
- *snapshotverifier* verifies that a VoltDB snapshot is complete and usable.

To use the `snapshotconvert` and `snapshotverifier` commands, be sure that the voltdb /bin directory is in your PATH, as described in the section on "Setting Up Your Environment" in the *Using VoltDB* manual. The following sections describe how to use these two commands.

snapshotconvert

snapshotconvert — Converts the tables in a VoltDB snapshot into text files.

Syntax

```
snapshotconvert {snapshot-id} --type {csv|tsv} \
        --table {table} [...] [--dir {directory}]... \
        [--outdir {directory}]

snapshotconvert --help
```

Description

SnapshotConverter converts one or more tables in a valid snapshot into either comma-separated (csv) or tab-separated (tsv) text files, creating one file per table.

Where:

{snapshot-id} is the unique identifier specified when the snapshot was created. (It is also the name of the .digest file that is part of the snapshot.) You must specify a snapshot ID.

{csv|tsv} is either "csv" or "tsv" and specifies whether the output file is comma-separated or tab-separated. This argument is also used as the filetype of the output files.

{table} is the name of the database table that you want to export to text file. You can specify the `--table` argument multiple times to convert multiple tables with a single command.

{directory} is the directory to search for the snapshot (`--dir`) or where to create the resulting output files (`--outdir`). You can specify the `--dir` argument multiple times to search multiple directories for the snapshot files. Both `--dir` and `--outdir` are optional; they default to the current directory path.

Example

The following command exports two tables from a snapshot of the flight reservation example used in the *Using VoltDB* manual. The utility searches for the snapshot files in the current directory (the default) and creates one file per table in the user's home directory:

```
$ snapshotconvert flightsnap --table CUSTOMER --table RESERVATION \
                  --type csv -- outdir ~/
```

snapshotverifier

snapshotverifier — Verifies that the contents of one or more snapshot files are complete and usable.

Syntax

```
snapshotverifier [snapshot-id] [--dir {directory}] ...

snapshotverifier --help
```

Description

SnapshotVerifier verifies one or more snapshots in the specified directories.

Where:

[snapshot-id] is the unique identifier specified when the snapshot was created. (It is also the name of the .digest file that is part of the snapshot.) If you specify a snapshot ID, only snapshots matching that ID are verified. If you do not specify an ID, all snapshots found will be verified.

{directory} is the directory to search for the snapshot. You can specify the `--dir` argument multiple times to search multiple directories for snapshot files. If you do not specify a directory, the default is to search the current directory.

Examples

The following command verifies all of the snapshots in the current directory:

```
$ snapshotverifier
```

This example verifies a snapshot with the unique identifier "flight" in either the directory `/etc/voltdb/save` or `~/mysaves`:

```
$ snapshotverifier flight --dir /etc/voltdb/save/ --dir ~/mysaves
```