



DSE 6.0 Developer Guide (Latest version)

© 2018 DataStax, Inc. All rights reserved.

DataStax, Titan, and TitanDB are registered trademark of DataStax, Inc. and its subsidiaries in the United States and/or other countries.

Apache Cassandra, Apache, Tomcat, Lucene, Solr, Hadoop, Spark, TinkerPop, and Cassandra are trademarks of the Apache Software Foundation or its subsidiaries in Canada, the United States and/or other countries.

Table of Contents

Getting started.....	9
About this guide.....	9
New features.....	10
Key features.....	13
DSE release notes.....	17
DSE 6.0.2 release notes.....	17
Cassandra enhancements for 6.0.2.....	21
Upgrade advice for 6.0.2.....	22
TinkerPop changes for DSE 6.0.2.....	27
DSE 6.0.1 release notes.....	27
Cassandra enhancements for 6.0.1.....	31
Upgrade advice 6.0.1.....	32
Tinkerpop changes 6.0.1.....	37
DSE 6.0.0 release notes.....	37
Cassandra enhancements 6.0.....	47
Upgrade advice for 6.0.0.....	53
TinkerPop Changes.....	56
Bulk loader release notes.....	56
Bulk Loader 1.1.0 release notes.....	57
Bulk Loader 1.0.2 release notes.....	57
Bulk Loader 1.0.1 release notes.....	58
Studio release notes.....	59
Studio 6.0.1 release notes.....	59
Studio 6.0.0 release notes.....	59
Installing DSE.....	61
Configuration.....	62
Recommended production settings.....	62

YAML and configuration properties.....	68
cassandra.yaml.....	68
dse.yaml.....	104
cassandra-rackdc.properties.....	149
cassandra-topology.properties.....	150
Cloud provider snitches.....	151
Amazon EC2 single-region snitch.....	151
Amazon EC2 multi-region snitch.....	153
Google Cloud Platform.....	154
Apache CloudStack snitch.....	155
Start-up parameters.....	155
Choosing a compaction strategy.....	161
NodeSync service and table options.....	163
About NodeSync.....	163
Starting and Stopping the NodeSync service.....	164
Enabling NodeSync validation.....	165
Tuning validations.....	165
Manually starting validation.....	167
Configuring Virtual Nodes.....	168
Virtual node (vnode) configuration.....	168
DSE advanced functionality.....	170
DSE Analytics.....	170
About DSE Analytics.....	170
Setting the replication factor for analytics keyspaces.....	171
DSE Analytics and Search integration.....	172
About DSE Analytics Solo.....	174
Analyzing data using Spark.....	176
DSEFS (DataStax Enterprise file system).....	273
DSE Search.....	304

About DSE Search.....	304
Configuring DSE Search.....	309
DSE Search operations.....	335
Solr interfaces.....	351
HTTP API SolrJ and other Solr clients.....	362
DSE Graph.....	363
About DSE Graph.....	363
DSE Graph Terminology.....	366
DSE Graph QuickStart.....	368
DSE Graph, OLTP, and OLAP.....	404
Graph anti-patterns.....	421
DSE Graph data modeling.....	423
Using DSE Graph.....	431
DSE Graph Analysis with DSE Analytics.....	607
DSE Graph Tools.....	623
Starting the Gremlin console.....	624
DSE Graph Reference.....	627
DSE Advanced Replication.....	742
About DSE Advanced Replication.....	742
Architecture.....	743
Traffic between the clusters.....	746
Terminology.....	748
Getting started.....	749
Keyspaces.....	760
Data types.....	760
Operations.....	761
CQL queries.....	780
Metrics.....	782

Managing invalid messages.....	790
Managing audit logs.....	791
Tools.....	793
DataStax Bulk Loader (dsbulk).....	793
dse.....	793
dse commands.....	793
dse client-tool.....	799
cassandra.....	800
dsetool.....	804
About dsetool.....	804
Connection options.....	805
core_indexing_status.....	807
create_core.....	809
createsystemkey.....	813
encryptconfigvalue.....	816
get_core_config.....	816
get_core_schema.....	818
help.....	820
index_checks.....	821
infer_solr_schema.....	823
inmemorystatus.....	825
list_index_files.....	826
list_subranges.....	828
listjt.....	830
managekmip list.....	831
managekmip expirekey.....	833
managekmip revoke.....	834
managekmip destroy.....	836

node_health.....	837
partitioner.....	839
perf.....	840
read_resource.....	843
rebuild_indexes.....	844
reload_core.....	846
ring.....	848
sparkmaster cleanup.....	849
sparkworker restart.....	850
status.....	852
stop_core_reindex.....	853
tieredtablestats.....	854
tsreload.....	857
unload_core.....	858
upgrade_index_files.....	859
write_resource.....	861
Stress tools.....	863
Interpreting the output of cassandra-stress.....	863
fs-stress tool.....	864
Operations.....	867
Starting and stopping DSE.....	867
Starting as a service.....	867
Starting as a stand-alone process.....	870
Stopping a node.....	872
Clearing data from DSE.....	873
Studio.....	874
About DataStax Studio.....	874
Studio security.....	874

About notebooks.....	875
Using Studio.....	876
Starting and stopping Studio.....	876
Creating a connection.....	877
Using graph.....	879
Using CQL.....	886
Using Spark SQL.....	887
Listing notebooks.....	888
Using notebook history.....	889
Defining run behavior.....	891
Exporting notebooks.....	892
Importing notebooks.....	893
Configuring Studio.....	894
Configuring Studio.....	894
configuration.yaml.....	895
Advanced configuration options.....	897
User data.....	898
JVM settings.....	899
Studio reference.....	899
Creating a simple notebook.....	899
Configuring a notebook.....	901
Default imports.....	902
Keyboard shortcuts.....	904
Notebook cells.....	905
FAQ.....	907
Troubleshooting DataStax Studio.....	908
CQL.....	909
WH - SSTable tool.....	910

Getting started

Information about using the Developer Guide, plus new and key features in DataStax Enterprise.

Information about using this guide, plus new and key features in DataStax Enterprise 6.0.

About the DataStax Enterprise 6.0 Developer Guide

Information for developers about creating enterprise-level applications with DataStax Enterprise 6.0.

The *Developer Guide* provides information for creating enterprise-level applications that require real-time always available storage, search, and analytics. DataStax Enterprise seamlessly integrates your code, allowing applications to utilize a breadth of techniques to produce a mobile app or online applications.

Tip: Developing applications requires a basic understanding of how DataStax Enterprise works and how it differs from a relational database. In conjunction with this guide, you should refer to the [Architecture Guide](#) for background information. This will save you a lot of time when developing your data models, applications, and using the features in DataStax Enterprise. To get started, be sure to read the [DataStax Enterprise 6.0 FAQ](#) and [Architecture in brief](#).

As a developer, you must be familiar with [data modeling](#) and CQL.

To ensure that you get the best experience in using this document, take a moment to look at the [Tips for using DataStax documentation](#). This page provides information on search, navigational aids, and providing feedback.

DataStax supplies a number of [drivers](#) so that CQL statements and search commands can be passed from client to cluster and back. Other tasks can be accomplished using [OpsCenter](#).

This guide includes documentation for:

[Install methods](#)

Types of installs generally used by developers.

[DSE Analytics \(page 170\)](#)

DSE Analytics uses Apache Spark™ to perform analytic queries over large sets of data. Topics include starting, configuring, running commands against a remote cluster, accessing data, and a number of examples.

[DSE Graph \(page 363\)](#)

A graph database for storing information about the relationships between entries. Topics include getting started, terminology, data modeling, anti-patterns, importing data, tools, and graph analytics.

[DSE Search \(page 304\)](#)

DSE Search simplifies using search applications for data stored in a database. DSE Search integrates Apache Solr™ to manage search indexes with a persistent store.

[DSEFS \(page 273\)](#)

DataStax Enterprise File System is distributed file system for storing very large sets of data and encapsulated that data across a DSE cluster. It can be stored for processing by DSE analytics and other tools.

DataStax Studio (page 874)

An IDE for syntax validation, type checking, validations specific to the domain, and content assistance for [CQL \(Cassandra Query Language\)](#).

Other information sources

Architecture Guide	How the DataStax Enterprise database works.
Administrators Guide	Information about capacity planning, installation, configuration, migration, performance monitoring, security, backup, data recovery and more.
CQL for DSE 6.0	Cassandra Query Language (CQL) is a query language for the DataStax Enterprise database. You can interact with the database is using the CQL shell, cqlsh , and DataStax Studio (page 874) .
Landing pages	Getting started with DSE, supported platforms, product compatibility, third-party software, resources for additional information, and earlier documentation.
OpsCenter	Installing and using DSE OpsCenter.
Lifecycle Manager	Using Lifecycle Manage to create, configure, and manage clusters.
DSE drivers	C/C++ driver , C# driver , Java driver , Node.js driver , PHP driver , Python driver , and Ruby driver .
Planning and testing DSE deployments	Includes hardware selection, estimating disk capacity, anti-patterns, and cluster testing.
DSE Troubleshooting Guide	Various troubleshooting topics including Linux settings, search, analytics, security, starting DSE, and installing.
Upgrade Guide	Information on upgrading various versions of DataStax Enterprise and upgrading from Apache Cassandra to DataStax Enterprise 6.0.
Sources of support	DataStax Support , DataStax Academy forums , Stack Overflow for DataStax Enterprise , Stack Overflow for the DataStax Java client driver and the DataStax PHP driver .

DataStax Enterprise 6.0 new features

Features released in DataStax Enterprise 6.0.

DataStax Enterprise, built on Apache Cassandra™, powers the Right-Now Enterprise with an always-on, distributed cloud database designed for hybrid cloud. DataStax Enterprise (DSE) 6.0 dramatically increases performance and eases operational management with new features and enhancements.

Be sure to read the DataStax Enterprise 6.0 [release notes \(page 17\)](#).

Feature	Description
NodeSync (page 163)	<p>DSE NodeSync (page 163) removes the need for manual repair operations in DSE's distribution of Cassandra and eliminates cluster outages that are attributed to manual repair failures. This equates to operational cost savings, reduced support cycles, and reduced application management pain. NodeSync also makes applications run more predictably, making capacity planning easier. NodeSync's advantages for operational simplicity extend across the whole data layer including database, search, and analytics.</p> <p>Be sure to read the DSE NodeSync: Operational Simplicity at its Best blog.</p>
Advanced Performance	<p>DSE Advanced Performance delivers numerous performance advantages over open-source Apache Cassandra including:</p> <ul style="list-style-type: none"> • Thread per core (TPC) and asynchronous architecture: A coordination-free design, DSE's thread-per-core architecture provides up to 2x more throughput for read and write operations. • Storage engine optimizations that provide up to half the latency of open source Cassandra and include optimized compaction. • DataStax Bulk Loader Up to 4x faster loads and unloads of data over current data loading utilities. Up to 4 times faster than current data loading utilities. Be sure to read the Introducing DataStax Bulk Loader blog. • Continuous paging (page 213) improves DSE Analytics read performance by up to 3x over open source Apache Cassandra and Apache Spark. <p>Be sure to read the DSE Advanced Performance blog.</p>
DSE TrafficControl	<p>DSE TrafficControl provides a backpressure mechanism to avoid overloading DSE nodes with client or replica requests that could make DSE nodes unresponsive or lead to long garbage collections and out of memory errors. DSE TrafficControl is enabled by default and comes pre-tuned to accommodate very different workloads, from simple reads and writes to the most extreme workloads. It requires no configuration.</p>
Automated Upgrades for patch releases	Part of OpsCenter LifeCycle Manager, the Upgrade Service handles patch upgrades of DSE clusters at the data center, rack, or node level with up to 60% less manual involvement. The Upgrade Service allows you to easily clone your existing configuration profile to ensure compatibility with DSE upgrades. Be sure to read the Taking the Pain Out of Database Upgrades blog.

Feature	Description
DSE Analytics (page 170)	<p>New features in DSE Analytics include:</p> <ul style="list-style-type: none"> • AlwaysOn SQL (page 245) with advanced security (page 250), ensures around-the-clock uptime for analytics queries with the freshest, secure insight. It is interoperable with existing business intelligence tools that utilize ODBC/JDBC and other Spark-based tools. Be sure to read the Introducing AlwaysOn SQL for DSE Analytics blog. • Structured Streaming (page 234) simple, efficient, and robust streaming of data from Apache Kafka, file systems, or other sources. • Enhanced Spark SQL (page 235) support allows you to execute Spark queries using a variation of the SQL language. Spark SQL includes APIs for returning Spark Datasets in Scala and Java, and interactively using an SQL shell or visually through DataStax Studio notebooks. <p>Be sure to read the What's New for DataStax Enterprise Analytics 6 blog.</p>
DSE Graph (page 363)	<p>New features in DSE Graph include:</p> <ul style="list-style-type: none"> • Better throughput for DSE Graph due to Advanced Performance improvements, resulting in DSE Graph handling more requests per node. • Smart Analytics Query Routing: the DSE Graph engine automatically routes a Gremlin OLAP traversal to the correct implementation (DSE Graph Frames or Gremlin OLAP) for the fastest and best execution. • Advanced Schema Management provides the ability to remove any graph schema element (page 464), not just vertex labels or properties. • The Batches in DSE Graph Fluent API adds the ability to execute DSE Graph statements in batches to speeds up writes to DSE Graph. • TinkerPop 3.3.0. DataStax has added a lot of great enhancements to the Apache TinkerPopTM tool suite. Enhancements have proved faster, more robust graph querying and provided a better developer experience. <p>Be sure to read the What's New in DSE Graph 6 blog.</p>
DSE Security	<p>New security features include:</p> <ul style="list-style-type: none"> • Private Schemas (page 97): Control who can see what parts of a table definition, critical for security compliance best practices. • Separation of Duties: Create administrator roles who can carry out everyday administrative tasks without having unnecessary access to data. • Auditing by Role: Focus your audits on the users you need to scrutinize. You can now elect to audit activity by user type and increase the signal to noise ratio by removing application tier system accounts from the audit trail. • Unified Authorization for DSE Analytics (page 208): Additional protection for data used for analytics operations. <p>Be sure to read the Safe data? Check. DataStax Enterprise Advanced Security blog.</p>

Feature	Description
DSE Search (page 304)	<p>Built with a production-certified version of Apache Solr™ 6, DSE Search requires less configuration, improved search data consistency, and a more synchronous write path for indexing data with less moving pieces to tune and monitor. DSE 5.1 introduced index management CQL and cqlsh commands to streamline operations and development. DSE 6.0 adds a wider array of CQL query functionality and indexing support.</p> <p>Be sure to read the What's New for Search in DSE 6 blog.</p>
Drivers	<p>DataStax drivers are updated for DSE 6.0, including:</p> <ul style="list-style-type: none"> The Batches in DSE Graph Fluent API adds the ability to execute DSE Graph statements in batches to speed up writes to DSE Graph. The C# and Node.js DataStax drivers include Batches in DSE Graph Fluent API, as well as the Java and Python drivers. <p>Be sure to read the What's New With Drivers for DSE 6 blog.</p>
DataStax Studio (page 874)	<p>Improvements to DSE Studio further ease DSE development include:</p> <ul style="list-style-type: none"> Notebook Sharing (page 875): Easily collaborate with your colleagues to develop DSE applications using the new import (page 893) and export (page 892) capabilities. Spark SQL support (page 887): Query and analyze data with Spark SQL using DataStax Studio's visual and intelligent notebooks, which provide syntax highlighting, auto-code completion and correction, and more. Interactive Graphs (page 880): explore and configure DSE Graph schemas with a whiteboard-like view that allows you to drag your vertices and edges. Notebook History (page 889): provides a historical dated record with descriptions and change events that makes it easy to track and rollback changes. <p>Be sure to read the Announcing DataStax Studio 6 blog.</p>

DataStax Enterprise 6.0 key features

A brief description of the DataStax Enterprise, its key features, and integrated tools.

DataStax powers the Right-Now Enterprise with the always-on, distributed cloud database built on Apache Cassandra™ and designed for hybrid cloud. Maintain hybrid cloud flexibility with all the benefits of a distributed cloud database on any public cloud as well as on-premises. DSE 6.0 delivers all the advantages of Apache Cassandra with none of the complexities for easy enterprise-wide adoption of data management at scale.

A note about terminology

In the DataStax Enterprise 6.0 documentation, the Cassandra database and related commercial-only features are referred to cumulatively as the DataStax Enterprise database, the database, or DataStax Enterprise depending on the context.

Database

DSE 6.0 builds on its best-in-class performance for both reads and writes -- critical for high-volume data ingestion requirements. DSE 6.0 doubles the performance, yielding twice the responsiveness and throughput versus the prior version with the same hardware.

The DSE database is a partitioned row store database. It is a massively scalable NoSQL database that provides automatic data distribution across all nodes in a cluster. There is nothing programmatic that a developer or administrator needs to do or code to distribute data across a cluster.

The database provides built-in and customizable replication, which stores redundant copies of data across the cluster. This means that if any node in a cluster goes down, one or more copies of that node's data are available on other nodes. Replication can be configured to work across one datacenter, many datacenters, and multiple cloud availability zones.

DataStax Enterprise advanced functionality

[DSE Analytics \(page 170\)](#)

Built on a production-certified version of the industry standard Apache Spark™, DSE Analytics allows you to easily stream data to or from existing databases, including Apache Hadoop™, HDFS, S3, Oracle, MySQL, IBM DB2, and external Apache Spark clusters. DSE Analytics builds on Apache Spark's ability to eliminate single points of failure, deliver faster performance over open source, and enable DSE's real-time search, analytics, and graph capabilities.

[DSE Graph \(page 363\)](#)

Handles large, complex, relationship-heavy data sets through a highly-scalable graph database, capable of executing both transactional and analytical workloads in an always-on, horizontally scalable data platform.

[DSE Search \(page 304\)](#)

Integrated with Apache Solr™ 6.0 to provide continuously available search. Index management CQL and [cqlsh commands](#) streamline operations and development.

[DSE Advanced Security](#)

A feature suite for protecting data in enterprise environments. It includes advanced mechanisms for authentication and authorization, encryption of data in-flight and at-rest, data auditing, and row-level access control (RLAC).

[DSE Advanced Replication \(page 742\)](#)

DSE Advanced Replication lets you have a single cluster that can define a primary hub with multiple spokes, allowing configurable distributed [data replication](#) from source clusters to destination clusters bi-directionally. It's designed to support microservice analytics commonly found in retail environments and tolerate sporadic connectivity that can occur in constrained environments, such as oil-and-gas remote sites and cruise ships.

Advanced Replication solves the complex setup and limitations when using Cassandra in these environments. Moreover, any type of workload is supported at both the edge and the hub, allowing for advanced Search and Analytic use cases at remote and central locations

[DSE Tiered Storage](#)

Part of the multiple storage options offered in DataStax Enterprise for optimizing performance and cost goals. It automates the smart movement of data across different types of storage media to improve performance, lower costs, and reduce manual processes.

DSE Multi-Instance

Provides multi-tenancy to run multiple DataStax Enterprise nodes on a single host machine to leverage large server capabilities. This allows you to utilize the price-performance sweet spot in the contemporary hardware market and ensures that cost saving goals are met without compromising performance and availability.

DSE In-Memory

Part of the multiple storage options offered in DataStax Enterprise for optimizing performance and cost goals. It provides the ability to set which parts (some or all) of a database to reside fully in RAM. DSE in-memory provides lightning-fast performance for read-intensive situations.

Other DataStax Enterprise docs

Planning and testing DataStax Enterprise deployments

Information on choosing hardware, capacity planning, estimating disk capacity, anti-patterns, planning for the cloud, and testing your cluster before deployment.

Troubleshooting DataStax Enterprise

Troubleshooting for installing and starting DSE, Linux settings, security, DSE Graph, DSE Analytics, DSE Search, DataStax Studio, and more.

Development and production tools

Integrated DataStax products

[cqlsh, Gremlin console \(page 624\)](#)

Developer tools

[DataStax Studio \(page 874\)](#), Javadoc, and demos

Production tools

[OpsCenter and Lifecycle Manager](#), [nodetool](#), [dsetool \(page 804\)](#), [DSE Graph Loader \(page 471\)](#), [DataStax Bulk Loader](#)

DataStax Drivers

DataStax drivers come in two types: DataStax drivers for DataStax Enterprise 5.0 and later and DataStax drivers for Apache Cassandra™.

Download drivers from [DataStax Academy](#). For version compatibility, see the [DataStax drivers page](#).

Drivers for DSE 5.0 and later

These drivers can only be used with DataStax Enterprise and support the advanced functionality of DataStax Enterprise 5.0 and later:

- [C/C++ driver](#)
- [C# driver](#)
- [Java driver](#)
- [Node.js driver](#)
- [PHP driver](#)
- [Python driver](#)

- [Ruby driver](#)

DataStax drivers for Apache Cassandra

These drivers can be used with DataStax Enterprise but do not support its advanced functionality:

- [C/C++ driver](#)
- [C# driver](#)
- [Java driver](#)
- [Node.js driver](#)
- [PHP driver](#)
- [Python driver](#)
- [Ruby driver](#)

DataStax Enterprise 6.0 release notes

DataStax Enterprise release notes include cluster requirements, upgrade guidance, components, changes and enhancements, issues, and resolved issues for DataStax Enterprise 6.0.x.

DataStax Enterprise release notes cover cluster requirements, upgrade guidance, components, changes and enhancements, issues, and resolved issues for DataStax Enterprise (DSE) 6.0.x.

Note: Each point release contains a highlights and executive summary section to provide guidance and add visibility to important improvements.

Requirement for uniform clusters

All nodes in each cluster must be uniformly licensed to use the same subscription. For example, if a cluster contains 5 nodes, all 5 nodes within that cluster must be either DataStax Basic, or all 5 nodes must be DataStax Enterprise. Mixing different subscriptions within a cluster is not permitted. “Cluster” means a collection of nodes running the software which communicate with one another via Gossip, and “Gossip” means the mechanism within the software enabling related nodes to communicate with one another. For more information, see [Enterprise Terms](#).

Before you upgrade

The latest version of DataStax Enterprise is 6.0.2.

Be sure to read all of the relevant upgrade documentation, including [Planning your DataStax Enterprise upgrade](#). Upgrades are supported from [DSE 5.1 to 6.0](#) and from [DSE 5.0 to 6.0](#).

- Product compatibility
 - # DataStax Enterprise 6.0 is compatible with:
 - # DSE OpsCenter 6.5
 - # DataStax Studio 6.0
 - # [DataStax Drivers](#)

Depending on the driver version, you might need to recompile your client application code. See [Upgrading DataStax drivers](#).
 - # DataStax Bulk Loader compatibility
 - # Can load data into DSE 5.0 or later
 - # Can unload data from any Apache CassandraTM 2.1 or later data source

DSE 6.0.2 release notes

19 July 2018

In this section:

- [6.0.2 Components \(page 18\)](#)

- [6.0.2 DSE core \(page 18\)](#)
- [6.0.2 DSE Analytics \(page 19\)](#)
- [6.0.2 DSEFS \(page 20\)](#)
- [6.0.2 DSE Graph \(page 20\)](#)
- [6.0.2 DSE Search \(page 21\)](#)
- [DataStax Bulk Loader 1.1.0 \(page 21\)](#)
- [Cassandra enhancements for DSE 6.0.2 \(page 21\)](#)
- [General upgrade advice for DSE 6.0.2 \(page 22\)](#)
- [TinkerPop changes for DSE 6.0.2 \(page 27\)](#)

6.0.2 Components

All components from DSE 6.0.0 are listed. Components that are updated for DSE 6.0.2 are indicated.

- Apache Solr™ 6.0.1.1.2321 (updated)
- Apache Spark™ 2.2.1.2
- Apache Tomcat® 8.0.47
- DSE Java Driver 1.6.5
- Netty 4.1.13.11.dse
- Spark Jobserver 0.8.0.45 DSE custom version
- TinkerPop 3.3.3 with additional production-certified [changes \(page 27\)](#) (updated)

DSE 6.0 is compatible with Apache Cassandra™ 3.11 and adds additional production-certified [changes \(page 21\)](#).

DSE 6.0.2 Highlights

High-value benefits of upgrading to DSE 6.0.2 include these highlights:

DSE Analytics and DSEFS

- Fixed issue where CassandraConnectionConf creates excessive database connections and reports too many HashedWheelTimer instances. (DSP-16365)

DSE Graph

- Fixed several edge cases of using search indexes. (DSP-14802, DSP-16292)

DSE Search

- Search index permissions can be applied at the keyspace level. (DSP-15385)
- Schemas with stored=true work because stored=true is ignored. The workaround for 6.0.x upgrades with schema.xml fields with “indexed=false, stored=true, docValues=true” is no longer required. (DSP-16392)
- Minor bug fixes and error handling improvements. (DSP-16435, DSP-16061, DSP-16078)

6.0.2 DSE core

Changes and enhancements:

- [sstableloader](#) supports custom config file locations. (DSP-16092)

- -d option to create local encryption keys without configuring the directory in dse.yaml. (DSP-15380)

Resolved issues:

- Tool sstablepartitions for identifying large partitions. (DB-803)
- Show delegated snitch in nodetool describecluster. (DB-2057)
- Use more precise grep patterns to prevent accidental matches in cassandra-env.sh. (DB-2114)
- Add missing equality sign to SASI schema snapshot. (DB-2129)
- For tables using DSE Tiered Storage, nodetool cleanup places cleaned SSTables in the wrong tier. (DB-2173)
- Support creating system keys before the output directory is configured in dse.yaml. (DSP-15380)
- Client prepared statements are not populated in system.prepared_statements table. (DSP-15900)
- Improved compatibility with external tables stored in the DSE Metastore in remote systems. (DSP-16561)

6.0.2 DSE Analytics

Changes and enhancements:

- Apache Hadoop Azure libraries for Hadoop 2.7.1 have been added to the Spark classpath to simplify integration with Microsoft Azure and Microsoft Azure Blob Storage. (DSP-15943)
- AlwaysOn SQL (AOSS) improvements:
 - # AlwaysOn SQL (AOSS) support for enabling Kerberos and SSL at the same time. (DSP-16087)
 - # Add 120 seconds wait time so that Spark Master recovery process completes before status check of AlwaysOn SQL (AOSS) app. (DSP-16249)
 - # AlwaysOn SQL (AOSS) driver continually runs on a node even when DSE is down. (DSP-16297)
 - # AlwaysOn SQL (AOSS) binds to [native_transport_address \(page 91\)](#). (DSP-16469)
 - # Improved defaults and errors for AlwaysOn SQL (AOSS) workpool. (DSP-16343)

Resolved issues:

- Problems with temporary and data directories for Spark applications. (DSP-15476, DSP-15880)
 - # DSE client applications, like Spark, will not start if user HOME environment variable is not defined, user home directory does not exist, or the current user does not have write permissions.
 - # Temporary data directory for AOSS is /var/log/spark/rdd, the same as the server-side temporary data location for Spark. [Configurable \(page 208\)](#) with SPARK_EXECUTOR_DIRS environment variable in spark-env.sh.

- # If TMPDIR environment variable is missing, /tmp is set for all DSE apps. If /tmp directory does not exist, it is created with 1777 permissions. If directory creation fails, perform a hard stop.
- CassandraConnectionConf creates excessive database connections and reports too many HashedWheelTimer instances. (DSP-16365)
- Need to disable cluster object JMX metrics report to prevent count exceptions spam in Spark driver log. (DSP-16442)
- Fixed Spark-Connector dependencies and published [SparkBuildExamples](#). (DSP-16699)

6.0.2 DSEFS

Changes and enhancements:

- DSEFS [operations \(page 280\)](#): chown, chgrp, and chmod support recursive (-R) and verbose (-v) flag. (DSP-14238)
- Client and internode connection improvements. (DSP-14284, DSP-16065)
 - # DSEFS clients close idle connections after 60 seconds, configurable in [dse.yaml \(page 139\)](#).
 - # Idle DSEFS internode connections are closed after 120 seconds. Configurable with new dse.yaml option [internode_idle_connection_timeout_ms \(page 139\)](#).
 - # Configurable connection pool with [core_max_concurrent_connections_per_host \(page 139\)](#).
- DSEFS clients close idle connections after 60 seconds, configurable in [dse.yaml \(page 139\)](#). (DSP-14284)
- Improvements to DataSourceInputStream remove possible lockup. (DSP-16409)
 - # If the second read is issued after a failed read, it is not blocked forever. The stream is automatically closed on errors, and subsequent reads will fail with IllegalStateException.
 - # The timeout message includes information about the underlying DataSource object.
 - # No more reads are issued to the underlying DataSource after it reports hasMoreData = false.
 - # The read loop has been simplified to properly move to the next buffer if the requested number of bytes hasn't been delivered yet.
 - # Empty buffer returned from the DataSource when hasMoreData = true is not treated as an EOF. The read method validates offset and length arguments.
- Security improvement: DSEFS uses an isolated native memory pool for file data and metadata sent between nodes. This isolation makes it harder to exploit potential memory management bugs. (DSP-16492)

Resolved issues:

- DSEFS silently fails when TCP port 5599 is not open between nodes. (DSP-16101)

6.0.2 DSE Graph

Changes and enhancements:

- Vertices and vertex properties created or modified with graphframes respect TTL as defined in the schema. In earlier versions, vertices and vertex properties had no TTL. Edges created or modified with graphframes continue to have no TTL. (DSP-15555)
- Improved Gremlin console authentication configuration. (DSP-9905)

Resolved issues:

- 0 (zero) is not treated as unlimited abort of max num errors. (DGL-307)
- Search indexes are broken for multi cardinality properties. (DSP-14802)
- DGF interceptor does not take into account GraphStep parameters with g.V(id) queries. (DSP-16172)
- The clause LIMIT does not work in a graph traversal with search predicate TOKEN, returning only a subset of expected results. (DSP-16292)

6.0.2 DSE Search

Changes and enhancements:

- The node health option [uptime_ramp_up_period_seconds \(page 118\)](#) default value in dse.yaml is reduced to 3 hours (10800 seconds). (DSP-15752)
- CQL solr_query supports Solr facet heatmaps. (DSP-16404)
- Improved handling of asynchronous I/O timeouts during search read-before-write. (DSP-16061)
- Schemas with stored=true work because stored=true is ignored. (DSP-16392)
- Use monotonically increasing time source for search query execution latency calculation. (DSP-16435)

Resolved issues:

- Search index [permissions](#) can be applied at keyspace level. (DSP-15835)
- The encryptors thread cache in ThreadLocalIndexEncryptionConfiguration leaves entries in the cache. (DSP-16078)
- Classpath conflict between Lucene and SASI versions of Snowball. (DSP-16116)
- Indexing fails if fields have 'indexed=false', 'stored=true', and `docValues=true'. (DSP-16392)

DataStax Bulk Loader 1.1.0

Changes and enhancements:

- DataStax Bulk Loader ([dsbulk](#)) version 1.1.0 is automatically installed with DataStax Enterprise 6.0.2, and can also be installed as a standalone tool. See [DataStax Bulk Loader 1.1.0 release notes](#). (DSP-16484)

Cassandra enhancements for DSE 6.0.2

DataStax Enterprise 6.0.2 is compatible with Apache Cassandra™ 3.11 and adds no production-certified enhancements.

DataStax Enterprise 6.0.2 is compatible with Apache Cassandra™ 3.11 and adds no production-certified enhancements.

General upgrade advice for DSE 6.0.2

General upgrade advice for DataStax Enterprise 6.0.2.

General upgrade advice for DataStax Enterprise (DSE) 6.0.2:

PLEASE READ: MAXIMUM TTL EXPIRATION DATE NOTICE (CASSANDRA-14092)
=====

(General upgrading instructions are available in the next section)

The maximum expiration timestamp that can be represented by the storage engine is 2038-01-19T03:14:06+00:00, which means that inserts with TTL that expire after this date are not currently supported. By default, INSERTS with TTL exceeding the maximum supported date are rejected, but it's possible to choose a different expiration overflow policy. See CASSANDRA-14092.txt for more details.

Prior to 5.0.12 (5.0.X) and 5.1.7 (5.1.x) there was no protection against INSERTS with TTL expiring after the maximum supported date, causing the expiration time field to overflow and the records to expire immediately. Clusters in the 4.X and lower series are not subject to this when assertions are enabled. Backed up SSTables can be potentially recovered and recovery instructions can be found on the CASSANDRA-14092.txt file.

If you use or plan to use very large TTLS (10 to 20 years), read CASSANDRA-14092.txt for more information.

GENERAL UPGRADING ADVICE FOR ANY VERSION
=====

Snapshottting is fast (especially if you have JNA installed) and takes effectively zero disk space until you start compacting the live data files again. Thus, best practice is to ALWAYS snapshot before any upgrade, just in case you need to roll back to the previous version. (Cassandra version X + 1 will always be able to read data files created by version X, but the inverse is not necessarily the case.)

When upgrading major versions of Cassandra, you will be unable to restore snapshots created with the previous major version using the 'sstableloader' tool. You can upgrade the file format of your snapshots using the provided 'sstableupgrade' tool.

4.0

====

New features

- SSTableDump now supports the -l option to output each partition as its own json object
 - See CASSANDRA-13848 for more detail
- The currentTimestamp, currentDate, currentTime and currentTimeUUID functions have been added.
 - See CASSANDRA-13132
- Support for arithmetic operations between `timestamp`/`date` and `duration` has been added.
 - See CASSANDRA-11936
- Support for arithmetic operations on number has been added. See CASSANDRA-11935
 - Preview expected streaming required for a repair (nodetool repair --preview), and validate the consistency of repaired data between nodes (nodetool repair --validate). See CASSANDRA-13257
 - Support for selecting Map values and Set elements has been added for SELECT queries. See CASSANDRA-7396
 - The initial build of materialized views can be parallelized. The number of concurrent builder threads is specified by the property `cassandra.yaml:concurrent_materialized_view_builders`. This property can be modified at runtime through both JMX and the new `setconcurrentviewbuilders` and `getconcurrentviewbuilders` nodetool commands. See CASSANDRA-12245 for more details.

Upgrading

- Cassandra 4.0 removed support for COMPACT STORAGE tables. All Compact Tables have to be migrated using `ALTER ... DROP COMPACT STORAGE` statement in 3.0/3.11.
 - Cassandra starting 4.0 will not start if flags indicate that the table is non-CQL.
 - Syntax for creating compact tables is also deprecated.
 - Support for legacy auth tables in the system_auth keyspace (users, permissions, credentials) and the migration code has been removed.

Migration

- of these legacy auth tables must have been completed before the upgrade to 4.0 and the legacy tables must have been removed. See the 'Upgrading' section
 - for version 2.2 for migration instructions.
 - Cassandra 4.0 removed support for the deprecated Thrift interface.

Amongst

- Tother things, this imply the removal of all yaml option related to thrift
 - (`start_rpc`, `rpc_port`, ...).

- Cassandra 4.0 removed support for any pre-3.0 format. This means you
 - cannot upgrade from a 2.x version to 4.0 directly, you have to upgrade to a 3.0.x/3.x version first (and run upgradesstable). In particular, this mean Cassandra 4.0 cannot load or read pre-3.0 sstables in any way: you will need to upgrade those sstable in 3.0.x/3.x first.
- Upgrades from 3.0.x or 3.x are supported since 3.0.13 or 3.11.0, previous versions will causes issues during rolling upgrades (CASSANDRA-13274).
- Cassandra will no longer allow invalid keyspace replication options, such as invalid datacenter names for NetworkTopologyStrategy. Operators MUST add new nodes to a datacenter before they can set set ALTER or CREATE keyspace replication policies using that datacenter. Existing keyspaces will continue to operate, but CREATE and ALTER will validate that all datacenters specified exist in the cluster.
- Cassandra 4.0 fixes a problem with incremental repair which caused repaired data to be inconsistent between nodes. The fix changes the behavior of both full and incremental repairs. For full repairs, data is no longer marked repaired. For incremental repairs, anticompaaction is run at the beginning of the repair, instead of at the end. If incremental repair was being used prior to upgrading, a full repair should be run after upgrading to resolve any inconsistencies.
- Config option index_interval has been removed (it was deprecated since 2.0)
- Deprecated repair JMX APIs are removed.
- The version of snappy-java has been upgraded to 1.1.2.6
- the miniumum value for internode message timeouts is 10ms.

Previously, any positive value was allowed. See `cassandra.yaml` entries like `read_request_timeout_in_ms` for more details.

- Config option `commitlog_sync_batch_window_in_ms` has been deprecated as it's documentation has been incorrect and the setting itself near useless.

Batch mode remains a valid commit log mode, however.

- There is a new commit log mode, group, which is similar to batch mode but blocks for up to a configurable number of milliseconds between disk flushes.

- Due to the parallelization of the initial build of materialized views,

the per token range view building status is stored in the new table `system.view_builds_in_progress`. The old table `system.views_builds_in_progress` is no longer used and can be removed. See CASSANDRA-12245 for more details.
- nodetool clearsnapshot now required the --all flag to remove all snapshots.

Previous behavior would delete all snapshots by default.

Deprecation

- Background read repair has been deprecated.
dclocal_read_repair_chance and
read_repair_chance table options have been deprecated, and will be removed entirely in 4.0.
See CASSANDRA-13910 for details.

3.11.2

=====

Upgrading

- See MAXIMUM TTL EXPIRATION DATE NOTICE above.
- Cassandra is now relying on the JVM options to properly shutdown on OutOfMemoryError. By default it will

rely on the OnOutOfMemoryError option as the ExitOnOutOfMemoryError and CrashOnOutOfMemoryError options

are not supported by the older 1.7 and 1.8 JVMs. A warning will be logged at startup if none of those JVM

options are used. See CASSANDRA-13006 for more details

3.11.0

=====

Upgrading

- Creating Materialized View with filtering on non-primary-key base column

(added in CASSANDRA-10368) is disabled, because the liveness of view row

is depending on multiple filtered base non-key columns and base non-key

column used in view primary-key. This semantic cannot be supported without

storage format change, see CASSANDRA-13826. For append-only use case, you

may still use this feature with a startup flag: "-Dcassandra.mv.allow_filtering_nonkey_columns_unsafe=true"

- The NativeAccessMBean isAvailable method will only return true if the

```

native library has been successfully linked. Previously it was
returning
    true if JNA could be found but was not taking into account link
failures.
- Primary ranges in the system.size_estimates table are now based on
the keyspace
    replication settings and adjacent ranges are no longer merged
(CASSANDRA-9639).
- In 2.1, the default for otc_coalescing_strategy was 'DISABLED'.
    In 2.2 and 3.0, it was changed to 'TIMEHORIZON', but that value was
shown
    to be a performance regression. The default for 3.11.0 and newer has
been reverted to 'DISABLED'. Users upgrading from Cassandra 2.2 or
3.0 should
    be aware that the default has changed.
- The StorageHook interface has been modified to allow to retrieve
read information from
    SSTableReader (CASSANDRA-13120).

```

Materialized Views (only when upgrading from DSE 5.1.1 or 5.1.2 or any version lower than DSE 5.0.10)

```

- Cassandra will no longer allow dropping columns on tables with
Materialized Views.
- A change was made in the way the Materialized View timestamp is
computed, which
    may cause an old deletion to a base column which is view primary
key (PK) column
    to not be reflected in the view when repairing the base table post-
upgrade. This
    condition is only possible when a column deletion to an MV primary
key (PK) column
    not present in the base table PK (via UPDATE base SET view_pk_col =
null or DELETE
    view_pk_col FROM base) is missed before the upgrade and received by
repair after the upgrade.
    If such column deletions are done on a view PK column which is not
a base PK, it's advisable
        to run repair on the base table of all nodes prior to the upgrade.
Alternatively it's possible
    to fix potential inconsistencies by running repair on the views
after upgrade or drop and
        re-create the views. See CASSANDRA-11500 for more details.
- Removal of columns not selected in the Materialized View (via
UPDATE base SET unselected_column
    = null or DELETE unselected_column FROM base) may not be properly
reflected in the view in some
    situations so we advise against doing deletions on base columns not
selected in views

```

until this is fixed on CASSANDRA-13826.

TinkerPop changes for DSE 6.0.2

A list of DataStax Enterprise 6.0.2 production-certified enhancements to TinkerPop 3.2.9.

DataStax Enterprise (DSE) 6.0.2 includes these production-certified enhancements to TinkerPop 3.2.9:

- Implemented TraversalSelectStep which allows to select() runtime-generated keys.
- Coerced BulkSet to g>List in GraphSON 3.0.
- Deprecated CredentialsGraph DSL in favor of CredentialsTraversalDsl which uses the recommended method for Gremlin DSL development.
- Allowed iterate() to be called after profile().
- Fixed regression issue where the HTTPChannelizer doesn't instantiate the specified AuthenticationHandler.
- Defaulted GLV tests for gremlin-python to run for GraphSON 3.0.
- Fixed a bug with Tree serialization in GraphSON 3.0.
- In gremlin-python, the GraphSON 3.0 g:Set type is now deserialized to List.

DSE 6.0.1 release notes

5 June 2018

In this section:

- [6.0.1 Components \(page 27\)](#)
- [6.0.1 Highlights \(page 28\)](#)
- [6.0.1 DSE core \(page 28\)](#)
- [6.0.1 DSE Analytics \(page 29\)](#)
- [6.0.1 DSEFS \(page 30\)](#)
- [6.0.1 DSE Graph \(page 30\)](#)
- [6.0.1 DSE Search \(page 30\)](#)
- [DataStax Bulk Loader 1.0.2 \(page 31\)](#)
- [Cassandra enhancements for DSE 6.0.1 \(page 31\)](#)
- [General upgrade advice DSE 6.0.1 \(page 32\)](#)

6.0.1 Components

All components from DSE 6.0.0 are listed. Components that are updated for DSE 6.0.1 are indicated.

- Apache Solr™ 6.0.1.1.2295 (updated)
- Apache Spark™ 2.2.1.2 (updated)
- Apache Tomcat® 8.0.47
- DSE Java Driver 1.6.5
- Netty 4.1.13.11.dse
- Spark Jobserver 0.8.0.45 DSE custom version (updated)

- TinkerPop 3.3.2 with additional production-certified [changes \(page 37\)](#) (updated)

DSE 6.0.1 is compatible with Apache Cassandra™ 3.11 and adds additional production-certified [changes \(page 31\)](#).

DSE 6.0.1 Highlights

High-value benefits of upgrading to DSE 6.0.1 include these highlights:

DataStax Enterprise core

- Fix binding JMX to any address. (DB-2081)
- DataStax Bulk Loader 1.0.2 is bundled with DSE 6.0.1. (DSP-16206)

DSE Analytics and DSEFS

- Upgrade to Spark 2.2.1 for bug fixes.
- Fixed issue where multiple Spark Masters can be started on the same machine. (DSP-15636)
- Improved Spark Master discovery and reliability. (DSP-15801, DSP-14405)
- Improved AlwaysOn SQL (AOSS) startup reliability. (DSP-15871, DSP-15468, DSP-15695, DSP-15839)
- Resolved the missing /tmp directory in DSEFS after fresh cluster installation. (DSP-16058)
- Fixed handling of Parquet files with partitions. (DSP-16067)
- Fixed the HashedWheelTimer leak in Spark Connector that affected BYOS. (DSP-15569)

DSE Search

- Fix for the known issue that prevented using TTL (time-to-live) with DSE Search live indexing (RT indexing). (DSP-16038, DSP-14216)
- Addresses security vulnerabilities in libraries packaged with DSE. (DSP-15978)
- Fix for using facetting with non-zero offsets. (DSP-15946)
- Fix for ORDER BY clauses in native CQL syntax. (DSP-16064)

6.0.1 DSE core

Changes and enhancements:

- Improved NodeSync usability with secure environments. (DB-2034)
- [sstableloader](#) supports custom config file locations. (DSP-16092)
- [LDAP tuning \(page 159\)](#) parameters allow all LDAP connection pool options to be set. (DSP-15948)

Resolved issues:

- Use the indexed item type as backing table key validator of 2i on collections. (DB-1121)
- Add getConcurrentCompactors to JMX in order to avoid loading DatabaseDescriptor to check its value in nodetool. (DB-1730)
- Send a final error message when a continuous paging session is cancelled. (DB-1798)
- Ignore empty counter cells on digest calculation. (DB-1881)
- Apply view batchlog mutation parallel with local view mutations. (DB-1900)
- Use same IO queue depth as Linux scheduler and advise against overriding it. (DB-1909)

- Fix startup error message rejecting COMPACT STORAGE after upgrade. (DB-1916)
- Improve user warnings on startup when libaio package is not installed. (DB-1917)
- Set MX4J_ADDRESS to 127.0.0.1 if not explicitly set. (DB-1950)
- Prevent OOM due to OutboundTcpConnection backlog by dropping request messages after the queue becomes too large. (DB-2001)
- Fix exception in trace log messages of non-frozen user types. (DB-2005)
- Limit max cached direct buffer on NIO to 1 MB. (DB-2028)
- Reusing table ID with CREATE TABLE causes failure on restart. (DB-2032)
- BulkLoader class exits without printing the stack trace. (DB-2033)
- Fix binding JMX to any address. (DB-2081)
- sstableloader does not decrypt passwords using config encryption in DSE. (DSP-13492)
- dse client-tool help doesn't work if `~/.dserc` file exists. (DSP-15869)

6.0.1 DSE Analytics

Known issues:

- The Spark Jobserver demo has an incorrect version for the Spark Jobserver API. (DSP-15832)

Workaround: In the demo's `gradle.properties` file, change the version from 0.6.2 to 0.6.2.238.

Changes and enhancements:

- Decreased the number of exceptions logged during master move from node to node. (DSP-14405)
- When querying remote cluster from Spark job, connector does not route requests to data replicas. (DSP-15202)
- Long CassandraRDD.where clauses throw StackOverflow exceptions. (DSP-15438)
- AlwaysOn SQL dependency on JPS is removed. The `jps_directory` entry in `dse.yaml` is removed. (DSP-15468)
- Improved [AlwaysOn SQL \(page 246\)](#) configuration. (DSP-15734)
- Improved security for Spark JobServer. All uploaded JARs, temporary files, and logs are created under the current user's home directory: `~/.spark-jobserver`. (DSP-15832)
- Improved process scanning for AlwaysOn SQL driver. (DSP-15839)
- In Portfolio demo, pricer is no longer required to be run with sudo. (DSP-15970)
- Scala 2.10 in BYOS is no longer supported. (DSP-15999)
- Improved validation for authentication configuration for [AlwaysOn SQL \(page 250\)](#). (DSP-16018)
- Optimize memoizing converters for UDTs. (DSP-16121)
- During misconfigured cluster bootstrap, the AlwaysOn SqlServer does not start due to missing `/tmp/hive` directory in DSEFS. (DSP-16058)

Resolved issues:

- A shard request timeout caused an assertion error from Lucene `getNumericDocValues` in the log. (DSP-14216)
- Multiple Spark Masters can be started on the same machine. (DSP-15636)

- Do not start AlwaysOn SQL until Spark Master is ready. (DSP-15695)
- DSE client tool returns wrong Spark Master address. (DSP-15801)
- In some situations, AlwaysOn SQL cannot start unless DSE node is restarted. (DSP-15871)
- Portfolio demo does not work on package installs. (DSP-15970)
- Java driver in Spark Connector uses daemon threads to prevent shutdown hooks from being blocked by driver thread pools. (DSP-16051)
- dse client-tool spark sql-schema --all exports definitions for solr_admin keyspace. (DSP-16073).
- HashedWheelTimer leak in Spark Connector, affecting BYOS. (DSP-15569)

6.0.1 DSEFS

Resolved issues:

- Can't quote file patterns in DSEFS shell. (DSP-15550)

6.0.1 DSE Graph

Changes and enhancements:

- DseGraphFrame performance improvement reduces number of joins for count() and other id only queries. (DSP-15554)
- Performance improvements for traversal execution with Fluent API and script-based executions. (DSP-15686)

Resolved issues:

- edge_threads and vertex_threads can end up being 0. (DGL-305)
- When using graph frames, cannot upload edges when ids for vertices are complex non-text ids. (DSP-15614)
- CassandraHiveMetastore is prevented from adding multiple partitions for file-based data sources. Fixes MSCK REPAIR TABLE command. (DSP-16067)

6.0.1 DSE Search

Changes and enhancements:

- Output Solr foreign filter cache warning only on classes other than DSE classes. (DSP-15625)
- Solr security upgrades bundle. (DSP-15978)

```
# Apache Directory API All: CVE-2015-3250
# Apache Hadoop Common: CVE-2016-5393, CVE-2017-15713, CVE-2016-3086
# Apache Tika parsers: CVE-2018-1339
# Bouncy Castle Provider: CVE-2018-5382
# Data Mapper for Jackson: CVE-2018-5968, CVE-2017-17485, CVE-2017-15095,
CVE-2018-7489, CVE-2018-5968, CVE-2017-7525
# Guava: Google Core Libraries for Java: CVE-2018-10237
# Simple XML: CVE-2017-1000190
# Xerces2-j: CVE-2013-4002
```

```
# uimaj-core: CVE-2017-15691
```

Resolved issues:

- Offline sstable tools fail if DSE Search index is present on a table. (DSP-15628)
- HTTP read on solr_stress doesn't inject random data into placeholders. (DSP-15727)
- Servlet container shutdown (Tomcat) prematurely stops logback context. (DSP-15807)
- ERROR 500 on distributed http json.facet with non-zero offset. (DSP-15946)
- Search index TTL Expiration thread loops without effect with live indexing (RT indexing). (DSP-16038)
- Search incorrectly assumes only single-row ORDER BY clauses on first clustering key. (DSP-16064)

DataStax Bulk Loader 1.0.2

- DataStax Bulk Loader 1.0.2 is bundled with DSE 6.0.1. (DSP-16206)

DataStax recommends using the latest DataStax Bulk Loader 1.1.0. For details, see [DataStax Bulk Loader](#).

Cassandra enhancements for DSE 6.0.1

DataStax Enterprise 6.0.1 is compatible with Apache Cassandra™ 3.11 and adds production-certified enhancements.

DataStax Enterprise 6.0.1 is compatible with Apache Cassandra™ 3.11 and adds these production-certified changes:

- cassandra-stress throws NPE if insert section isn't specified in user profile (CASSANDRA-14426)
- nodetool listsnapshots is missing local system keyspace snapshots (CASSANDRA-14381)
- Remove string formatting lines from BufferPool hot path (CASSANDRA-14416)
- Detect OpenJDK jvm type and architecture (CASSANDRA-12793)
- Don't use guava collections in the non-system keyspace jmx attributes (CASSANDRA-12271)
- Allow existing nodes to use all peers in shadow round (CASSANDRA-13851)
- Fix cqlsh to read connection.ssl cqlshrc option again (CASSANDRA-14299)
- Downgrade log level to trace for CommitLogSegmentManager (CASSANDRA-14370)
- CQL fromJson(null) throws NullPointerException (CASSANDRA-13891)
- Serialize empty buffer as empty string for json output format (CASSANDRA-14245)
- Cassandra not starting when using enhanced startup scripts in windows (CASSANDRA-14418)
- Fix progress stats and units in compactionstats (CASSANDRA-12244)
- Better handle missing partition columns in system_schema.columns (CASSANDRA-14379)
- Deprecate background repair and probabilistic read_repair_chance table options (CASSANDRA-13910)

- Delay hints store excise by write timeout to avoid race with decommission (CASSANDRA-13740)
- Add missed CQL keywords to documentation (CASSANDRA-14359)
- Avoid deadlock when running nodetool refresh before node is fully up (CASSANDRA-14310)
- Handle all exceptions when opening sstables (CASSANDRA-14202)
- Handle incompletely written hint descriptors during startup (CASSANDRA-14080)
- Handle repeat open bound from SRP in read repair (CASSANDRA-14330)
- CqlRecordReader no longer quotes the keyspace when connecting, as the java driver will (CASSANDRA-10751)
- Fix compaction failure caused by reading un-flushed data (CASSANDRA-12743)
- Fix JSON queries with IN restrictions and ORDER BY clause (CASSANDRA-14286)
- CQL fromJson(null) throws NullPointerException (CASSANDRA-13891)
- Check checksum before decompressing data (CASSANDRA-14284)

General upgrade advice DSE 6.0.1

General upgrade advice for DataStax Enterprise 6.0.1.

General upgrade advice for DataStax Enterprise (DSE) 6.0.1:

PLEASE READ: MAXIMUM TTL EXPIRATION DATE NOTICE (CASSANDRA-14092)

(General upgrading instructions are available in the next section)

The maximum expiration timestamp that can be represented by the storage engine is 2038-01-19T03:14:06+00:00, which means that inserts with TTL that expire after this date are not currently supported. By default, INSERTS with TTL exceeding the maximum supported date are rejected, but it's possible to choose a different expiration overflow policy. See CASSANDRA-14092.txt for more details.

Prior to 5.0.12 (5.0.X) and 5.1.7 (5.1.x) there was no protection against INSERTS with TTL expiring after the maximum supported date, causing the expiration time field to overflow and the records to expire immediately. Clusters in the 4.X and lower series are not subject to this when assertions are enabled. Backed up SSTables can be potentially recovered and recovery instructions can be found on the CASSANDRA-14092.txt file.

If you use or plan to use very large TTLS (10 to 20 years), read CASSANDRA-14092.txt for more information.

GENERAL UPGRADING ADVICE FOR ANY VERSION

Snapshotting is fast (especially if you have JNA installed) and takes effectively zero disk space until you start compacting the live data files again. Thus, best practice is to **ALWAYS** snapshot before any upgrade, just in case you need to roll back to the previous version. (Cassandra version X + 1 will always be able to read data files created by version X, but the inverse is not necessarily the case.)

When upgrading major versions of Cassandra, you will be unable to restore snapshots created with the previous major version using the 'sstableloader' tool. You can upgrade the file format of your snapshots using the provided 'sstableupgrade' tool.

4.0

====

New features

- SSTableDump now supports the -l option to output each partition as its own json object
 - See CASSANDRA-13848 for more detail
- The currentTimestamp, currentDate, currentTime and currentTimeUUID functions have been added.
 - See CASSANDRA-13132
 - Support for arithmetic operations between `timestamp`/`date` and `duration` has been added.
 - See CASSANDRA-11936
 - Support for arithmetic operations on number has been added. See CASSANDRA-11935
 - Preview expected streaming required for a repair (nodetool repair --preview), and validate the consistency of repaired data between nodes (nodetool repair --validate). See CASSANDRA-13257
 - Support for selecting Map values and Set elements has been added for SELECT queries. See CASSANDRA-7396
 - The initial build of materialized views can be parallelized. The number of concurrent builder threads is specified by the property `cassandra.yaml:concurrent_materialized_view_builders`.
 - This property can be modified at runtime through both JMX and the new `setconcurrentviewbuilders` and `getconcurrentviewbuilders` nodetool commands. See CASSANDRA-12245 for more details.

Upgrading

- Cassandra 4.0 removed support for COMPACT STORAGE tables. All Compact Tables have to be migrated using `ALTER ... DROP COMPACT STORAGE` statement in 3.0/3.11.
 - Cassandra starting 4.0 will not start if flags indicate that the table is non-CQL.

Syntax for creating compact tables is also deprecated.

- Support for legacy auth tables in the system_auth keyspace (users, permissions, credentials) and the migration code has been removed.

Migration

- of these legacy auth tables must have been completed before the upgrade to
- 4.0 and the legacy tables must have been removed. See the 'Upgrading' section
- for version 2.2 for migration instructions.
- Cassandra 4.0 removed support for the deprecated Thrift interface.

Amongst

- Tother things, this imply the removal of all yaml option related to thrift
 - ('start_rpc', rpc_port, ...).
- Cassandra 4.0 removed support for any pre-3.0 format. This means you
 - cannot upgrade from a 2.x version to 4.0 directly, you have to upgrade to
 - a 3.0.x/3.x version first (and run upgradestable). In particular, this mean Cassandra 4.0 cannot load or read pre-3.0 sstables in any way: you
 - will need to upgrade those sstable in 3.0.x/3.x first.
- Upgrades from 3.0.x or 3.x are supported since 3.0.13 or 3.11.0, previous versions will causes issues during rolling upgrades (CASSANDRA-13274).
 - Cassandra will no longer allow invalid keyspace replication options, such as invalid datacenter names for NetworkTopologyStrategy. Operators

MUST

- add new nodes to a datacenter before they can set set ALTER or CREATE keyspace replication policies using that datacenter. Existing keyspaces
 - will continue to operate, but CREATE and ALTER will validate that all datacenters specified exist in the cluster.
- Cassandra 4.0 fixes a problem with incremental repair which caused repaired data to be inconsistent between nodes. The fix changes the behavior of both
 - full and incremental repairs. For full repairs, data is no longer marked
 - repaired. For incremental repairs, antcompaction is run at the beginning
 - of the repair, instead of at the end. If incremental repair was being used
- prior to upgrading, a full repair should be run after upgrading to resolve
 - any inconsistencies.
- Config option index_interval has been removed (it was deprecated since 2.0)
- Deprecated repair JMX APIs are removed.

- The version of snappy-java has been upgraded to 1.1.2.6
- the minimum value for internode message timeouts is 10ms.

Previously, any

positive value was allowed. See `cassandra.yaml` entries like `read_request_timeout_in_ms` for more details.

- Config option `commitlog_sync_batch_window_in_ms` has been deprecated as it's

documentation has been incorrect and the setting itself near useless.

Batch mode remains a valid commit log mode, however.

- There is a new commit log mode, `group`, which is similar to batch mode

but blocks for up to a configurable number of milliseconds between disk flushes.

- Due to the parallelization of the initial build of materialized views,

the per token range view building status is stored in the new table `'system.view_builds_in_progress'`. The old table

`'system.views_builds_in_progress'`

is no longer used and can be removed. See CASSANDRA-12245 for more details.

- nodetool `clearsnapshot` now required the `--all` flag to remove all snapshots.

Previous behavior would delete all snapshots by default.

Deprecation

-
- Background read repair has been deprecated.
- `dclocal_read_repair_chance` and `read_repair_chance` table options have been deprecated, and will be removed entirely in 4.0.
- See CASSANDRA-13910 for details.

3.11.2

Upgrading

-
- See MAXIMUM TTL EXPIRATION DATE NOTICE above.
 - Cassandra is now relying on the JVM options to properly shutdown on `OutOfMemoryError`. By default it will rely on the `OnOutOfMemoryError` option as the `ExitOnOutOfMemoryError` and `CrashOnOutOfMemoryError` options are not supported by the older 1.7 and 1.8 JVMs. A warning will be logged at startup if none of those JVM options are used. See CASSANDRA-13006 for more details

3.11.0

Upgrading

- Creating Materialized View with filtering on non-primary-key base column
(added in CASSANDRA-10368) is disabled, because the liveness of view row
is depending on multiple filtered base non-key columns and base non-key
column used in view primary-key. This semantic cannot be supported without
storage format change, see CASSANDRA-13826. For append-only use case, you
may still use this feature with a startup flag: "-Dcassandra.mv.allow_filtering_nonkey_columns_unsafe=true"
- The NativeAccessMBean isAvailable method will only return true if the native library has been successfully linked. Previously it was returning true if JNA could be found but was not taking into account link failures.
- Primary ranges in the system.size_estimates table are now based on the keyspace
replication settings and adjacent ranges are no longer merged (CASSANDRA-9639).
- In 2.1, the default for otc_coalescing_strategy was 'DISABLED'.
In 2.2 and 3.0, it was changed to 'TIMEHORIZON', but that value was shown
to be a performance regression. The default for 3.11.0 and newer has been reverted to 'DISABLED'. Users upgrading from Cassandra 2.2 or 3.0 should
be aware that the default has changed.
- The StorageHook interface has been modified to allow to retrieve read information from SSTableReader (CASSANDRA-13120).

Materialized Views (only when upgrading from DSE 5.1.1 or 5.1.2 or any version lower than DSE 5.0.10)

- Cassandra will no longer allow dropping columns on tables with Materialized Views.
- A change was made in the way the Materialized View timestamp is computed, which
may cause an old deletion to a base column which is view primary key (PK) column
to not be reflected in the view when repairing the base table post-upgrade. This
condition is only possible when a column deletion to an MV primary key (PK) column
not present in the base table PK (via UPDATE base SET view_pk_col = null or DELETE
view_pk_col FROM base) is missed before the upgrade and received by repair after the upgrade.
If such column deletions are done on a view PK column which is not a base PK, it's advisable

```

    to run repair on the base table of all nodes prior to the upgrade.
Alternatively it's possible
    to fix potential inconsistencies by running repair on the views
after upgrade or drop and
    re-create the views. See CASSANDRA-11500 for more details.
    - Removal of columns not selected in the Materialized View (via
UPDATE base SET unselected_column
    = null or DELETE unselected_column FROM base) may not be properly
reflected in the view in some
    situations so we advise against doing deletions on base columns not
selected in views
    until this is fixed on CASSANDRA-13826.

```

TinkerPop changes for 6.0.1

A list of DataStax Enterprise 6.0.1 production-certified enhancements to TinkerPop 3.2.8.

DataStax Enterprise (DSE) 6.0.1 includes these production-certified enhancements to TinkerPop 3.2.9:

- Performance enhancement to Bytecode deserialization. (TINKERPOP-1936)
- Path history isn't preserved for keys in mutations. (TINKERPOP-1947)
- Traversal construction performance enhancements (TINKERPOP-1950)
- Bump to Groovy 2.4.15 - resolves a Groovy bug preventing Lambda creation in GLVs in some cases. (TINKERPOP-1953)

DSE 6.0.0 release notes

17 April 2018

In this section:

- [6.0.0 Components \(page 38\)](#)
- [6.0 New features \(page 38\)](#)
- [6.0.0 DSE core \(page 38\)](#)
- [6.0.0 DSE Advanced Replication \(page 41\)](#)
- [6.0.0 DSE Analytics \(page 41\)](#)
- [6.0.0 DSEFS \(page 43\)](#)
- [6.0.0 DSE Graph \(page 44\)](#)
- [6.0.0 DSE Search \(page 45\)](#)
- [6.0.0 DataStax Studio \(page 47\)](#)
- [DataStax Bulk Loader 1.0.1 \(page 47\)](#)
- [Cassandra enhancements for DSE 6.0 \(page 47\)](#)
- [General upgrade advice for DSE 6.0.0 \(page 53\)](#)

Warning: DSE 6.0.0 Do not use TTL (time-to-live) with DSE Search live indexing (RT indexing). To use these features together, upgrade to DSE 6.0.1. ([DSP-16038 \(page 46\)](#))

6.0.0 Components

- Apache Solr™ 6.0.1.1.2234
- Apache Spark™ 2.2.0.14
- Apache Tomcat® 8.0.47
- DSE Java Driver 1.6.5
- Netty 4.1.13.11.dse
- Spark Jobserver 0.8.0.44 (DSE custom version)
- TinkerPop 3.3.2 with additional production-certified [changes \(page 56\)](#)

DSE 6.0 is compatible with Apache Cassandra™ 3.11 and adds additional production-certified [changes \(page 47\)](#).

6.0 New features

See [DataStax Enterprise 6.0 new features \(page 10\)](#).

6.0.0 DSE core

Experimental features. These features are experimental and are not supported for production:

- [SASI indexes](#).
- [OpsCenter Labs features](#) in OpsCenter.

Known issues:

- DSE 6.0 will not start with OpsCenter 6.1 installed. [OpsCenter 6.5](#) is required for managing DSE 6.0 clusters. See [DataStax OpsCenter compatibility with DSE](#). (DSP-15996)

Changes and enhancements:

Important: Support for Thrift-compatible tables (COMPACT STORAGE) is dropped.

Before upgrading to DSE 6.0, you must migrate all tables that have COMPACT STORAGE to CQL table format.

Upgrades from [DSE 5.0.x](#) or [DSE 5.1.x](#) with Thrift-compatible tables require DSE 5.1.6 or later or DSE 5.0.12 or later.

- For TWCS, flush to separate SSTables based on write time. (DB-42)
- Allow to aggregate by time intervals. Allow aggregates in GROUP BY results. (DB-75)
- Allow user-defined functions (UDFs) within GROUP BY clause and allow non-deterministic UDFs within GROUP BY clause. New CQL keywords ([DETERMINISTIC](#) and [MONOTONIC](#)). The cassandra.yaml file [enable_user_defined_functions_threads \(page 74\)](#) option has no changes to default behavior of true; set to false to use UDFs in GROUP BY clauses. (DB-672)
- Improved architecture with Thread Per Core (TPC) asynchronous read and write paths. (DB-707)

New DSE [start-up parameters \(page 158\)](#):

```
# -Ddse.io.aio.enable (page 158)
# -Ddse.io.aio.force (page 158)
```

Observable metrics with [nodetool tpstats](#).

- New options in `cassandra.yaml`. (DB-111, DB-707, DB-945, DB-1381, DB-1656)
 - # [aggregated_request_timeout_in_ms \(page 88\)](#)
 - # [batchlog_endpoint_strategy \(page 95\)](#) to improve batchlog endpoint selection. (DB-1367)
 - # [client_timeout_sec \(page 104\)](#), [cancel_timeout_sec \(page 104\)](#), [file_cache_size_in_mb \(page 80\)](#), [tpc_cores \(page 82\)](#), [tpc_io_cores \(page 82\)](#), [io_global_queue_depth \(page 82\)](#)
 - # The `rpc_*` properties are deprecated and renamed to [native_transport_*](#) ([page 90](#)). (DB-1130)
 - # [streaming_connections_per_host \(page 81\)](#)
 - # `key_cache_*` settings are no longer used in new SSTable format, but retained to support existing SSTable format
- Removed options in `cassandra.yaml`:
 - # `buffer_pool_use_heap_if_exhausted`, `concurrent_counter_writes`, `concurrent_materialized_view_writes`, `concurrent_reads`, `concurrent_writes`, `credentials_update_interval_in_ms`, `credentials_validity_in_ms`, `max_client_wait_time_ms`, `max_threads`, `native_transport_max_threads`, `otc_backlog_expiration_interval_ms`, `request_scheduler`.
 - # Deprecated option: `memtable_cleanup_threshold`.
- Default value changes in `cassandra.yaml`:
 - # [batch_size_warn_threshold_in_kb \(page 83\)](#): 64
 - # [column_index_size_in_kb \(page 80\)](#): 16
 - # [memtable_flush_writers \(page 80\)](#): 4
 - # [roles_validity_in_ms \(page 97\)](#): 120000 (2 minutes)
 - # [permissions_validity_in_ms \(page 98\)](#): 120000 (2 minutes)
- Legacy auth tables no longer supported. (DB-897)
- Authentication and authorization improvements. RLAC (setting [row-level permissions](#)) speed is improved. (DB-909)
- Incremental [repair is opt-in](#). (DB-1126)
- Background [read-repair](#). (DB-1771)
- Authentication filters used in DSE Search moved to DSE core. (DSP-12531)
- Improved [authentication](#) and security. (DSP-14173)

Supporting changes:

- # Allow to [grant/revoke](#) multiple permissions in one statement. (DB-792)
- # Database administrators can manage `role` permissions without having access to the data. (DB-757)
- # Filter rows from system keyspaces and system_schema tables based on user permissions. New [system_keyspaces_filtering \(page 97\)](#) option in `cassandra.yaml` returns information based on user access to keyspaces. (DB-404)

- # Removed cassandra.yaml options credentials_validity_in_ms and credentials_update_interval_in_ms. For upgrade impact, see [Upgrading from DataStax Enterprise 5.1 to 6.0](#). (DB-909)
- # Warn when the cassandra superuser logs in. (DB-104)
- # Auditing by role: new dse.yaml [audit options \(page 140\)](#) included_roles and excluded_roles. (DSP-15733)
- libaio package dependency for DataStax Enterprise 6.0 installations on RHEL-based systems using Yum and on Debian-based systems using APT install. For optimal performance in [tarball installations](#), DataStax recommends installing the libaio package. (DSP-14228)
- The default number of threads used by performance objects increased from 1 to 4. [Upgrade restrictions](#) apply. (DSP-14515)
- All tables are created without COMPACT STORAGE. (DSP-14735)
- Support for Thrift-compatible tables (COMPACT STORAGE) is dropped. Before upgrading, migrate all tables that have COMPACT STORAGE to CQL table format. DSE 6.0 will not start if COMPACT STORAGE tables are present. See [Upgrading from DSE 5.1.x](#) or [Upgrading from DSE 5.0.x](#). (DSP-14839)
- The minimum supported version of [Oracle Java SE Runtime Environment 8 \(JDK\)](#) is 1.8u151. (DSP-14818)
- [sstabledump](#) supports the -l option to output each partition as its own JSON object. (DSP-15079)
- Audit improvements, new and changed [filtering event categories](#). (DSP-15724)
- Upgrades to [OpsCenter 6.5](#) are required before starting DSE 6.0. OpsCenter 6.5 is required for managing DSE 6.0 clusters. See [DataStax OpsCenter compatibility](#) with DSE. (DSP-15996)

Resolved issues:

- Warn when the cassandra superuser logs in. (DB-104)
- Prevent multiple serializations of mutation. (DB-370)
- Internal implementation of paging by bytes. (DB-414)
- Connection refused should be logged less frequently. (DB-455)
- Refactor messaging service code. (DB-497)
- Change protocol to allow sending keyspace independent of query string. (DB-600)
- Add result set metadata to prepared statement MD5 hash calculation. (DB-608)
- Fix LWT asserts for immutable TableMetadata.a (DB-728)
- MigrationManager should use toDebugString() when logging TableMetadata. (DB-739)
- Create administrator roles who can carry out everyday administrative tasks without having unnecessary access to data. (DB-757)
- When repairing Paxos commits, only block on nodes are being repaired. (DB-761)
- Allow to grant/revoke multiple permissions in one statement (DB-792)
- SystemKeyspace.snapshotOnVersionChange() never called in production code. (DB-797)
- Error in counting iterated SSTables when choosing whether to defrag in timestamp ordered path. (DB-1018)

- Check for mismatched versions when answering schema pulls. (DB-1026)
- Expose ports (storage, native protocol, JMX) in system local and peers tables. (DB-1040)
- Rename ITrigger interface method from augment to augmentNonBlocking. (DB-1046)
- Load mapped buffer into physical memory after mlocking it for MemoryOnlyStrategy. (DB-1052)
- New STARTUP message parameters identify clients. (DB-1054)
- Emit client warning when a GRANT/REVOKE/RESTRICT/UNRESTRICT command has no effect. (DB-1083)
- Update bundled Python driver to 2.2.0.post0-d075d57. (DB-1152)
- Forbid advancing KeyScanningIterator before exhausting or closing the current iterator. (DB-1199)
- Ensure that empty clusterings with kind==CLUSTERING are Clustering.EMPTY. (DB-1248)
- New `nodetool abortrebuild` command stops a currently running rebuild operation. (DB-1234)
- Batchlog replays do not leverage remote coordinators. (DB-1337)
- Avoid copying EMPTY_STATIC_ROW to heap again with offheap memtable. (DB-1375)
- Allow DiskBoundaryManager to cache different directories. (DB-1454)
- Abort repair when there is only one node. (DB-1511)
- OutOfMemory during view update. (DB-1493)
- Drop response on view lock acquisition timeout and add ViewLockAcquisitionTimeouts metric. (DB-1522)
- Handle race condition on dropping keyspace and opening keyspace. (DB-1570)
- dsetool ring prints ERROR when data_file_directories is removed from cassandra.yaml. (DSP-13547)
- Driver: Jackson-databind is vulnerable to remote code execution (RCE) attacks. (DSP-13498)
- LDAP library issue. (DSP-15927)

6.0.0 DSE Advanced Replication

Changes and enhancements:

- Support for DSE Advanced Replication V1 is removed. For V1 installations, you must first [upgrade to DSE 5.1.x](#) and migrate your DSE Advanced Replication to V2, and then upgrade to [DSE 6.0](#). (DSP-13376)
- Enhanced CLI security prevents injection attacks and sanitizes and validates the command line inputs. (DSP-13682)

Resolved issues:

- Improve logging on unsupported operation failure and remove the failed mutation from replog. (DSP-15043)
- Channel creation fails with NPE when using mixed case destination name. (DSP-15538)

6.0.0 DSE Analytics

Experimental features. These features are experimental and are not supported for production:

- Importing graphs using [DseGraphFrame \(page 619\)](#).

Known issues:

- DSE Analytics: Additional configuration is required when enabling context-per-jvm in the [Spark Jobserver \(page 258\)](#). (DSP-15163)

Changes and enhancements:

- Previously deprecated environment variables, including SPARK_CLASSPATH, are removed in Spark 2.2.0. (DSP-8379)
- [AlwaysOn SQL service \(page 245\)](#), a HA (highly available) Spark SQL Thrift server. (DSP-10996)

```
# JPS is an option required for nodes with AlwaysON SQL enabled.  
# The spark_config_settings and hive_config_settings are removed from dse.yaml.  
The configuration is provided in the spark-alwayson-sql.conf file in DSEHOME/  
resources/spark/conf with the same default contents as DSEHOME/resources/spark/  
conf/spark-defaults.conf. (DSP-15837)
```
- Cassandra File System (CFS) is removed. Use [DSEFS \(page 273\)](#) instead. Before [upgrading](#) to DSE 6.0, remove CFS keyspaces. See the [From CFS to DSEFS](#) dev blog post. (DSP-12470)
- Optimization for SearchAnalytics with SELECT COUNT(*) and no predicates. (DSP-12669)
- Authenticate [JDBC users \(page 250\)](#) to Spark SQL Thrift Server. Queries that are executed during JDBC session are run as the user who authenticated through JDBC. (DSP-13395)
- Solr optimization is [automatic \(page 173\)](#); spark.sql.dse.solr.enabled is deprecated, use spark.sql.dse.search.enabled instead. (DSP-13398)
- Optimization for SearchAnalytics with SELECT COUNT(*) and no predicates. (DSP-13398)
- dse spark-beeline command is removed, use dse beeline instead. (DSP-13468)
- cfs-stress tool is replaced by [fs-stress tool \(page 864\)](#). (DSP-13549)
- [Encryption](#) for data stored on the server and encryption of Spark spill files is supported. (DSP-13841)
- Improved security with Spark. (DSP-13991)
- Spark local applications no longer use /var/lib/spark/rdd, instead configure and use .sparkdirectory for processes started by the user. (DSP-14380)
- Input metrics are not thread-safe and are not used properly in CassandraJoinRDD and CassandraLeftJoinRDD. (DSP-14569)
- AlwaysOn SQL workpool option adds high availability (HA) for the JDBC or ODBC connections for analytics node. (DSP-14719)
- CFS is removed. Before upgrade, move HiveMetaStore from CFS to DSEFS and update URL references. (DSP-14831)
- Include [SPARK-21494](#) to use correct app id when authenticating to external service. (DSP-14140)

- Upgrade to DSE 6.0 must be complete on all nodes in the cluster before Spark Worker and Spark Master will start. (DSP-14735)
- [Spark Cassandra Connector \(page 226\)](#) in DSE 6.0.0, has the following changes:
 - # Changes to default values: spark.output.concurrent.writes: 100, spark.task.maxFailures: 10 (DSP-15164)
 - # spark.cassandra.connection.connections_per_executor_max is removed; use new properties spark.cassandra.connection.local_connections_per_executor, spark.cassandra.connection.remote_connections_per_executor_min, and spark.cassandra.connection.remote_connections_per_executor_max. (DSP-15193)
 - # All Spark-related parameters are now camelCase. Parameters are case-sensitive. The snake_versions are automatically translated to the camelCaseVersions except when the parameters are used as table options. In SparkSQL and with spark.read.options(...), the parameters are case-insensitive because of internal SQL implementation.
 - # The DSE artifact is com.datastax.dse : spark-connector: 6.0.0.
 - # The DseSparkDependencies JAR is still required. (DSP-15694)
- Use NodeSync (continuous repair) and LOCAL_QUORUM for reading from Spark recovery storage. (DSP-15219)

Supporting changes:

- # Spark Master will not start until LOCAL_QUORUM is achieved for dse_analytics keyspace.
- # Spark Master recovery data is first attempted to be updated with LOCAL_QUORUM, and if that fails, then attempt with LOCAL_ONE. Recovery data are always queried with LOCAL_QUORUM (unlike previous versions of DSE where we used LOCAL_ONE)
- # DSE Analytics internal data moved from spark_system to dse_analytics keyspace.

Note: DataStax strongly recommends enabling [NodeSync \(page 163\)](#) for continuous repair on all tables in the dse_analytics keyspace. NodeSync is required on the rm_shared_data keyspace that stores Spark recovery information.

Resolved issues:

- DSE does not work with Spark Crypto based encryption. (DSP-14140)

6.0.0 DSEFS

Changes and enhancements:

- Wildcard characters are supported in DSEFS shell [commands \(page 282\)](#). (DSP-10583)
- DSEFS should support all DSE authentication schemes. (DSP-12956)
- Improved [authorization \(page 293\)](#) security sets the default permission to 755 for directories and 644 for files. New DSEFS clusters create the root directory / with 755 permission to prevent non-super users from modifying root content; for example, by using mkdir or put commands. (DSP-13609)
- Enable [SSL for DSEFS \(page 296\)](#) encryption. (DSP-13771)

- HTTP communication logging level changed from DEBUG to TRACE. (DSP-14400)
- DSEFS shell history has been moved to `~/.dse/dsefs_history`. (DSP-15070)
- New tool to move hive metastore from CFS to DSEFS and update references.
- Add [echo \(page 279\)](#) command to DSEFS shell. (DSP-15446)
- Changes in `dse.yaml` for [advanced DSEFS \(page 136\)](#) settings.
- Alternatives wildcards are Hadoop compatible. (DSP-15249)

6.0.0 DSE Graph

Known issues:

- Dropping a property of vertex label with materialized view (MV) indices breaks graph. To drop a property key for a vertex label that has a materialized view index, additional steps are required to prevent data loss or cluster errors. See [Dropping graph schema \(page 464\)](#). (DSP-15532)
- Secondary indexes used for DSE Graph queries have higher latency in DSE 6.0 than in the previous version. (DB-1928)

Changes and enhancements:

- Improved and simplified data batch loading of pre-formatted data. (DGL-235)

Supporting changes:

- ```
Schema discovery and schema generation are deprecated. (DGL-246)
Standard vertex IDs are deprecated. Use custom vertex IDs instead. (DSP-13485)
Standard IDs are deprecated. (DGL-247)
Transformations are deprecated. (DGL-248)
```
- Schema API changes: all `.remove()` methods are renamed to `.drop()` and `schema.clear()` is renamed to `schema.drop()`. Schema API supports removing vertex/edge labels and property keys. Unify use of `drop` | `remove` | `clear` in the Schema API and use `.drop()` everywhere. (DSP-8385, DSP-14150)
  - Include materialized view (MV) indexes in query optimizer only if the MV was fully built. (DSP-10219)
  - DSE profiling of graph statements from the gremlin shell. (DSP-13484)
  - Improve Graph OLAP performance by smart routing query to DseGraphFrame engine with [DseGraphFrameInterceptorStrategy \(page 610\)](#). (DSP-13489)
  - OSS TinkerPop 3.3 supports Spark 2.2. (DSP-13632)
  - Partitioned vertex tables (PVT) are removed. (DSP-13676)
  - Graph online analytical processing (OLAP) supports `drop()` with DseGraphFrame interceptor. Simple queries can be used in [drop operations \(page 613\)](#). (DSP-13998)
  - DSE Graphs vertices and edges [tables \(page 372\)](#) are accessible from SparkSQL and automated to [dse\\_graph SparkSQL \(page 237\)](#) database. (DSP-12046)
  - More Gremlin APIs are supported in DSEGraphFrames: `dedup`, `sort`, `limit`, `filter`, `as()`/`select()`, `or()`. (DSP-13649)
  - Some `gremlin_server` properties in earlier versions of DSE are no longer required in the DSE 6.0 `dse.yaml`. If these properties exist in the `dse.yaml` file after upgrading to DSE 6.0, logs display [warnings](#). You can ignore these warnings or modify `dse.yaml` so that only the required [gremlin\\_server \(page 148\)](#) properties are present. (DSP-14308)

- Spark Jobserver is the DSE custom version 0.8.0.44. Applications must use the compatible Spark Jobserver API in DataStax [repository](#). (DSP-14152)
- Edge label names and property key names allow only [a-zA-Z0-9], underscore, hyphen, and period. The string formatting for vertices with text custom IDs has changed. (DSP-14710)

Supporting changes (DSP-15167):

```
schema.describe() displays the entire schema, even if it contains illegal names.
In-place upgrades allow existing schemas with invalid edge label names and
property key names.
Schema elements with illegal names cannot be uploaded or added.
```

- Invoking `toString` on a custom vertex ID containing a text property, or on an edge ID that is incident upon a vertex with a custom vertex ID, now returns a value that encloses the text property value in double quotation marks and escapes the value's internal double-quotes. This change protects older formats from irresolvable parsing ambiguity. For example:

```
// old
{~label=v, x=foo}
{~label=w, x=a "b}
// new
{~label=v, x="foo" }
{~label=w, x="a" "b" }
```

- Support for `math()`-step ([math \(page 709\)](#)) to enable scientific calculator functionality within Gremlin. (DSP-14786)
- The `GraphQueryThreads` JMX attribute has been removed. Thread selection occurs with Thread Per Core (TPC) asynchronous request processing architecture. (DSP-15222)

Resolved issues:

- `spark.sql.hive.metastore.barrierPrefixes` is set to `org.apache.spark.sql.cassandra` to properly use `CassandraConnector` in DSE HiveMetastore implementation. (DSP-14120)
- Intermittent `KryoException`: Buffer underflow error when running `order by` query in OLTP mode. (DSP-12694)
- `DseGraphFrame` does not support infix `and()` and `or()`. (DSP-12013)
- Graph could be left in an inconsistent state if a schema migration fails. (DSP-15532)
- `DseGraphFrames properties().count()` step return vertex count instead of multi properties count. (DSP-15049)
- GraphSON parsing error prevents proper type detection under certain conditions. (DSP-14066)

## 6.0.0 DSE Search

Experimental features. These features are experimental and are not supported for production:

- The `dsetool index_checks (page 821)` use an Apache Lucene® experimental feature.

Known issues:

- Search index TTL Expiration thread loops without effect with live indexing (RT indexing). (DSP-16038)

Changes and enhancements:

- Writes are flushed to disk in segments that use a new Lucene codec that does not exist in earlier versions. Unique key values are no longer stored as both docValues and Lucene stored fields. The unique key values are now stored only as docValues in a new codec to store managed fields like Lucene. Downgrades to versions earlier than DSE 6.0 are not supported. (DSP-8465)
- Document inserts and updates using HTTP are removed. Before [upgrading](#), ensure you are using CQL for all inserts and updates. (DSP-9725).
- Delete by id is removed. Delete by query no longer accepts wildcard queries, including queries that match all documents (for example, `<delete><query>*:*</query></delete>`). Instead, use CQL to [DELETE](#) by Primary Key or the [TRUNCATE](#) command. (DSP-13436)
- DSENRTCachingDirectoryFactory is removed. Before [upgrading](#), change your search index config. (DSP-10126)
- The `<dataDir>` parameter in the `solrconfig.xml` file is not supported. Instead, follow the steps in [Set the location of search indexes](#). (DSP-13199)
- Improved performance by early termination of sorting. Ideal for queries that need only a few results returned, from a large number of total matches. (DSP-13253)
- Native CQL syntax for [search queries](#). (DSP-13411)
- Logging configuration improvements.

```
The solrvalidation.log is removed. You can safely remove appender
SolrValidationErrorAppender and the logger SolrValidationErrorLogger from
logback.xml. Indexing errors manifest as:
failures at the coordinator if they represent failures that might succeed at some
later point in time using the hint replay mechanism
as messages in the system.log if the failures are due to non-recoverable
indexing validation errors (for data that is written to the database, but not
indexed properly)
```

- dsetool [core\\_indexing\\_status \(page 807\)](#) --progress option is always true. (DSP-13465)
- The Tika functionality bundled with Apache Solr is removed. Instead, use the stand-alone Apache Tika project. (DSP-13892)
- The DSE custom update request processor (URP) implementation is deprecated. Use the [field input/output \(page 354\)](#) (FIT) transformer API instead. (DSP-14360)
- The stored flag in [search index schemas \(page 315\)](#) is deprecated and is no longer added to auto-generated schemas. If the flag exists in custom schemas, it is ignored. (DSP-14425)
- Tuning improvements for indexing. (DSP-14785)
  - # Indexing is no longer asynchronous. Document updates are written to the Lucene RAM buffer synchronously with the mutation backing table.
  - # See [Configuring and tuning indexing performance](#).

- # The back\_pressure\_threshold\_per\_core in dse.yaml affects only index rebuilding/reindexing.
- # The max\_solr\_concurrency\_per\_core and solr\_indexing\_error\_log\_options options in dse.yaml are removed. DSE 6.0 will not start with these options present.
- StallMetrics MBean is removed. Before upgrading to DSE 6.0, change operators that use the MBean. (DSP-14860)
- Optimize [Paging](#) when limit is smaller than the page size. (DSP-15207)

Resolved issues. Includes all bug fixes up to DSE 5.1.8. Additional 6.0.0 fixes:

- Isolate Solr Resource Loading at startup to the Local DC. (DSP-10911)

Resolved issues:

- Numeric overflows should display the original input that caused the overflow. (DAT-237)
- Null words are not supported by all connectors. (DAT-241)
- Addresses might not be properly translated when cluster has custom native port. (DAT-245)

DataStax Studio 6.0.0

- For use with DSE 6.0.x, DataStax Studio 6.0.0 is installed as a standalone tool. (DSP-13999, DSP-15623)

For details, see [DataStax Studio 6.0.0 release notes \(page 59\)](#).

DataStax Bulk Loader 1.0.1

- DataStax Bulk Loader ([dsbulk](#)) version 1.0.1 is automatically installed with DataStax Enterprise 6.0.0, and can also be installed as a standalone tool. (DSP-13999, DSP-15623)

For details, see [DataStax Bulk Loader 1.0.1 release notes \(page 58\)](#).

## Cassandra enhancements for DSE 6.0

DataStax Enterprise 6.0.0 is compatible with Apache Cassandra™ 3.11 and adds production-certified enhancements.

DataStax Enterprise 6.0.0 is compatible with Apache Cassandra™ 3.11 and adds these production-certified enhancements:

- Add DEFAULT, UNSET, MBEAN and MBEANS to `ReservedKeywords`. (CASSANDRA-14205)
- Add Unittest for schema migration fix (CASSANDRA-14140)
- Print correct snitch info from nodetool describecluster (CASSANDRA-13528)
- Close socket on error during connect on OutboundTcpConnection (CASSANDRA-9630)
- Enable CDC unittest (CASSANDRA-14141)
- Split CommitLogStressTest to avoid timeout (CASSANDRA-14143)
- Improve commit log chain marker updating (CASSANDRA-14108)
- Fix updating base table rows with TTL not removing view entries (CASSANDRA-14071)
- Reduce garbage created by DynamicSnitch (CASSANDRA-14091)
- More frequent commitlog chained markers (CASSANDRA-13987)

- RPM package spec: fix permissions for installed jars and config files (CASSANDRA-14181)
- More PEP8 compliance for cqlsh (CASSANDRA-14021)
- Fix support for SuperColumn tables (CASSANDRA-12373)
- Fix missing original update in TriggerExecutor (CASSANDRA-13894)
- Improve short read protection performance (CASSANDRA-13794)
- Fix counter application order in short read protection (CASSANDRA-12872)
- Fix MV timestamp issues (CASSANDRA-11500)
- Fix AssertionError in short read protection (CASSANDRA-13747)
- Gossip thread slows down when using batch commit log (CASSANDRA-12966)
- Allow native function calls in CQLSSTableWriter (CASSANDRA-12606)
- Copy session properties on cqlsh.py do\_login (CASSANDRA-13847)
- Fix load over calculated issue in IndexSummaryRedistribution (CASSANDRA-13738)
- Obfuscate password in stress-graphs (CASSANDRA-12233)
- ReverseIndexedReader may drop rows during 2.1 to 3.0 upgrade (CASSANDRA-13525)
- Avoid reading static row twice from old format sstables (CASSANDRA-13236)
- Fix possible NPE on upgrade to 3.0/3.X in case of IO errors (CASSANDRA-13389)
- Add duration data type (CASSANDRA-11873)
- Properly report LWT contention (CASSANDRA-12626)
- Stress daemon help is incorrect (CASSANDRA-12563)
- Remove ALTER TYPE support (CASSANDRA-12443)
- Fix assertion for certain legacy range tombstone pattern (CASSANDRA-12203)
- Remove support for non-JavaScript UDFs (CASSANDRA-12883)
- Better handle invalid system roles table (CASSANDRA-12700)
- Upgrade netty version to fix memory leak with client encryption (CASSANDRA-13114)
- Fix trivial log format error (CASSANDRA-14015)
- Allow SSTabledump to do a JSON object per partition (CASSANDRA-13848)
- Remove unused and deprecated methods from AbstractCompactionStrategy (CASSANDRA-14081)
- Fix Distribution.average in cassandra-stress (CASSANDRA-14090)
- Presize collections (CASSANDRA-13760)
- Add GroupCommitLogService (CASSANDRA-13530)
- Parallelize initial materialized view build (CASSANDRA-12245)
- Fix flaky SecondaryIndexManagerTest.assert[Not]MarkedAsBuilt (CASSANDRA-13965)
- Make LWTs send resultset metadata on every request (CASSANDRA-13992)
- Fix flaky indexWithFailedInitializationIsNotQueryableAfterPartialRebuild (CASSANDRA-13963)
- Introduce leaf-only iterator (CASSANDRA-9988)
- Allow only one concurrent call to StatusLogger (CASSANDRA-12182)
- Refactoring to specialised functional interfaces (CASSANDRA-13982)
- Speculative retry should allow more friendly parameters (CASSANDRA-13876)

- Throw exception if we send/receive repair messages to incompatible nodes (CASSANDRA-13944)
- Replace usages of MessageDigest with Guava's Hasher (CASSANDRA-13291)
- Add nodetool command to print hinted handoff window (CASSANDRA-13728)
- Fix some alerts raised by static analysis (CASSANDRA-13799)
- Checksum SSTable metadata (CASSANDRA-13321, CASSANDRA-13593)
- Add result set metadata to prepared statement MD5 hash calculation (CASSANDRA-10786)
- Add incremental repair support for --hosts, --force, and subrange repair (CASSANDRA-13818)
- Refactor GcCompactionTest to avoid boxing (CASSANDRA-13941)
- Expose recent histograms in JmxHistograms (CASSANDRA-13642)
- Add SERIAL and LOCAL\_SERIAL support for cassandra-stress (CASSANDRA-13925)
- LCS needlessly checks for L0 STCS candidates multiple times (CASSANDRA-12961)
- Correctly close netty channels when a stream session ends (CASSANDRA-13905)
- Update Iz4 to 1.4.0 (CASSANDRA-13741)
- Throttle base partitions during MV repair streaming to prevent OOM (CASSANDRA-13299)
- Improve short read protection performance (CASSANDRA-13794)
- Fix AssertionError in short read protection (CASSANDRA-13747)
- Use compaction threshold for STCS in L0 (CASSANDRA-13861)
- Fix problem with min\_compress\_ratio: 1 and disallow ratio < 1 (CASSANDRA-13703)
- Add extra information to SASI timeout exception (CASSANDRA-13677)
- Rework CompactionStrategyManager.getScanners synchronization (CASSANDRA-13786)
- Add additional unit tests for batch behavior, TTLs, Timestamps (CASSANDRA-13846)
- Add keyspace and table name in schema validation exception (CASSANDRA-13845)
- Emit metrics whenever we hit tombstone failures and warn thresholds (CASSANDRA-13771)
- Allow changing log levels via nodetool for related classes (CASSANDRA-12696)
- Add stress profile yaml with LWT (CASSANDRA-7960)
- Reduce memory copies and object creations when acting on ByteBufs (CASSANDRA-13789)
- simplify mx4j configuration (Cassandra-13578)
- Fix trigger example on 4.0 (CASSANDRA-13796)
- force minumum timeout value (CASSANDRA-9375)
- Add bytes repaired/unrepaired to nodetool tablestats (CASSANDRA-13774)
- Don't delete incremental repair sessions if they still have sstables (CASSANDRA-13758)
- Fix pending repair manager index out of bounds check (CASSANDRA-13769)
- Don't use RangeFetchMapCalculator when RF=1 (CASSANDRA-13576)
- Don't optimise trivial ranges in RangeFetchMapCalculator (CASSANDRA-13664)

- Use an ExecutorService for repair commands instead of new Thread(..).start() (CASSANDRA-13594)
- Fix race / ref leak in anticompaaction (CASSANDRA-13688)
- Fix race / ref leak in PendingRepairManager (CASSANDRA-13751)
- Enable ppc64le runtime as unsupported architecture (CASSANDRA-13615)
- Improve sstablemetadata output (CASSANDRA-11483)
- Support for migrating legacy users to roles has been dropped (CASSANDRA-13371)
- Introduce error metrics for repair (CASSANDRA-13387)
- Refactoring to primitive functional interfaces in AuthCache (CASSANDRA-13732)
- Update metrics to 3.1.5 (CASSANDRA-13648)
- batch\_size\_warn\_threshold\_in\_kb can now be set at runtime (CASSANDRA-13699)
- Avoid always rebuilding secondary indexes at startup (CASSANDRA-13725)
- Upgrade JMH from 1.13 to 1.19 (CASSANDRA-13727)
- Upgrade SLF4J from 1.7.7 to 1.7.25 (CASSANDRA-12996)
- Default for start\_native\_transport now true if not set in config (CASSANDRA-13656)
- Don't add localhost to the graph when calculating where to stream from (CASSANDRA-13583)
- Allow skipping equality-restricted clustering columns in ORDER BY clause (CASSANDRA-10271)
- Use common nowInSec for validation compactions (CASSANDRA-13671)
- Improve handling of IR prepare failures (CASSANDRA-13672)
- Send IR coordinator messages synchronously (CASSANDRA-13673)
- Flush system.repair table before IR finalize promise (CASSANDRA-13660)
- Fix column filter creation for wildcard queries (CASSANDRA-13650)
- Add 'nodetool getbatchlogreplaythrottle' and 'nodetool setbatchlogreplaythrottle' (CASSANDRA-13614)
- fix race condition in PendingRepairManager (CASSANDRA-13659)
- Allow noop incremental repair state transitions (CASSANDRA-13658)
- Run repair with down replicas (CASSANDRA-10446)
- Added started & completed repair metrics (CASSANDRA-13598)
- Added started & completed repair metrics (CASSANDRA-13598)
- Improve secondary index (re)build failure and concurrency handling (CASSANDRA-10130)
- Improve calculation of available disk space for compaction (CASSANDRA-13068)
- Change the accessibility of RowCacheSerializer for third party row cache plugins (CASSANDRA-13579)
- Allow sub-range repairs for a preview of repaired data (CASSANDRA-13570)
- NPE in IR cleanup when columnfamily has no sstables (CASSANDRA-13585)
- Fix Randomness of stress values (CASSANDRA-12744)
- Allow selecting Map values and Set elements (CASSANDRA-7396)
- Fast and garbage-free Streaming Histogram (CASSANDRA-13444)
- Update repairTime for keyspaces on completion (CASSANDRA-13539)

- Add configurable upper bound for validation executor threads (CASSANDRA-13521)
- Bring back maxHintTTL property (CASSANDRA-12982)
- Add testing guidelines (CASSANDRA-13497)
- Add more repair metrics (CASSANDRA-13531)
- RangeStreamer should be smarter when picking endpoints for streaming (CASSANDRA-4650)
- Avoid rewrapping an exception thrown for cache load functions (CASSANDRA-13367)
- Log time elapsed for each incremental repair phase (CASSANDRA-13498)
- Add multiple table operation support to cassandra-stress (CASSANDRA-8780)
- Fix incorrect cqlsh results when selecting same columns multiple times (CASSANDRA-13262)
- Fix WriteResponseHandlerTest is sensitive to test execution order (CASSANDRA-13421)
- Improve incremental repair logging (CASSANDRA-13468)
- Start compaction when incremental repair finishes (CASSANDRA-13454)
- Add repair streaming preview (CASSANDRA-13257)
- Cleanup isIncremental/repairedAt usage (CASSANDRA-13430)
- Change protocol to allow sending key space independent of query string (CASSANDRA-10145)
- Make gc\_log and gc\_warn settable at runtime (CASSANDRA-12661)
- Take number of files in L0 in account when estimating remaining compaction tasks (CASSANDRA-13354)
- Skip building views during base table streams on range movements (CASSANDRA-13065)
- Improve error messages for +/- operations on maps and tuples (CASSANDRA-13197)
- Remove deprecated repair JMX APIs (CASSANDRA-11530)
- Fix version check to enable streaming keep-alive (CASSANDRA-12929)
- Make it possible to monitor an ideal consistency level separate from actual consistency level (CASSANDRA-13289)
- Outbound TCP connections ignore internode authenticator (CASSANDRA-13324)
- Upgrade junit from 4.6 to 4.12 (CASSANDRA-13360)
- Cleanup ParentRepairSession after repairs (CASSANDRA-13359)
- Upgrade snappy-java to 1.1.2.6 (CASSANDRA-13336)
- Incremental repair not streaming correct sstables (CASSANDRA-13328)
- Upgrade the JNA version to 4.3.0 (CASSANDRA-13300)
- Add the currentTimestamp, currentDate, currentTime and currentTimeUUID functions (CASSANDRA-13132)
- Remove config option index\_interval (CASSANDRA-10671)
- Reduce lock contention for collection types and serializers (CASSANDRA-13271)
- Make it possible to override MessagingService.Verb ids (CASSANDRA-13283)
- Avoid synchronized on prepareForRepair in ActiveRepairService (CASSANDRA-9292)
- Adds the ability to use uncompressed chunks in compressed files (CASSANDRA-10520)

- Don't flush sstables when streaming for incremental repair (CASSANDRA-13226)
- Remove unused method (CASSANDRA-13227)
- Fix minor bugs related to #9143 (CASSANDRA-13217)
- Output warning if user increases RF (CASSANDRA-13079)
- Remove pre-3.0 streaming compatibility code for 4.0 (CASSANDRA-13081)
- Add support for + and - operations on dates (CASSANDRA-11936)
- Fix consistency of incrementally repaired data (CASSANDRA-9143)
- Increase commitlog version (CASSANDRA-13161)
- Make TableMetadata immutable, optimize Schema (CASSANDRA-9425)
- Refactor ColumnCondition (CASSANDRA-12981)
- Parallelize streaming of different keyspaces (CASSANDRA-4663)
- Improved compactions metrics (CASSANDRA-13015)
- Speed-up start-up sequence by avoiding un-needed flushes (CASSANDRA-13031)
- Use Caffeine (W-TinyLFU) for on-heap caches (CASSANDRA-10855)
- Thrift removal (CASSANDRA-11115)
- Remove pre-3.0 compatibility code for 4.0 (CASSANDRA-12716)
- Add column definition kind to dropped columns in schema (CASSANDRA-12705)
- Add (automate) Nodetool Documentation (CASSANDRA-12672)
- Update bundled cqlsh python driver to 3.7.0 (CASSANDRA-12736)
- Reject invalid replication settings when creating or altering a keyspace (CASSANDRA-12681)
- Clean up the SSTableReader#getScanner API wrt removal of RateLimiter (CASSANDRA-12422)
- Use new token allocation for non bootstrap case as well (CASSANDRA-13080)
- Avoid byte-array copy when key cache is disabled (CASSANDRA-13084)
- Require forceful decommission if number of nodes is less than replication factor (CASSANDRA-12510)
- Allow IN restrictions on column families with collections (CASSANDRA-12654)
- Log message size in trace message in OutboundTcpConnection (CASSANDRA-13028)
- Add timeUnit Days for cassandra-stress (CASSANDRA-13029)
- Add mutation size and batch metrics (CASSANDRA-12649)
- Add method to get size of endpoints to TokenMetadata (CASSANDRA-12999)
- Expose time spent waiting in thread pool queue (CASSANDRA-8398)
- Conditionally update index built status to avoid unnecessary flushes (CASSANDRA-12969)
- cqlsh auto completion: refactor definition of compaction strategy options (CASSANDRA-12946)
- Add support for arithmetic operators (CASSANDRA-11935)
- Add histogram for delay to deliver hints (CASSANDRA-13234)
- Fix cqlsh automatic protocol downgrade regression (CASSANDRA-13307)
- Changing `max\_hint\_window\_in\_ms` at runtime (CASSANDRA-11720)

- Nodetool repair can hang forever if we lose the notification for the repair completing/failing (CASSANDRA-13480)
- Anticompaction can cause noisy log messages (CASSANDRA-13684)
- Switch to client init for sstabledump (CASSANDRA-13683)
- CQLSH: Don't pause when capturing data (CASSANDRA-13743)

## General upgrade advice for DSE 6.0.0

General upgrade advice for DataStax Enterprise 6.0.0

General upgrade advice for DataStax Enterprise (DSE) 6.0.0:

```
GENERAL UPGRADING ADVICE FOR ANY VERSION
=====

```

Snapshotting is fast (especially if you have JNA installed) and takes effectively zero disk space until you start compacting the live data files again. Thus, best practice is to **ALWAYS** snapshot before any upgrade, just in case you need to roll back to the previous version. (Cassandra version X + 1 will always be able to read data files created by version X, but the inverse is not necessarily the case.)

When upgrading major versions of Cassandra, you will be unable to restore snapshots created with the previous major version using the 'sstableloader' tool. You can upgrade the file format of your snapshots using the provided 'sstableupgrade' tool.

```
DSE 6.0
=====

```

Operations

```

- Added batchlog_endpoint_strategy option and 2 new strategies to
cassandra.yaml. The default is
 "random_remote", which works as in previous versions. Two new
strategies help to choose better
 endpoints and improve batchlog write latency. See cassandra.yaml for
details. (DB-1367)
- New command 'nodetool abortrebuild' allows to abort a currently
running rebuild operation.
 The command must be executed on the node where the rebuild operation
is running. Streams
 may continue until they finish or timeout.
- Added "-Dcassandra.replace_consistency" to support streaming each
range from more than one source.
 By default, node replacement only streams one copy of data which may
be stale wrt to QUORUM/LOCAL_QUORUM.
 After replacing a down node with "-
Dcassandra.replace_consistency=LOCAL_DC_QUORUM", replacement node will be
up
 with consistent data assuming data is inserted with LOCAL_QUORUM.
Note that streaming multiple copies causes more

```

```
IO and compactions. Alternatively, operators can run repair before
bootstrapping the replacement with the
 default replace_consistency=ONE.
 Available options: "ONE"(default), "LOCAL_DC_QUORUM",
"GLOBAL_QUORUM".
 - Rebuild functionality now requires to specify the streaming source
endpoints and/or DC/racks
 and no longer implicitly prefers the local data center. Specifying
no sources or DC/racks is
 now considered an error.
 See `nodetool help rebuild` about which combinations of DCs/racks or
endpoints to include or
 exclude are possible. (DB-581)
 - Nodetool repair will now run full repairs by default, use "-inc"
option to run incremental repair.
 - Incremental repairs are disallowed on tables with Materialized Views
or CDC. Run full repairs on these
 tables.
 - A new HINTS stage was created to handle incoming hints. The maximum
size of this thread pool is governed by
 the system property cassandra.hints.max_receive_threads, which has
default value max_hints_delivery_threads
 from cassandra.yaml.
 - A new BACKGROUND_IO stage was created to handle blocking background
IO operations like cache loading,
 commit log segment allocation and replay. The maximum size of this
thread pool is governed by the system
 property cassandra.io.background.max_pool_size.
 - DSE is logging by default a heap histogram on OutOfMemoryError. To
disable that behavior
 set the 'cassandra.printHeapHistogramOnOutOfMemoryError' System
property to 'false'.
```

## Upgrading

```

 - The ITrigger interface has been modified from "augment" to
"augmentNonBlocking" for non-blocking
 internal architecture (DB-1046).
 - Removed support for directly setting JMX remote port via the
com.sun.management.jmxremote.port
 system property, deprecated since 3.6. See CASSANDRA-11725 and
DB-1040 for more details.
 - The table system_auth.resource_role_permissions_index is no longer
used and should be dropped
 after all nodes are on 6.0.
 - Incremental repairs are no longer supported on tables with
materialized views or CDC until
 its limitations are addressed. See CASSANDRA-12888 for details.
 - Changes to permissions (CassandraAuthorizer, DseAuthorizer), roles
(CassandraRoleManager,
 DseRoleManager) and credentials (PasswordAuthenticator) are
broadcasted throughout the cluster.
```

This eliminates the need to have short-ish validity (permissions\_validity\_in\_ms, roles\_validity\_in\_ms, credentials\_validity\_in\_ms) and update-interval (permissions\_update\_interval\_in\_ms, roles\_update\_interval\_in\_ms, credentials\_update\_interval\_in\_ms) settings in `cassandra.yaml`. Consider raising the values for \*\_validity\_in\_ms and \*\_update\_interval\_in\_ms, if configured. Please note, that only "reachable" nodes will receive the invalidation messages and unreachable nodes may still hold outdated information after DCL command.

- REVOKE (and the new UNRESTRICT statement) now error out when the statement would not change anything. This means, an error is raised when the statement targets permissions which are not present for that resource.
- The caches for JMX permissions and credentials no longer exist and therefore the JMX beans for these caches have been removed, too. Also the configuration options for the credentials cache need to be removed from the `cassandra.yaml` file.

#### Deprecation

The following properties were renamed and are now deprecated before they are removed in the next major release:

- `rpc_address` -> `native_transport_address`
- `rpc_interface` -> `native_transport_interface`
- `rpc_interface_prefer_ipv6` -> `native_transport_interface_prefer_ipv6`
- `broadcast_rpc_address` -> `native_transport_broadcast_address`
- `rpc_keepalive` -> `native_transport_keepalive`
- Removed org.apache.cassandra.metrics.Table.{AntiCompactedBytes|NonAntiCompactedBytes} metrics.
- Removed following properties from `cassandra.yaml`: `concurrent_reads`, `concurrent_writes`, `concurrent_counter_writes`, `concurrent_materialized_view_writes`, `streaming_socket_timeout_in_ms`

#### 4.0

====

#### New features

SSTableDump now supports the `-l` option to output each partition as its own json object  
See CASSANDRA-13848 for more detail

- The `currentTimestamp`, `currentDate`, `currentTime` and `currentTimeUUID` functions have been added.  
See CASSANDRA-13132
- Support for arithmetic operations between `'timestamp'/'date'` and `'duration'` has been added.

```
See CASSANDRA-11936
- Support for arithmetic operations on number has been added. See
CASSANDRA-11935
- Preview expected streaming required for a repair (nodetool repair --
preview), and validate the
consistency of repaired data between nodes (nodetool repair --
validate). See CASSANDRA-13257
- Support for selecting Map values and Set elements has been added for
SELECT queries. See CASSANDRA-7396
- The initial build of materialized views can be parallelized. The
number of concurrent builder
threads is specified by the property
`cassandra.yaml:concurrent_materialized_view_builders`.
This property can be modified at runtime through both JMX and the
new `setconcurrentviewbuilders`
and `getconcurrentviewbuilders` nodetool commands. See
CASSANDRA-12245 for more details.
```

## TinkerPop Changes

A list of DataStax Enterprise 6.0.0 production-certified changes in addition to TinkerPop 3.2.8.

DataStax Enterprise (DSE) 6.0.0 includes all changes from previous releases plus these production-certified changes that are in addition to TinkerPop 3.2.8:

- Made iterate() a first class step. (TINKERPOP-1834)
- Fixed a bug in NumberHelper that led to wrong min/max results if numbers exceeded the Integer limits. (TINKERPOP-1873)
- Improved error messaging for failed serialization and deserialization of request/response messages.
- Fixed bug in handling of Direction.BOTH in Messenger implementations to pass the message to the opposite side of the `StarGraph` in VertexPrograms for OLAP traversals. (TINKERPOP-1862)
- Fixed a bug in Gremlin Console which prevented handling of gremlin.sh flags that had an equal sign (=) between the flag and its arguments. (TINKERPOP-1879)
- Fixed a bug where SparkMessenger was not applying the edgeFunction`from MessageScope`in VertexPrograms for OLAP-based traversals. (TINKERPOP-1872)
- TinkerPop drivers prior to 3.2.4 won't authenticate with Kerberos anymore. A long-deprecated option on the Gremlin Server protocol was removed.

## DataStax Bulk Loader release notes

Release notes for DataStax Bulk Loader 1.1.x and 1.0.x.

Release notes for DataStax Bulk Loader 1.1.x and 1.0.x.

DataStax Bulk Loader 1.1.x and 1.0.x can migrate data in CSV or JSON format into DSE from another DSE or Apache Cassandra<sup>TM</sup> cluster.

- Can unload data from any Cassandra 2.1 or later data source

- Can load data to DSE 5.0 or later

## DataStax Bulk Loader 1.1.0 release notes

Changes, enhancements, and resolved issues for DataStax Bulk Loader 1.1.0.

18 June 2018

DataStax Bulk Loader 1.1.0 release notes include:

- [1.1.0 Changes and enhancements \(page 57\)](#)
- [1.1.0 Resolved issues \(page 57\)](#)

### 1.1.0 Changes and enhancements

After upgrade to 1.01.0, be sure to review and adjust scripts to use changed settings.

- Combine batch.mode and batch.enabled into a single setting: batch.mode. If you are using the batch.enabled setting in scripts, change to batch.mode with value DISABLED. (DAT-287)
- Improve handling of Univocity exceptions. (DAT-286)
- Logging improvements. (DAT-290)
  - # Log messages are logged only to `operation.log`. Logging does not print to `stdout`.
  - # Configurable logging levels with the `log.verbosity` setting.
  - # The setting `log.ansiEnabled` is changed to `log.ansiMode`.
- New count workflow. (DAT-291, DAT-299)
  - # Supports counting rows in a table.
  - # Configurable counting mode.
  - # When mode = partitions, configurable number of partitions to count. Support to count the number of rows for the  $n$  biggest partitions in a table.
- Counter tables are supported for load and unload. (DAT-292)
- Improve validation to include user-supplied queries and mappings. (DAT-294)
- The `codec.timestamp CQL_DATE_TIME` setting is renamed to `CQL_TIMESTAMP`. Adjust scripts to use the new setting. (DAT-298)

### 1.1.0 Resolved issues:

- Generated query does not contain all token ranges when a range wraps around the ring. (DAT-295)
- Empty map values do not work when loading using `dsbulk`. (DAT-297)
- DSBulk cannot handle columns of type `list<timestamp>`. (DAT-288)
- Generated queries do not respect indexed mapping order. (DAT-289)
- DSBulk fails to start with Java 10+. (DAT-300)

## DataStax Bulk Loader 1.0.2 release notes

Release notes for DataStax Bulk Loader 1.0.2.

5 June 2018

DataStax Bulk Loader 1.0.2 release notes include:

- [1.0.2 Changes and enhancements \(page 58\)](#)

#### 1.0.2 Changes and enhancements

- DataStax Bulk Loader 1.0.2 is bundled with DSE 6.0.1. (DSP-16206)
- Configure whether to use ANSI colors and other escape sequences in [log messages](#) printed to standard output and standard error. (DAT-249)

## DataStax Bulk Loader 1.0.1 release notes

Release notes for DataStax Bulk Loader 1.0.1.

17 April 2018

DataStax Bulk Loader 1.0.1 release notes include:

- [1.0.1 Changes and enhancements \(page 58\)](#)
- [1.0.1 Resolved issues \(page 58\)](#)

#### 1.0.1 Changes and enhancements

- DataStax Bulk Loader ([dsbulk](#)) version 1.0.1 is automatically installed with DataStax Enterprise, and can also be installed as a standalone tool. DataStax Bulk Loader 1.0.1 is supported for use with DSE 5.0 and later. (DSP-13999, DSP-15623)
- Support to manage special characters on the [command line](#) and in the configuration file. (DAT-229)
- Improve error messages for incorrect mapping. (DAT-235)
- Improved [monitoring options](#). (DAT-238)
- Detect console width on Windows. (DAT-240)
- Null words are supported by all connectors. The schema.nullStrings is changed to codec.nullWords. Renamed the convertTo and convertFrom methods. See [Codec options](#) and [Schema options](#). (DAT-241)
- Use Logback to improve filtering to make stack traces more readable and useful. On ANSI-compatible terminals, the date prints in green, the hour in cyan, the level is blue (INFO) or red (WARN), and the message prints in black. (DAT-242)
- Improved messaging for completion with errors. (DAT-243)
- Settings schema.allowExtraFields and schema.allowMissingFields are added to [reference.conf](#). (DAT-244)
- Support is dropped for using :port to specify the port to connect to. Specify the port for all hosts only with [driver.port](#). (DAT-245)

#### 1.0.1 Resolved issues

- Numeric overflows should display the original input that caused the overflow. (DAT-237)
- Null words are not supported by all connectors. (DAT-241)

- Addresses might not be properly translated when cluster has custom native port. (DAT-245)

## DataStax Studio release notes

Release notes for DataStax Studio 6.0.x.

Release notes for DataStax Studio 6.0.x.

### DataStax Studio 6.0.1 release notes

Release notes for DataStax Studio 6.0.1.

18 July 2018

DataStax Studio release notes for 6.0.1 include:

- [6.0.1 Changes and enhancement \(page 59\)](#)
- [6.0.1 Resolved issues \(page 59\)](#)

#### 6.0.1 Changes and enhancements

- Correctly handle and display all numerical values and formats that DSE supports. (STUDIO-2253)
- New Schema Reset icon (). Click to reset drop-down selections in schema view. (STUDIO-2266)

#### 6.0.1 Resolved issues

- Spark SQL cell execution causes incorrect Query execution exceeded configured execution timeout message. (STUDIO-2251)
- Content assist freezes and becomes unresponsive. (STUDIO-2265)

For troubleshooting, see [Content assist not working](#).

### DataStax Studio 6.0.0 release notes

Release notes for DataStax Studio 6.0.0.

17 April 2018

DataStax Studio release notes for 6.0.0 include:

- [6.0.0 Known issues \(page 59\)](#)
- [6.0.0 Changes and enhancement \(page 60\)](#)

#### 6.0.0 Known issues

- After some operations in graph view, the graph is displayed off center. To resolve, double-click anywhere in the graph view to recenter the graph.
- If Java 9 or later is installed, you might see this error. (STUDIO-2213)

```
java.lang.ClassNotFoundException: javax.xml.bind.JAXBContext
```

To resolve, uninstall Java 9 and install Java 8 on the machine where Studio is installed. Studio supports only Java 8.

- Spark SQL cell execution causes incorrect `Query execution exceeded configured execution timeout` message. (STUDIO-2251)

To workaround, set `executionTimeoutMs` in `configuration.yaml` to a value other than zero. For example, to set a 10-minute timeout: `executionTimeoutMs: 600000`.

## 6.0.0 Changes and enhancements

- Notebook sharing with [export \(page 892\)](#) and [import \(page 893\)](#) capabilities. (STUDIO-577)
- [Notebook history \(page 889\)](#) provides a historical dated record to track and rollback changes. (STUDIO-945)
- Support to query and analyze data with [Spark SQL \(page 887\)](#) in Studio
- [Interactive graph \(page 880\)](#)
- Improved graph view with added ability to remove selected vertices/edges from view and expand neighborhood to edge labels. (STUDIO-167)
- Support for [cancel execution \(page 882\)](#) in Gremlin cells. (STUDIO-145)

# Installing DataStax Enterprise

Guidance for choosing the best DataStax Enterprise installation method for your purposes.

See [Which install method should I use?](#).

# Configuration

Information about configuring DataStax Enterprise, such as recommended production setting, configuration files, snitch configuration, start-up parameters, heap dump settings, using virtual nodes, and more.

## Recommended production settings

Recommended settings for Linux platforms on DataStax Enterprise.

The following sections provide recommendations for optimizing your DataStax Enterprise installation on Linux:

- Use the latest Java Virtual Machine ([page 62](#))
- Synchronize clocks ([page 62](#))
- TCP settings ([page 63](#))
- Disable CPU frequency scaling ([page 64](#))
- Make sure that new settings persist after reboot ([page 64](#))
- Optimize SSDs ([page 64](#))
- Use the optimum --setra setting for RAID on SSD ([page 65](#))
- Disable zone\_reclaim\_mode on NUMA systems ([page 65](#))
- Set user resource limits ([page 65](#))
- Disable swap ([page 66](#))
- Check the Java Hugepages setting ([page 67](#))
- Set the heap size for optional Java garbage collection in DataStax Enterprise ([page 67](#))
- Determining the heap size when using Concurrent-Mark-Sweep (CMS) garbage collection in DataStax Enterprise ([page 67](#))
- Set the heap size for optimal Java garbage collection ([page 67](#))
- Apply optimum blockdev --setra settings for RAID on spinning disks ([page 68](#))

### Use the latest Java Virtual Machine

Configure your system to use the latest version of [Oracle Java SE 8 JRE or JDK](#) (recommended). The minimum supported version is 1.8u151. Java 9 and later are not supported.

**Note:** JDK provides more classes and tools for support and troubleshooting operations.

### Synchronize clocks

Synchronize the clocks on all nodes and application servers. Use NTP (Network Time Protocol) or other methods.

This is required because DataStax Enterprise (DSE) overwrites a column only if there is another version whose timestamp is more recent, which can happen when machines in are different locations.

DSE timestamps are encoded as microseconds since UNIX epoch without timezone information. The timestamp for all writes in DSE is UTC (Universal Time Coordinated). DataStax recommends converting to local time only when generating output to be read by humans.

## TCP settings

During low traffic intervals, a firewall configured with an idle connection timeout can close connections to local nodes and nodes in other data centers. To prevent connections between nodes from timing out, set the following network kernel settings:

```
sudo sysctl -w \
net.ipv4.tcp_keepalive_time=60 \
net.ipv4.tcp_keepalive_probes=3 \
net.ipv4.tcp_keepalive_intvl=10
```

These values set the TCP keepalive timeout to 60 seconds with 3 probes, 10 seconds gap between each. The settings detect dead TCP connections after 90 seconds ( $60 + 10 + 10 + 10$ ). There is no need to be concerned about the additional traffic as it's negligible; permanently leaving these settings isn't an issue. See [Securing DataStax Enterprise ports](#).

To handle thousands of concurrent connections used by the database, change these Linux kernel settings:

```
sudo sysctl -w \
net.core.rmem_max=16777216 \
net.core.wmem_max=16777216 \
net.core.rmem_default=16777216 \
net.core.wmem_default=16777216 \
net.core.optmem_max=40960 \
net.ipv4.tcp_rmem=4096 87380 16777216 \
net.ipv4.tcp_wmem=4096 65536 16777216
```

To persist the TCP settings:

1. Add the following to the `/etc/sysctl.conf` file:

```
net.ipv4.tcp_keepalive_time=60
net.ipv4.tcp_keepalive_probes=3
net.ipv4.tcp_keepalive_intvl=10
net.core.rmem_max=16777216
net.core.wmem_max=16777216
net.core.rmem_default=16777216
net.core.wmem_default=16777216
net.core.optmem_max=40960
net.ipv4.tcp_rmem=4096 87380 16777216
net.ipv4.tcp_wmem=4096 65536 16777216
```

2. Load the settings using one of the following commands:

```
sudo sysctl -p /etc/sysctl.conf
sudo sysctl -p /etc/sysctl.d/filename.conf
```

## Disable CPU frequency scaling

Recent Linux systems include a feature called *CPU frequency scaling* or *CPU speed scaling*. It allows a server's clock speed to be dynamically adjusted so that the server can run at lower clock speeds when the demand or load is low. This reduces the server's power consumption and heat output (which significantly impacts cooling costs). Unfortunately, this behavior has a detrimental effect on servers running DataStax Enterprise because throughput can get capped at a lower rate.

On most Linux systems, a `CPUfreq` governor manages the scaling of frequencies based on defined rules and the default `ondemand` governor switches the clock frequency to maximum when the demand is high and switches to the lowest frequency when the system is idle.

Do not use governors that lower the CPU frequency. To ensure optimal performance, reconfigure all CPUs to use the `performance` governor, which locks the frequency at maximum. This governor will not switch frequencies, which means there will be no power savings but the servers will always run at maximum throughput. On most systems, set the governor as follows:

```
for CPUFREQ in /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor
do
 [-f $CPUFREQ] || continue
 echo -n performance > $CPUFREQ
done
```

For more information, see [High server load and latency when CPU frequency scaling is enabled](#) in the DataStax Help Center.

## Make sure that new settings persist after reboot

**Caution:** Depending on your environment, some of the following settings may not be persisted after reboot. Check with your system administrator to ensure they are viable for your environment.

## Optimize SSDs

The default SSD configurations on most Linux distributions are not optimal. Follow these steps to ensure the best settings for SSDs:

1. Ensure that the `SysFS` rotational flag is set to `false` (zero).

This overrides any detection by the operating system to ensure the drive is considered an SSD.

2. Apply the same rotational flag setting for any block devices created from SSD storage, such as mdarrays.
3. Set the IO scheduler to either `deadline` or `noop`:

- The noop scheduler is the right choice when the target block device is an array of SSDs behind a high-end IO controller that performs IO optimization.
  - The deadline scheduler optimizes requests to minimize IO latency. If in doubt, use the deadline scheduler.
- 4. Set the readahead value for the block device to 8 KB.**

This setting tells the operating system not to read extra bytes, which can increase IO time and pollute the cache with bytes that weren't requested by the user.

For example, if the SSD is /dev/sda, in /etc/rc.local:

```
echo deadline > /sys/block/sda/queue/scheduler
#OR...
#echo noop > /sys/block/sda/queue/scheduler
touch /var/lock/subsys/local
echo 0 > /sys/class/block/sda/queue/rotational
echo 8 > /sys/class/block/sda/queue/read_ahead_kb
```

## Use the optimum --setra setting for RAID on SSD

The optimum readahead setting for RAID on SSDs (in Amazon EC2) is 8KB, the same as it is for non-RAID SSDs. For details, see [Optimizing SSDs \(page 64\)](#).

## Disable zone\_reclaim\_mode on NUMA systems

The Linux kernel can be inconsistent in enabling/disabling zone\_reclaim\_mode. This can result in odd performance problems.

To ensure that zone\_reclaim\_mode is disabled:

```
echo 0 > /proc/sys/vm/zone_reclaim_mode
```

For more information, see [Peculiar Linux kernel performance problem on NUMA systems](#).

## Set user resource limits

Use the ulimit -a command to view the current limits. Although limits can also be temporarily set using this command, DataStax recommends making the changes permanent:

Package installations:

Ensure that the following settings are included in the /etc/security/limits.d/cassandra.conf file:

```
<cassandra_user> - memlock unlimited
<cassandra_user> - nofile 100000
<cassandra_user> - nproc 32768
<cassandra_user> - as unlimited
```

Tarball installations:

In RHEL version 6.x, ensure that the following settings are included in the /etc/security/limits.conf file:

```
<cassandra_user> - memlock unlimited
```

```
<cassandra_user> - nofile 100000
<cassandra_user> - nproc 32768
<cassandra_user> - as unlimited
```

If you run DataStax Enterprise as root, some Linux distributions, such as Ubuntu, require setting the limits for root explicitly instead of using *cassandra\_user*:

```
root - memlock unlimited
root - nofile 100000
root - nproc 32768
root - as unlimited
```

For RHEL 6.x-based systems, also set the **nproc** limits in `/etc/security/limits.d/90-nproc.conf`:

```
cassandra_user - nproc 32768
```

For all installations, add the following line to `/etc/sysctl.conf`:

```
vm.max_map_count = 1048575
```

For installations on Debian and Ubuntu operating systems, the `pam_limits.so` module is not enabled by default. Edit the `/etc/pam.d/su` file and uncomment this line:

```
session required pam_limits.so
```

This change to the PAM configuration file ensures that the system reads the files in the `/etc/security/limits.d` directory.

To make the changes take effect, reboot the server or run the following command:

```
sudo sysctl -p
```

For more information, see [Insufficient user resource limits errors](#).

### Disable swap

DataStax strongly recommends disabling swap entirely (`sudo swapoff --all`). Because the database has multiple replicas and transparent failover, it is preferable for a replica to be killed immediately when memory is low rather than go into swap. This allows traffic to be immediately redirected to a functioning replica instead of continuing to hit the replica that has high latency due to swapping. If your system has a lot of DRAM, swapping still lowers performance significantly because the OS swaps out executable code so that more DRAM is available for caching disks.

```
sudo swapoff --all
```

To make this change permanent, remove all swap file entries from `/etc/fstab`.

**Note:** If you insist on using swap, you can set `vm.swappiness=1`. This allows the kernel swap out the absolute least used parts.

To make this change permanent, remove all swap file entries from `/etc/fstab`.

For more information, see [Nodes seem to freeze after some period of time](#).

## Check the Java Hugepages setting

Many modern Linux distributions ship with Transparent Hugepages enabled by default. When Linux uses Transparent Hugepages, the kernel tries to allocate memory in large chunks (usually 2MB), rather than 4K. This can improve performance by reducing the number of pages the CPU must track. However, some applications still allocate memory based on 4K pages. This can cause noticeable performance problems when Linux tries to defrag 2MB pages. For more information, see the [Cassandra Java Huge Pages](#) blog and this [RedHat bug report](#).

To solve this problem, disable `defrag` for hugepages. Enter:

```
echo never | sudo tee /sys/kernel/mm/transparent_hugepage/defrag
```

For more information, including a temporary fix, see [No DataStax Enterprise processing but high CPU usage](#).

## Set the heap size for optional Java garbage collection in DataStax Enterprise

The default JVM garbage collection (GC) is G1.

**Note:** DataStax does not recommend using G1 when using Java 7. This is due to a problem with class unloading in G1. In Java 7, PermGen fills up indefinitely until a full GC is performed.

Heap size is usually between  $\frac{1}{4}$  and  $\frac{1}{2}$  of system memory. Do not devote all memory to heap because it is also used for offheap cache and file system cache.

The easiest way to determine the optimum heap size for your environment is:

1. Set the `MAX_HEAP_SIZE` in the `cassandra-env.sh` file to a high arbitrary value on a single node.
2. View the heap used by that node:
  - Enable GC logging and check the logs to see trends.
  - Use [List View](#) in OpsCenter.
3. Use the value for setting the heap size in the cluster.

**Note:** This method decreases performance for the test node, but generally does not significantly reduce cluster performance.

If you don't see improved performance, contact the [DataStax Services team](#) for additional help in tuning the JVM.

## Determining the heap size when using Concurrent-Mark-Sweep (CMS) garbage collection in DataStax Enterprise

There are many nuances for tuning CMS. It requires time, expertise, and repeated testing to get the best results. DataStax recommends contacting the [DataStax Services team](#) instead. [Tuning Java resources](#) provides the basic information to get you started.

## Set the heap size for optimal Java garbage collection

See [Tuning Java resources](#).

## Apply optimum blockdev --setra settings for RAID on spinning disks

Typically, a readahead of 128 is recommended.

Check to ensure `setra` is not set to 65536:

```
sudo blockdev --report /dev/spinning_disk
```

To set `setra`:

```
sudo blockdev --setra 128 /dev/spinning_disk
```

**Note:** The recommended setting for RAID on SSDs is the same as that for SSDs that are not being used in a RAID installation. For details, see [Optimizing SSDs \(page 64\)](#).

## YAML and configuration properties

Information on how to configure DataStax Enterprise using the `cassandra.yaml`, `dse.yaml`, `cassandra-rackdc.properties`, and `cassandra-topology.properties` files.

### cassandra.yaml configuration file

The `cassandra.yaml` file is the main configuration file for DataStax Enterprise.

The `cassandra.yaml` file is the main configuration file for DataStax Enterprise. The [dse.yaml \(page 104\)](#) file is the primary configuration file for security, DSE Search, DSE Graph, and DSE Analytics.

**Important:** After changing properties in the `cassandra.yaml` file, you must restart the node for the changes to take effect.

### Syntax

For the properties in each section, the parent setting has zero spaces. Each child entry requires at least two spaces. Adhere to the YAML syntax and retain the spacing.

- Literal default values are shown as `literal`.
- Calculated values are shown as `calculated`.
- Default values that are not defined are shown as `Default: none`.
- Internally defined default values are described.

**Note:** Default values can be defined internally, commented out, or have implementation dependencies on other properties in the `cassandra.yaml` file. Additionally, some commented-out values may not match the actual default values. The commented out values are recommended alternatives to the default values.

## Organization

The configuration properties are grouped into the following sections:

- [Quick start \(page 69\)](#)  
The minimal properties needed for configuring a cluster.
- [Default directories \(page 70\)](#)  
If you have changed any of the default directories during installation, set these properties to the new locations. Make sure you have root access.
- [Commonly used \(page 71\)](#)  
Properties most frequently used when configuring DataStax Enterprise.
- [Performance tuning \(page 76\)](#)  
Tuning performance and system resource utilization, including commit log, compaction, memory, disk I/O, CPU, reads, and writes.
- [Advanced \(page 83\)](#)  
Properties for advanced users or properties that are less commonly used.
- [Security \(page 96\)](#)  
[DSE Unified Authentication](#) provides authentication, authorization, and role management.
- [Continuous paging options \(page 103\)](#) Properties configure memory, threads, and duration when pushing pages continuously to the client.

## Quick start properties

The minimal properties needed for configuring a cluster.

```
cluster_name: 'Test Cluster'
listen_address: localhost
listen_interface: wlan0
listen_interface_prefer_ipv6: false
```

**Tip:** See [Initializing a DataStax Enterprise cluster](#).

### **cluster\_name**

The name of the cluster. This setting prevents nodes in one logical cluster from joining another. All nodes in a cluster must have the same value.

Default: 'Test Cluster'

### **listen\_address**

The IP address or hostname that the database binds to for connecting this node to other nodes.

#### **Warning:**

- Never set `listen_address` to 0.0.0.0.

- Set `listen_address` or `listen_interface`, do not set both.

Default: `localhost`

### `listen_interface`

The interface that the database binds to for connecting to other nodes. Interfaces must correspond to a single address. IP aliasing is not supported.

**Warning:** Set `listen_address` or `listen_interface`, not both.

Default: commented out (`wlan0`)

### `listen_interface_prefer_ipv6`

Use IPv4 or IPv6 when interface is specified by name.

- `false` - use first IPv4 address.
- `true` - use first IPv6 address.

When only a single address is used, that address is selected without regard to this setting.

Default: commented out (`false`)

## Default directories

```
data_file_directories:
 - /var/lib/cassandra/data
commitlog_directory: /var/lib/cassandra/commitlog
cdc_raw_directory: /var/lib/cassandra/cdc_raw
hints_directory: /var/lib/cassandra/hints
saved_caches_directory: /var/lib/cassandra/saved_caches
```

If you have changed any of the default directories during installation, set these properties to the new locations. Make sure you have root access.

### `data_file_directories`

The directory where table data is stored on disk. The database distributes data evenly across the location, subject to the granularity of the configured compaction strategy. If not set, the directory is `$DSE_HOME/data/data`.

**Tip:** For production, DataStax recommends [RAID 0 and SSDs](#).

Default: `- /var/lib/cassandra/data`

### `commitlog_directory`

The directory where the commit log is stored. If not set, the directory is `$DSE_HOME/data/commitlog`.

For optimal write performance, place the commit log on a separate disk partition, or ideally on a separate physical device from the data file directories. Because the commit log is append only, a hard disk drive (HDD) is acceptable.

Default: `/var/lib/cassandra/commitlog`

### `cdc_raw_directory`

The directory where the change data capture (CDC) commit log segments are stored on flush. DataStax recommends a physical device that is separate from the data directories. If not set, the directory is `$DSE_HOME/data/cdc_raw`.

Default: `/var/lib/cassandra/cdc_raw`

### `hints_directory`

The directory in which hints are stored. If not set, the directory is \$CASSANDRA\_HOME/data/hints.

Default: /var/lib/cassandra/hints

### **saved\_caches\_directory**

The directory location where table key and row caches are stored. If not set, the directory is \$DSE\_HOME/data/saved\_caches.

Default: /var/lib/cassandra/saved\_caches

## **Commonly used properties**

Properties most frequently used when configuring DataStax Enterprise.

Before starting a node for the first time, DataStax recommends that you carefully evaluate your requirements.

- [Common initialization properties \(page 71\)](#)
- [Common compaction settings \(page 75\)](#)
- [Common memtable settings \(page 75\)](#)
- [Common automatic backup settings \(page 75\)](#)

## **Common initialization properties**

```
commit_failure_policy: stop
prepared_statements_cache_size_mb:
disk_optimization_strategy: ssd
disk_failure_policy: stop
endpoint_snitch: com.datastax.bdp.snitch.DseSimpleSnitch
seed_provider:
 - org.apache.cassandra.locator.SimpleSeedProvider
 - seeds: "127.0.0.1"
enable_user_defined_functions: false
enable_scripted_user_defined_functions: false
enable_user_defined_functions_threads: true
```

**Note:** Be sure to set the properties in the [Quick start section \(page 69\)](#) as well.

### **commit\_failure\_policy**

Policy for commit disk failures:

- die - Shut down the node and kill the JVM, so the node can be replaced.
- stop - Shut down the node, leaving the node effectively dead, available for inspection using JMX.
- stop\_commit - Shut down the commit log, letting writes collect but continuing to service reads.
- ignore - Ignore fatal errors and let the batches fail.

Default: stop

### **prepared\_statements\_cache\_size\_mb**

Maximum size of the native protocol prepared statement cache. Change this value only if there are more prepared statements than fit in the cache.

Generally, the calculated default value is appropriate and does not need adjusting. DataStax recommends contacting the [DataStax Services team](#) before changing this value.

**Note:** Specifying a value that is too large results in long running GCs and possibly out-of-memory errors. Keep the value at a small fraction of the heap.

Constantly re-preparing statements is a performance penalty. When not set, the default is automatically calculated to `heap / 256` or 10 MB, whichever is greater.

Default: `calculated`

#### **disk\_optimization\_strategy**

The strategy for optimizing disk reads.

- `ssd` - solid state disks
- `spinning` - spinning disks

When commented out, the default is `ssd`.

Default: commented out (`ssd`)

#### **disk\_failure\_policy**

Sets how the database responds to disk failure. Recommend settings: `stop` or `best_effort`. Valid values:

- `die` - Shut down gossip and client transports, and kill the JVM for any file system errors or single SSTable errors, so the node can be replaced.
- `stop_paranoid` - Shut down the node, even for single SSTable errors.
- `stop` - Shut down the node, leaving the node effectively dead, but available for inspection using JMX.
- `best_effort` - Stop using the failed disk and respond to requests based on the remaining available SSTables. This setting allows obsolete data at consistency level of ONE.
- `ignore` - Ignore fatal errors and lets the requests fail; all file system errors are logged but otherwise ignored.

**Tip:** See [Recovering from a single disk failure using JBOD](#).

Default: `stop`

#### **endpoint\_snitch**

A class that implements the `IEndpointSnitch` interface. The database uses the snitch to locate nodes and route requests.

**Important:** Use only snitch implementations bundled with DSE.

- `DseSimpleSnitch`

Appropriate only for development deployments. Proximity is determined by DSE workload, which places transactional, analytics, and search nodes into their separate datacenters. Does not recognize datacenter or rack information.

- `GossipingPropertyFileSnitch`

Recommended for production. Reads rack and datacenter for the local node in `cassandra-rackdc.properties` file and propagates these values to other nodes

via gossip. For migration from the PropertyFileSnitch, uses the `cassandra-topology.properties` file if it is present.

- **PropertyFileSnitch**

Determines proximity by rack and datacenter that are explicitly configured in `cassandra-topology.properties` file.

- **Ec2Snitch**

For EC2 deployments in a single region. Loads region and availability zone information from the Amazon EC2 API. The region is treated as the datacenter, the availability zone is treated as the rack, and uses only private IP addresses. For this reason, Ec2Snitch does not work across multiple regions.

- **Ec2MultiRegionSnitch**

Uses the public IP as the [broadcast\\_address \(page 84\)](#) to allow cross-region connectivity. This means you must also set [seed \(page 73\)](#) addresses to the public IP and open the [storage\\_port \(page 89\)](#) or [ssl\\_storage\\_port \(page 102\)](#) on the public IP firewall. For intra-region traffic, the database switches to the private IP after establishing a connection.

- **RackInferingSnitch**

Proximity is determined by rack and datacenter, which are assumed to correspond to the 3rd and 2nd octet of each node's IP address, respectively. Best used as an example for writing a custom snitch class (unless this happens to match your deployment conventions).

- **GoogleCloudSnitch**

Use for deployments on [Google Cloud Platform](#) across one or more regions. The region is treated as a datacenter and the availability zones are treated as racks within the datacenter. All communication occurs over private IP addresses within the same logical network.

- **CloudstackSnitch**

Use the CloudstackSnitch for [Apache Cloudstack](#) environments.

**Tip:** See [Snitches](#).

Default: `com.datastax.bdp.snitch.DseSimpleSnitch`

#### **seed\_provider**

The addresses of hosts that are designated as contact points in the cluster. A joining node contacts one of the nodes in the -seeds list to learn the topology of the ring.

**Important:** Use only seed provider implementations bundled with DSE.

- `class_name` - The class that handles the seed logic. It can be customized, but this is typically not required.

Default: `org.apache.cassandra.locator.SimpleSeedProvider`

- - seeds - A comma delimited list of addresses that are used by [gossip](#) for bootstrapping new nodes joining a cluster. If your cluster includes multiple nodes, you must change the list from the default value to the IP address of one of the nodes.

Default: "127.0.0.1"

**Attention:** Making every node a seed node is **not** recommended because of increased maintenance and reduced gossip performance. Gossip optimization is not critical, but it is recommended to use a small seed list (approximately three nodes per datacenter).

**Tip:** See [Initializing a single datacenter per workload type](#) and [Initializing multiple datacenters per workload type](#).

Default: `org.apache.cassandra.locator.SimpleSeedProvider`

#### **enable\_user\_defined\_functions**

Whether to enable user defined functions (UDFs). UDFs present a security risk, since they are executed on the server side. UDFs are executed in a sandbox to contain the execution of malicious code.

- true - Enabled. Supports Java as the code language. Detect endless loops and unintended memory leaks.
- false - Disabled.

Default: `false` (disabled)

#### **enable\_scripted\_user\_defined\_functions**

Whether to enable the use of JavaScript language in UDFs.

- true - Enabled. Allow JavaScript in addition to Java as a code language.
- false - Disabled. Only allow Java as a code language.

**Note:** If [enable\\_user\\_defined\\_functions \(page 74\)](#) is `false`, this setting has no impact.

Default: `false`

#### **enable\_user\_defined\_functions\_threads**

Whether to enable asynchronous UDF execution which requires a function to complete before being executed again.

- true - Enabled. Only one instance of a function can run at one time. Asynchronous execution prevents UDFs from running too long or forever and destabilizing the cluster.
- false - Disabled. Allows multiple instances of the same function to run simultaneously. Required to use UDFs within [GROUP BY](#) clauses.

**Caution:** Disabling asynchronous UDF execution implicitly disables the security manager. You must monitor the read timeouts for UDFs that run too long or forever, which can cause the cluster to destabilize.

Default: `true`

## Common compaction settings

```
compaction_throughput_mb_per_sec: 16
compaction_large_partition_warning_threshold_mb: 100
```

### **compaction\_throughput\_mb\_per\_sec**

The MB per second to throttle compaction for the entire system. The faster the database inserts data, the faster the system must compact in order to keep the SSTable count down.

- 16 to 32 x rate of write throughput in MB/second, recommended value.
- 0 - disable compaction throttling

**Tip:** See [Configuring compaction](#).

Default: 16

### **compaction\_large\_partition\_warning\_threshold\_mb**

The partition size threshold before logging a warning.

Default: 100

## Common memtable settings

```
memtable_heap_space_in_mb: 2048
memtable_offheap_space_in_mb: 2048
```

### **memtable\_heap\_space\_in\_mb**

The amount of on-heap memory allocated for memtables. The database uses the total of this amount and the value of memtable\_offheap\_space\_in\_mb to set a threshold for automatic memtable flush.

**Tip:** See [memtable\\_cleanup\\_threshold \(page 80\)](#) and [Tuning the Java heap](#).

Default: *calculated 1/4 of heap size (2048)*

### **memtable\_offheap\_space\_in\_mb**

The amount of off-heap memory allocated for memtables. The database uses the total of this amount and the value of memtable\_heap\_space\_in\_mb to set a threshold for automatic memtable flush.

**Tip:** See [memtable\\_cleanup\\_threshold \(page 80\)](#) and [Tuning the Java heap](#).

Default: *calculated 1/4 of heap size (2048)*

## Common automatic backup settings

```
incremental_backups: false
snapshot_before_compaction: false
```

### **incremental\_backups**

Whether to enable incremental backups.

- true - Enable incremental backups to create a hard link to each SSTable flushed or streamed locally in a `backups` subdirectory of the keyspace data. Incremental backups enable storing backups off site without transferring entire snapshots.
- false - Do not enable incremental backups.

**Important:** The database does not automatically clear incremental backup files. DataStax recommends setting up a process to clear incremental backup hard links each time a new snapshot is created.

**Tip:** See [Enabling incremental backups](#).

Default: `false`

### **snapshot\_before\_compaction**

Whether to take a snapshot before each compaction. A snapshot is useful to back up data when there is a data format change.

**Important:** Be careful using this option, the database does not clean up older snapshots automatically.

**Tip:** See [Configuring compaction](#).

Default: `false`

## Performance tuning properties

Tuning performance and system resource utilization, including commit log, compaction, memory, disk I/O, CPU, reads, and writes.

Performing tuning properties include:

- [Commit log settings \(page 76\)](#)
- [Change-data-capture \(CDC\) space settings \(page 78\)](#)
- [Common compaction settings \(page 75\)](#)
- [Common memtable settings \(page 75\)](#)
- [Cache and index settings \(page 80\)](#)
- [Disk settings \(page 81\)](#)

## Commit log settings

```
commitlog_sync: periodic
commitlog_sync_period_in_ms: 10000
commitlog_sync_group_window_in_ms: 1000
commitlog_sync_batch_window_in_ms: 2 //deprecated
commitlog_segment_size_in_mb: 32
commitlog_total_space_in_mb: 8192
commitlog_compression:
- class_name: LZ4Compressor
parameters:
-
```

### **commitlog\_sync**

The method that the database uses to acknowledge writes in milliseconds.

- periodic - Send ACK signal for writes immediately. Commit log is synced every `commitlog_sync_period_in_ms`.
- group - Send ACK signal for writes after the commit log has been flushed to disk. Wait up to `commitlog_sync_group_window_in_ms` between flushes.
- batch - Send ACK signal for writes after the commit log has been flushed to disk. Each incoming write triggers the flush task.

Default: `periodic`

#### **commitlog\_sync\_period\_in\_ms**

Use with `commitlog_sync: periodic`. Time interval between syncing the commit log to disk. Periodic syncs are acknowledged immediately.

Default: 10000

#### **commitlog\_sync\_group\_window\_in\_ms**

Use with `commitlog_sync: group`. The time that the database waits between flushing the commit log to disk. DataStax recommends using `group` instead of `batch`.

Default: commented out (1000)

#### **commitlog\_sync\_batch\_window\_in\_ms**

Deprecated. Use with `commitlog_sync: batch`. The maximum length of time that queries may be batched together.

Default: commented out (2)

#### **commitlog\_segment\_size\_in\_mb**

The size of an individual commitlog file segment. A commitlog segment may be archived, deleted, or recycled after all its data has been flushed to SSTables. This data can potentially include commitlog segments from every table in the system. The default size is usually suitable, but for commitlog archiving you might want a finer granularity; 8 or 16 MB is reasonable.

##### **Restriction:**

If you set [max\\_mutation\\_size\\_in\\_kb \(page 77\)](#) explicitly, then you must set `commitlog_segment_size_in_mb` to:

```
2 * max_mutation_size_in_kb / 1024
```

The value must be positive and less than 2048.

**Tip:** See [Commit log archive configuration](#).

Default: 32

#### **max\_mutation\_size\_in\_kb**

The maximum size of a mutation before the mutation is rejected. Before increasing the commitlog segment size of the commitlog segments, investigate why the mutations are larger than expected. Look for underlying issues with access patterns and data model, because increasing the commitlog segment size is a limited fix.

When not set, the default is calculated as `(commitlog_segment_size_in_mb * 1024) / 2`.

Default: `calculated`

#### **commitlog\_total\_space\_in\_mb**

Total space for commit logs. If the total space used by all commit logs exceeds this threshold, the database rounds up to the next nearest segment multiple and flushes memtables to disk for the oldest commitlog segments, removing those log segments from the commit log. This flushing reduces the amount of data to replay on start-up, and prevents infrequently updated tables from keeping commitlog segments indefinitely. If the `commitlog_total_space_in_mb` is small, the result is more flush activity on less-active tables.

**Tip:** See [Configuring memtable thresholds](#).

**Default for 64-bit JVMs:** *calculated (8192 or 25% of the total space of the commit log value, whichever is smaller)*

**Default for 32-bit JVMs:** *calculated (32 or 25% of the total space of the commit log value, whichever is smaller )*

#### **commitlog\_compression**

The compressor to use if commit log is compressed.

- - class\_name: LZ4, Snappy, or Deflate
- parameters: optional parameters for the compressor

When not set, uncompressed)

Default: commented out

### **Change-data-capture (CDC) space settings**

```
cdc_enabled: false
cdc_total_space_in_mb: 4096
cdc_free_space_check_interval_ms: 250
```

See also [cdc\\_raw\\_directory \(page 70\)](#).

#### **cdc\_enabled**

Whether to enable change data capture (CDC) functionality on a per-node basis. This modifies the logic used for write path allocation rejection.

- true - use cdc functionality to reject mutations that contain a CDC-enabled table if at space limit threshold in `cdc_raw_directory`
- false - standard behavior, never reject.

Default: `false`

#### **cdc\_total\_space\_in\_mb**

Total space to use for change-data-capture (CDC) logs on disk. If space allocated for CDC exceeds this value, the database throws `WriteTimeoutException` on mutations, including CDC-enabled tables. A `CDCCompactor` (a consumer) is responsible for parsing the raw CDC logs and deleting them when parsing is completed.

Default: *calculated (4096 or 1/8th of the total space of the drive where the [cdc\\_raw\\_directory \(page 70\)](#) resides)*

#### **cdc\_free\_space\_check\_interval\_ms**

Interval between checks for new available space for CDC-tracked tables when the [cdc\\_total\\_space\\_in\\_mb \(page 78\)](#) threshold is reached and the `CDCCompactor` is running behind or experiencing back pressure. When not set, the default is 250.

Default: commented out (250)

## Compaction settings

```
#concurrent_compactors: 1
concurrent_validations: 0
concurrent_materialized_view_builders: 2
sstable_preemptive_open_interval_in_mb: 50
```

**Tip:** See also [compaction\\_throughput\\_mb\\_per\\_sec \(page 75\)](#) in the common compaction settings section and [Configuring compaction](#).

### concurrent\_compactors

The number of concurrent compaction processes allowed to run simultaneously on a node, not including validation [compactions](#) for [anti-entropy](#) repair. Simultaneous compactions help preserve read performance in a mixed read-write workload by limiting the number of small SSTables that accumulate during a single long-running compaction. If your data directories are backed by [SSDs](#), increase this value to the number of cores. If compaction running too slowly or too fast, adjust [compaction\\_throughput\\_mb\\_per\\_sec \(page 75\)](#) first.

**Important:** Increasing concurrent compactors leads to more use of available disk space for compaction, because concurrent compactions happen in parallel, especially for STCS. Ensure that adequate disk space is available before increasing this configuration.

Generally, the calculated default value is appropriate and does not need adjusting. DataStax recommends contacting the [DataStax Services team](#) before changing this value.

Default: *calculated* The fewest number of disks or number of cores, with a minimum of 2 and a maximum of 8 per CPU core.

### concurrent\_validations

Number of simultaneous repair validations to allow. When not set, the default is unbounded. Values less than one are interpreted as unbounded.

Default: commented out (0) unbounded

### concurrent\_materialized\_view\_builders

Number of simultaneous materialized view builder tasks to allow.

Default: 2

### sstable\_preemptive\_open\_interval\_in\_mb

The size of the SSTables to trigger preemptive opens. The compaction process opens SSTables before they are completely written and uses them in place of the prior SSTables for any range previously written. This process helps to smoothly transfer reads between the SSTables by reducing page cache churn and keeps hot rows hot.

**Important:** A low value has a negative performance impact and will eventually cause heap pressure and GC activity. The optimal value depends on hardware and workload.

Default: 50

## Memtable settings

```
memtable_allocation_type: heap_buffers
memtable_cleanup_threshold: 0.34
file_cache_size_in_mb: 4096
memtable_flush_writers: 4
```

### **memtable\_allocation\_type**

The method the database uses to allocate and manage memtable memory.

- heap\_buffers - On heap NIO (non-blocking I/O) buffers.
- offheap\_buffers - Off heap (direct) NIO buffers.
- offheap\_objects - Native memory, eliminating NIO buffer heap overhead.

Default: heap\_buffers

### **memtable\_cleanup\_threshold (deprecated)**

Ratio used for automatic memtable flush.

Generally, the calculated default value is appropriate and does not need adjusting. DataStax recommends contacting the [DataStax Services team](#) before changing this value.

When not set, the default is calculated  $1 / (\text{memtable\_flush\_writers (page 80)} + 1)$

Default: commented out (0.34)

### **file\_cache\_size\_in\_mb**

Maximum memory to use for buffer pooling and SSTable chunk cache. 32 MB is reserved for pooling buffers, the remaining memory is the cache for holding uncompressed SSTable chunks. This pool is allocated off heap and is in addition to the memory allocated for heap. When manually tuning the size, consider:

- Other processes on machine
- Offheap data
- JVM heap
- If the node has an Analytics (Spark) workload

Memory is allocated only when needed.

Default:  $\text{calculated}(\text{MACHINE_MEMORY} - \text{HEAP_SIZE}) * 0.33$

### **memtable\_flush\_writers**

The number of memtable flush writer threads. These threads are blocked by disk I/O, and each one holds a memtable in memory while blocked. If your data directories are backed by SSDs, increase this setting to the number of cores.

Default: 2 The smaller of number of disks or number of cores with a minimum of 2 and a maximum of 8

## Cache and index settings

```
column_index_size_in_kb: 16
index_summary_capacity_in_mb:
index_summary_resize_interval_in_minutes: 60
```

### **column\_index\_size\_in\_kb**

Granularity of the index of rows within a partition. For huge rows, decrease this setting to improve seek time. Lower density nodes might benefit from decreasing this value to 4, 2, or 1.

**Important:** If you use key cache, do not make this setting too large because key cache will be overwhelmed.

If you're unsure of the size of the rows, it's best to use the default setting.

Default: 64

#### **index\_summary\_capacity\_in\_mb**

Size of the memory pool for SSTable index summaries. If the memory usage of all index summaries exceeds this limit, any SSTables with low read rates shrink their index summaries to meet this limit. This is a best-effort process. In extreme conditions, the database may use more than this amount of memory.

Default: *calculated* (5% of the heap size)

#### **index\_summary\_resize\_interval\_in\_minutes**

How often to re-sample index summaries. Periodic re-sampling redistributes memory from the fixed-size pool to SSTables proportional their recent read rates.

- -1 - disable re-sampling of index summaries, and leave existing index summaries at their current sampling level
- *interval* - the time interval between index summary re-sampling

Default: 60

### Disk settings

```
stream_throughput_outbound_megabits_per_sec: 200
inter_dc_stream_throughput_outbound_megabits_per_sec: 200
streaming_keep_alive_period_in_secs: 300
streaming_connections_per_host: 1
```

#### **stream\_throughput\_outbound\_megabits\_per\_sec**

Throttle for the throughput of all outbound streaming file transfers on a node. The database does mostly sequential I/O when streaming data during bootstrap or repair which can saturate the network connection and degrade client (RPC) performance. When not set, the value is 200 Mbps.

Default: commented out (200)

#### **inter\_dc\_stream\_throughput\_outbound\_megabits\_per\_sec**

Throttle for all streaming file transfers between datacenters, and for network stream traffic as configured with [stream\\_throughput\\_outbound\\_megabits\\_per\\_sec \(page 81\)](#). When not set, the value is 200 Mbps.

Default: commented out (200)

#### **streaming\_keep\_alive\_period\_in\_secs**

Interval to send keep-alive messages. The stream session fails when a keep-alive message is not received for 2 keep-alive cycles. When not set, the default is 300 seconds (5 minutes) so that a stalled stream times out in 10 minutes.

Default: commented out (300)

#### **streaming\_connections\_per\_host**

Maximum number of connections per host for streaming. Increase this value when you notice that joins are CPU-bound, rather than network-bound. For example, a few nodes with large files.

## Fsync settings

```
trickle_fsync: true
trickle_fsync_interval_in_kb: 10240
```

### **trickle\_fsync**

When set to true, causes fsync to force the operating system to flush the dirty buffers at the set interval `trickle_fsync_interval_in_kb`. Enable this parameter to prevent sudden dirty buffer flushing from impacting read latencies. Recommended for use with SSDs, but not with HDDs.

Default: `false`

### **trickle\_fsync\_interval\_in\_kb**

The size of the fsync in kilobytes.

Default: `10240`

### **max\_value\_size\_in\_mb**

The maximum size of any value in SSTables. SSTables are marked as corrupted when the threshold is exceeded.

Default: `256`

## Thread Per Core (TPC) parameters

```
#tpc_cores:
tpc_io_cores:
io_global_queue_depth: 128
```

### **tpc\_cores**

Number of TPC event loops. If not set, the default is the number of cores (processors on the machine) minus one.

**Attention:** Do not tune. DataStax recommends contacting the [DataStax Services team](#) before changing this value.

Default: commented out (`number of cores -1`)

### **tpc\_io\_cores**

The number of [tpc\\_cores \(page 82\)](#) to use for asynchronous disk reads.

**Attention:** Do not tune. DataStax recommends contacting the [DataStax Services team](#) before changing this value.

Default: commented out (`min(io_global_queue_depth/4, tpc_cores)`)

### **io\_global\_queue\_depth**

Global IO queue depth used for reads when AIO is enabled (the default for SSDs). The optimal queue depth as found with the fio tool for a given disk setup.

**Attention:** Do not tune. DataStax recommends contacting the [DataStax Services team](#) before changing this value.

Default: `128`

## NodeSync parameters

```
nodesync:
 rate_in_kb: 1024
```

By default, the NodeSync service runs on every node.

**Tip:** Manage the NodeSync service using the [nodetool nodesyncservice](#) command.

**Tip:** See [Setting the rate \(page 166\)](#).

### rate\_in\_kb

The maximum bytes per second for data validation on the local node. The optimum validation rate for each node may vary.

Default: 1024

## Advanced properties

Properties for advanced users or properties that are less commonly used.

### *Advanced initialization properties*

```
batch_size_warn_threshold_in_kb: 64
batch_size_fail_threshold_in_kb: 640
unlogged_batch_across_partitions_warn_threshold: 10
broadcast_address: 1.2.3.4
listen_on_broadcast_address: false
initial_token:
num_tokens: 128
allocate_tokens_for_local_replication_factor: 3
partitioner: org.apache.cassandra.dht.Murmur3Partitioner
tracetype_query_ttl: 86400
tracetype_repair_ttl: 604800
```

### auto\_bootstrap

This setting has been removed from default configuration.

- true - causes new (non-seed) nodes migrate the right data to themselves automatically
- false - When initializing a fresh cluster *without* data

**Tip:** See [Initializing a DataStax Enterprise cluster](#).

When not set, the internal default is true.

Default: not present

### batch\_size\_warn\_threshold\_in\_kb

Threshold to log a warning message when any multiple-partition batch size exceeds this value in kilobytes.

**Caution:** Increasing this threshold can lead to node instability.

Default: 64

### batch\_size\_fail\_threshold\_in\_kb

Threshold to fail and log WARN on any multiple-partition batch whose size exceeds this value. The default value is 10X the value of `batch_size_warn_threshold_in_kb`.

Default: 640

#### **unlogged\_batch\_across\_partitions\_warn\_threshold**

Threshold to log a WARN message on any batches not of type LOGGED that span across more partitions than this limit.

Default: 10

#### **broadcast\_address**

The *public* IP address this node uses to broadcast to other nodes outside the network or across regions in multiple-region EC2 deployments. If this property is commented out, the node uses the same IP address or hostname as `listen_address`. A node does not need a separate `broadcast_address` in a single-node or single-datacenter installation, or in an EC2-based network that supports automatic switching between private and public communication. It is necessary to set a separate `listen_address` and `broadcast_address` on a node with multiple physical network interfaces or other topologies where not all nodes have access to other nodes by their private IP addresses. For specific configurations, see the instructions for [listen\\_address \(page 69\)](#).

Default: [listen\\_address \(page 69\)](#)

#### **listen\_on\_broadcast\_address**

Whether to enable the node to communicate on both interfaces.

- true - If this node uses multiple physical network interfaces, set a unique IP address for [broadcast\\_address \(page 84\)](#)
- false - if this node is on a network that automatically routes between public and private networks, like Amazon EC2 does

**Tip:** See [listen\\_address \(page 69\)](#).

Default: `false`

#### **initial\_token**

The token to start the contiguous range. Set this property for single-node-per-token architecture, in which a node owns exactly one contiguous range in the ring space. Setting this property overrides [num\\_tokens \(page 84\)](#).

If your installation is not using `vnodes` or this node's [num\\_tokens \(page 84\)](#) is set it to 1 or is commented out, you should always set an `initial_token` value when setting up a production cluster for the first time, and when adding capacity.

See [Generating tokens](#).

Use this parameter only with [num\\_tokens \(page 84\)](#) (`vnodes`) in special cases such as [Restoring from a snapshot](#).

Default: 1 (disabled)

#### **num\_tokens**

Set this property for virtual node token architecture. Determines the number of token ranges to assign to this [virtual node \(page 168\)](#) (`vnode`). Use a number between 1 and 128, where 1 disables `vnodes`. When the token number varies between nodes in a datacenter, the `vnode` logic assigns a proportional number of ranges relative to other nodes in the datacenter. In general, if all nodes have equal hardware capability, each node should have the same `num_tokens` value.

- **Random selection algorithm:**

**Note:** Over time loads in a datacenter using this algorithm become uneven distributed. The random selection algorithm method is **not** recommended by DataStax. Instead, use the allocation algorithm.

Assign token ranges randomly. A higher `num_token` value increases the probability that the data and workload are evenly distributed. Use 8 when randomly assigning token ranges.

- **Allocation algorithm:** Assign token ranges using the allocation algorithm which optimizes the workload balance using the target keyspace replication factor. Enabled when the [allocate\\_tokens\\_for\\_local\\_replication\\_factor \(page 85\)](#) is set. DataStax recommends setting the number of tokens to 8 to distribute the workload with ~10% variance between nodes.

To migrate an existing cluster from single node per token range to vnodes, see [Enabling virtual nodes on an existing production cluster](#).

**Note:** All other nodes in the datacenter must have the same token architecture, that is single-token, random algorithm vnode or allocation algorithm vnode architecture.

Default: 1 (disabled)

#### **allocate\_tokens\_for\_local\_replication\_factor**

The target replication factor (RF) of keyspaces in the datacenter when adding a vnode to an existing cluster or setting up nodes in a new datacenter. Triggers the recommended **algorithmic allocation** for the RF and `num_tokens` for this node. The allocation algorithm attempts to choose tokens in a way that optimizes replicated load over the nodes in the datacenter for the specified RF. The load assigned to each node is close to proportional to the number of vnodes.

**Tip:** See [Virtual node \(vnode\) configuration \(page 168\)](#), and for set up instructions see [Adding vnodes to an existing cluster](#) or [Adding a datacenter to a cluster](#).

**Note:** The allocation algorithm is supported only for the Murmur3Partitioner and RandomPartitioner partitioners. The Murmur3Partitioner is the default partitioning strategy for new DSE clusters and the right choice for new clusters in almost all cases.

Default: 3

#### **partitioner**

The class that distributes rows (by partition key) across all nodes in the cluster. Any `IPartitioner` may be used, including your own as long as it is in the class path. For new clusters use the default partitioner.

DataStax Enterprise provides the following partitioners for backward compatibility:

- `RandomPartitioner`
- `ByteOrderedPartitioner` (deprecated)
- `OrderPreservingPartitioner` (deprecated)

**Important:** Use only partitioner implementations bundled with DSE.

**Tip:** See [Partitioners](#).

Default: `org.apache.cassandra.dht.Murmur3Partitioner`

#### **tracetype\_query\_ttl**

TTL for different trace types used during logging of the query process.

Default: 86400

#### **tracetype\_repair\_ttl**

TTL for different trace types used during logging of the repair process.

Default: 604800

*Advanced automatic backup setting*

```
auto_snapshot: true
```

#### **auto\_snapshot**

Whether to enable snapshots of the data before truncating a keyspace or dropping a table. To prevent data loss, DataStax strongly advises using the default setting. If you set `auto_snapshot` to false, you lose data on truncation or drop.

Default: `true`

*Global row properties*

```
column_index_cache_size_in_kb: 2
row_cache_class_name: org.apache.cassandra.cache.OHCProvider
row_cache_size_in_mb: 0
row_cache_save_period: 0
row_cache_keys_to_save: 100
```

When creating or modifying tables, you can enable or disable the row cache for that table by setting the caching parameter. Other row cache tuning and configuration options are set at the global (node) level. The database uses these settings to automatically distribute memory for each table on the node based on the overall workload and specific table usage. You can also configure the save periods for these caches globally.

**Tip:** See [Configuring caches](#).

#### **column\_index\_cache\_size\_in\_kb**

Threshold for the total size of all index entries for a partition that the database stores in the partition key cache. If the total size of all index entries for a partition exceeds this amount, the database stops putting entries for this partition into the partition key cache.

Default: 2

#### **row\_cache\_class\_name**

The classname of the row cache provider to use. Valid values:

- `org.apache.cassandra.cache.OHCProvider` - fully off-heap
- `org.apache.cassandra.cache.SerializingCacheProvider` - partially off-heap, available in earlier releases

**Important:** Use only row cache provider implementations bundled with DSE.

When not set, the default is org.apache.cassandra.cache.OHCProvider (fully off-heap)

Default: commented out (org.apache.cassandra.cache.OHCProvider)

#### **row\_cache\_size\_in\_mb**

Maximum size of the row cache in memory. The row cache can save time, but it is space-intensive because it contains the entire row. Use the row cache only for hot rows or static rows. If you reduce the size, you may not get your hottest keys loaded on start up.

- 0 - disable row caching
- *MB* - Maximum size of the row cache in memory

Default: 0 (disabled)

#### **row\_cache\_save\_period**

The number of seconds that rows are kept in cache. Caches are saved to [saved\\_caches\\_directory \(page 71\)](#). This setting has limited use as described in **row\_cache\_size\_in\_mb**.

Default: 0 (disabled)

#### **row\_cache\_keys\_to\_save**

The number of keys from the row cache to save. All keys are saved.

Default: commented out (100)

### *Counter caches properties*

```
counter_cache_size_in_mb:
counter_cache_save_period: 7200
counter_cache_keys_to_save: 100
```

Counter cache helps to reduce counter locks' contention for hot counter cells. In case of RF = 1 a counter cache hit causes the database to skip the read before write entirely. With RF > 1 a counter cache hit still helps to reduce the duration of the lock hold, helping with hot counter cell updates, but does not allow skipping the read entirely. Only the local (clock, count) tuple of a counter cell is kept in memory, not the whole counter, so it is relatively cheap.

**Note:** If you reduce the counter cache size, the database may load the hottest keys start-up.

#### **counter\_cache\_size\_in\_mb**

When no value is set, the database uses the smaller of minimum of 2.5% of Heap or 50 megabytes (MB). If your system performs counter deletes and relies on low **gc\_grace\_seconds**, you should disable the counter cache. To disable, set to 0.

Default: *calculated*

#### **counter\_cache\_save\_period**

The time, in seconds, after which the database saves the counter cache (keys only). The database saves caches to [saved\\_caches\\_directory \(page 71\)](#).

Default: 7200 (2 hours)

#### **counter\_cache\_keys\_to\_save**

Default value: disabled. [note](#) Number of keys from the counter cache to save. When this property is disabled, the database saves all keys.

## Tombstone settings

```
tombstone_warn_threshold: 1000
tombstone_failure_threshold: 100000
```

When executing a scan, within or across a partition, the database must keep tombstones in memory to allow them to return to the coordinator. The coordinator uses tombstones to ensure that other replicas know about the deleted rows. Workloads that generate numerous tombstones may cause performance problems and exhaust the server heap. Adjust these thresholds only if you understand the impact and want to scan more tombstones. You can adjust these thresholds at runtime using the `StorageServiceMBean`.

**Tip:** See [Cassandra anti-patterns: Queues and queue-like datasets](#).

### **tombstone\_warn\_threshold**

The database issues a warning if a query scans more than this number of tombstones.

Default: 1000

### **tombstone\_failure\_threshold**

The database aborts a query if it scans more than this number of tombstones.

Default: 100000

## Network timeout settings

```
read_request_timeout_in_ms: 5000
range_request_timeout_in_ms: 10000
aggregated_request_timeout_in_ms: 120000
write_request_timeout_in_ms: 2000
counter_write_request_timeout_in_ms: 5000
cas_contention_timeout_in_ms: 1000
truncate_request_timeout_in_ms: 60000
request_timeout_in_ms: 10000
cross_dc_rtt_in_ms: 0
```

### **read\_request\_timeout\_in\_ms**

Default: 5000. How long the coordinator waits for read operations to complete before timing it out.

### **range\_request\_timeout\_in\_ms**

Default: 10000. How long the coordinator waits for sequential or index scans to complete before timing it out.

### **aggregated\_request\_timeout\_in\_ms**

How long the coordinator waits for sequential or index scans to complete. Lowest acceptable value is 10 ms. This timeout does not apply to aggregated queries such as `SELECT`, `COUNT(*)`, `MIN(x)`, and so on.

Default: 120000 (2 minutes)

### **write\_request\_timeout\_in\_ms**

How long the coordinator waits for write requests to complete with at least one node in the local datacenter. Lowest acceptable value is 10 ms.

**Tip:** See [Hinted handoff: repair during write path](#).

Default: 2000 (2 seconds)

#### **counter\_write\_request\_timeout\_in\_ms**

How long the coordinator waits for counter writes to complete before timing it out.

Default: 5000 (5 seconds)

#### **cas\_contention\_timeout\_in\_ms**

How long the coordinator continues to retry a CAS (compare and set) operation that contends with other proposals for the same row. If the coordinator cannot complete the operation within this timespan, it aborts the operation.

Default: 1000 (1 second)

#### **truncate\_request\_timeout\_in\_ms**

How long the coordinator waits for a truncate (the removal of all data from a table) to complete before timing it out. The long default value allows the database to take a snapshot before removing the data. If [auto\\_snapshot \(page 86\)](#) is disabled (not recommended), you can reduce this time.

Default: 60000 (1 minute)

#### **request\_timeout\_in\_ms**

The default timeout value for other miscellaneous operations. Lowest acceptable value is 10 ms.

**Tip:** See [Hinted handoff: repair during write path](#).

Default: 10000

#### **cross\_dc\_rtt\_in\_ms**

How much to increase the cross-datacenter timeout

(`write_request_timeout_in_ms + cross_dc_rtt_in_ms`) for requests that involve only nodes in a remote datacenter. This setting is intended to reduce hint pressure.

**Tip:** DataStax recommends using `LOCAL_*` consistency levels (CL) for read and write requests in multi-datacenter deployments to avoid timeouts that may occur when remote nodes are chosen to satisfy the CL, such as QUORUM.

Default: commented out (0)

#### **slow\_query\_log\_timeout\_in\_ms**

Default: 500. How long before a node logs slow queries. Select queries that exceed this value generate an aggregated log message to identify slow queries. To disable, set to 0.

### **Inter-node settings**

```
storage_port: 7000
cross_node_timeout: false
internode_send_buff_size_in_bytes:
internode_recv_buff_size_in_bytes:
internode_compression: dc
inter_dc_tcp_nodelay: false
```

#### **storage\_port**

The port for inter-node communication. Follow security best practices, do not expose this port to the internet. Apply firewall rules.

**Tip:** See [Securing DataStax Enterprise ports](#).

Default: 7000

#### **cross\_node\_timeout**

Whether to enable operation timeout information exchange between nodes to accurately measure request timeouts. If this property is disabled, the replica assumes any requests are forwarded to it instantly by the coordinator. During overload conditions this means extra time is required for processing already-timed-out requests.

**Caution:** Before enabling this property make sure NTP (network time protocol) is installed and the times are synchronized among the nodes.

Default: `false`

#### **internode\_send\_buff\_size\_in\_bytes**

The sending socket buffer size, in bytes, for inter-node calls.

**Tip:** See [TCP settings \(page 63\)](#).

The sending socket buffer size and [internode\\_recv\\_buff\\_size\\_in\\_bytes \(page 90\)](#) is limited by `net.core.wmem_max`. If this property is not set, `net.ipv4.tcp_wmem` determines the buffer size. For more details run `man tcp` and refer to:

- `/proc/sys/net/core/wmem_max`
- `/proc/sys/net/core/rmem_max`
- `/proc/sys/net/ipv4/tcp_wmem`
- `/proc/sys/net/ipv4/tcp_rmem`

Default: not set

#### **internode\_recv\_buff\_size\_in\_bytes**

The receiving socket buffer size in bytes for inter-node calls.

Default: not set

#### **internode\_compression**

Controls whether traffic between nodes is compressed. Valid values:

- `all` - Compresses all traffic
- `dc` - Compresses traffic between datacenters only
- `none` - No compression.

Default: `dc`

#### **inter\_dc\_tcp\_nodelay**

Whether to enable `tcp_nodelay` for inter-datacenter communication. When disabled, the network sends larger, but fewer, network packets. This reduces overhead from the TCP protocol itself. However, disabling `inter_dc_tcp_nodelay` may increase latency by blocking cross datacenter responses.

Default: `false`

## **Native transport (CQL Binary Protocol)**

```
start_native_transport: true
native_transport_port: 9042
native_transport_port_ssl: 9142
```

```
native_transport_max_frame_size_in_mb: 256
native_transport_max_concurrent_connections: -1
native_transport_max_concurrent_connections_per_ip: -1
native_transport_address: localhost
native_transport_interface: eth0
native_transport_interface_prefer_ipv6: false
native_transport_broadcast_address: 1.2.3.4
native_transport_keepalive: true
```

**Tip:** See also [native\\_transport\\_port\\_ssl \(page 102\)](#) in [SSL Ports \(page 102\)](#).

### **start\_native\_transport**

Enables or disables the native transport server.

Default: `true`

### **native\_transport\_port**

The port where the CQL native transport listens for clients. For security reasons, do not expose this port to the internet. Firewall it if needed.

Default: `9042`

### **native\_transport\_max\_frame\_size\_in\_mb**

The maximum allowed size of a frame. Frame (requests) larger than this are rejected as invalid.

Default: `256`

### **native\_transport\_max\_concurrent\_connections**

The maximum number of concurrent client connections.

Default: `-1` (unlimited)

### **native\_transport\_max\_concurrent\_connections\_per\_ip**

The maximum number of concurrent client connections per source IP address.

Default: `-1` (unlimited)

### **native\_transport\_address**

When left blank, uses the configured hostname of the node. Unlike the `listen_address`, this value can be set to `0.0.0.0`, but you must set the `native_transport_broadcast_address` to a value other than `0.0.0.0`.

**Note:** Set `native_transport_address` OR `native_transport_interface`, not both.

Default: `localhost`

### **native\_transport\_interface**

IP aliasing is not supported.

**Note:** Set `native_transport_address` OR `native_transport_interface`, not both.

Default: `eth0`

### **native\_transport\_interface\_prefer\_ipv6**

Use IPv4 or IPv6 when interface is specified by name.

- `false` - use first IPv4 address.
- `true` - use first IPv6 address.

When only a single address is used, that address is selected without regard to this setting.

Default: commented out (`false`)

### **native\_transport\_broadcast\_address**

Native transport address to broadcast to drivers and other DSE nodes. This cannot be set to 0.0.0.0.

- blank - will be set to the value of native\_transport\_address
- *IP\_address* - when native\_transport\_address is set to 0.0.0.0

Default: commented out (1.2.3.4)

#### **native\_transport\_keepalive**

Whether to enable keepalive on native connections.

Default: true

### **Advanced fault detection settings**

Settings to handle poorly performing or failing components.

```
gc_log_threshold_in_ms: 200
gc_warn_threshold_in_ms: 1000
otc_coalescing_strategy: DISABLED
otc_coalescing_window_us: 200
otc_coalescing_enough_coalesced_messages: 8
```

#### **gc\_log\_threshold\_in\_ms**

The threshold for log messages at the INFO level. Adjust to minimize logging.

Default: commented out (200)

#### **gc\_warn\_threshold\_in\_ms**

Threshold for GC pause. Any GC pause longer than this interval is logged at the WARN level. By default, the database logs any GC pause greater than 200 ms at the INFO level.

**Tip:** See [Configuring logging](#).

Default: commented out (1000)

#### **otc\_coalescing\_strategy**

Strategy to use for coalescing messages in OutboundTcpConnection. Supported strategies are: FIXED, MOVINGAVERAGE, TIMEHORIZON, and DISABLED.

Suitable for VMs, but not noticeably performant in other environments. The OutboundTcpConnection (otc) strategy to:

- Increase message throughput (doubling or more).
- Process multiple messages with one trip to read from a socket.
- Perform all the task submission work at the same time.
- Reduce context switching.
- Increase cache friendliness of network message processing.

**Important:** Use only strategy implementations bundled with DSE.

Default: commented out (DISABLED)

#### **otc\_coalescing\_window\_us**

How many microseconds to wait for coalescing. For fixed strategy, the amount of time after the first message is received before it is sent with any accompanying messages. For moving average, this is the maximum wait time and the interval that messages must arrive on average to enable coalescing.

Default: commented out (200)

#### **otc\_coalescing\_enough\_coalesced\_messages**

The threshold for the number of messages. Do not coalesce messages when this value is exceeded. Should be more than 2 and less than 128.

Default: commented out (8)

#### **seed\_gossip\_probability**

The percentage of time that gossip messages are sent to a seed node during each round of gossip. Decreases the time to propagate gossip changes across the cluster.

Default: 1.0 (100%)

### **Backpressure settings**

```
back_pressure_enabled: false
back_pressure_strategy:
 - class_name: org.apache.cassandra.net.RateBasedBackPressure
 parameters:
 - high_ratio: 0.90
 factor: 5
 flow: FAST
```

#### **back\_pressure\_enabled**

Whether to enable for the coordinator to apply the specified back pressure strategy to each mutation that is sent to replicas.

Default: false

#### **back\_pressure\_strategy**

To add new strategies, implement org.apache.cassandra.net.BackpressureStrategy and provide a public constructor that accepts a Map<String, Object>.

**Important:** Use only strategy implementations bundled with DSE.

#### **class\_name**

The default class\_name uses the ratio between incoming mutation responses and outgoing mutation requests.

Default: org.apache.cassandra.net.RateBasedBackPressur

#### **high\_ratio**

When outgoing mutations are below this value, they are rate limited according to the incoming rate decreased by the factor (described below). When above this value, the rate limiting is increased by the factor.

Default: 0.90

#### **factor**

A number between 1 and 10. When backpressure is below high ratio, outgoing mutations are rate limited according to the incoming rate decreased by the given factor; if above high ratio, the rate limiting is increased by the given factor.

Default: 5

#### **flow**

The flow speed to apply rate limiting:

- FAST - rate limited to the speed of the fastest replica
- SLOW - rate limit to the speed of the slowest replica

Default: `FAST`

### **dynamic\_snitch\_badness\_threshold**

The performance threshold for dynamically routing client requests away from a poorly performing node. Specifically, it controls how much worse a poorly performing node has to be before the `dynamic snitch` prefers other replicas. A value of 0.2 means the database continues to prefer the static snitch values until the node response time is 20% worse than the best performing node. Until the threshold is reached, incoming requests are statically routed to the closest replica as determined by the snitch. A value of zero to 1.0 for the `read_repair_chance` table property maximizes cache capacity across the nodes.

Default: `0.1`

### **dynamic\_snitch\_reset\_interval\_in\_ms**

Time interval after which the database resets all node scores. This allows a bad node to recover.

Default: `600000`

### **dynamic\_snitch\_update\_interval\_in\_ms**

The time interval, in milliseconds, between the calculation of node scores. Because score calculation is CPU intensive, be careful when reducing this interval.

Default: `100`

## **Hinted handoff options**

```
hinted_handoff_enabled: true
hinted_handoff_disabled_datacenters:
- DC1
- DC2
max_hint_window_in_ms: 10800000 # 3 hours
hinted_handoff_throttle_in_kb: 1024
max_hints_delivery_threads: 2
hints_directory: /var/lib/cassandra/hints
hints_flush_period_in_ms: 10000
max_hints_file_size_in_mb: 128
#hints_compression:
- class_name: LZ4Compressor
parameters:
-
batchlog_replay_throttle_in_kb: 1024
batchlog_endpoint_strategy: random_remote
```

**Tip:** See [Hinted handoff: repair during write path](#).

### **hinted\_handoff\_enabled**

Enables or disables hinted handoff. A hint indicates that the write needs to be replayed to an unavailable node. The database writes the hint to a hints file on the coordinator node.

- `false` - do not enable hinted handoff
- `true` - globally enable hinted handoff, except for datacenters specified for `hinted_handoff_disabled_datacenters`

Default: `true`

**hinted\_handoff\_disabled\_datacenters**

A blacklist of datacenters that will not perform hinted handoffs. To disable hinted handoff on a certain datacenter, add its name to this list.

Default: commented out

**max\_hint\_window\_in\_ms**

Maximum amount of time during which the database generates hints for an unresponsive node. After this interval, the database does not generate any new hints for the node until it is back up and responsive. If the node goes down again, the database starts a new interval. This setting can prevent a sudden demand for resources when a node is brought back online and the rest of the cluster attempts to replay a large volume of hinted writes.

**Tip:** See [About failure detection and recovery](#).

Default: 10800000 (3 hours)

**hinted\_handoff\_throttle\_in\_kb**

Maximum amount of traffic per delivery thread in kilobytes per second. This rate reduces proportionally to the number of nodes in the cluster. For example, if there are two nodes in the cluster, each delivery thread uses the maximum rate. If there are three, each node throttles to half of the maximum, since the two nodes are expected to deliver hints simultaneously.

**Note:** When applying this limit, the calculated hint transmission rate is based on the uncompressed hint size, even if [internode\\_compression \(page 90\)](#) or [hints\\_compression \(page 95\)](#) is enabled.

Default: 1024

**hints\_flush\_period\_in\_ms**

The time, in milliseconds, to wait before flushing hints from internal buffers to disk.

Default: 10000

**max\_hints\_delivery\_threads**

Number of threads the database uses to deliver hints. In multiple datacenter deployments, consider increasing this number because cross datacenter handoff is generally slower.

Default: 2

**max\_hints\_file\_size\_in\_mb**

The maximum size for a single hints file, in megabytes.

Default: 128

**hints\_compression**

The compressor for hint files. Supported compressors: [LZ](#), [Snappy](#), and [Deflate](#).

When not set, the database does not compress hints files.

Default: [LZ4Compressor](#)

**batchlog\_replay\_throttle\_in\_kb**

Total maximum throttle, in KB per second, for replaying hints. Throttling is reduced proportionally to the number of nodes in the cluster

Default: 1024

**batchlog\_endpoint\_strategy**

Strategy to choose the batchlog storage endpoints.

- random\_remote - Default, purely random. Prevents the local rack, if possible.  
Same behavior as earlier releases.

- `dynamic_remote` - Uses DynamicEndpointSnitch to select batchlog storage endpoints. Prevents the local rack, if possible. This strategy offers the same availability guarantees as `random_remote`, but selects the fastest endpoints according to the DynamicEndpointSnitch. DynamicEndpointSnitch tracks reads but not writes. Write-only, or mostly-write, workloads might not benefit from this strategy. Note: this strategy will fall back to `random_remote` if `dynamic_snitch` is not enabled.
- `dynamic` - Mostly the same as `dynamic_remote`, except that local rack is not excluded, which offers lower availability guarantee than `random_remote` or `dynamic_remote`. Note: this strategy will fall back to `random_remote` if `dynamic_snitch` is not enabled.

Default: `random_remote`

## Security properties

DSE Advanced Security fortifies DataStax Enterprise (DSE) databases against potential harm due to deliberate attack or user error. Configuration properties include authentication and authorization, permissions, roles, encryption of data in-flight and at-rest, and data auditing. [DSE Unified Authentication](#) provides authentication, authorization, and role management. Enabling DSE Unified Authentication requires additional configuration in `dse.yaml`, see [Configuring DSE Unified Authentication](#).

```
authenticator: com.datastax.bdp.cassandra.auth.DseAuthenticator
internode_authenticator:
org.apache.cassandra.auth.AllowAllInternodeAuthenticator
authorizer: com.datastax.bdp.cassandra.auth.DseAuthorizer
role_manager: com.datastax.bdp.cassandra.auth.DseRoleManager
system_keyspaces_filtering: false
roles_validity_in_ms: 120000
roles_update_interval_in_ms: 120000
permissions_validity_in_ms: 120000
permissions_update_interval_in_ms: 120000
```

### authenticator

The authentication backend. The only supported authenticator is `DseAuthenticator` for external authentication with multiple authentication schemes such as Kerberos, LDAP, and internal authentication. Authenticators other than `DseAuthenticator` are deprecated and not supported. Some security features might not work correctly if other authenticators are used.

**Important:** Use only authentication implementations bundled with DSE.

Default: `com.datastax.bdp.cassandra.auth.DseAuthenticator`

### internode\_authenticator

Internode authentication backend to enable secure connections from peer nodes.

**Important:** Use only authentication implementations bundled with DSE.

Default: `org.apache.cassandra.auth.AllowAllInternodeAuthenticator`

### authorizer

The authorization backend. Authorizers other than `DseAuthorizer` are not supported. `DseAuthorizer` supports enhanced permission management of DSE-

specific resources. Authorizers other than DseAuthorizer are deprecated and not supported. Some security features might not work correctly if other authorizers are used.

**Important:** Use only authorization implementations bundled with DSE.

Default: `com.datastax.bdp.cassandra.auth.DseAuthorizer`

#### **system\_keyspaces\_filtering**

Whether to enable system keyspace filtering so that users can access and view only schema information for rows in the system and system\_schema keyspaces to which they have access.

**Attention:** Security requirements and user permissions apply. Enable this feature only after appropriate user permissions are granted.

**Tip:** See [Controlling access to keyspaces and tables](#) and [Configuring the security keyspaces replication factors](#).

Default: `false`

#### **role\_manager**

The DSE Role Manager supports LDAP roles and internal roles supported by the CassandraRoleManager. Role options are stored in the `dse_security` keyspace. When using the DSE Role Manager, increase the replication factor of the `dse_security` keyspace. Role managers other than DseRoleManager are deprecated and not supported. Some security features might not work correctly if other role managers are used.

**Important:** Use only role manager implementations bundled with DSE.

Default: `com.datastax.bdp.cassandra.auth.DseRoleManager`

#### **roles\_validity\_in\_ms**

Validity period for roles cache in milliseconds. Determines how long to cache the list of roles assigned to the user; users may have several roles, either through direct assignment or inheritance (a role that has been granted to another role). Adjust this setting based on the complexity of your role hierarchy, tolerance for role changes, the number of nodes in your environment, and activity level of the cluster.

Fetching permissions can be an expensive operation, so this setting allows flexibility. Granted roles are cached for authenticated sessions in `AuthenticatedUser`. After the specified time elapses, role validity is rechecked. Disabled automatically when internal authentication is not enabled when using `DseAuthenticator`.

- 0 - disable role caching
- *milliseconds* - how long to cache the list of roles assigned to the user

Default: `120000` (2 minutes)

#### **roles\_update\_interval\_in\_ms**

Refresh interval for roles cache. After this interval, cache entries become eligible for refresh. On next access, the database schedules an async reload, and returns the old value until the reload completes. If `roles_validity_in_ms` is non-zero, then this value must also be non-zero. When not set, the default is the same value as `roles_validity_in_ms`.

Default: commented out (120000)

#### **permissions\_validity\_in\_ms**

How long permissions in cache remain valid to manage performance impact of permissions queries. Fetching permissions can be resource intensive. Set the cache validity period to your security tolerances. The cache is used for the standard authentication and the row-level access control (RLAC) cache. The cache is quite effective at small durations.

- 0 - disable permissions cache
- *milliseconds* - time, in milliseconds

**Caution:** [REVOKE](#) does not automatically invalidate cached permissions.

Permissions are invalidated the next time they are refreshed.

Default: 120000 (2 minutes)

#### **permissions\_update\_interval\_in\_ms**

Sets refresh interval for the standard authentication cache and the row-level access control (RLAC) cache. After this interval, cache entries become eligible for refresh. On next access, the database schedules an async reload and returns the old value until the reload completes. If permissions\_validity\_in\_ms is non-zero, the value for roles\_update\_interval\_in\_ms must also be non-zero. When not set, the default is the same value as [permissions\\_validity\\_in\\_ms \(page 98\)](#).

Default: commented out (2000)

#### **permissions\_cache\_max\_entries**

The maximum number of entries that are held by the standard authentication cache and row-level access control (RLAC) cache. With the default value of 1000, the RLAC permissions cache can have up to 1000 entries in it, and the standard authentication cache can have up to 1000 entries. This single option applies to both caches. To size the permissions cache for use with [Setting up Row Level Access Control \(RLAC\)](#), use this formula:

```
numRlacUsers * numRlacTables + 100
```

If this option is not present in `cassandra.yaml`, manually enter it to use a value other than 1000. See [Enabling DSE Unified Authentication](#).

Default: not set (1000)

### **Inter-node encryption options**

Node-to-node (internode) encryption protects data that is transferred between nodes in a cluster using SSL.

```
server_encryption_options:
 internode_encryption: none
 keystore: resources/dse/conf/.keystore
 keystore_password: cassandra
 truststore: resources/dse/conf/.truststore
 truststore_password: cassandra
 # More advanced defaults below:
 # protocol: TLS
 # algorithm: SunX509
 # store_type: JKS
```

```
cipher_suites:
[TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA,TLS_DHE_RSA_WITH_AES_128_CBC_S
require_client_auth: false
require_endpoint_verification: false
```

**server\_encryption\_options**

Inter-node encryption options. If enabled, you must also generate keys and provide the appropriate key and truststore locations and passwords. No custom encryption options are supported.

**Tip:** The passwords used in these options must match the passwords used when generating the keystore and truststore. For instructions on generating these files, see [Creating a Keystore to Use with JSSE](#).

**Tip:** See [Securing internal transactional node connections](#).

**internode\_encryption**

Encryption options for of inter-node communication using the TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA cipher suite for authentication, key exchange, and encryption of data transfers. Use the DHE/ECDHE ciphers, such as TLS\_DHE\_RSA\_WITH\_AES\_128\_CBC\_SHA if running in (Federal Information Processing Standard) FIPS 140 compliant mode.

- all - Encrypt all inter-node communications
- none - No encryption
- dc - Encrypt the traffic between the datacenters (server only)
- rack - Encrypt the traffic between the racks (server only)

Default: none

**keystore**

Relative path from DSE installation directory or absolute path to the Java keystore (JKS) suitable for use with Java Secure Socket Extension (JSSE), which is the Java version of the Secure Sockets Layer (SSL), and Transport Layer Security (TLS) protocols. The keystore contains the private key used to encrypt outgoing messages.

Default: resources/dse/conf/.keystore

**keystore\_password**

Password for the keystore. This must match the password used when generating the keystore and truststore.

Default: cassandra

**truststore**

Relative path from DSE installation directory or absolute path to truststore containing the trusted certificate for authenticating remote servers.

Default: resources/dse/conf/.truststore

**truststore\_password**

Password for the truststore.

Default: cassandra

**protocol**

Default: commented out (TLS)

**algorithm**

Default: commented out (SunX509)

**store\_type**

Default: commented out (JKS)

**cipher\_suites**

Supported ciphers:

- TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA
- TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA
- TLS\_DHE\_RSA\_WITH\_AES\_128\_CBC\_SHA
- TLS\_DHE\_RSA\_WITH\_AES\_256\_CBC\_SHA
- TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA
- TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA

Default: commented out

**require\_client\_auth**

Whether to enable certificate authentication.

Default: commented out (false)

**require\_endpoint\_verification**

Whether to verify the connected host and the host name in the certificate match.

When not set, the default is false.

Default: commented out (false)

## Client-to-node encryption options

Client-to-node encryption protects in-flight data from client machines to a database cluster using SSL (Secure Sockets Layer) and establishes a secure channel between the client and the coordinator node.

```
client_encryption_options:
 enabled: false
 # If enabled and optional is set to true, encrypted and unencrypted
 # connections over native transport are handled.
 optional: false
 keystore: resources/dse/conf/.keystore
 keystore_password: cassandra
 # require_client_auth: false
 # Set truststore and truststore_password if require_client_auth is true
 # truststore: resources/dse/conf/.truststore
 # truststore_password: cassandra
 # More advanced defaults below:
 # protocol: TLS
 # algorithm: SunX509
 # store_type: JKS
 # cipher_suites:
 [TLS_RSA_WITH_AES_128_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA, TLS_DHE_RSA_WITH_AES_128_CBC_SHA]
```

**Tip:** See [Securing client to cluster connections](#).

**client\_encryption\_options**

Whether to enable client-to-node encryption. You must also generate keys and provide the appropriate key and truststore locations and passwords. There are no custom encryption options enabled for DataStax Enterprise.

Advanced settings:

**enabled**

Whether to enable client-to-node encryption.

Default: `false`

**optional**

Whether to allow unsecured connections when client encryption is enabled.

Default: `false`

**keystore**

Relative path from DSE installation directory or absolute path to the Java keystore (JKS) suitable for use with Java Secure Socket Extension (JSSE), which is the Java version of the Secure Sockets Layer (SSL), and Transport Layer Security (TLS) protocols. The keystore contains the private key used to encrypt outgoing messages.

Default: `resources/dse/conf/.keystore`

**keystore\_password**

Password for the keystore.

Default: `cassandra`

**truststore**

Relative path from DSE installation directory or absolute path to truststore containing the trusted certificate for authenticating remote servers. Required if `require_client_auth` is true.

Default: `resources/dse/conf/.truststore`

**truststore\_password**

Password for the truststore. This must match the password used when generating the keystore and truststore. Required if `require_client_auth` is true.

Default: `cassandra`

**protocol**

Default: commented out (`TLS`)

**algorithm**

Default: commented out (`SunX509`)

**store\_type**

Default: commented out (`JKS`)

**cipher\_suites**

Supported ciphers:

- `TLS_RSA_WITH_AES_128_CBC_SHA`
- `TLS_RSA_WITH_AES_256_CBC_SHA`
- `TLS_DHE_RSA_WITH_AES_128_CBC_SHA`
- `TLS_DHE_RSA_WITH_AES_256_CBC_SHA`
- `TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA`
- `TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA`

Default: commented out

## Transparent data encryption options

**transparent\_data\_encryption\_options**

DataStax Enterprise supports this option only for backward compatibility. When using DSE, [configure data encryption options \(page 115\)](#) in the `dse.yaml`; see [Transparent data encryption](#).

TDE properties:

- `enabled`: (Default: false)
- `chunk_length_kb`: (Default: 64)
- `cipher`: options:

```
AES
CBC
PKCS5Padding
```

- `key_alias`: testing:1
- `iv_length`: 16

**Note:** `iv_length` is commented out in the default `cassandra.yaml` file. Uncomment only if cipher is set to AES. The value must be 16 (bytes).

- `key_provider`:

```
class_name: org.apache.cassandra.security.JKSKeyProvider
parameters:
 # keystore: conf/.keystore
 # keystore_password: cassandra
 # store_type: JCEKS
 # key_password: cassandra
```

## SSL Ports

```
ssl_storage_port: 7001
native_transport_port_ssl: 9142
```

**Tip:** See [Securing DataStax Enterprise ports](#).

### `ssl_storage_port`

The SSL port for encrypted communication. Unused unless enabled in `encryption_options`. Follow security best practices, do not expose this port to the internet. Apply firewall rules.

Default: 7001

### `native_transport_port_ssl`

Dedicated SSL port where the CQL native transport listens for clients with encrypted communication. For security reasons, do not expose this port to the internet. Firewall it if needed.

- commented out (disabled) - the `native_transport_port` will encrypt all traffic
- port number different than `native_transport_port` - use encryption for `native_transport_port_ssl`, keep `native_transport_port` unencrypted to use both unencrypted and encrypted traffic

Default: 9142

## Continuous paging options

```
continuous_paging:
 max_concurrent_sessions: 60
 max_session_pages: 4
 max_page_size_mb: 8
 max_local_query_time_ms: 5000
 client_timeout_sec: 600
 cancel_timeout_sec: 5
 paused_check_interval_ms: 1
```

### **continuous\_paging**

Options to tune continuous paging that pushes pages, when requested, continuously to the client:

- Maximum memory used:

```
max_concurrent_sessions # max_session_pages # max_page_size_mb
```

Default: *calculated (60 # 4 # 8 = 1920 MB)*

### **Guidance**

- Because memtables and SSTables are used by the continuous paging query, you can define the maximum period of time during which memtables cannot be flushed and compacted SSTables cannot be deleted.
- If fewer threads exist than sessions, a session cannot execute until another one is swapped out.
- Distributed queries (CL > ONE or non-local data) are swapped out after every page, while local queries at CL = ONE are swapped out after max\_local\_query\_time\_ms.

### **max\_concurrent\_sessions**

The maximum number of concurrent sessions. Additional sessions are rejected with an unavailable error.

Default: 60

### **max\_session\_pages**

The maximum number of pages that can be buffered for each session. If the client is not reading from the socket, the producer thread is blocked after it has prepared max\_session\_pages.

Default: 4

### **max\_page\_size\_mb**

The maximum size of a page, in MB. If an individual CQL row is larger than this value, the page can be larger than this value.

Default: 8

### **max\_local\_query\_time\_ms**

The maximum time for a local continuous query to run. When this threshold is exceeded, the session is swapped out and rescheduled. Swapping and rescheduling ensures the release of resources that prevent the memtables from flushing and ensures fairness when max\_threads < max\_concurrent\_sessions.

Adjust when high write workloads exist on tables that have continuous paging requests.

Default: 5000

#### **client\_timeout\_sec**

How long the server will wait, in seconds, for clients to request more pages if the client is not reading and the server queue is full.

Default: 600

#### **cancel\_timeout\_sec**

How long to wait before checking if a paused session can be resumed. Continuous paging sessions are paused because of backpressure or when the client has not request more pages with backpressure updates.

Default: 5

#### **paused\_check\_interval\_ms**

How long to wait, in milliseconds, before checking if a continuous paging sessions can be resumed, when that session is paused because of backpressure.

Default: 1

### Fault detection setting

```
phi_convict_threshold: 8
```

#### **phi\_convict\_threshold**

The sensitivity of the failure detector on an exponential scale. Generally, this setting does not need adjusting.

**Tip:** See [About failure detection and recovery](#).

When not set, the internal value is 8.

Default: commented out (8)

## dse.yaml configuration file

The DataStax Enterprise configuration file for security, DSE Search, DSE Graph, and DSE Analytics.

The dse.yaml file is the primary configuration file for security, DSE Search, DSE Graph, and DSE Analytics.

**Important:** After changing properties in the `dse.yaml` file, you must restart the node for the changes to take effect.

|                       |                                                                |
|-----------------------|----------------------------------------------------------------|
| Package installations | <code>/etc/dse/dse.yaml</code>                                 |
| Tarball installations | <code>installation_location/resources/dse/conf/dse.yaml</code> |

The [cassandra.yaml \(page 68\)](#) file is the primary configuration file for the DataStax Enterprise database.

### Syntax

For the properties in each section, the parent setting has zero spaces. Each child entry requires at least two spaces. Adhere to the YAML syntax and retain the spacing. For

example, no spaces before the parent `node_health_options` entry, and at least two spaces before the child settings:

```
node_health_options:
 refresh_rate_ms: 50000
 uptime_ramp_up_period_seconds: 10800

 dropped_mutation_window_minutes: 30
```

## Organization

The DataStax Enterprise configuration properties are grouped into the following sections:

- [Security and authentication options \(page 105\)](#)
- [DSE In-Memory \(page 117\)](#)
- [Node health \(page 118\)](#)
- [Health-based routing \(page 118\)](#)
- [Lease metrics \(page 119\)](#)
- [DSE Search options \(page 119\)](#)
- [DSE Analytics options \(page 131\)](#)
- [Performance Service options \(page 123\)](#)
- [Audit logging \(page 140\)](#)
- [audit\\_logging\\_options \(page 140\)](#)
- [DSE Tiered Storage \(page 143\)](#)
- [DSE Advanced Replication \(page 144\)](#)
- [Inter-node messaging \(page 145\)](#)
- [DSE Multi-Instance \(page 146\)](#)
- [DSE Graph options \(page 146\)](#)

## Security and authentication options

- [Authentication options \(page 105\)](#)
- [Role management options \(page 108\)](#)
- [Authorization options \(page 108\)](#)
- [Kerberos options \(page 109\)](#)
- [LDAP options \(page 110\)](#)
- [Encrypt sensitive system resources \(page 114\)](#)
- [Encrypted configuration properties settings \(page 115\)](#)
- [KMIP encryption options \(page 116\)](#)
- [DSE Search index encryption settings \(page 117\)](#)

## Authentication options

Authentication options for the DSE Authenticator that allows you to use multiple schemes for authentication in a DataStax Enterprise cluster. Additional [authenticator \(page 96\)](#) configuration is required in `cassandra.yaml`.

**Note:** Internal and LDAP schemes can also be used for role management, see [role\\_management\\_options \(page 108\)](#).

**Tip:** See [Enabling DSE Unified Authentication](#).

```
authentication_options:
enabled: false
default_scheme: internal
other_schemes:
- ldap
- kerberos
scheme_permissions: false
transitional_mode: disabled
allow_digest_with_kerberos: true
plain_text_without_ssl: warn
```

### **authentication\_options**

Options for the DseAuthenticator to authenticate users when the authenticator option in `cassandra.yaml` is set to `com.datastax.bdp.cassandra.auth.DseAuthenticator`. Authenticators other than DseAuthenticator are not supported.

#### **enabled**

Enables user authentication.

- true - The DseAuthenticator authenticates users.
- false - The DseAuthenticator does not authenticate users and allows all connections.

When not set, the default is false.

Default: commented out `false`

#### **default\_scheme**

Sets the first scheme to validate a user against when the driver does not request a specific scheme.

- internal - Plain text authentication using the internal password authentication.
- ldap - Plain text authentication using pass-through LDAP authentication.
- kerberos - GSSAPI authentication using the Kerberos authenticator.

Default: commented out (`internal`

)

#### **other\_schemes**

List of schemes that are also checked if validation against the first scheme fails and no scheme was specified by the driver. Same scheme names as `default_scheme`.

#### **scheme\_permissions**

Whether roles need to have permission granted to them in order to use specific authentication schemes. These permissions can be granted only when the DseAuthorizer is used. Set to one of the following values:

- true - Use multiple schemes for authentication. Every role requires permissions to a scheme in order to be assigned.

- false - Do not use multiple schemes for authentication. Prevents unintentional role assignment that might occur if user or group names overlap in the authentication service.

**Tip:** See [Binding a role to an authentication scheme](#).

When not set, the default is false.

Default: commented out (`false`)

#### **allow\_digest\_with\_kerberos**

Controls whether DIGEST-MD5 authentication is also allowed with Kerberos. The DIGEST-MD5 mechanism is not directly associated with an authentication scheme, but is used by Kerberos to pass credentials between nodes and jobs.

- true - DIGEST-MD5 authentication is also allowed with Kerberos. In analytics clusters, set to true to use Hadoop inter-node authentication with Hadoop and Spark jobs.
- false - DIGEST-MD5 authentication is not used with Kerberos.

Analytics nodes require true to use internode authentication with Hadoop and Spark jobs. When not set, the default is true.

Default: commented out (`true`)

#### **plain\_text\_without\_ssl**

Controls how the DseAuthenticator responds to plain text authentication requests over unencrypted client connections. Set to one of the following values:

- block - Block the request with an authentication error.
- warn - Log a warning about the request but allow it to continue.
- allow - Allow the request without any warning.

Default: commented out (`warn`)

#### **transitional\_mode**

Whether to enable transitional mode for temporary use during authentication setup in an already established environment.

Transitional mode allows access to the database using the `anonymous` role, which has all permissions except `AUTHORIZE`.

- disabled - Transitional mode is disabled. All connections must provide valid credentials and map to a login-enabled role.
- permissive - Only super users are authenticated and logged in. All other authentication attempts are logged in as the `anonymous` user.
- normal - Allow all connections that provide credentials. Maps all authenticated users to their role AND maps all other connections to `anonymous`.
- strict - Allow only authenticated connections that map to a login-enabled role OR connections that provide a blank username and password as `anonymous`.

**Important:** Credentials are required for all connections after authentication is enabled; use a blank username and password to login with `anonymous` role in transitional mode.

Default: commented out (`disabled`)

## Role management options

```
#role_management_options:
mode: internal
```

**Tip:** See [Enabling DSE Unified Authentication](#).

### role\_management\_options

Options for the DSE Role Manager. To enable role manager, set:

- [authorization\\_options \(page 105\)](#) enabled to true
- [role\\_manager \(page 97\)](#) in `cassandra.yaml` to `com.datastax.bdp.cassandra.auth.DseRoleManager`

**Tip:** See [Setting up logins and users](#).

When [scheme\\_permissions \(page 106\)](#) is enabled, all roles must have permission to execute on the authentication scheme, see [Binding a role to an authentication scheme](#).

### mode

Set to one of the following values:

- internal - Scheme that manages roles per individual user in the internal database. Allows nesting roles for permission management.
- ldap - Scheme that assigns roles by looking up the user name in LDAP and mapping the group attribute ([ldap\\_options \(page 110\)](#)) to an internal role name. To configure an LDAP scheme, complete the steps in [Defining an LDAP scheme](#).

**Attention:** Internal role management allows nesting roles for permission management; when using LDAP mode role, nesting is disabled. Using `GRANT role_name TO role_name` results in an error.

Default: commented out (`internal`)

## Authorization options

```
#authorization_options:
enabled: false
transitional_mode: disabled
allow_row_level_security: false
```

**Tip:** See [Enabling DSE Unified Authentication](#).

### authorization\_options

Options for the DSE Authorizer.

### enabled

Whether to use the DSE Authorizer for role-based access control (RBAC).

- true - use the DSE Authorizer for role-based access control (RBAC)
- false - do not use the Dse Authorizer

When not set, the default is false.

Default: commented out (`false`)

#### **transitional\_mode**

Allows the DSE Authorizer to operate in a temporary transitional mode during setup of authorization in a cluster. Set to one of the following values:

- disabled - Transitional mode is disabled.
- normal - Permissions can be passed to resources, but are not enforced.
- strict - Permissions can be passed to resources, and are enforced on authenticated users. Permissions are not enforced against anonymous users.

Default: commented out (`disabled`)

#### **allow\_row\_level\_security**

Whether to enable row-level access control (RLAC) permissions; use the same setting on all nodes.

- true - use row-level security
- false - do not use row-level

When not set, the default is false.

Default: commented out (`false`)

### **Kerberos options**

```
kerberos_options:
 keytab: resources/dse/conf/dse.keytab
 service_principal: dse/_HOST@REALM
 http_principal: HTTP/_HOST@REALM
 qop: auth
```

**Tip:** See [Defining a Kerberos scheme](#).

#### **kerberos\_options**

Options to configure security for a DataStax Enterprise cluster using Kerberos.

##### **keytab**

The file path of `dse.keytab`.

##### **service\_principal**

The `service_principal` that the DataStax Enterprise process runs under must use the form `dse_user/_HOST@REALM`, where:

- `dse_user` is the name of the user that starts the DataStax Enterprise process.
- `_HOST` is converted to a reverse DNS lookup of the broadcast address.
- `REALM` is the name of your Kerberos realm. In the Kerberos principal, `REALM` must be uppercase.

##### **http\_principal**

The `http_principal` is used by the Tomcat application container to run DSE Search. The Tomcat web server uses the GSSAPI mechanism (SPNEGO) to negotiate the GSSAPI security mechanism (Kerberos). Set `REALM` to the name of your Kerberos realm. In the Kerberos principal, `REALM` must be uppercase.

##### **qop**

A comma-delimited list of Quality of Protection (QOP) values that clients and servers can use for each connection. The client can have multiple QOP values, while the server can have only a single QOP value. The valid values are:

- auth - Authentication only.
- auth-int - Authentication plus integrity protection for all transmitted data.
- auth-conf - Authentication plus integrity protection and encryption of all transmitted data.

Encryption using auth-conf is separate and independent of whether encryption is done using SSL. If both auth-conf and SSL are enabled, the transmitted data is encrypted twice. DataStax recommends choosing only one method and using it for both encryption and authentication.

## LDAP options

Define LDAP options to authenticate users against an external LDAP service and/or for Role Management using LDAP group look up.

**Tip:** See [Enabling DSE Unified Authentication](#).

```
ldap_options:
server_host:
server_port: 389
search_dn:
search_password:
use_ssl: false
use_tls: false
truststore_path:
truststore_password:
truststore_type: jks
user_search_base:
user_search_filter: (uid={0})
user_memberof_attribute: memberof
group_search_type: directory_search
group_search_base:
group_search_filter: (uniqueMember={0})
group_name_attribute: cn
credentials_validity_in_ms: 0
search_validity_in_seconds: 0
connection_pool:
max_active: 8
max_idle: 8
```

Microsoft Active Directory (AD) example, for both authentication and role management:

```
ldap_options:
 server_host: win2012ad_server.mycompany.lan
 server_port: 389
 search_dn:
 cn=lookup_user,cn=users,dc=win2012domain,dc=mycompany,dc=lan
 search_password: lookup_user_password
 use_ssl: false
```

```

use_tls: false
truststore_path:
truststore_password:
truststore_type: jks
#group_search_type: directory_search
group_search_type: memberof_search
#group_search_base:
#group_search_filter:
group_name_attribute: cn
user_search_base: cn=users,dc=win2012domain,dc=mycompany,dc=lan
user_search_filter: (sAMAccountName={0})
user_memberof_attribute: memberOf
connection_pool:
 max_active: 8
 max_idle: 8

```

**Tip:** See [Defining an LDAP scheme](#).

### ldap\_options

Options to configure LDAP security. When not set, LDAP authentication is not used.

Default: commented out

### server\_host

The host name of the LDAP server.

**Important:** LDAP on the same host (localhost) is appropriate only in single node test or development environments.

Default: commented out

### server\_port

The port on which the LDAP server listens.

- 389 - the default port for unencrypted connections
- 636 - typically used for encrypted connections; the default SSL port for LDAP is 636

Default: commented out (389)

### search\_dn

Distinguished name (DN) of an account with read access to the user\_search\_base and group\_search\_base. For example:

- OpenLDAP: uid=lookup,ou=users,dc=springsource,dc=com
- [Microsoft Active Directory \(AD\)](#): cn=lookup, cn=users, dc=springsource, dc=com

**Warning:** Do not create/use an LDAP account or group called `cassandra`.

The DSE database comes with a default login role, `cassandra`, that has access to all database objects and uses the consistency level QUOROM.

When not set, an anonymous bind is used for the search on the LDAP server.

Default: commented out

### search\_password

The password of the `search_dn` account.

Default: commented out

**use\_ssl**

Whether to use an SSL-encrypted connection.

- true - use an SSL-encrypted connection, set [server\\_port \(page 111\)](#) to the LDAP port for the server (typically port 636)
- false - do not enable SSL connections to the LDAP server

Default: commented out (`false`)

**use\_tls**

Whether to enable TLS connections to the LDAP server.

- true - enable TLS connections to the LDAP server, set [server\\_port \(page 111\)](#) to the TLS port of the LDAP server.
- false - do not enable TLS connections to the LDAP server

Default: commented out (`false`)

**truststore\_path**

The path to the truststore for SSL certificates.

Default: commented out

**truststore\_password**

The password to access the trust store.

Default: commented out

**truststore\_type**

The type of truststore.

Default: commented out (`jks`)

**user\_search\_base**

Distinguished name (DN) of the object to start the recursive search for user entries for authentication and role management memberof searches. For example to search all users in example.com, `ou=users,dc=example,dc=com`.

- For your LDAP domain, set the `ou` and `dc` elements. Typically set to `ou=users,dc=domain,dc=top_level_domain`. For example, `ou=users,dc=example,dc=com`.
- Active Directory uses a different search base, typically `CN=search,CN=Users,DC=ActDir_domname,DC=internal`. For example, `CN=search,CN=Users,DC=example-sales,DC=internal`.

Default: commented out

**user\_search\_filter**

Attribute that identifies the user that the search filter uses for looking up user names.

- `uid={0}` - when using LDAP
- `samAccountName={0}` - when using AD (Microsoft Active Directory). For example, `(sAMAccountName={0})`

Default: commented out (`uid={0}`)

**user\_memberof\_attribute**

Attribute that contains a list of group names; role manager assigns DSE roles that exactly match any group name in the list. Required when managing roles using

`group_search_type: memberof_search` with LDAP (`role_manager.mode:ldap (page 108)`). The directory server must have memberof support, which is a default user attribute in Microsoft Active Directory (AD).

Default: commented out (`memberof`)

#### **group\_search\_type**

Required when managing roles with LDAP (`role_manager.mode: ldap (page 108)`). Define how group membership is determined for a user. Choose from one of the following values:

- `directory_search` - Filters the results by doing a subtree search of `group_search_base (page 113)` to find groups that contain the user name in the attribute defined in the `group_search_filter (page 113)`. (Default)
- `memberof_search` - Recursively search for user entries using the `user_search_base` and `user_search_filter`. Get groups from the user attribute defined in `user_memberof_attribute`. The directory server must have memberof support.

Default: commented out (`directory_search`)

#### **group\_search\_base**

The unique distinguished name (DN) of the group record from which to start the group membership search on.

Default: commented out

#### **group\_search\_filter**

Set to any valid LDAP filter.

Default: commented out (`uniqueMember={0}`)

#### **group\_name\_attribute**

The attribute in the group record that contains the LDAP group name. Role names are case-sensitive and must match exactly on DSE for assignment. Unmatched groups are ignored.

Default: commented out (`cn`)

#### **credentials\_validity\_in\_ms**

The duration period of the credentials cache.

- 0 - disable credentials cache
- duration period in milliseconds - enable a search cache and improve performance by reducing the number of requests that are sent to the LDAP server

When not set, the default is 0 (disabled).

Default: commented out (0)

#### **search\_validity\_in\_seconds**

The duration period for the search cache.

- 0 - disable search credentials cache
- duration period in seconds - enables a search cache and improves performance by reducing the number of requests that are sent to the LDAP server

Default: commented out (0, disabled)

#### **connection\_pool**

The configuration settings for the connection pool for making LDAP requests.

#### **max\_active**

The maximum number of active connections to the LDAP server.

Default: commented out (8)

#### **max\_idle**

The maximum number of idle connections in the pool awaiting requests.

Default: commented out (8)

## **Encrypt sensitive system resources**

Options to encrypt sensitive system resources using a local encryption key or a remote KMIP key.

```
system_info_encryption:
 enabled: false
 cipher_algorithm: AES
 secret_key_strength: 128
 chunk_length_kb: 64
 key_provider: KmipKeyProviderFactory
 kmip_host: kmip_host_name
```

**Note:** DataStax recommends using a remote encryption key from a KMIP provider when using Transparent Data Encryption (TDE) features. Use a local encryption key only if a KMIP server is not available.

#### **system\_info\_encryption**

Options to set encryption settings for system resources that might contain sensitive information, including the `system.batchlog` and `system.paxos` tables, hint files, and the database commit log.

##### **enabled**

Whether to enable encryption of system resources. See [Encrypting system resources](#).

**Note:** The `system_trace` keyspace is NOT encrypted by enabling the `system_information_encryption` section. In environments that also have tracing enabled, manually configure encryption with compression on the `system_trace` keyspace. See [Transparent data encryption](#).

Default: `false`

##### **cipher\_algorithm**

The name of the JCE cipher algorithm used to encrypt system resources.

**Table 1: Supported cipher algorithms names**

| <b>cipher_algorithm</b> | <b>secret_key_strength</b> |
|-------------------------|----------------------------|
| AES                     | 128, 192, or 256           |
| DES                     | 56                         |
| DESede                  | 112 or 168                 |
| Blowfish                | 32-448                     |

| cipher_algorithm | secret_key_strength |
|------------------|---------------------|
| RC2              | 40-128              |

Default: AES

#### secret\_key\_strength

Length of key to use for the system resources. See [Table 1 \(page 114\)](#).

**Note:** DSE uses a matching local key or requests the key type from the KMIP server. For KMIP, if an existing key does not match, the KMIP server automatically generates a new key.

Default: 128

#### chunk\_length\_kb

Optional. Size of SSTable chunks when data from the system.batchlog or system.paxos are written to disk.

**Note:** To encrypt existing data, run `nodetool upgradesstables -a system batchlog paxos` on all nodes in the cluster.

Default: 64

#### key\_provider

KMIP key provider to enable encrypting sensitive system data with a KMIP key. Comment out if using a local encryption key.

Default: commented out (`KmipKeyProviderFactory`)

#### kmip\_host

The KMIP key server host. Set to the `kmip_group_name` that defines the KMIP host in [kmip\\_hosts \(page 116\)](#) section. DSE requests a key from the KMIP host and uses the key generated by the KMIP provider.

Default: commented out

### Encrypted configuration properties settings

Settings for using encrypted passwords in sensitive configuration file properties.

```
system_key_directory: /etc/dse/conf
config_encryption_active: false
config_encryption_key_name: (key_filename | KMIP_key_URL)
```

#### system\_key\_directory

Path to the directory where local encryption/decryption key files are stored, also called system keys. Distribute the system keys to all nodes in the cluster. Ensure that the DSE account is the folder owner and has read/write (600) permissions.

See [Setting up local encryption keys](#).

**Note:** This directory is not used for KMIP keys.

Default: `/etc/dse/conf`

#### config\_encryption\_active

Whether to enable decryption of configuration property values using the specified [config\\_encryption\\_key\\_name \(page 116\)](#). When set to true, the configuration

values must be encrypted or commented out. See [Encrypting configuration file properties](#)

Default: `false`

#### **config\_encryption\_key\_name**

Set to the local encryption key filename or KMIP key URL to use for configuration file property value decryption.

**Note:** Use `dsetool dsetool encryptconfigvalue (page 816)` to generate encrypted values for the configuration file properties.

Default: `system_key`. The default name is not configurable.

### **KMIP encryption options**

Options for KMIP encryption keys and communication between the DataStax Enterprise node and the KMIP key server or key servers. Enables DataStax Enterprise encryption features to use encryption keys that stored on a server that is not running DataStax Enterprise.

```
kmip_hosts:
 your_kmip_groupname:
 hosts: kmip1.yourdomain.com, kmip2.yourdomain.com
 keystore_path: path/to/kmip/keystore.jks
 keystore_type: jks
 keystore_password: password
 truststore_path: path/to/kmip/truststore.jks
 truststore_type: jks
 truststore_password: password
```

#### **kmip\_hosts**

Connection settings for key servers that support the KMIP protocol.

#### **kmip\_groupname**

The unique name of the KMIP host/cluster that is specified in the table schema. A user-defined name for a group of options to configure a KMIP server or servers, key settings, and certificates. Configure options for a `kmip_groupname` section for each KMIP key server or group of KMIP key servers. Using separate key server configuration settings allows use of different key servers to encrypt table data, and eliminates the need to enter key server configuration information in DDL statements and other configurations. Multiple KMIP hosts are supported.

Default: commented out

#### **hosts**

A comma-separated list KMIP hosts (`host[:port]`) using the FQDN (Fully Qualified Domain Name). DSE queries the host in the listed order, so add KMIP hosts in the intended failover sequence.

For example, if the host list contains `kmip1.yourdomain.com`, `kmip2.yourdomain.com`, DSE tries `kmip1.yourdomain.com` and then `kmip2.yourdomain.com`.

#### **keystore\_path**

The path to a Java keystore created from the KMIP agent PEM files.

Default: commented out (`/etc/dse/conf/KMIP_keystore.jks`)

#### **keystore\_type**

The type of keystore.  
 Default: commented out (`jks`)

**keystore\_password**  
 The password to access the keystore.  
 Default: commented out (`password`)

**truststore\_path**  
 The path to a Java truststore that was created using the KMIP root certificate.  
 Default: commented out (`/etc/dse/conf/KMIP_truststore.jks`)

**truststore\_type**  
 The type of truststore.  
 Default: commented out (`jks`)

**truststore\_password**  
 The password to access the truststore.  
 Default: commented out (`password`)

**key\_cache\_millis**  
 Milliseconds to locally cache the encryption keys that are read from the KMIP hosts.  
 The longer the encryption keys are cached, the fewer requests are made to the KMIP key server, but the longer it takes for changes, like revocation, to propagate to the DataStax Enterprise node. DataStax Enterprise uses concurrent encryption, so multiple threads fetch the secret key from the KMIP key server at the same time. DataStax recommends using the default value.  
 Default: commented out (`300000`)

**timeout**  
 Socket timeout in milliseconds.  
 Default: commented out (`1000`)

## DSE Search index encryption settings

```
solr_encryption_options:
decryption_cache_offheap_allocation: true
decryption_cache_size_in_mb: 256
```

### **solr\_encryption\_options**

Settings to tune encryption of search indexes.

#### **decryption\_cache\_offheap\_allocation**

Whether to allocate shared DSE Search decryption cache off JVM heap.

- `true` - allocate shared DSE Search decryption cache off JVM heap
- `false` - do not allocate shared DSE Search decryption cache off JVM heap

When not set, the default is `true`.

Default: commented out (`true`)

#### **decryption\_cache\_size\_in\_mb**

The maximum size of shared DSE Search decryption cache in megabytes (MB).

Default: commented out (`256`)

## DSE In-Memory options

To use the [DSE In-Memory](#), choose one of these options to specify how much system memory to use for all in-memory tables: fraction or size.

```
max_memory_to_lock_fraction: 0.20
```

```
max_memory_to_lock_mb: 10240
```

### **max\_memory\_to\_lock\_fraction**

A fraction of the system memory. The default value of 0.20 specifies to use up to 20% of system memory. This `max_memory_to_lock_fraction` value is ignored if `max_memory_to_lock_mb` is set to a non-zero value. To specify a fraction, use instead of `max_memory_to_lock_mb`.

Default: commented out (0.20)

### **max\_memory\_to\_lock\_mb**

A maximum amount of memory in megabytes (MB).

- not set - use the fraction specified with `max_memory_to_lock_fraction`
- number greater than 0 - maximum amount of memory in megabytes (MB)

Default: commented out (10240)

## **Node health options**

```
node_health_options:

 refresh_rate_ms: 50000
 uptime_ramp_up_period_seconds: 10800

 dropped_mutation_window_minutes: 30
```

### **node\_health\_options**

Node health options are always enabled.

#### **refresh\_rate\_ms**

Default: 60000

#### **uptime\_ramp\_up\_period\_seconds**

The amount of continuous uptime required for the node's uptime score to advance the [node health score](#) from 0 to 1 (full health), assuming there are no recent dropped mutations. The health score is a composite score based on dropped mutations and uptime.

**Tip:** If a node is repairing after a period of downtime, you might want to increase the uptime period to the expected repair time.

Default: commented out (10800 3 hours)

#### **dropped\_mutation\_window\_minutes**

The historic time window over which the rate of dropped mutations affect the node health score.

Default: 30

## **Health-based routing**

```
enable_health_based_routing: true
```

### **enable\_health\_based\_routing**

Whether to consider node health for replication selection for distributed DSE Search queries. Health-based routing enables a trade-off between index consistency and query throughput.

- true - consider node health when multiple candidates exist for a particular token range.
- false - ignore node health for replication selection. When the primary concern is performance, do not enable health-based routing.

Default: true

## Lease metrics

```
lease_metrics_options:
 enabled:false
 ttl_seconds: 604800
```

### lease\_metrics\_options

Lease holder statistics help monitor the lease subsystem for [automatic management \(page 217\)](#) of Job Tracker and Spark Master nodes.

#### enabled

Enables (true) or disables (false) log entries related to lease holders. Most of the time you do not want to enable logging.

Default: false

#### ttl\_seconds

Defines the time, in milliseconds, to persist the log of lease holder changes. Logging of lease holder changes is always on, and has a very low overhead.

Default: 604800

## DSE Search options

- [Scheduler settings for DSE Search indexes \(page 119\)](#)
- [async\\_bootstrap\\_reindex \(page 120\)](#)
- [CQL Solr paging \(page 120\)](#)
- [Solr CQL query option \(page 121\)](#)
- [DSE Search resource upload limit \(page 121\)](#)
- [Shard transport options \(page 121\)](#)
- [DSE Search indexing settings \(page 121\)](#)

### Scheduler settings for DSE Search indexes

To ensure that records with TTLs are purged from search indexes when they expire, the search indexes are periodically checked for expired documents.

```
ttl_index_rebuild_options:
 fixed_rate_period: 300
 initial_delay: 20
 max_docs_per_batch: 4096
 thread_pool_size: 1
```

### ttl\_index\_rebuild\_options

Section of options to control the schedulers in charge of querying for and removing expired records, and the execution of the checks.

#### fix\_rate\_period

Time interval to check for expired data in seconds.

Default: 300

#### **initial\_delay**

The number of seconds to delay the first TTL check to speed up start-up time.

Default: 20

#### **max\_docs\_per\_batch**

The maximum number of documents to check and delete per batch by the TTL rebuild thread. All documents determined to be expired are deleted from the index during each check, to avoid memory pressure, their unique keys are retrieved and deletes issued in batches.

Default: 4096

#### **thread\_pool\_size**

The maximum number of cores that can execute TTL cleanup concurrently. Set the `thread_pool_size` to manage system resource consumption and prevent many search cores from executing simultaneous TTL deletes.

Default: 1

## **Reindexing of bootstrapped data**

```
async_bootstrap_reindex: false
```

#### **async\_bootstrap\_reindex**

For DSE Search, configure whether to asynchronously reindex bootstrapped data.

Default: false

- If enabled, the node joins the ring immediately after bootstrap and reindexing occurs asynchronously. Do not wait for post-bootstrap reindexing so that the node is not marked down.
- If disabled, the node joins the ring after reindexing the bootstrapped data.

## **CQL Solr paging**

Options to specify the paging behavior.

```
cql_solr_query_paging: off
```

#### **cql\_solr\_query\_paging**

- driver - Respects driver paging settings. Specifies to use [Solr pagination \(cursors\)](#) only when the driver uses pagination. Enabled automatically for DSE SearchAnalytics workloads.
- off - Paging is off. Ignore driver paging settings for [CQL queries](#) and use normal Solr paging unless:
  - # The current workload is an analytics workload, including SearchAnalytics. SearchAnalytics nodes always use driver paging settings.
  - # The cqlsh query parameter paging is set to driver.  
Even when `cql_solr_query_paging: off`, paging is dynamically enabled with the `"paging": "driver"` parameter in [JSON queries](#).

When not set, the default is off.

Default: commented out (`off`)

## Solr CQL query option

Available option for CQL Solr queries.

```
cql_solr_query_row_timeout: 10000
```

### **cql\_solr\_query\_row\_timeout**

The maximum time in milliseconds to wait for each row to be read from the database during CQL Solr queries.

Default: commented out (10000 10 seconds)

## DSE Search resource upload limit

```
solr_resource_upload_limit_mb: 10
```

### **solr\_resource\_upload\_limit\_mb**

Option to disable or configure the maximum file size of the search index config or schema. Resource files can be uploaded, but the search index config and schema are stored internally in the database after upload.

- 0 - disable resource uploading
- upload size - The maximum upload size limit in megabytes (MB) for a DSE Search resource file (search index config or schema).

Default: 10

## Shard transport options

```
shard_transport_options:
 netty_client_request_timeout: 60000
```

### **shard\_transport\_options**

Fault tolerance option for inter-node communication between DSE Search nodes.

### **netty\_client\_request\_timeout**

Timeout behavior during distributed queries. The internal timeout for all search queries to prevent long running queries. The client request timeout is the maximum cumulative time (in milliseconds) that a distributed search request will wait idly for shard responses.

Default: 60000 (1 minute)

## DSE Search indexing settings

```
back_pressure_threshold_per_core: 1024
flush_max_time_per_core: 5
load_max_time_per_core: 5
enable_index_disk_failure_policy: false
solr_data_dir: /MyDir
solr_field_cache_enabled: false
ram_buffer_heap_space_in_mb: 1024
ram_buffer_offheap_space_in_mb: 1024
```

**Tip:** See [Configuring and tuning indexing performance](#).

**back\_pressure\_threshold\_per\_core**

The maximum number of queued partitions during search index rebuilding and reindexing. This maximum number safeguards against excessive heap use by the indexing queue. If set lower than the number of threads per core (TPC), not all TPC threads can be actively indexing.

Default: commented out (1024)

**flush\_max\_time\_per\_core**

The maximum time, in minutes, to wait for the flushing of asynchronous index updates that occurs at DSE Search commit time or at flush time. Expert level knowledge is required to change this value. Always set the value reasonably high to ensure flushing completes successfully to fully sync DSE Search indexes with the database data. If the configured value is exceeded, index updates are only partially committed and the commit log is not truncated which can undermine data durability.

**Note:** When a timeout occurs, it usually means this node is being overloaded and cannot flush in a timely manner. Live indexing increases the time to flush asynchronous index updates.

Default: commented out (5)

**load\_max\_time\_per\_core**

The maximum time, in minutes, to wait for each DSE Search index to load on startup or create/reload operations. This advanced option should be changed only if exceptions happen during search index loading. When not set, the default is 5 minutes.

Default: commented out (5)

**enable\_index\_disk\_failure\_policy**

Whether to apply the configured disk failure policy if IOExceptions occur during index update operations.

- true - apply the configured Cassandra disk failure policy to index write failures
- false - do not apply the disk failure policy

When not set, the default is false.

Default: commented out (false)

**solr\_data\_dir**

The directory to store index data. By default, each DSE Search index is saved in `solrconfig_data_dir/keyspace_name.table_name`, or as specified by the `dse.solr.data.dir` system property.

**Tip:** See [Managing the location of DSE Search data](#).

Default: commented out (/MyDir)

**solr\_field\_cache\_enabled**

The Apache Lucene® field cache is deprecated. Instead, for fields that are sorted, faceted, or grouped by, set `docValues="true"` on the field in the search index schema. Then reload the search index and reindex. When not set, the default is false.

Default: commented out (false)

**ram\_buffer\_heap\_space\_in\_mb**

Global Lucene RAM buffer usage threshold for heap to force segment flush. Setting too low might induce a state of constant flushing during periods of ongoing write

activity. For NRT, forced segment flushes also de-schedule pending auto-soft commits to avoid potentially flushing too many small segments. When not set, the default is 1024.

Default: commented out (1024)

#### **ram\_buffer\_offheap\_space\_in\_mb**

Global Lucene RAM buffer usage threshold for offheap to force segment flush.

Setting too low might induce a state of constant flushing during periods of ongoing write activity. For NRT, forced segment flushes also de-schedule pending auto-soft commits to avoid potentially flushing too many small segments. When not set, the default is 1024.

Default: commented out (1024)

### **Performance Service options**

- [Global Performance Service options \(page 123\)](#)
- [Performance Service options \(page 123\)](#)
- [DSE Search Performance Service options \(page 127\)](#)
- [Spark Performance Service options \(page 130\)](#)

#### **Global Performance Service options**

Available options to configure the thread pool that is used by most plug-ins. A dropped task warning is issued when the performance service requests more tasks than `performance_max_threads + performance_queue_capacity`. When a task is dropped, collected statistics might not be current.

```
performance_core_threads: 4
performance_max_threads: 32
performance_queue_capacity: 32000
```

##### **performance\_core\_threads**

Number of background threads used by the performance service under normal conditions. Default: 4

##### **performance\_max\_threads**

Maximum number of background threads used by the performance service.

##### **performance\_queue\_capacity**

The number of queued tasks in the backlog when the number of `performance_max_threads` are busy. Default: 32000

### **Performance Service options**

These settings are used by the Performance Service to configure collection of performance metrics on transactional nodes. Performance metrics are stored in the `dse_perf` keyspace and can be queried with CQL using any CQL-based utility, such as `cqlsh` or any application using a CQL driver. To temporarily make changes for diagnostics and testing, use the [dsetool perf \(page 840\)](#) subcommands.

**Tip:** See [Collecting system level diagnostics](#).

#### **graph\_events**

Graph event information.

```
graph_events:
 ttl_seconds: 600
```

**ttl\_seconds**

The TTL in milliseconds.

Default: 600

**cql\_slow\_log\_options**

Options to configure reporting distributed sub-queries for search (query executions on individual shards) that take longer than a specified period of time.

```
cql_slow_log_options:
enabled: true
threshold: 200.0
minimum_samples: 100
ttl_seconds: 259200
skip_writing_to_db: true
num_slowest_queries: 5
```

**Tip:** See [Collecting slow queries](#).

**enabled**

Enables (true) or disables (false) log entries for slow queries. When not set, the default is true.

Default: commented out (`true`)

**threshold**

The threshold in milliseconds or as a percentile.

- A value greater than 1 is expressed in time and will log queries that take longer than the specified number of milliseconds.
- A value of 0 to 1 is expressed as a percentile and will log queries that exceed this percentile.

Default: commented out (`200.0` 0.2 seconds)

**minimum\_samples**

The initial number of queries before activating the percentile filter.

Default: commented out (`100`)

**ttl\_seconds**

Time, in milliseconds, to keep the slow query log entries.

Default: commented out (`259200`)

**skip\_writing\_to\_db**

Whether to keep slow queries in-memory only and not write data to database.

- false - write slow queries to the database; the threshold must be  $\geq 2000$  ms to prevent a high load on the database
- true - skip writing to database, keep slow queries only in memory

Default: commented out (`true`)

**num\_slowest\_queries**

The number of slow queries to keep in-memory.

Default: commented out (`5`)

**cql\_system\_info\_options**

Options to configure collection of system-wide performance information about a cluster.

```
cql_system_info_options:
 enabled: false
 refresh_rate_ms: 10000
```

**enabled**

Whether to collect system-wide performance information about a cluster.

- false - do not collect metrics
- true - enable collection of metrics

Default: false

**refresh\_rate\_ms**

The length of the sampling period in milliseconds; the frequency to update the statistics.

Default: 10000 (10 seconds)

**resource\_level\_latency\_tracking\_options**

Options to configure collection of object I/O performance statistics.

```
resource_level_latency_tracking_options:
 enabled: false
 refresh_rate_ms: 10000
```

**Tip:** See [Collecting system level diagnostics](#).

**enabled**

Whether to collect object I/O performance statistics.

- false - do not collect metrics
- true - enable collection of metrics

Default: false

**refresh\_rate\_ms**

The length of the sampling period in milliseconds; the frequency to update the statistics.

Default: 10000 (10 seconds)

**db\_summary\_stats\_options**

Options to configure collection of summary statistics at the database level.

```
db_summary_stats_options:
 enabled: false
 refresh_rate_ms: 10000
```

**Tip:** See [Collecting database summary diagnostics](#).

**enabled**

Whether to collect database summary performance information.

- false - do not collect metrics

- true - enable collection of metrics

Default: false

#### refresh\_rate\_ms

The length of the sampling period in milliseconds; the frequency to update the statistics.

Default: 10000 (10 seconds)

#### cluster\_summary\_stats\_options

Options to configure collection of statistics at a cluster-wide level.

```
cluster_summary_stats_options:
 enabled: false
 refresh_rate_ms: 10000
```

**Tip:** See [Collecting cluster summary diagnostics](#).

#### enabled

Whether to collect statistics at a cluster-wide level.

- false - do not collect metrics
- true - enable collection of metrics

Default: false

#### refresh\_rate\_ms

The length of the sampling period in milliseconds; the frequency to update the statistics.

Default: 10000 (10 seconds)

#### spark\_cluster\_info\_options

Options to configure collection of data associated with Spark cluster and Spark applications.

```
spark_cluster_info_options:
 enabled: false
 refresh_rate_ms: 10000
```

**Tip:** See [Monitoring Spark with Spark Performance Objects](#).

#### enabled

Whether to collect Spark performance statistics.

- false - do not collect metrics
- true - enable collection of metrics

Default: false

#### refresh\_rate\_ms

The length of the sampling period in milliseconds; the frequency to update the statistics.

Default: 10000 (10 seconds)

#### histogram\_data\_options

Histogram data for the dropped mutation metrics are stored in the dropped\_messages table in the dse\_perf keyspace.

```
histogram_data_options:
```

```

 enabled: false
 refresh_rate_ms: 10000
 retention_count: 3

```

**Tip:** See [Collecting histogram diagnostics](#).

#### **enabled**

- false - do not collect metrics
- true - enable collection of metrics

Default: false

#### **refresh\_rate\_ms**

The length of the sampling period in milliseconds; the frequency to update the statistics.

Default: 10000 (10 seconds)

#### **retention\_count**

Default: 3

#### **user\_level\_latency\_tracking\_options**

User-resource latency tracking settings.

```

user_level_latency_tracking_options:
 enabled: false
 refresh_rate_ms: 10000
 top_stats_limit: 100
 quantiles: false

```

**Tip:** See [Collecting user activity diagnostics](#).

#### **enabled**

- false - do not collect metrics
- true - enable collection of metrics

Default: false

#### **refresh\_rate\_ms**

The length of the sampling period in milliseconds; the frequency to update the statistics.

Default: 10000 (10 seconds)

#### **top\_stats\_limit**

Limit the number of individual metrics.

Default: 100

#### **quantiles**

Default: false

### **DSE Search Performance Service options**

These settings are used by the [DataStax Enterprise Performance Service](#).

```

solr_slow_sub_query_log_options:
 enabled: false
 ttl_seconds: 604800
 threshold_ms: 3000

```

```
async_writers: 1

solr_update_handler_metrics_options:
 enabled: false
 ttl_seconds: 604800
 refresh_rate_ms: 60000

solr_request_handler_metrics_options:
 enabled: false
 ttl_seconds: 604800
 refresh_rate_ms: 60000

solr_index_stats_options:
 enabled: false
 ttl_seconds: 604800
 refresh_rate_ms: 60000

solr_cache_stats_options:
 enabled: false
 ttl_seconds: 604800
 refresh_rate_ms: 60000

solr_latency_snapshot_options:
 enabled: false
 ttl_seconds: 604800
 refresh_rate_ms: 60000
```

### **solr\_slow\_sub\_query\_log\_options**

See [Collecting slow search queries](#).

#### **enabled**

- false - do not collect metrics
- true - enable collection of metrics

Default: false

#### **ttl\_seconds**

The length of the sampling period in milliseconds; the frequency to update the statistics.

Default: 604800 (about 10 minutes)

#### **async\_writers**

The number of server threads dedicated to writing in the log. More than one server thread might degrade performance.

Default: 1

#### **threshold\_ms**

Default: 3000

### **solr\_update\_handler\_metrics\_options**

Options to collect search index direct update handler statistics over time.

**Tip:** See [Collecting handler statistics](#).

#### **enabled**

- false - do not collect metrics
- true - enable collection of metrics

Default: `false`

#### **ttl\_seconds**

The length of the sampling period in milliseconds; the frequency to update the statistics.

Default: 604800 (about 10 minutes)

#### **refresh\_rate\_ms**

The length of the sampling period in milliseconds; the frequency to update the statistics.

Default: 60000 (1 minute)

#### **solr\_index\_stats\_options**

Options to record search index statistics over time.

**Tip:** See [Collecting index statistics](#).

#### **enabled**

- false - do not collect metrics
- true - enable collection of metrics

Default: `false`

#### **ttl\_seconds**

The length of the sampling period in milliseconds; the frequency to update the statistics.

Default: 604800 (about 10 minutes)

#### **refresh\_rate\_ms**

The length of the sampling period in milliseconds; the frequency to update the statistics.

Default: 60000 (1 minute)

#### **solr\_cache\_stats\_options**

See [Collecting cache statistics](#).

#### **enabled**

- false - do not collect metrics
- true - enable collection of metrics

Default: `false`

#### **ttl\_seconds**

The length of the sampling period in milliseconds; the frequency to update the statistics.

Default: 604800 (about 10 minutes)

#### **refresh\_rate\_ms**

The length of the sampling period in milliseconds; the frequency to update the statistics.

Default: 60000 (1 minute)

#### **solr\_latency\_snapshot\_options**

See [Collecting Apache Solr performance statistics](#).

#### **enabled**

- false - do not collect metrics
- true - enable collection of metrics

Default: `false`

#### **ttl\_seconds**

The length of the sampling period in milliseconds; the frequency to update the statistics.

Default: 604800 (about 10 minutes)

#### **refresh\_rate\_ms**

The length of the sampling period in milliseconds; the frequency to update the statistics.

Default: 60000 (1 minute)

### **Spark Performance Service options**

**Tip:** See [Monitoring Spark application information](#).

```
spark_application_info_options:
 enabled: false
 refresh_rate_ms: 10000
 driver:
 sink: false
 connectorSource: false
 jvmSource: false
 stateSource: false
 executor:
 sink: false
 connectorSource: false
 jvmSource: false
```

#### **spark\_application\_info\_options**

Statistics options.

##### **enabled**

- false - do not collect metrics
- true - enable collection of metrics

Default: `false`

##### **refresh\_rate\_ms**

The length of the sampling period in milliseconds; the frequency to update the statistics.

Default: 10000 (10 seconds)

##### **driver**

Options to configure collection of metrics at the Spark Driver.

##### **connectorSource**

Whether to collect Spark Cassandra Connector metrics at the Spark Driver.

- false - do not collect metrics
- true - enable collection of metrics

Default: `false`

##### **jvmSource**

Whether to collect JVM heap and garbage collection (GC) metrics from the Spark Driver.

- false - do not collect metrics
- true - enable collection of metrics

Default: `false`

#### **stateSource**

Whether to collect application state metrics at the Spark Driver.

- false - do not collect metrics
- true - enable collection of metrics

Default: `false`

#### **executor**

Options to configure collection of metrics at Spark executors.

#### **sink**

Whether to write metrics collected at Spark executors.

- false - do not collect metrics
- true - enable collection of metrics

Default: `false`

#### **connectorSource**

Whether to collect Spark Cassandra Connector metrics at Spark executors.

- false - do not collect metrics
- true - enable collection of metrics

Default: `false`

#### **jvmSource**

Whether to collect JVM heap and GC metrics at Spark executors.

- false - do not collect metrics
- true - enable collection of metrics

Default: `false`

### **DSE Analytics options**

- [Spark \(page 131\)](#)
- [Starting Spark drivers and executors \(page 135\)](#)
- [DSE File System \(DSEFS\) options \(page 136\)](#)
- [Spark Performance Service \(page 130\)](#)

### **Spark resource and encryption options**

```
spark_shared_secret_bit_length: 256
spark_security_enabled: false
spark_security_encryption_enabled: false

spark_daemon_readiness_assertion_interval: 1000

resource_manager_options:
 worker_options:
```

```

cores_total: 0.7
memory_total: 0.6

workpools:
 - name: alwayson_sql
 cores: 0.25
 memory: 0.25

spark_ui_options:
 encryption: inherit
 encryption_options:
 enabled: false
 keystore: .keystore
 keystore_password: cassandra
 require_client_auth: false
 truststore: .truststore
 truststore_password: cassandra
 # Advanced settings
 # protocol: TLS
 # algorithm: SunX509
 # store_type: JKS
 # cipher_suites:
 [TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA,TLS_DHE_RSA_WITH_AES_128_CBC_S]

```

**spark\_shared\_secret\_bit\_length**

The length of a shared secret used to authenticate Spark components and encrypt the connections between them. This value is not the strength of the cipher for encrypting connections. Default: 256

**spark\_security\_enabled**

Enables Spark security based on shared secret infrastructure. Enables mutual authentication and optional encryption between DSE Spark Master and Workers and of communication channels except the web UI. Default: false

**spark\_security\_encryption\_enabled**

Enables encryption of Spark connections except the web UI. Uses DIGEST-MD5 SASL-based encryption mechanism. Requires `spark_security_enabled: true`.

**spark\_daemon\_readiness\_assertion\_interval**

Time interval, in milliseconds, between subsequent retries by the Spark plugin for Spark Master and Worker readiness to start. Default: 1000

**resource\_manager\_options**

DataStax Enterprise can control the memory and cores offered by particular Spark Workers in semi-automatic fashion. You can define the total amount of physical resources available to Spark Workers, and optionally add named work pools with specific resources dedicated to them.

**worker\_options**

If the option is not specified, the default value 0.6 is used. The amount of system resources that are made available to the Spark Worker.

**cores\_total**

If the option is not specified, the default value 0.7 is used. The number of total system cores available to Spark. This setting can be the exact number of cores or a fraction of the total system cores.

When the value is expressed as a fraction, the available resources are calculated in the following way:

```
Spark Worker cores = cores_total * total system cores
```

The lowest value that you can assign to Spark Worker cores is 1 core. If the results are lower, no exception is thrown and the values are automatically limited. The range of fractional values is 0.01 to 1.

#### **memory\_total**

The amount of total system memory available to Spark. This setting can be the exact amount of memory or a fraction of the total system memory. When the value is an absolute value, you can use standard suffixes like M for megabyte and G for gigabyte.

When the value is expressed as a fraction, the available resources are calculated in the following way:

```
Spark Worker memory = memory_total * (total system memory - memory assigned to DataStax Enterprise)
```

The lowest values that you can assign to Spark Worker memory is 64 MB. If the results are lower, no exception is thrown and the values are automatically limited. The fractional range of values is 0.01 to 1. If the option is not specified, the default value 0.6 is used.

#### **workpools**

Named work pools that can use a portion of the total resources defined under `worker_options`. A default work pool named `default` is used if no work pools are defined in this section. If work pools are defined, the resources allocated to the work pools are taken from the total amount, with the remaining resources available to the `default` work pool. The total amount of resources defined in the `workpools` section must not exceed the resources available to Spark in `worker_options`. A work pool named `alwaysOn_sql` is created by default for AlwaysOn SQL. By default, it is configured to use 25% of the resources available to Spark.

##### **name**

The name of the work pool.

##### **cores**

The number of system cores to use in this work pool expressed as either an absolute value or a fractional value. This option follows the same rules as `cores_total`.

##### **memory**

The amount of memory to use in this work pool expressed as either an absolute value or a fractional value. This option follows the same rules as `memory_total`.

##### **spark\_ui\_options**

Specify the source for SSL settings for Spark Master and Spark Worker UIs. The `spark_ui_options` apply only to Spark daemon UIs, and do not apply to user applications even when the user applications are run in cluster mode.

##### **encryption**

- `inherit` - inherit the SSL settings from the client encryption options.

- custom - use the following [encryption\\_options \(page 134\)](#) from dse.yaml.

Default: inherit

**encryption\_options**

Set encryption options for HTTPS of Spark Master and Worker UI. The spark\_encryption\_options are not valid for DSE 5.1 and later.

**enabled**

Whether to enable Spark encryption for Spark client-to-Spark cluster and Spark internode communication.

Default: false

**keystore**

The keystore for Spark encryption keys.

The relative file path is the base Spark configuration directory that is defined by the SPARK\_CONF\_DIR environment variable. The default Spark configuration directory is resources/spark/conf.

Default: resources/dse/conf/.ui-keystore

**keystore\_password**

The password to access the key store.

Default: cassandra

**require\_client\_auth**

Whether to require truststore for client authentication. When not set, the default is false.

Default: commented out (false)

**truststore**

The truststore for Spark encryption keys.

The relative file path is the base Spark configuration directory that is defined by the SPARK\_CONF\_DIR environment variable. The default Spark configuration directory is resources/spark/conf.

Default: commented out (resources/dse/conf/.ui-truststore)

**truststore\_password**

The password to access the truststore.

Default: commented out (cassandra)

**protocol**

Defines the encryption protocol.

Default: commented out (TLS)

**algorithm**

Defines the key manager algorithm.

Default: commented out (TLSUnX509SunX509S)

**store\_type**

Defines the keystore type.

Default: commented out (JKS)

**cipher\_suites**

Defines the cipher suites for Spark encryption:

- TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA
- TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA
- TLS\_DHE\_RSA\_WITH\_AES\_128\_CBC\_SHA

- TLS\_DHE\_RSA\_WITH\_AES\_256\_CBC\_SHA
- TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA
- TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA

Default: commented out

## Starting Spark drivers and executors

```
spark_process_runner:
 runner_type: default
 run_as_runner_options:
 user_slots:
 - slot1
 - slot2
```

### **spark\_process\_runner:**

Options to configure how Spark driver and executor processes are created and managed.

#### **runner\_type**

- default - Use the default runner type.
- run\_as - Use the `run_as_runner_options` options. See [Running Spark processes as separate users \(page 222\)](#).

#### **run\_as\_runner\_options**

The slot users for separating Spark processes users from the DSE service user.

See [Running Spark processes as separate users \(page 222\)](#).

Default: slot1, slot2

## AlwaysOn SQL options

Properties to enable and configure [AlwaysOn SQL \(page 245\)](#).

```
AlwaysOn SQL options
alwayson_sql_options:
enabled: false
thrift_port: 10000
web_ui_port: 9077
reserve_port_wait_time_ms: 100
alwayson_sql_status_check_wait_time_ms: 500
workpool: alwayson_sql
log_dsefs_dir: /spark/log/alwayson_sql
auth_user: alwayson_sql
runner_max_errors: 10
```

### **alwayson\_sql\_options**

The AlwaysOn SQL options enable and configure the server on this node.

#### **enabled**

Whether to enable AlwaysOn SQL for this node. The node must be an analytics node. When not set, the default is false.

Default: commented out (`false`)

#### **thrift\_port**

The Thrift port on which AlwaysOn SQL listens.

Default: commented out (10000)

**web\_ui\_port**

The port on which the AlwaysOn SQL web UI is available.

Default: commented out (9077)

**reserve\_port\_wait\_time\_ms**

The wait time in milliseconds to reserve the `thrift_port` if it is not available.

Default: commented out (100)

**server\_status\_check\_wait\_time\_ms**

The time in milliseconds to wait for a health check status of the AlwaysOn SQL server.

Default: commented out (500)

**workpool**

The [work pool name \(page 133\)](#) used by AlwaysOn SQL.

Default: commented out (`alwayson_sql`)

**log\_dsefs\_dir**

Location in DSEFS of the AlwaysOn SQL log files.

Default: commented out (`/spark/log/alwayson_sql`)

**auth\_user**

The role to use for internal communication by AlwaysOn SQL if authentication is enabled. Custom roles must be created with `login=true`.

Default: commented out (`alwayson_sql`)

**runner\_max\_errors**

The maximum number of errors that can occur during AlwaysOn SQL service runner thread runs before stopping the service. A service stop requires a manual restart.

Default: commented out (10)

## DSE File System (DSEFS) options

Properties to enable and configure the DSE File System ([DSEFS \(page 273\)](#)).

**Note:** DSEFS replaced the Cassandra File System (CFS). DSE version 6.0 and later do not support CFS.

```
dsefs_options:
 enabled:
 keyspace_name: dsefs
 work_dir: /var/lib/dsefs
 public_port: 5598
 private_port: 5599
 data_directories:
 - dir: /var/lib/dsefs/data
 storage_weight: 1.0
 min_free_space: 5368709120
```

```
service_startup_timeout_ms: 30000
service_close_timeout_ms: 600000
server_close_timeout_ms: 2147483647 # Integer.MAX_VALUE
compression_frame_max_size: 1048576
query_cache_size: 2048
query_cache_expire_after_ms: 2000
```

```

gossip_options:
round_delay_ms: 2000
startup_delay_ms: 5000
shutdown_delay_ms: 10000
rest_options:
request_timeout_ms: 330000
connection_open_timeout_ms: 55000
client_close_timeout_ms: 60000
server_request_timeout_ms: 300000
idle_connection_timeout_ms: 60000
internode_idle_connection_timeout_ms: 120000
core_max_concurrent_connections_per_host: 8
transaction_options:
transaction_timeout_ms: 3000
conflict_retry_delay_ms: 200
conflict_retry_count: 40
execution_retry_delay_ms: 1000
execution_retry_count: 3
block_allocator_options:
overflow_margin_mb: 1024
overflow_factor: 1.05

```

**dsefs\_options**

Enable and configure options for DSEFS.

**enabled**

Whether to enable DSEFS.

- true - enables DSEFS on this node, regardless of the workload.
- false - disables DSEFS on this node, regardless of the workload.
- blank or commented out (#) - DSEFS will start only if the node is configured to run analytics workloads.

Default: commented out (blank)

**keyspace\_name**

The keyspace where the DSEFS metadata is stored. You can optionally configure multiple DSEFS file systems within a single datacenter by specifying different keyspace names for each cluster.

Default: commented out (`dsefs`)

**work\_dir**

The local directory for storing the local node metadata, including the node identifier. The volume of data stored in this directory is nominal and does not require configuration for throughput, latency, or capacity. This directory must not be shared by DSEFS nodes.

Default: commented out (`/var/lib/dsefs`)

**public\_port**

The public port on which DSEFS listens for clients.

**Note:** DataStax recommends that all nodes in the cluster have the same value. Firewalls must [open this port](#) to trusted clients. The service on this port is bound to the [native\\_transport\\_address \(page 91\)](#).

Default: commented out (5598)

**private\_port**

The private port for DSEFS inter-node communication.

**Caution:** Do not open this port to firewalls; this private port must be not visible from outside of the cluster.

Default: commented out (5599)

**data\_directories**

One or more data locations where the DSEFS data is stored.

**- dir**

Mandatory attribute to identify the set of directories. DataStax recommends segregating these data directories on physical devices that are different from the devices that are used for DataStax Enterprise. Using multiple directories on JBOD improves performance and capacity.

Default: commented out (/var/lib/dsefs/data)

**storage\_weight**

The weighting factor for this location specifies how much data to place in this directory, relative to other directories in the cluster. This soft constraint determines how DSEFS distributes the data. For example, a directory with a value of 3.0 receives about three times more data than a directory with a value of 1.0.

Default: commented out (1.0)

**min\_free\_space**

The reserved space, in bytes, to not use for storing file data blocks. You can use a unit of measure suffix to specify other size units. For example: terabyte (1 TB), gigabyte (10 GB), and megabyte (5000 MB).

Default: commented out (5368709120)

## Advanced properties for DSEFS

**service\_startup\_timeout\_ms**

Wait time, in milliseconds, before the DSEFS server times out while waiting for services to bootstrap.

Default: commented out (30000)

**service\_close\_timeout\_ms**

Wait time, in milliseconds, before the DSEFS server times out while waiting for services to close.

Default: commented out (600000)

**server\_close\_timeout\_ms**

Wait time, in milliseconds, that the DSEFS server waits during shutdown before closing all pending connections.

Default: commented out (2147483647)

**compression\_frame\_max\_size**

The maximum accepted size of a compression frame defined during file upload.

Default: commented out (1048576)

**query\_cache\_size**

Maximum number of elements in a single DSEFS Server query cache.

Default: commented out (2048)

**query\_cache\_expire\_after\_ms**

The time to retain the DSEFS Server query cache element in cache. The cache element expires when this time is exceeded.

Default: commented out (2000)

**gossip options**

Options to configure DSEFS gossip rounds.

#### **round\_delay\_ms**

The delay, in milliseconds, between gossip rounds.

Default: commented out (2000)

#### **startup\_delay\_ms**

The delay time, in milliseconds, between registering the location and reading back all other locations from the database.

Default: commented out (5000)

#### **shutdown\_delay\_ms**

The delay time, in milliseconds, between announcing shutdown and shutting down the node.

Default: commented out (30000)

#### **rest\_options**

Options to configure DSEFS rest times.

#### **request\_timeout\_ms**

The time, in milliseconds, that the client waits for a response that corresponds to a given request.

Default: commented out (330000)

#### **connection\_open\_timeout\_ms**

The time, in milliseconds, that the client waits to establish a new connection.

Default: commented out (55000)

#### **client\_close\_timeout\_ms**

The time, in milliseconds, that the client waits for pending transfer to complete before closing a connection.

Default: commented out (60000)

#### **server\_request\_timeout\_ms**

The time, in milliseconds, to wait for the server rest call to complete.

Default: commented out (300000)

#### **idle\_connection\_timeout\_ms**

The time, in milliseconds, for RestClient to wait before closing an idle connection.

If RestClient does not close connection after timeout, the connection is closed after 2\*idle\_connection\_timeout\_ms.

- time - wait time to close idle connection
- 0 - disable closing idle connections

Default: commented out (60000)

#### **internode\_idle\_connection\_timeout\_ms**

Wait time, in milliseconds, before closing idle internode connection. The internode connections are primarily used to exchange data during replication. Do not set lower than the default value for heavily utilized DSEFS clusters.

Default: commented out (0) (disabled)

#### **core\_max\_concurrent\_connections\_per\_host**

Maximum number of connections to a given host per single CPU core. DSEFS keeps a connection pool for each CPU core.

Default: 120000

#### **transaction\_options**

Options to configure DSEFS transaction times.

#### **transaction\_timeout\_ms**

## Configuration

Transaction run time, in milliseconds, before the transaction is considered for timeout and rollback.

Default: 3000

### **conflict\_retry\_delay\_ms**

Wait time, in milliseconds, before retrying a transaction that was ended due to a conflict. Default: 200

### **conflict\_retry\_count**

The number of times to retry a transaction before giving up. Default: 40

### **execution\_retry\_delay\_ms**

Wait time, in milliseconds, before retrying a failed transaction payload execution.

Default: 1000

### **execution\_retry\_count**

The number of payload execution retries before signaling the error to the application. Default: 3

### **block\_allocator\_options**

Controls how much additional data can be placed on the local coordinator before the local node overflows to the other nodes. The trade-off is between data locality of writes and balancing the cluster. A local node is preferred for a new block allocation, if:

```
used_size_on_the_local_node < average_used_size_per_node *
overflow_factor + overflow_margin
```

### **overflow\_margin\_mb**

- *margin\_size* - overflow margin size in megabytes
- 0 - disable block allocation overflow

Default: commented out (1024)

### **overflow\_factor**

- *factor* - overflow factor on an exponential scale
- 1.0 - disable block allocation overflow

Default: commented out (1.05)

## Audit database activities

Track database activity using the audit log feature. To get the maximum information from data auditing, turn on data auditing on every node.

**Tip:** See [Setting up database auditing](#).

### **audit\_logging\_options**

Options to enable and configure database activity logging.

#### **enabled**

Whether to enable database activity auditing.

- true - enables database activity auditing
- false - disables database activity auditing

Default: false

#### **logger**

The logger to use for recording events:

- SLF4JAuditWriter - Capture events in a log file.
- CassandraAuditWriter - Capture events in a table, `dse_audit.audit_log`.

**Tip:** Configure logging level, sensitive data masking, and log file name/location in the `logback.xml` file.

Default: `SLF4JAuditWriter`

#### **included\_categories**

Comma separated list of event categories that are captured, where the category names are:

- QUERY - Data retrieval events.
- DML - (Data manipulation language) Data change events.
- DDL - (Data definition language) Database schema change events.
- DCL - (Data change language) Role and permission management events.
- AUTH - (Authentication) Login and authorization related events.
- ERROR - Failed requests.
- UNKNOWN - Events where the category and type are both UNKNOWN.

Event categories that are not listed are not captured.

**Warning:** Use either `included_categories` or `excluded_categories` but not both. When specifying included categories leave [excluded\\_categories \(page 141\)](#) blank or commented out.

Default: none (include all categories)

#### **excluded\_categories**

Comma separated list of categories to ignore, where the categories are:

- QUERY - Data retrieval events.
- DML - (Data manipulation language) Data change events.
- DDL - (Data definition language) Database schema change events.
- DCL - (Data change language) Role and permission management events.
- AUTH - (Authentication) Login and authorization related events.
- ERROR - Failed requests.
- UNKNOWN - Events where the category and type are both UNKNOWN.

Events in all other categories are logged.

**Warning:** Use either `included_categories` or `excluded_categories` but not both. When specifying excluded categories leave [included\\_categories \(page 141\)](#) blank or commented out.

Default: none (exclude no categories )

#### **included\_keyspaces**

The keyspaces for which events are logged. Specify keyspace names in a comma separated list or use a regular expression to filter on keyspace name.

**Warning:** DSE supports using either `included_keyspaces` or `excluded_keyspaces` but not both. When specifying included categories leave [excluded\\_keyspaces \(page 142\)](#) blank or comment it out.

Default: none (include all keyspaces)

#### **excluded\_keyspaces**

Log events for all keyspaces which are not listed. Specify a comma separated list keyspace names or use a regular expression to filter on keyspace name. Only use this option if `included_keyspaces` is blank or commented out.

Default: none (exclude no keyspaces)

#### **included\_roles**

The roles for which events are logged. Log events for the listed roles. Specify roles in a comma separated list.

**Warning:** DSE supports using either `included_roles` or `excluded_roles` but not both. When specifying `included_roles` leave [excluded\\_keyspaces \(page 142\)](#) blank or comment it out.

Default: none (include all roles)

#### **excluded\_roles**

The roles for which events are not logged. Specify a comma separated list role names. Only use this option if `included_roles` is blank or commented out.

Default: none (exclude no roles)

### Cassandra audit writer options

```
retention_time: 0
cassandra_audit_writer_options:
 mode: sync
 batch_size: 50
 flush_time: 250
 queue_size: 30000
 write_consistency: QUORUM
dropped_event_log: /var/log/cassandra/dropped_audit_events.log
day_partition_millis: 3600000
```

#### **retention\_time**

The amount of time, in hours, audit events are retained by supporting loggers. Only the CassandraAuditWriter supports retention time.

- 0 - retain events forever
- hours - the number of hours to retain audit events

Default: 0 (retain events forever)

#### **cassandra\_audit\_writer\_options**

Audit writer options.

#### **mode**

The mode the writer runs in.

- sync - A query is not executed until the audit event is successfully written.
- async - Audit events are queued for writing to the audit table, but are not necessarily logged before the query executes. A pool of writer threads consumes the audit events from the queue, and writes them to the audit table in batch queries.

**Important:** While async substantially improves performance under load, if there is a failure between when a query is executed, and its audit

event is written to the table, the audit table might be missing entries for queries that were executed.

Default: sync

#### **batch\_size**

Available only when mode: async. Must be greater than 0.

The maximum number of events the writer dequeues before writing them out to the table. If warnings in the logs reveal that batches are too large, decrease this value or increase the value of [batch\\_size\\_warn\\_threshold\\_in\\_kb \(page 83\)](#) in `cassandra.yaml`.

Default: 50

#### **flush\_time**

Available only when mode: async.

The maximum amount of time in milliseconds before an event is removed from the queue by a writer before being written out. This flush time prevents events from waiting too long before being written to the table when there are not a lot of queries happening.

Default: 500

#### **queue\_size**

The size of the queue feeding the asynchronous audit log writer threads. When there are more events being produced than the writers can write out, the queue fills up, and newer queries are blocked until there is space on the queue. If a value of 0 is used, the queue size is unbounded, which can lead to resource exhaustion under heavy query load.

Default: 30000

#### **write\_consistency**

The consistency level that is used to write audit events.

Default: QUORUM

#### **dropped\_event\_log**

The directory to store the log file that reports dropped events. When not set, the default is `/var/log/cassandra/dropped_audit_events.log`.

Default: commented out (`/var/log/cassandra/dropped_audit_events.log`)

#### **day\_partition\_millis**

The interval, in milliseconds, between changing nodes to spread audit log information across multiple nodes. For example, to change the target node every 12 hours, specify 43200000 milliseconds. When not set, the default is 3600000 (1 hour).

Default: commented out (3600000) (1 hour)

### **DSE Tiered Storage options**

Options to define one or more disk configurations for [DSE Tiered Storage](#). Specify multiple disk configurations as unnamed tiers by a collection of paths that are defined in priority order, with the fastest storage media in the top tier. With heterogeneous storage configurations across the cluster, specify each disk configuration with `config_name:config_settings`, and then use this configuration in [CREATE TABLE](#) or [ALTER TABLE](#) statements.

```
tiered_storage_options:
strategy1:
tiers:
- paths:
- /mnt1
- /mnt2
- paths: [/mnt3, /mnt4]
- paths: [/mnt5, /mnt6]
#
local_options:
k1: v1
k2: v2
#
'another strategy':
tiers: [paths: [/mnt1]]
```

**tiered\_storage\_options**

Options to configure the smart movement of data across different types of storage media so that data is matched to the most suitable drive type, according to the performance and cost characteristics it requires

**strategy1**

The first disk configuration strategy. Create a strategy2, strategy3, and so on. In this example, strategy1 is the configurable name of the tiered storage configuration strategy.

**tiers**

The unnamed tiers in this section define a storage tier with the paths and file paths that define the priority order.

**local\_options**

Local configuration options overwrite the tiered storage settings for the table schema in the local dse.yaml file. See [Testing DSE Tiered Storage configurations](#).

**- paths**

The section of file paths that define the data directories for this tier of the disk configuration. Typically list the fastest storage media first. These paths are used only to store data that is configured to use tiered storage. These paths are independent of any settings in the cassandra.yaml file.

**- /filepath**

The file paths that define the data directories for this tier of the disk configuration.

**DSE Advanced Replication configuration settings**

DSE Advanced Replication configuration options to replicate data from remote clusters to central data hubs.

```
advanced_replication_options:
enabled: false
conf_driver_password_encryption_enabled: false
advanced_replication_directory: /var/lib/cassandra/advrep
security_base_path: /base/path/to/advrep/security/files/
```

**advanced\_replication\_options**

Options to enable and configure DSE Advanced Replication.

**enabled**

Whether to enable an edge node to collect data in the replication log.

Default: commented out (`false`)

**conf\_driver\_password\_encryption\_enabled**

Whether to enable encryption of driver passwords. When enabled, the stored driver password is expected to be encrypted. See [Encrypting configuration file properties](#).

Default: commented out (`false`)

**advanced\_replication\_directory**

The directory for storing advanced replication CDC logs. A directory `replication_logs` will be created in the specified directory.

Default: commented out (`/var/lib/cassandra/advrep`)

**security\_base\_path**

The base path to prepend to paths in the Advanced Replication configuration locations, including locations to SSL keystore, SSL truststore, and so on.

Default: commented out (`/base/path/to/advrep/security/files/`)

## Inter-node messaging options

Configuration options for the internal messaging service used by several components of DataStax Enterprise. All internode messaging requests use this service.

```
internode.messaging_options:
 port: 8609
 # frame_length_in_mb: 256
 # server_acceptor_threads: 8
 # server_worker_threads: 16
 # client_max_connections: 100
 # client_worker_threads: 16
 # handshake_timeout_seconds: 10
 # client_request_timeout_seconds: 60
```

**internode.messaging\_options**

Configuration options for inter-node messaging.

**port**

The mandatory port for the inter-node messaging service.

Default: 8609

**frame\_length\_in\_mb**

Maximum message frame length. When not set, the default is 256.

Default: commented out (256)

**server\_acceptor\_threads**

The number of server acceptor threads. When not set, the default is the number of available processors.

Default: commented out

**server\_worker\_threads**

The number of server worker threads. When not set, the default is the number of available processors \* 8.

Default: commented out

**client\_max\_connections**

The maximum number of client connections. When not set, the default is 100.  
Default: commented out (100)

### **client\_worker\_threads**

The number of client worker threads. When not set, the default is the number of available processors \* 8.  
Default: commented out

### **handshake\_timeout\_seconds**

Timeout for communication handshake process. When not set, the default is 10.  
Default: commented out (10)

### **client\_request\_timeout\_seconds**

Timeout for non-query search requests like core creation and distributed deletes.  
When not set, the default is 60.  
Default: commented out (60)

## **DSE Multi-Instance server\_id**

### **server\_id**

In DSE Multi-Instance /etc/dse-nodeId/dse.yaml files, the server\_id option is generated to uniquely identify the physical server on which multiple instances are running. The server\_id default value is the media access control address (MAC address) of the physical server. You can change server\_id when the MAC address is not unique, such as a virtualized server where the host's physical MAC is cloned.

## **DSE Graph options**

- [DSE Graph system-level options \(page 146\)](#)
- [DSE Graph Gremlin Server options \(page 148\)](#)

## **DSE Graph system-level options**

These graph options are system-level configuration options and options that are shared between graph instances. Add an option if it is not present in the provided dse.yaml file.

```
graph:
 # analytic_evaluation_timeout_in_minutes: 10080
 # realtime_evaluation_timeout_in_seconds: 30
 # schema_agreement_timeout_in_ms: 10000
 # system_evaluation_timeout_in_seconds: 180
 # adjacency_cache_size_in_mb: 128
 # index_cache_size_in_mb: 128
 # max_query_queue: 10000
 # max_query_threads (no explicit default)
 # max_query_params: 16
```

### **graph**

These graph options are system-level configuration options and options that are shared between graph instances.

Option names and values expressed in ISO 8601 format used in earlier DSE 5.0 releases are still valid. The ISO 8601 format is deprecated.

### **analytic\_evaluation\_timeout\_in\_minutes**

Maximum time to wait for an OLAP analytic (Spark) traversal to evaluate. When not set, the default is 10080 (168 hours).

Default: commented out (10080)

#### **realtime\_evaluation\_timeout\_in\_seconds**

Maximum time to wait for an OLTP real-time traversal to evaluate. When not set, the default is 30.

Default: commented out (30)

#### **schema\_agreement\_timeout\_in\_ms**

Maximum time to wait for the database to agree on schema versions before timing out. When not set, the default is 10000 (10 seconds).

Default: commented out (10000)

#### **system\_evaluation\_timeout\_in\_seconds**

Maximum time to wait for a graph system-based request to execute, like creating a new graph. When not set, the default is 180 (3 minutes).

Default: commented out (180)

#### **adjacency\_cache\_size\_in\_mb**

The amount of RAM to allocate to each graph's adjacency (edge and property) cache. When not set, the default is 128.

Default: commented out (128)

#### **index\_cache\_size\_in\_mb**

The amount of ram to allocate to the index cache. When not set, the default is 128.

Default: commented out (128)

#### **max\_query\_queue**

The maximum number of CQL queries that can be queued as a result of Gremlin requests. Incoming queries are rejected if the queue size exceeds this setting.

When not set, the default is 10000.

Default: commented out (10000)

#### **max\_query\_threads**

The maximum number of threads to use for queries to the database. When this option is not set, the default is calculated:

- If gremlinPool is present and nonzero:  
10 \* the gremlinPool setting
- If gremlinPool is not present in this file or set to zero:  
The number of available CPU cores

See [gremlinPool](#).

Default: calculated

#### **max\_query\_params**

The maximum number of parameters that can be passed on a graph query request for TinkerPop drivers and drivers using the Cassandra native protocol. Passing very large numbers of parameters on requests is an anti-pattern, because the script evaluation time increases proportionally. DataStax recommends reducing the number of parameters to speed up script compilation times. Before you increase this value, consider alternate methods for parameterizing scripts, like passing a single map. If the graph query request requires many arguments, pass a list.

Default: commented out (16)

## DSE Graph Gremlin Server options

The Gremlin Server is configured using [Apache TinkerPop](#) specifications.

```
gremlin_server:
 # port: 8182
 # threadPoolWorker: 2
 # gremlinPool: 0
 #
 # scriptEngines:
 # gremlin-groovy:
 # config:
 # sandbox_enabled: false
 # sandbox_rules:
 # whitelist_packages:
 # - package.name
 # whitelist_types:
 # - fully.qualified.type.name
 # whitelist_supers:
 # - fully.qualified.class.name
 # blacklist_packages:
 # - package.name
 # blacklist_supers:
 # - fully.qualified.class.name
```

### **gremlin\_server**

The top-level configurations in Gremlin Server.

#### **port**

The available communications port for Gremlin Server. When not set, the default is 8182.

Default: commented out (8182)

#### **threadPoolWorker**

The number of worker threads that handle non-blocking read and write (requests and responses) on the Gremlin Server channel, including routing requests to the right server operations, handling scheduled jobs on the server, and writing serialized responses back to the client. When not set, the default is 2.

Default: commented out (2)

#### **gremlinPool**

The number of Gremlin threads available to execute actual scripts in a ScriptEngine. This pool represents the workers available to handle blocking operations in Gremlin Server.

- 0 - the value of the JVM property `cassandra.available_processors`, if that property is set
- When not set - the value of `Runtime.getRuntime().availableProcessors()`

Default: commented out (0)

#### **scriptEngines**

Section to configure gremlin server scripts.

##### **gremlin-groovy**

Section for gremlin-groovy scripts.

##### **sandbox\_enabled**

Sandbox is enabled by default. To disable the gremlin groovy sandbox entirely, set to false.

#### **sandbox\_rules**

Section for sandbox rules.

#### **whitelist\_packages**

List of packages, one package per line, to whitelist.

##### **-package.name**

Retain the hyphen before the fully qualified package name.

#### **whitelist\_types**

List of types, one type per line, to whitelist.

##### **-fully.qualified.type.name**

Retain the hyphen before the fully qualified type name.

#### **whitelist\_supers**

List of super classes, one class per line, to whitelist. Retain the hyphen before the fully qualified class name.

##### **-fully.qualified.class.name**

Retain the hyphen before the fully qualified class name.

#### **blacklist\_packages**

List of packages, one package per line, to blacklist.

##### **-package.name**

Retain the hyphen before the fully qualified package name.

#### **blacklist\_supers**

List of super classes, one class per line, to blacklist. Retain the hyphen before the fully qualified class name.

##### **-fully.qualified.class.name**

Retain the hyphen before the fully qualified class name.

See also [Configuring the Gremlin console in the remote.yaml file](#).

## **cassandra-rackdc.properties file**

Configuration file for the GossipingPropertyFileSnitch, Ec2Snitch, and Ec2MultiRegionSnitch.

The GossipingPropertyFileSnitch, Ec2Snitch, and Ec2MultiRegionSnitch use the cassandra-rackdc.properties configuration file to determine which datacenters and racks nodes belong to. They inform the database about the network topology to route requests efficiently and distribute replicas evenly. Settings for this file depend on the type of snitch:

- [GossipingPropertyFileSnitch \(page 149\)](#)
- [Configuring the Amazon EC2 single-region snitch \(page 151\)](#)
- [Configuring Amazon EC2 multi-region snitch \(page 153\)](#)

This page also includes instructions for [migrating \(page 150\)](#) from the PropertyFileSnitch to the GossipingPropertyFileSnitch.

### **GossipingPropertyFileSnitch**

This snitch is recommended for production. It uses rack and datacenter information for the local node defined in the cassandra-rackdc.properties file and propagates this information to other nodes via gossip.

To configure a node to use GossipingPropertyFileSnitch, edit the `cassandra-rackdc.properties` file as follows:

- Define the datacenter and rack that include this node. The default settings:

```
dc=DC1
rack=RAC1
```

**Note:** datacenter and rack names are case-sensitive. For examples, see [Initializing a single datacenter per workload type](#) and [Initializing multiple datacenters per workload type](#).

- To save bandwidth, add the `prefer_local=true` option. This option tells DataStax Enterprise to use the local IP address when communication is not across different datacenters.

## Migrating from the PropertyFileSnitch to the GossipingPropertyFileSnitch

To allow migration from the PropertyFileSnitch, the GossipingPropertyFileSnitch uses the `cassandra-topology.properties` file when present. Delete the file after the migration is complete. For more information about migration, see [Switching snitches](#).

**Note:** The GossipingPropertyFileSnitch always loads `cassandra-topology.properties` when that file is present. Remove the file from each node on any new cluster or any cluster migrated from the PropertyFileSnitch.

## cassandra-topology.properties file

Configuration file for setting datacenters and rack names and using the PropertyFileSnitch.

The PropertyFileSnitch uses the `cassandra-topology.properties` for datacenters and rack names and to determine network topology so that requests are routed efficiently and allows the database to distribute replicas evenly.

**Note:** The [GossipingPropertyFileSnitch \(page 149\)](#) snitch is recommended for production. See [Migrating from the PropertyFileSnitch to the GossipingPropertyFileSnitch \(page 150\)](#).

## PropertyFileSnitch

This snitch determines proximity as determined by rack and datacenter. It uses the network details located in the `cassandra-topology.properties` file. When using this snitch, you can define your datacenter names to be whatever you want. Make sure that the datacenter names correlate to the name of your datacenters in the [keyspace definition](#). Every node in the cluster should be described in the `cassandra-topology.properties` file, and this file should be exactly the same on every node in the cluster.

## Setting datacenters and rack names

If you had non-uniform IPs and two physical datacenters with two racks in each, and a third logical datacenter for replicating analytics data, the `cassandra-topology.properties` file might look like this:

**Note:** Datacenter and rack names are case-sensitive.

```
datacenter One

175.56.12.105=DC1:RAC1
175.50.13.200=DC1:RAC1
175.54.35.197=DC1:RAC1

120.53.24.101=DC1:RAC2
120.55.16.200=DC1:RAC2
120.57.102.103=DC1:RAC2

datacenter Two

110.56.12.120=DC2:RAC1
110.50.13.201=DC2:RAC1
110.54.35.184=DC2:RAC1

50.33.23.120=DC2:RAC2
50.45.14.220=DC2:RAC2
50.17.10.203=DC2:RAC2

Analytics Replication Group

172.106.12.120=DC3:RAC1
172.106.12.121=DC3:RAC1
172.106.12.122=DC3:RAC1

default for unknown nodes
default =DC3:RAC1
```

## Configuring snitches for cloud providers

Set up `Ec2Snitch`, `Ec2MultiRegionSnitch`, `GoogleCloudSnitch`, or `CloudstackSnitch`.

Configure a cloud provider snitch that corresponds to the provider.

## Configuring the Amazon EC2 single-region snitch

Use the `Ec2Snitch` snitch for an Amazon EC2 deployments that are in a single region.

Use the `Ec2Snitch` for simple cluster deployments on Amazon EC2 where all nodes in the cluster are within a single region. Because private IPs are used, this snitch does not work across multiple regions.

In EC2 deployments, the region name is treated as the datacenter name and availability zones are treated as racks within a datacenter. For example, if a node is in the **us-east-1** region, **us-east** is the datacenter name and **1** is the rack location. (Racks are important for distributing replicas, but not for datacenter naming.)

If you are using only a single datacenter, you do not need to specify any properties.

If you need multiple datacenters, set the **dc\_suffix** options in the `cassandra-rackdc.properties` file. Any other lines are ignored.

For example, for each node within the **us-east** region, specify the datacenter in its `cassandra-rackdc.properties` file:

**Note:** datacenter names are case-sensitive.

- **node0**

```
dc_suffix=_1_cassandra
```

- **node1**

```
dc_suffix=_1_cassandra
```

- **node2**

```
dc_suffix=_1_cassandra
```

- **node3**

```
dc_suffix=_1_cassandra
```

- **node4**

```
dc_suffix=_1_analytics
```

- **node5**

```
dc_suffix=_1_search
```

This results in three datacenters for the region:

```
us-east_1_cassandra
us-east_1_analytics
us-east_1_search
```

**Note:** The datacenter naming convention in this example is based on the workload. You can use other conventions, such as DC1, DC2 or 100, 200.

## Keyspace strategy options

When defining your [keyspace strategy options](#), use the EC2 region name, such as ``us-east``, as your datacenter name.

## Configuring Amazon EC2 multi-region snitch

Use the `Ec2MultiRegionSnitch` snitch for Amazon EC2 deployments where the cluster spans multiple regions.

Use the `Ec2MultiRegionSnitch` for deployments on Amazon EC2 where the cluster spans multiple regions.

You must configure settings in both the `cassandra.yaml` file and the property file (`cassandra-rackdc.properties`) used by the `Ec2MultiRegionSnitch`.

### Configuring `cassandra.yaml` for cross-region communication

The `Ec2MultiRegionSnitch` uses public IP designated in the `broadcast_address` to allow cross-region connectivity. Configure each node as follows:

1. In the `cassandra.yaml`, set the [listen\\_address \(page 69\)](#) to the *private* IP address of the node, and the [broadcast\\_address \(page 84\)](#) to the *public* IP address of the node. This allows DataStax Enterprise nodes in one EC2 region to bind to nodes in another region, thus enabling multiple datacenter support. For intra-region traffic, DataStax Enterprise switches to the private IP after establishing a connection.
2. Set the addresses of the seed nodes in the `cassandra.yaml` file to that of the *public* IP. Private IP are not routable between networks. For example:

```
seeds: 50.34.16.33, 60.247.70.52
```

To find the public IP address, from each of the seed nodes in EC2:

```
curl http://instance-data/latest/meta-data/public-ipv4
```

**Note:** Do not make all nodes seeds, see [Internode communications \(gossip\)](#).

3. Be sure that the [storage\\_port \(page 89\)](#) or [ssl\\_storage\\_port \(page 102\)](#) is open on the public IP firewall.

### Configuring the snitch for cross-region communication

In EC2 deployments, the region name is treated as the datacenter name and availability zones are treated as racks within a datacenter. For example, if a node is in the **us-east-1** region, **us-east** is the datacenter name and **1** is the rack location. (Racks are important for distributing replicas, but not for datacenter naming.)

For each node, specify its datacenter in the `cassandra-rackdc.properties`. The `dc_suffix` option defines the datacenters used by the snitch. Any other lines are ignored.

In the example below, there are two DataStax Enterprise datacenters and each datacenter is named for its workload. The datacenter naming convention in this example is based on the workload. You can use other conventions, such as DC1, DC2 or 100, 200. (datacenter names are case-sensitive.)

| Region: us-east                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | Region: us-west                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>Node and datacenter:</p> <ul style="list-style-type: none"> <li>• <b>node0</b><br/>dc_suffix=_1_transactional</li> <li>• <b>node1</b><br/>dc_suffix=_1_transactional</li> <li>• <b>node2</b><br/>dc_suffix=_2_transactional</li> <li>• <b>node3</b><br/>dc_suffix=_2_transactional</li> <li>• <b>node4</b><br/>dc_suffix=_1_analytics</li> <li>• <b>node5</b><br/>dc_suffix=_1_search</li> </ul> <p>This results in four <b>us-east</b> datacenters:</p> <pre>us-east_1_transactional us-east_2_transactional us-east_1_analytics us-east_1_search</pre> | <p>Node and datacenter:</p> <ul style="list-style-type: none"> <li>• <b>node0</b><br/>dc_suffix=_1_transactional</li> <li>• <b>node1</b><br/>dc_suffix=_1_transactional</li> <li>• <b>node2</b><br/>dc_suffix=_2_transactional</li> <li>• <b>node3</b><br/>dc_suffix=_2_transactional</li> <li>• <b>node4</b><br/>dc_suffix=_1_analytics</li> <li>• <b>node5</b><br/>dc_suffix=_1_search</li> </ul> <p>This results in four <b>us-west</b> datacenters:</p> <pre>us-west_1_transactional us-west_2_transactional us-west_1_analytics us-west_1_search</pre> |

## Keyspace strategy options

When defining your [keyspace strategy options](#), use the EC2 region name, such as ``us-east``, as your datacenter name.

## Configuring the Google Cloud Platform snitch

Use the `GoogleCloudSnitch` snitch for DataStax Enterprise deployments on Google Cloud Platform across one or more regions.

Use the `GoogleCloudSnitch` for DataStax Enterprise deployments on [Google Cloud Platform](#) across one or more regions. The region is treated as a datacenter and the availability zones are treated as racks within the datacenter. All communication occurs over private IP addresses within the same logical network.

The region name is treated as the datacenter name and zones are treated as racks within a datacenter. For example, if a node is in the **us-central1-a** region, **us-central1** is the datacenter name and **a** is the rack location. (Racks are important for distributing replicas, but not for datacenter naming.) This snitch can work across multiple regions without additional configuration.

If you are using only a single datacenter, you do not need to specify any properties.

If you need multiple datacenters, set the **dc\_suffix** options in the `cassandra-rackdc.properties` file. Any other lines are ignored.

For example, for each node within the **us-central1** region, specify the datacenter in its `cassandra-rackdc.properties` file:

**Note:** Datacenter names are case-sensitive.

| Node  | dc_suffix                               |
|-------|-----------------------------------------|
| node0 | <code>dc_suffix=_a_transactional</code> |
| node1 | <code>dc_suffix=_a_transactional</code> |
| node2 | <code>dc_suffix=_a_transactional</code> |
| node3 | <code>dc_suffix=_a_transactional</code> |
| node4 | <code>dc_suffix=_a_analytics</code>     |
| node5 | <code>dc_suffix=_a_search</code>        |

## Configuring the Apache CloudStack snitch

Use the `CloudstackSnitch` snitch for Apache CloudStack environments.

Use the `CloudstackSnitch` for [Apache CloudStack](#) environments. Because zone naming is free-form in Apache CloudStack, this snitch uses the widely-used `<country> <az>` notation.

## Using start-up parameters

Modify DataStax Enterprise system properties from the startup command line.

Use the system property (`-D`) switch to modify the DataStax Enterprise (DSE) settings during start up.

**Tip:** To automatically pass the settings each time DSE starts, uncomment or add the switch to the `jvm.options` file.

### Synopsis

On the command line use the following syntax:

```
dse cassandra -Dparameter_name=value
```

In the `jvm.options` file use the following syntax:

```
-Dparameter_name=value
```

**Warning:** Only pass the parameter to the startup operation once. If the same switch is passed to the start operation multiple times, for example from both the `jvm.options` file and on the command line, DSE may fail to start or may use the wrong parameter.

## DSE system property switches

Change DSE system properties using the following switches.

### -Dcassandra.auto\_bootstrap

Set [auto\\_bootstrap \(page 83\)](#) to `false` during the initial set up of a node to override the default setting in the `cassandra.yaml` file.

Default: `true`.

### -Dcassandra.available\_processors

Number of processors available to DSE. In a multi-instance deployment, each instance independently assumes that all CPU processors are available to it. Use this setting to specify a smaller set of processors.

Default: `all_processors`.

### -Dcassandra.config

Set to the directory location of the `cassandra.yaml` file.

Default: depends on the type of installation.

### -Dcassandra.consistent.rangemovement

Set to `true`, makes bootstrapping behavior effective.

Default: `false`.

### -Dcassandra.disable\_auth\_caches\_remote\_configuration

Set to `true` to disable authentication caches, for example the caches used for credentials, permissions, and roles. This will mean those config options can only be set (persistently) in `cassandra.yaml` and will require a restart for new values to take effect.

Default: `false`.

### -Dcassandra.expiration\_date\_overflow\_policy

Set the policy (`REJECT` or `CAP`) for any TTL (time to live) timestamps that exceeds the maximum value supported by the storage engine, `2038-01-19T03:14:06+00:00`.

The database storage engine can only encode TTL timestamps through January 19 2038 03:14:07 UTC due to the [Year 2038 problem](#).

- `REJECT`: Reject requests that contain an expiration timestamp later than `2038-01-19T03:14:06+00:00`.
- `CAP`: Allow requests and insert expiration timestamps later than `2038-01-19T03:14:06+00:00` as `2038-01-19T03:14:06+00:00`.

Default: `REJECT`.

### -Dcassandra.force\_default\_indexing\_page\_size

Set to `true` to disable dynamic calculation of the page size used when indexing an entire partition during initial index build or a rebuild. Fixes the page size to the default of 10000 rows per page.

Default: `false`.

### -Dcassandra.ignore\_dc

Set to `true` to ignore the datacenter name change on startup. Applies only when using `DseSimpleSnitch`.

Default: `false`.

### -Dcassandra.initial\_token

Use when DSE is not using virtual nodes (vnodes). Set to the initial partitioner token for the node on the first start up.

Default: `blank` (not set).

**Note:** Vnodes automatically select tokens.

#### -Dcassandra.join\_ring

Set to `false` to prevent the node from joining a ring on startup.

**Tip:** Add the node to the ring afterwards using `nodetool join` and a JMX call.

Default: `true`.

#### -Dcassandra.load\_ring\_state

Set to `false` to clear all gossip state for the node on restart.

Default: `true`.

#### -Dcassandra.metricsReporterConfigFile

Enables pluggable metrics reporter and configures it from the specified file.

Default: `blank` (not set).

#### -Dcassandra.native\_transport\_port

Set to the port number that CQL native transport listens for clients.

Default: `9042`.

#### -Dcassandra.native\_transport\_startup\_delay\_seconds

Set to the number of seconds to delay the native transport server start up.

Default: `0` (no delay).

#### -Dcassandra.partitioner

Set to the partitioner name.

Default: `org.apache.cassandra.dht.Murmur3Partitioner`.

#### -Dcassandra.partition\_sstables\_by\_token\_range

Set to `false` to disable JBOD SSTable partitioning by token range to multiple `data_file_directories`.

**Caution:** Advanced setting that should only be used with guidance from [DataStax Support](#).

Default: `true`.

#### -Dcassandra.replace\_address

Set to the [listen\\_address](#) (page 69) or the [broadcast\\_address](#) (page 84) when replacing a dead node with a new node. The new node must be in the same state as before bootstrapping, without any data in its data directory.

**Note:** The `broadcast_address` defaults to the `listen_address` except when the ring is using the [Configuring Amazon EC2 multi-region snitch](#) (page 153).

#### -Dcassandra.replayList

Allows restoring specific tables from an archived commit log.

#### -Dcassandra.ring\_delay\_ms

Set to the number of milliseconds the node waits to hear from other nodes before formally joining the ring.

Default: `30000`.

#### -Dcassandra.ssl\_storage\_port

Sets the SSL port for encrypted communication.

Default: `7001`.

#### -Dcassandra.start\_native\_transport

Enables or disables the native transport server. See [start\\_native\\_transport](#) (page 91) in `cassandra.yaml`.

Default: `true`.

**-Dcassandra.storage\_port**

Sets the port for inter-node communication.

Default: 7000.

**-Dcassandra.write\_survey**

Set to `true` to enable a tool for testing new compaction and compression strategies. `write_survey` allows you to experiment with different strategies and benchmark write performance differences without affecting the production workload. See [Testing compaction and compression](#).

Default: `false`.

**-Ddse.io.aio.enable**

Set to `false` to have all read operations use the `AsynchronousFileChannel` regardless of the operating system or disk type.

The default setting `true` allows dynamic switching of libraries for read operations as follows:

- `LibAIO` on solid state drives (SSD) and EXT4/XFS
- `AsynchronousFileChannel` for read operations on hard disk drives and all non-Linux operating systems

**Caution:** Advanced setting that should only be used with guidance from [DataStax Support](#).

Default: `true`.

**-Ddse.io.aio.force**

Set to `true` to force all read operations to use `LibAIO` regardless of the disk type or operating system.

**Caution:** Advanced setting that should only be used with guidance from [DataStax Support](#).

Default: `false`.

## DSE Search switches

**-Ddse.search.client.timeout.secs**

Set the timeout in seconds for native driver search core management calls using the `dsetool` search-specific commands.

Default: 600 (10 minutes).

**-Ddse.search.query.threads**

Sets the number of Search queries that can execute in parallel. Consider increasing this value or reducing client/driver requests per connection if [EnqueuedRequestCount \(page 349\)](#) does not stabilize near zero.

Default: The default is two times the number of CPUs (including hyperthreading).

**-Ddse.solr.data.dir**

Set the `path` to store DSE Search data. See [Set the location of search indexes](#).

**-Dsolr.offheap.enable**

The DSE Search per-segment filter cache is moved off-heap by using native memory to reduce on-heap memory consumption and garbage collection overhead.

The off-heap filter cache is enabled by default. To disable, set to false to pass the offheap JVM system property at startup time. When not set, the default is true.

Default: true

## Netty switches

### -Dnetty.eventloop.yield\_extra\_spins

Set to the number of iterations issue a thread yield when the queues are empty.

Enabling this option increases context switches and impacts performance, both throughput and latency.

Default: 0 (disabled).

### -Dnetty.eventloop.park\_extra\_spins

Set to the number of iterations call a thread park for 1 nanosecond when the queues are empty. Enabling this option increases context switches and impacts performance, both throughput and latency.

Default: 0 (disabled).

### -Dnetty.eventloop.busy\_extra\_spins=N

Set to the number of iterations in the epoll event loops performed when queues are empty before moving on to the next backoff stage. Increasing the value reduces latency while increasing CPU usage when the loops are idle.

Default: 10.

### -Dnetty.epoll\_check\_interval\_nanos

Sets the granularity for calling an epoll select in nanoseconds, which is a system call. Setting the value too low impacts performance because by making too many system calls. Setting the value too high, impacts performance by delaying the discovery of new events.

Default: 2000.

### -Dnetty.schedule\_check\_interval\_nanos

Set the granularity for checking if scheduled events are ready to execute in nanoseconds. Specifying a value below 1 nanosecond is not productive. Too high a values delays scheduled tasks.

Default: 1000.

## LDAP tuning switches

### -Ddse.ldap.connection.timeout.ms

The number of milliseconds before the connection timeout.

Default:

### -Ddse.ldap.pool.min.idle

Finer control over the connection pool for DataStax Enterprise LDAP authentication connector. The min idle settings determines the minimum number of connections allowed in the pool before the evictor thread will create new connections. This setting has no effect if the evictor thread isn't configured to run.

Default:

### -Ddse.ldap.pool.exhausted.action

Determines what the pool does when it is full. It can be one of:

- fail - the pool will throw an exception
- block - the pool will block for max wait ms (default)
- grow - the pool will just keep growing (not recommended)

Default: `block`

**-Ddse.ldap.pool.max.wait**

When the `dse.ldap.pool.exhausted.action` is `block`, sets the number of milliseconds to block the pool before throwing an exception.

Default:

**-Ddse.ldap.pool.test.borrow**

Tests a connection when it is borrowed from the pool.

Default:

**-Ddse.ldap.pool.test.return**

Tests a connection returned to the pool.

Default:

**-Ddse.ldap.pool.test.idle**

Tests any connections in the eviction loop that are not being evicted. Only works if the time between eviction runs is greater than 0ms.

Default:

**-Ddse.ldap.pool.time.between.evictions**

Determines the time in ms (milliseconds) between eviction runs. When run with the `dse.ldap.pool.test.idle` this becomes a basic keep alive for connections.

Default:

**-Ddse.ldap.pool.num.tests.per.eviction**

Number of connections in the pool that are tested each connection run. If this is set the same as max active (the pool size) then all connections will be tested each eviction run.

Default:

**-Ddse.ldap.pool.min.evictable.idle.time.ms**

Determines the minimum time in ms (milliseconds) that a connection can sit in the pool before it becomes available for eviction.

Default:

**-Ddse.ldap.pool.soft.min.evictable.idle.time.ms**

Determines the minimum time in ms (milliseconds) that a connection can sit the pool before it becomes available for eviction with the proviso that the number of connections doesn't fall below `dse.ldap.pool.min.evictable.idle.time.ms`.

Default:

## NodeSync switches

**-Ddse.nodesync.controller\_update\_interval\_sec**

Set the frequency to execute NodeSync auto-tuning process in seconds.

Default: 300 (5 minutes).

**-Ddse.nodesync.log\_reporter\_interval\_sec**

Set the frequency of short INFO progress report in seconds.

Default: 600 (10 minutes).

**-Ddse.nodesync.min\_validation\_interval\_sec**

Set to the minimum number of seconds between validations of the same segment, mostly to avoid busy spinning on new/empty clusters.

Default: 300 (5 minutes).

**-Ddse.nodesync.min\_warn\_interval\_sec**

Set to the minimum number of seconds between logging warnings.

**Tip:** Avoid logging warnings too often.

Default: 36000 (10 hours).

**-Ddse.nodesync.rate\_checker\_interval\_sec**

Set the frequency in seconds of comparing the current configured rate to tables and their deadline. Log a warning if rate considered too low.

Default: 1800 (30 minutes).

**-Ddse.nodesync.segment\_lock\_timeout\_sec**

Set the Time-to-live (TTL) on locks inserted in the status table in seconds.

Default: 600 (10 minutes).

**-Ddse.nodesync.segment\_size\_target\_bytes**

Set to the targeted maximum size for segments in bytes.

Default: 26214400 (200 MB).

**-Ddse.nodesync.size\_checker\_interval\_sec**

Set the frequency to check if the depth used for a table should be updated due to data size changes in seconds.

Default: 7200 (2 hours).

### Starting a node without joining the ring:

- Command line:

```
dse cassandra -Dcassandra.join_ring=false
```

- jvm.options:

```
-Dcassandra.join_ring=false
```

### Replacing a dead node:

- Command line:

```
dse cassandra -Dcassandra.replace_address=10.91.176.160
```

- jvm.options:

```
-Dcassandra.replace_address=10.91.176.160
```

## Choosing a compaction strategy

Information on how to select the best compaction strategy.

To implement the chosen compaction strategy:

- To understand how compaction and compaction strategies work, read [How is data maintained?](#)
- Review your application's requirements use this information to answer the questions below.
- [Configure the table \(page 162\)](#) to use the most appropriate strategy.

#### 4. Test the compaction strategies (page 162) against your data.

### Which compaction strategy is best?

The following questions are based on the experiences of developers and users with the strategies.

#### Does your table process time series data?

If so, your best choice is [Compaction strategies](#). If not, the following questions introduce other considerations to guide your choice.

#### Does your table handle more reads than writes, or more writes than reads?

[LCS](#) is a good choice if your table processes twice as many reads as writes or more — especially randomized reads. If the proportion of reads to writes is closer, the performance hit exacted by LCS may not be worth the benefit. Be aware that LCS can be quickly overwhelmed by a high volume of writes.

#### Does the data in your table change often?

One advantage of LCS is that it keeps related data in a small set of SSTables. If your data is *immutable* or not subject to frequent [upserts](#), [STCS](#) accomplishes the same type of grouping without the LCS performance hit.

#### Do you require predictable levels of read and write activity?

LCS keeps the SSTables within predictable sizes and numbers. For example, if your table's read/write ratio is small, and it is expected to conform to a Service Level Agreements (SLAs) for reads, it may be worth taking the write performance penalty of LCS in order to keep read rates and latency at predictable levels. And you may be able to overcome this write penalty through horizontal scaling (adding more nodes).

#### Will your table be populated by a batch process?

On both batch reads and batch writes, STCS performs better than LCS. The batch process causes little or no fragmentation, so the benefits of LCS are not realized; batch processes can overwhelm LCS-configured tables.

#### Does your system have limited disk space?

LCS handles disk space more efficiently than STCS: it requires about 10% *headroom* in addition to the space occupied by the data it handles. STCS and DTCS generally require, in some cases, as much as 50% more than the data space. ([DateTieredStorageStrategy](#) (DTCS) is deprecated.)

#### Is your system reaching its limits for I/O?

LCS is significantly more I/O intensive than DTCS or STCS. Switching to LCS may introduce extra I/O load that offsets the advantages.

### Configuring and running compaction

Set the compaction strategy for a table in the parameters for the `CREATE TABLE` or `ALTER TABLE` command. For details, see [table\\_options](#).

You can start compaction manually using the `nodetool compact` command.

### Testing compaction strategies

Suggestions for determining which compaction strategy is best for your system:

- Create a three-node cluster using one of the compaction strategies, stress test the cluster using [cassandra-stress](#) and measure the results.

- Set up a node on your existing cluster and use the [write survey mode](#) to sample live data.

## Configuring NodeSync

Set up continuous background repair.

### About NodeSync

Easy to use continuous background repair that has low overhead and provides consistent performance.

NodeSync is an easy to use continuous background repair that has low overhead and provides consistent performance.

- Continuously validates that data is in sync on all replica.
- Always running but low impact on cluster performance
- Fully automatic, no manual intervention needed
- Completely replace anti-entropy repairs

### NodeSync service

By default, each node runs the NodeSync service. The service is idle unless it has something to validate. NodeSync is enabled/disabled on each table individually. The service continuously validates local data ranges for NodeSync-enabled tables and repairs any inconsistency found. The local data ranges are split into small segments, which act as validation save points. Segments are **prioritized** in order to try to meet the per-table deadline target.

### Segments

A segment is a small local token range of a table. NodeSync recursively splits local ranges in half a certain number of times (depth) to create segments. The depth is calculated using the total table size, assuming equal distribution of data. Typically segments cover no more than 200 MB. The token ranges can be no smaller than a single partition, so very large partitions can result in segments larger than the configured size.

### Validation process and status

After a segment is selected for validation, NodeSync reads the entirety of the data it covers from all replica (using paging), checks for inconsistencies, and repairs if needed. When a node validates a segment, it “locks” it in a system table to avoid work duplication by other nodes. It is **not** a race-free lock; there is a possibility of duplicated work which saves the complexity and cost of true distributed locking.

Segment validation is saved on completion in the `system_distributed.nodesync_status` table, which is used internally for resuming on failure, prioritization, segment locking, and by tools. It is not meant to be read directly.

- Validation status is:

```
successful: All replicas responded and all inconsistencies (if any) were properly repaired.
```

```
full_in_sync: All replica were already in sync.
```

```

full_repaired: Some replica were repaired.

unsuccessful: Either some replicas did not respond or repairs on inconsistent
replicas failed.

partial_in_sync: Not all replica responded, but all respondents were in sync.

partial_repaired: Not all replica responded, some that responded were
repaired.

uncompleted: At most 1 node was available/responded; no validation
happened.

failed: Some unexpected errors occurred. (Check the node logs.)

```

**Note:** If validation of a large segment is interrupted, increase the amount of redundant work.

## Limitations

- For debugging/tuning, understanding of traditional repair will be mostly unhelpful, since NodeSync depends on the read repair path
- No special optimizations for remote DC - may perform poorly on particularly bad WAN links
- In aggregate, CPU consumption of NodeSync might exceed traditional repair
- NodeSync only makes internal adjustments to try to hit the configured rate - operators must ensure this configured throughput is sufficient to meet the gc\_grace\_seconds commitment and can be achieved by the hardware

## Starting and Stopping the NodeSync service

On each node start or stop the NodeSync service.

The NodeSync service automatically starts with the [dse cassandra \(page 800\)](#) command. You can manually start and stop the service on each node.

1. Verify the status of the NodeSync service:

```
nodetool nodesyncservice status
```

The output should indicate running.

```
The NodeSync service is running
```

2. Disable the NodeSync service:

```
nodetool nodesyncservice disable
```

**Note:** On the next restart of DataStax Enterprise (DSE), the NodeSync service will start up.

3. Verify the status of the NodeSync service:

```
nodetool nodesyncservice status
```

The output should indicate not running.

```
The NodeSync service is not running
```

## Enabling NodeSync validation

To continuously verify data consistency in the background without the need for anti-entropy repairs, enable NodeSync on one or more tables.

By default, NodeSync is disabled when a table is created. It is also disabled on tables that were migrated from earlier versions. To continuously verify data consistency in the background without the need for anti-entropy repairs, enable NodeSync on one or more tables.

**Note:** Data only needs to be validated if the table is in more than one datacenter or is in a datacenter where the keyspace has a replication factor of 2 or more.

- Enable on an existing table:

# Change the NodeSync setting on a single table using CQL syntax:

```
ALTER TABLE table_name WITH
nodesync={'enabled': 'true'};
```

# All tables in a keyspace using `nodesync enable`:

```
nodesync enable -v -k keyspace_name "*"
```

# A list of tables using `nodesync enable`:

```
nodesync enable keyspace_name.table_name keyspace_name.table_name
```

- Create a table with `nodesync enabled`:

```
CREATE TABLE table_name (column_list) WITH
nodesync={'enabled': 'true'};
```

## Tuning validations

Configure validation rate to meet target deadlines.

NodeSync tries to validate all tables within their respective deadlines, while respecting the configured rate limit. If a table is 10GB and has a `deadline_target_sec=10` and the [rate\\_in\\_kb \(page 83\)](#) is set to 1MB/sec, validation **will not** happen quickly enough.

Configure the rate and deadlines realistically, take data sizes into account and adapt with data growth.

NodeSync records warnings to the `system.log`, if it detects any of the following conditions:

- [rate\\_in\\_kb \(page 83\)](#) is too low to validate all tables within their deadline, even under ideal circumstances.

- [rate\\_in\\_kb \(page 83\)](#) cannot be sustained by the node (too high for the node load/hardware).

## Setting the rate

Estimate rate impacts and set rates.

### Estimating rate setting impact

The [rate\\_in\\_kb \(page 83\)](#) sets the per node rate of the local NodeSync service. It controls the maximum number of bytes per second used to validate data. There is a fundamental tradeoff between how fast NodeSync validates data and how many resources it consumes. The rate is a limit on the amount of resources used and a target that NodeSync tries to achieve by auto-tuning internals. The set rate might not be achieved in practice, because validation can complete at a slower rate on new or small cluster or the node might temporarily or permanently lack available resources.

### Initial rate setting

There is no strong requirement to keep all nodes validating at the same rate. Some nodes will simply validate more data than others. When setting the rate, use the simplest method first by using the defaults.

1. Check the [rate\\_in\\_kb \(page 83\)](#) setting within the `nodesync` section in the `cassandra.yaml` file.
2. Try increasing or decreasing the value at run time:

```
nodetool nodesyncservice setrate value_in_kb_sec
```

3. Check the configured rate.

```
nodetool nodesyncservice getrate
```

**Tip:** The configured rate is different from the effective [rate](#), which can be found in the NodeSync Service metrics.

### Simulating rates

When adjusting rates, use the [NodeSync rate simulator](#) to help determine the configuration settings by computing the rate necessary to validate all tables within their allowed deadlines.

Unfortunately, no perfect value exists because NodeSync also deals with many unknown or difficult to predict factors, such as:

- **Failures** - When a node fails, it does not participate in NodeSync validation while it is offline.
- **Temporary overloads** - During periods of overload, such as an unexpected events, nodes can not achieve the configured rate.
- **Data size variation** - The rate required to repair all tables within a fixed amount of time directly depends on the size of the data to validate, which is typically a moving target.

All these factors can impact the overall NodeSync rate. Therefore build safety margins within the configured rate. The NodeSyncServiceRate simulator helps to set the rate.

## Setting the deadline

Adjust maximum time between validations of the same data.

Each table with NodeSync enabled has a `deadline_target_sec` property. This is the target for the maximum time between 2 validations of the same data. As long as the deadline is met, all parts of the ring (for the table) are validated at least that often.

The deadline (`deadline_target_sec`) relates to grace period (`gc_grace_seconds`). The deadline should always be less than or equal to the grace period. As long as the deadline is met, no data is resurrected due to tombstone purging.

The deadline defaults to whichever is longer, the grace period or four hours. Typically an acceptable default, unless the table has a grace period of zero. For testing, the deadline value can be less than the grace period. Verify for a few weeks if a lower `gc_grace` value is realistic without taking risk before changing it.

NodeSync prioritizes segments in order to try to meet the deadline. The next segment to validate at any given time is the one the closest to missing its deadline. For example, if table 1 has half the deadline of table 2, table 1 validates approximately twice as often as table 2.

Use OpsCenter to get a graphical representation of the NodeSync validation status. See [Viewing NodeSync Status](#).

The syntax to change the per-table `nodesync` property:

```
ALTER TABLE table_name
WITH nodesync = { 'enabled': 'true',
 'deadline_target_sec': value };
```

## Manually starting validation

Force NodeSync to repair specific segments.

Force NodeSync to repair specific segments. Once a user validation is submitted, it takes precedence over normal NodeSync work. Normal work resumes automatically after the validation finishes.

This is an advanced tool. Usually, it is better to let NodeSync prioritize segments on its own.

- Submitting user validations:

```
nodesync validation submit keyspace_name.table_name
```

- Listing user validations:

```
nodesync validation list
```

- Canceling user validations:

```
nodesync validation cancel validation_id
```

## Configuring Virtual Nodes

Topics about setting up and enabling virtual nodes (vnodes).

### Virtual node (vnode) configuration

A description of virtual nodes (vnodes) and how to use them in different types of datacenters. Also steps for disabling vnodes.

Virtual nodes simplify many tasks in DataStax Enterprise, such as eliminating the need to determine the partition range (calculate and assign tokens), rebalancing the cluster when adding or removing nodes, and replacing dead nodes. For a complete description of virtual nodes and how they work, see [Virtual nodes](#).

DataStax Enterprise requires the same token architecture on all nodes in a datacenter. The nodes must all be vnode-enabled or single-token architecture. Across the entire cluster, datacenter architecture can vary. For example, a single cluster with:

- A transaction-only datacenter running OLTP.
- A single-token architecture analytics datacenter (no vnodes).
- A search datacenter with vnodes.

### Guidelines for using virtual nodes

Whether virtual nodes (vnodes) are enabled or disabled depends on the initial `cassandra.yaml` settings. There are two methods of distributing token ranges. DataStax recommends using the allocation algorithm. Use the same method on all systems in the datacenter.

#### Allocation algorithm

Optimizes token range distribution between nodes and racks in the datacenter based on the keyspace replication factor ([allocate\\_tokens\\_for\\_local\\_replication\\_factor \(page 85\)](#)) of the datacenter. Distributes the token ranges proportionately using the [num\\_tokens \(page 84\)](#) settings. All systems in the datacenter should have the same `num_token` settings unless the systems performance varies between systems. To distribute more of the workload to the higher performance hardware, increase the number of tokens for those systems.

The allocation algorithm efficiently balances the workload using fewer tokens; when systems are added to a datacenter, the algorithm maintains the balance. Using a higher number of tokens more evenly distributes the workload, but also significantly increases token management overhead.

**Caution:** When adding multiple nodes to the cluster using the allocation algorithm, ensure that nodes are added one at a time. If nodes are added concurrently, the algorithm assigns the same tokens to different nodes.

The recommended number of vnodes (tokens) depends on the workload. This distributes the workload between systems with a ~10% variance and has minimal impact on performance. Set the number of vnode tokens (`num_tokens`) based on the workload distribution requirements of the datacenter:

**Table 2: Allocation algorithm workload distribution variance**

| Replication factor | 4 vnode (tokens) | 8 vnode (tokens) | 64 vnode (tokens) | 128 vnode (tokens) |
|--------------------|------------------|------------------|-------------------|--------------------|
| 2                  | ~17.5%           | ~12.5%           | ~3%               | ~1%                |
| 3                  | ~14%             | ~10%             | ~2%               | ~1%                |
| 5                  | ~11%             | ~7%              | ~1%               | ~1%                |

## Enabling vnodes

In the `cassandra.yaml` file:

1. Uncomment [num\\_tokens \(page 84\)](#) and set the required number of tokens.
2. (Recommended) To use the allocation algorithm uncomment [allocate\\_tokens\\_for\\_local\\_replication\\_factor \(page 85\)](#) and set it to the target replication factor for the keyspaces in the datacenter. If the replication varies, alternate between the replication factor (RF) settings.
3. Comment out the [initial\\_token \(page 84\)](#) or leave unset.

To upgrade existing clusters to vnodes, see [Enabling virtual nodes on an existing production cluster](#).

## Disabling vnodes

**Important:** If you do not use vnodes, you must make sure that each node is responsible for roughly an equal amount of data. To ensure that each node is responsible for an equal amount of data, assign each node an [initial-token \(page 68\)](#) value and calculate the tokens for each datacenter as described in [Generating tokens](#).

1. In the `cassandra.yaml` file:
  - a. Comment out the [num\\_tokens \(page 84\)](#) and [allocate\\_tokens\\_for\\_local\\_replication\\_factor \(page 85\)](#).
  - b. Uncomment the [initial\\_token \(page 84\)](#) and set it to 1 or to the value of a [generated token](#) for a multi-node cluster.

# Using DataStax Enterprise advanced functionality

Information on using DSE Analytics, DSE Search, DSE Graph, DSEFS (DataStax Enterprise file system), and DSE Advanced Replication.

Information on using DSE Analytics, DSE Search, DSE Graph, DSEFS (DataStax Enterprise file system), and DSE Advanced Replication.

## DSE Analytics

DataStax Enterprise 6.0 Analytics includes integration with Apache Spark.

DataStax Enterprise (DSE) integrates real-time and batch operational analytics capabilities with an enhanced version of Apache Spark™. With DSE Analytics you can easily generate ad-hoc reports, target customers with personalization, and process real-time streams of data. The analytics toolset lets you write code once and then use it for both real-time and batch workloads.

## About DSE Analytics

Use DSE Analytics to analyze huge databases. DSE Analytics includes integration with Apache Spark.

DataStax Enterprise (DSE) integrates real-time and batch operational analytics capabilities with an enhanced version of Apache Spark™. With DSE Analytics you can easily generate ad-hoc reports, target customers with personalization, and process real-time streams of data. The analytics toolset lets you write code once and then use it for both real-time and batch workloads.

DSE Analytics jobs can use the DataStax Enterprise File System (DSEFS) to handle the large data sets typical of analytic processing. DSEFS replaces CFS (Cassandra File System).

### DSE Analytics features

#### No single point of failure

DSE Analytics supports a peer-to-peer, distributed cluster for running Spark jobs. Being peers, any node in the cluster can load data files, and any analytics node can assume the responsibilities of Spark Master.

#### Spark Master management

DSE Analytics provides automatic Spark Master management.

#### Analytics without ETL

Using DSE Analytics, you run Spark jobs directly against data in the database. You can perform real-time and analytics workloads at the same time without one workload affecting the performance of the other. Starting some cluster nodes as Analytics nodes and others as pure transactional real-time nodes automatically replicates data between nodes.

#### DataStax Enterprise file system ([DSEFS \(page 273\)](#))

DSEFS (DataStax Enterprise file system) is a fault-tolerant, general-purpose, distributed file system within DataStax Enterprise. It is designed for use cases that need to leverage a distributed file system for data ingestion, data staging, and state management for Spark Streaming applications (such as checkpointing or write-ahead logging). DSEFS is similar to HDFS, but avoids the deployment complexity and single point of failure typical of HDFS. DSEFS is HDFS-compatible and is designed to work in place of HDFS in Spark and other systems.

### **DSE Analytics Solo**

[DSE Analytics Solo \(page 174\)](#) datacenters are devoted entirely to DSE Analytics processing, for deployments that require separation of analytics jobs from transactional data.

### **Integrated security**

DSE Analytics uses the [advanced security features of DSE](#), simplifying configuration and deployment.

### **AlwaysOn SQL**

[AlwaysOn SQL \(page 245\)](#) is a highly-available service that provides JDBC and ODBC interfaces to applications accessing DSE Analytics data.

## **Setting the replication factor for analytics keyspace**

Guidelines and steps to set the replication factor for keyspace on DSE Analytics nodes.

Keyspaces and tables are automatically created when DSE Analytics nodes are started for the first time. The replication factor must be adjusted for these keyspace in order for the analytics features to work properly and to avoid data loss.

The keyspace used by DSE Analytics are the following:

- dse\_analytics
- dse\_leases
- dsefs
- "HiveMetaStore"

All analytics keyspace are initially created with the `SimpleStrategy` replication strategy and a replication factor (RF) of 1. Each of these must be updated in production environments to avoid data loss. After starting the cluster, alter the keyspace to use the `NetworkTopologyStrategy` replication strategy with an appropriate settings for the replication factor and datacenters. For most environments using DSE Analytics, a suitable replication factor will be either 3 or the cluster size, whichever is smaller.

For example, use a CQL statement to configure the `dse_leases` keyspace for a replication factor of 3 in both DC1 and DC2 datacenters using `NetworkTopologyStrategy`:

```
ALTER KEYSPACE dse_leases
WITH REPLICATION = {
 'class': 'NetworkTopologyStrategy',
 'DC1': '3',
 'DC2': '3'
};
```

The datacenter name used is case-sensitive. If needed, use the `dsetool status` command to confirm the exact datacenter spelling.

After adjusting the replication factor, `nodetool repair` must be run on each node in the affected datacenters. For example to repair the altered keyspace `dse_leases`:

```
nodetool repair -full dse_leases
```

Repeat the above steps for each of the analytics keyspaces listed above. For more information see [Changing keyspace replication strategy](#).

## DSE Analytics and Search integration

DSE SearchAnalytics clusters can use DSE Search queries within DSE Analytics jobs.

An integrated DSE SearchAnalytics cluster allows analytics jobs to be performed using [CQL queries](#). This integration allows finer-grained control over the types of queries that are used in analytics workloads, and improves performance by reducing the amount of data that is processed. However, a DSE SearchAnalytics cluster does not provide workload isolation and there are no detailed guidelines for provisioning and performance in production environments.

Nodes that are started in `SearchAnalytics` mode allow you to create analytics queries that use DSE Search indexes. These queries return RDDs that are used by Spark jobs to analyze the returned data.

The following code shows how to use a DSE Search query from the DSE Spark console.

```
val table = sc.cassandraTable("music", "solr")
 val result = table.select("id", "artist_name")
 .where("solr_query='artist_name:Miles*' ")
 .take(10)
```

You can use Spark `Datasets/DataFrames` instead of RDDs.

```
val table = spark.read.format("org.apache.spark.sql.cassandra")
 .options(Map("keyspace" -> "music", "table" -> "solr"))
 .load()
 val result =
 table.select("id", "artist_name").where("solr_query='artist_name:Miles*' ")
 .show(10)
```

You may alternately use a Spark SQL query.

```
val result = spark.sql("SELECT id, artist_name FROM music.solr WHERE
 solr_query = 'artist_name:Miles*' LIMIT 10")
```

For a detailed example, see [Running the Wikipedia demo with SearchAnalytics \(page 267\)](#).

### Configuring a DSE SearchAnalytics cluster

1. Create DSE SearchAnalytics nodes in a mixed-workload cluster, as described in [Initializing a single datacenter per workload type](#).

The name of the datacenter is set to `SearchAnalytics` when using the `DseSimpleSnitch`. Do not modify existing search or analytics nodes that use

`DseSimpleSnitch` to be `SearchAnalytics` nodes. If you use another snitch like `GossipingPropertyFileSnitch` you can have a mixed workload within a datacenter.

2. Perform load testing to ensure your hardware has enough CPU and memory for the additional resource overhead that is required by Spark and Solr.

**Note:** `SearchAnalytics` nodes always use driver paging settings. See [Using pagination \(cursors\) with CQL Solr queries](#).

`SearchAnalytics` nodes might consume more resources than search or analytics nodes. Resource requirements of the nodes greatly depend on the type of query patterns you are using.

## Considerations for DSE `SearchAnalytics` clusters

Care should be taken when enabling both Search and Analytics on a DSE node. Since both workloads will be enabled, ensure proper resources are provisioned for these simultaneous workloads. This includes sufficient memory and compute resources to accommodate the specific indexing, query, and processing appropriate to the use case.

`SearchAnalytics` clusters are appropriate for production environments, provided these environments provide sufficient resources for the specific workload, as is the case for all DSE clusters.

All of the fields that are queried on DSE `SearchAnalytics` clusters must be defined in the [search index schema definition \(page 315\)](#). Fields that are not defined in the search index schema columns are excluded from the results returned from Spark queries.

## Using predicate push down on search indexes in Spark SQL

Search predicate push down allows queries in `SearchAnalytics` datacenters to use Solr-indexed columns in Spark SQL queries.

Search predicate push down allows queries in `SearchAnalytics` datacenters to use Solr-indexed columns in Spark SQL queries. To enable Search predicate push down, set the `spark.sql.dse.search.enabled` property to `on` or `auto`. By default, `spark.sql.dse.search.enabled` is set to `auto`.

When in `auto` mode the predicate push down will do a COUNT operation against the Search indices both with and without the predicate filters applied. If the number of records with the predicate filter is less than the result of the following formula:

```
spark.sql.dse.search.auto_ratio * the total number of records
```

the optimization occurs automatically.

The property `spark.sql.dse.search.auto_ratio` is user configurable. The default value is 0.03.

The performance of DSE Search is directly related to the number of records returned in a query. Requests which require a large portion of the dataset are likely better served by a full table scan without using predicate push downs.

To enable Solr predicate push down on a Scala dataset:

```
val solrEnabledDataSet = spark.read
```

```
.format("org.apache.spark.sql.cassandra")
.options(Map(
 "keyspace" -> "ks",
 "table" -> "tab",
 "spark.sql.dse.search.enabled" -> "on"))
.load()
```

To create a temporary table in Spark SQL with Solr predicate push down enabled:

```
CREATE TEMPORARY TABLE temp USING org.apache.spark.sql.cassandra OPTIONS
(
 table "tab",
 keyspace "ks",
 spark.sql.dse.search.enabled "on");
```

Set the `spark.sql.dse.search.enabled` property globally by adding it to the [server configuration file \(page 216\)](#).

The optimizer works on the push down level so only predicates which are being pushed to the source can be optimized. Use the `explain` command to see exactly what predicates are being pushed to the `CassandraSourceRelation`.

```
val query = spark.sql("query")
query.explain
```

## Logging optimization plans

The optimization plans for a query using predicate push downs are logged by setting the `org.apache.spark.sql.SolrPredicateRules` logger to `DEBUG` in the [Spark logging configuration files \(page 221\)](#).

```
<logger name="org.apache.spark.sql.SolrPredicateRules" level="DEBUG" />
```

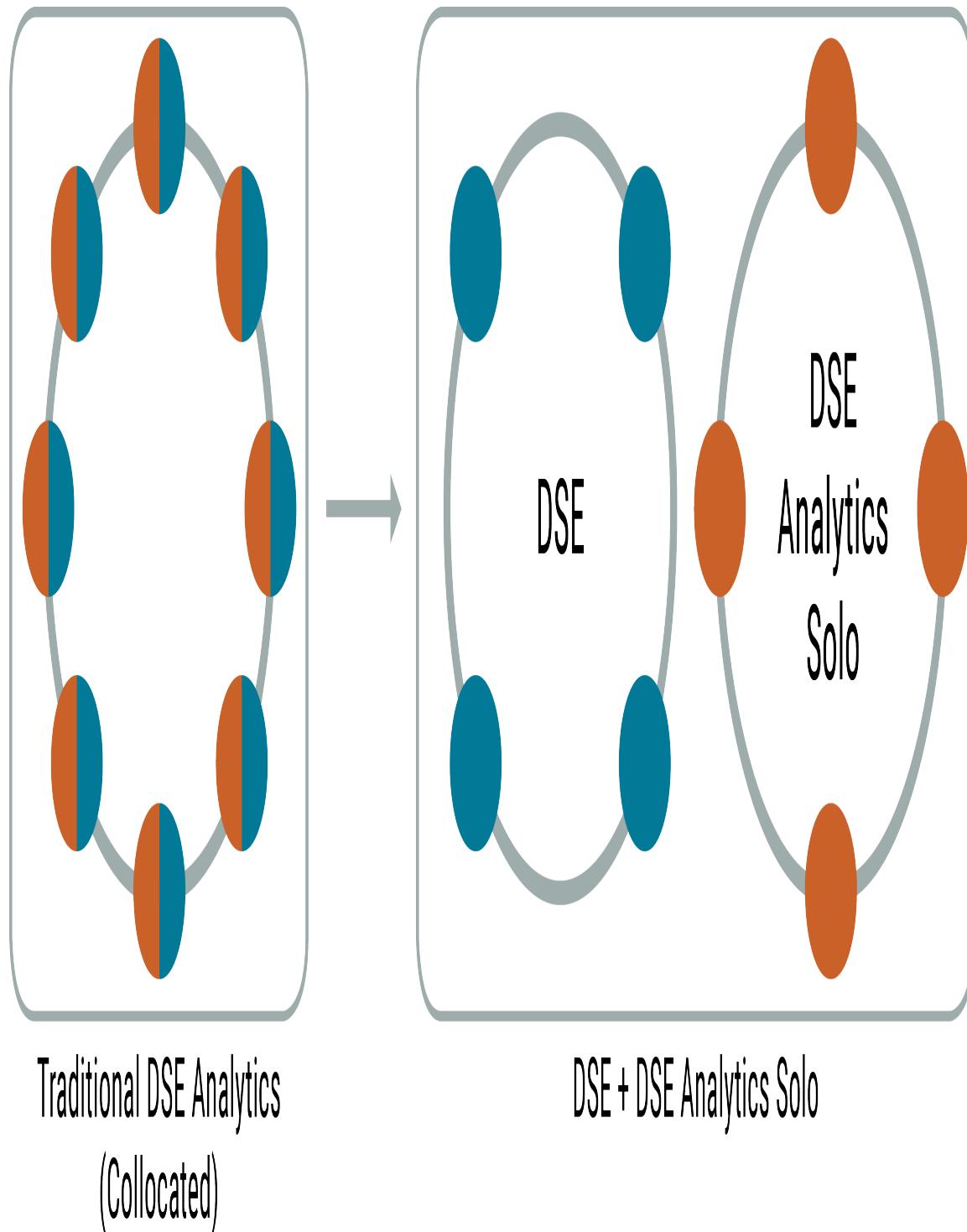
## About DSE Analytics Solo

DSE Analytics Solo datacenters provide analytics processing with Spark and distributed storage using DSEFS without storing transactional database data.

DSE Analytics Solo datacenters provide analytics processing with Spark and distributed storage using DSEFS without storing transactional database data.

DataStax Enterprise is flexible when deploying analytic processing in concert with transactional workloads. There are two main ways to deploy DSE Analytics: collocated with the database processing nodes, and on segregated machines in their own datacenter.

Figure 1: Traditional and DSE Analytics Solo deployments



Traditional DSE Analytics deployments have both the DataStax database process and the Spark process running on the same machine. This allows for simple deployment of analytic processing when the analysis is not as intensive, or the database is not as heavily used.

DSE Analytics Solo allows customers to deploy DSE Analytics processing on segregated hardware configurations in a different datacenter from the transactional DSE nodes. This ensures consistent behavior of both engines in a configuration that does not compete for computer resources. This configuration is good for processing-intensive analytic workloads.

DSE Analytics Solo allows the flexibility to have more nodes dedicated to data processing than are used for database transactions. This is particularly good for situations where the processing needs far exceed the transactional resource needs. For example, suppose you have a Spark Streaming job that will analyze and filter 99.9% of the incoming data, storing only a few records after analysis. The resources required by the transactional datacenter are much smaller than the resources required to analyze the data.

DSE Analytics Solo is more elastic in terms of scaling up, or down, the analytic processing in the cluster. This is particularly useful when you need extra analytics processing, such as end of the day or end of the quarter surges in analytics jobs. Since a DSE Analytics Solo node does not store database data, when new nodes are added to a cluster there is very little data moved across the network to the new nodes. In an analytics and transactional collocated environment, adding a node means moving transactional data between the existing nodes and the new nodes.

For information on creating a DSE Analytics Solo datacenter, see [Creating a DSE Analytics Solo datacenter \(page 229\)](#).

## Analyzing data using Spark

Spark is the default mode when you start an analytics node in a packaged installation. Spark runs locally on each node.

Spark is the default mode when you start an analytics node in a packaged installation.

### About Spark

Information about Spark architecture and capabilities.

Apache Spark is a framework for analyzing large data sets across a cluster, and is enabled when you start an Analytics node. Spark runs locally on each node and executes in memory when possible. Spark uses multiple threads instead of multiple processes to achieve parallelism on a single node, avoiding the memory overhead of several JVMs.

Apache Spark integration with DataStax Enterprise includes:

- [Spark Cassandra Connector \(page 204\)](#) for accessing data stores in DSE
- [DSE Resource Manager for managing \(page 208\)](#) Spark components in a DSE cluster
- [Spark Job Server \(page 258\)](#)
- [Spark SQL \(page 235\)](#) support
- [AlwaysOn SQL \(page 245\)](#)
- [Spark SQL Thrift Server \(page 243\)](#)
- [Spark streaming \(page 232\)](#)
- [DataFrames \(page 242\)](#) API to manipulate data within Spark
- [SparkR integration \(page 245\)](#)

## Spark architecture

The software components for a single DataStax Enterprise analytics node are:

- Spark Worker
- DataStax Enterprise File System (DSEFS)
- The database

A Spark Master acts purely as a resource manager for Spark applications. Spark Workers launch executors that are responsible for executing part of the job that is submitted to the Spark Master. Each application has its own set of executors. Spark architecture is described in the [Apache documentation](#).

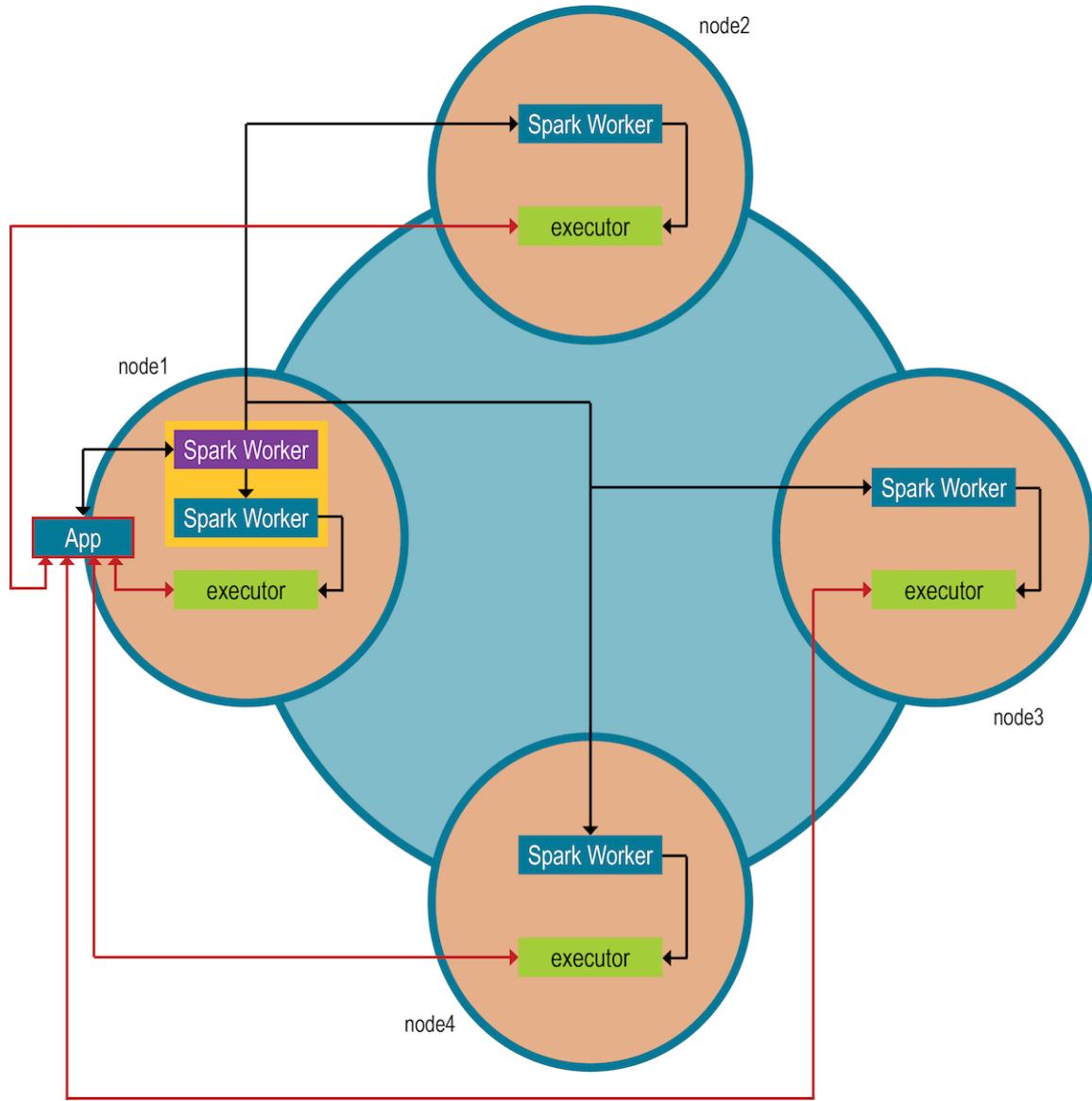
DSE Spark nodes use a different resource manager than standalone Spark nodes. The DSE Resource Manager simplifies integration between Spark and DSE. In a DSE Spark cluster, client applications use the CQL protocol to connect to any DSE node, and that node redirects the request to the Spark Master.

The communication between the Spark client application (or driver) and the Spark Master is secured the same way as connections to DSE, which means that plain password authentication as well as Kerberos authentication is supported, with or without SSL encryption. Encryption and authentication can be configured per application, rather than per cluster. Authentication and encryption between the Spark Master and Worker nodes can be enabled or disabled regardless of the application settings.

Spark supports multiple applications. A single application can spawn multiple jobs and the jobs run in parallel. An application reserves some resources on every node and these resources are not freed until the application finishes. For example, every session of Spark shell is an application that reserves resources. By default, the scheduler tries to allocate the application to the highest number of different nodes. For example, if the application declares that it needs four cores and there are ten servers, each offering two cores, the application most likely gets four executors, each on a different node, each consuming a single core. However, the application can get also two executors on two different nodes, each consuming two cores. You can configure the application scheduler. Spark Workers and Spark Master are part of the main DSE process. Workers spawn executor JVM processes which do the actual work for a Spark application (or driver). Spark executors use native integration to access data in local transactional nodes through the [Open Source Spark-Cassandra Connector](#). The memory settings for the executor JVMs are set by the user submitting the driver to DSE.

In deployment for each Analytics datacenter one node runs the Spark Master, and Spark Workers run on each of the nodes. The Spark Master comes with [automatic high availability \(page 178\)](#).

Figure 2: Spark integration with DataStax Enterprise



As you run Spark, you can access data in the Hadoop Distributed File System (HDFS), or the DataStax Enterprise File System (DSEFS) by using the URL for the respective file system.

### Highly available Spark Master

The Spark Master High Availability mechanism uses a special table in the `dse_analytics` keyspace to store information required to recover Spark workers and the application. Unlike the high availability mechanism mentioned in Spark documentation, DataStax Enterprise does not use ZooKeeper.

If the original Spark Master fails, the reserved one automatically takes over. To find the current Spark Master, run:

```
dse client-tool spark leader-address
```

DataStax Enterprise provides [Automatic Spark Master management \(page 217\)](#).

## Unsupported features

The following Spark features and APIs are not supported:

- Writing to blob columns from Spark

Reading columns of all types is supported; however, you must convert collections of blobs to byte arrays before serializing.

## Using Spark with DataStax Enterprise

DataStax Enterprise integrates with Apache Spark to allow distributed analytic applications to run using database data.

DataStax Enterprise integrates with Apache Spark to allow distributed analytic applications to run using database data.

## Starting Spark

How you start Spark depends on the installation and if want to run in Spark mode or SearchAnalytics mode:

Before you start Spark, set [Authorizing remote procedure calls \(RPC\)](#) for the `DseClientTool` object.

**Note:** RPC permission for the `DseClientTool` object is required to run Spark because the `DseClientTool` object is called implicitly by the Spark launcher.

**Note:** By default DSEFS is required to execute Spark applications. DSEFS should not be disabled when Spark is enabled on a DSE node. If there is a strong reason not to use DSEFS as the default file system, reconfigure Spark to use a different file system. For example to use a local file system set the following properties in `spark-daemon-defaults.conf`:

```
spark.hadoop.fs.defaultFS=file:///
spark.hadoop.hive.metastore.warehouse.dir=file:///tmp/warehouse
```

How you start Spark depends on the installation and if you want to run in Spark mode or SearchAnalytics mode:

### Package installations:

To start the Spark trackers on a cluster of analytics nodes, edit the `/etc/default/dse` file to set **SPARK\_ENABLED** to 1.

When you [start DataStax Enterprise as a service \(page 867\)](#), the node is launched as a Spark node. You can enable additional components.

Mode	Option in /etc/default/dse	Description
Spark	SPARK_ENABLED=1	Start the node in Spark mode.
SearchAnalytics mode	SPARK_ENABLED=1 SEARCH_ENABLED=1	SearchAnalytics mode requires testing in your environment before it is used in production clusters. In dse.yaml, <a href="#">cql_solr_query_paging: driver (page 120)</a> is required.

**Tarball installations:**

To start the Spark trackers on a cluster of analytics nodes, use the **-k** option:

```
installation_location/bin/dse cassandra -k
```

**Note:** Nodes started with **-k** are automatically assigned to the default Analytics datacenter if you do not configure a datacenter in the snitch property file.

You can enable additional components:

Mode	Option	Description
Spark	<b>-k</b>	Start the node in Spark mode.
SearchAnalytics mode	<b>-k -s</b>	In dse.yaml, <a href="#">cql_solr_query_paging: driver (page 120)</a> is required.

For example:

To start a node in SearchAnalytics mode, use the **-k** and **-s** options.

```
installation_location/bin/dse cassandra -k -s
```

or

```
installation_location/bin/dse cassandra -ks
```

Starting the node with the Spark option starts a node that is designated as the master, as shown by the Analytics(SM) workload in the output of the `dsetool ring` command:

```
dsetool ring
Address DC Rack Workload
Graph Status State Load Owns Token
 Health [0,1]

 0
10.200.175.149 Analytics rack1 Analytics(SM)
 no Up Normal 185 KiB ?
-9223372036854775808 0.90
10.200.175.148 Analytics rack1 Analytics(SW)
 no Up Normal 194.5 KiB ?
 0.90
Note: you must specify a keyspace to get ownership information.
```

If you use `sudo` to start DataStax Enterprise, remove the `~./spark` directory before you restart the cluster:

```
sudo rm -r ~/.spark
```

## Launching Spark

After starting a Spark node, use `dse` commands to launch Spark.

Usage:

Package installations: `dse spark`

Tarball installations: `installation_location/bin/ dse spark`

You can use [Cassandra specific properties \(page 226\)](#) to start Spark. Spark binds to the `listen_address` that is specified in `cassandra.yaml`.

DataStax Enterprise supports these commands for launching Spark on the DataStax Enterprise command line:

### `dse spark`

Enters interactive Spark shell, offers basic auto-completion.

Package installations: `dse spark`

Tarball installations: `installation_location/bin/ dse spark`

### `dse spark-submit`

Launches applications on a cluster like `spark-submit`. Using this interface you can use Spark cluster managers without the need for separate configurations for each application. The syntax for package installations is:

```
dse spark-submit --class class_name jar_file other_options
```

For example, if you write a class that defines an option named `d`, enter the command as follows:

```
dse spark-submit --class com.datastax.HttpSparkStream target/
HttpSparkStream.jar -d $NUM_SPARK_NODES
```

**Note:** The JAR file can be located in a DSEFS directory. If the DSEFS cluster is secured, provide authentication credentials as described in [DSEFS authentication \(page 291\)](#).

The `dse spark-submit` command supports the [same options](#) as Apache Spark's `spark-submit`. For example, to submit an application using cluster mode using the `supervise` option to restart in case of failure:

```
dse spark-submit --deploy-mode cluster --supervise --class
com.datastax.HttpSparkStream target/HttpSparkStream.jar -d
$NUM_SPARK_NODES
```

**Note:** The directory in which you run the `dse` Spark commands must be writable by the current user.

Internal authentication is supported.

Use the optional environment variables `DSE_USERNAME` and `DSE_PASSWORD` to increase security and prevent the user name and passwords from appearing in the Spark log files or in the process list on the Spark Web UI. To specify a user name and password using environment variables, add the following to your Bash `.profile` or `.bash_profile`:

```
export DSE_USERNAME=user
export DSE_PASSWORD=secret
```

These environment variables are supported for all Spark and [dse client-tool \(page 799\)](#) commands.

**Note:** DataStax recommends using the environment variables instead of passing user credentials on the command line.

You can provide authentication credentials in several ways, see [Credentials for authentication](#).

## Specifying Spark URLs

You do not need to specify the Spark Master address when starting Spark jobs with DSE. If you connect to any Spark node in a datacenter, DSE will automatically discover the Master address and connect the client to the Master.

Specify the URL for any Spark node using the following format:

```
dse://[Spark node address[:port number]]?[parameter name=parameter value;]...
```

By default the URL is `dse://?`, which is equivalent to `dse://localhost:9042`. Any parameters you set in the URL will override the configuration read from DSE's Spark configuration settings.

You can specify the work pool in which the application will be run by adding the `workpool=work pool name` as a URL parameter. For example, `dse://1.1.1.1:123?workpool=workpool2`.

Valid parameters are `CassandraConnectorConf` [settings](#) with the `spark.cassandra.` prefix stripped. For example, you can set the `spark.cassandra.connection.local_dc` option to `dc2` by specifying `dse://?connection.local_dc=dc2`.

Or to specify multiple `spark.cassandra.connection.host` addresses for high-availability if the specified connection point is down: `dse://1.1.1.1:123?connection.host=1.1.2.2,1.1.3.3`.

If the `connection.host` parameter is specified, the host provided in the standard URL is prepended to the list of hosts set in `connection.host`. If the port is specified in the standard URL, it overrides the port number set in the `connection.port` parameter.

Connection options when using `dse spark-submit` are retrieved in the following order: from the Master URL, then the Spark Cassandra Connector options, then the DSE configuration files.

## Detecting Spark application failures

DSE has a failure detector for Spark applications, which detects whether a running Spark application is dead or alive. If the application has failed, the application will be removed from the DSE Spark Resource Manager.

The failure detector works by keeping an open TCP connection from a DSE Spark node to the Spark Driver in the application. No data is exchanged, but regular TCP connection keep-alive control messages are sent and received. When the connection is interrupted, the failure detector will attempt to reacquire the connection every 1 second for the duration of the `appReconnectionTimeoutSeconds` timeout value (5 seconds by default). If it fails to reacquire the connection during that time, the application is removed.

A custom timeout value is specified by adding `appReconnectionTimeoutSeconds=value` in the master URI when submitting the application. For example to set the timeout value to 10 seconds:

```
dse spark --master dse://?appReconnectionTimeoutSeconds=10
```

## Running Spark commands against a remote cluster

To run Spark commands against a remote cluster, you must export the DSE configuration from one of the remote nodes to the local client machine.

To run Spark commands against a remote cluster, you must export the DSE configuration from one of the remote nodes to the local client machine.

To run a driver application remotely, there must be full public network communication between the remote nodes and the client machine.

1. Export the DataStax Enterprise client configuration from the remote node to the client node:

- a. On the remote node:

```
dse client-tool configuration export dse-config.jar
```

- b. Copy the exported JAR to the client nodes.

```
scp dse-config.jar user@clientnode1.example.com:
```

- c. On the client node:

```
dse client-tool configuration import dse-config.jar
```

2. Run the Spark command against the remote node.

```
dse spark-submit submit options myApplication.jar
```

To set the driver host to a publicly accessible IP address, pass in the `spark.driver.host` option.

```
dse spark-submit --conf spark.driver.host=IP address
myApplication.jar
```

## Accessing database data from Spark

DataStax Enterprise integrates Spark with DataStax Enterprise database. Database tables are fully usable from Spark.

DataStax Enterprise integrates Spark with DataStax Enterprise database. Database tables are fully usable from Spark.

## Accessing the database from a Spark application

To access the database from a Spark application, follow instructions in the Spark example [Portfolio Manager demo using Spark \(page 260\)](#).

## Accessing database data from the Spark shell

DataStax Enterprise uses the Spark Cassandra Connector to provide database integration for Spark. By running the Spark shell in DataStax Enterprise, you have access to enriched Spark context objects for accessing transactional nodes directly. See the Spark Cassandra Connector [Java Doc](#) on GitHub.

To access database data from the Spark Shell, just run the `dse spark` command and follow instructions in subsequent sections.

The Spark Shell creates a default Spark session named `spark`, an instance of `org.apache.spark.sql.SparkSession`.

The Spark Shell creates two contexts by default: `sc` (an instance of `org.apache.spark.SparkContext`) and `sqlContext` (an instance of `org.apache.spark.sql.hive.HiveContext`).

**Note:**

In previous versions of DSE, the default `HiveContext` instance was named `hc`. If your application uses `hc` instead of `sqlContext`, you can work around this change by adding a line:

```
val hc = sqlContext
```

Previous versions also created a `CassandraSqlContext` instance named `csc`. Starting in DSE 5.0, this is no longer the case. Use the `sqlContext` object instead.

## Using the Spark session

The Spark session object is the primary entry point for Spark applications, and allows you to run SQL queries on database tables.

A Spark session is encapsulated in an instance of `org.apache.spark.sql.SparkSession`. The session object has information about the Spark Master, the Spark application, and the configuration options.

The DSE Spark shell automatically configures and creates a Spark session object named `spark`. Use this object to begin querying database tables in DataStax Enterprise.

```
spark.sql("SELECT * FROM keyspace.table_name")
```

**Note:**

In Spark 1.6 and earlier, there were separate `HiveContext` and `SQLContext` objects. Starting in Spark 2.0, the `SparkSession` encapsulates both.

Spark applications can use multiple sessions to use different underlying data catalogs. You can use an existing Spark session to create a new session by calling the `newSession` method.

```
val newSpark = spark.newSession()
```

## Building a Spark session using the Builder API

The Builder API allows you to create a Spark session manually.

```
import org.apache.spark.sql.SparkSession
val sparkSession = SparkSession.builder
 .master("dse://localhost?")
 .appName("my-spark-app")
 .enableHiveSupport()
 .config("spark.executor.logs.rolling.maxRetainedFiles", "3")
 .config("spark.executor.logs.rolling.strategy", "size")
 .config("spark.executor.logs.rolling.maxSize", "50000")
```

```
.getOrCreate
```

## Stopping a Spark session

Use the `stop` method to end the Spark session.

```
spark.stop
```

## Getting and setting configuration options

Use the `spark.conf.get` and `spark.conf.set` methods to retrieve or set Spark configuration options for the session.

```
spark.conf.set("spark.executor.logs.rolling.maxRetainedFiles", "3")
spark.conf.get("spark.executor.logs.rolling.maxSize")
```

## Using the Spark context

To get a Spark RDD that represents a database table, load data from a the table into Spark using the sc-dot (sc.) syntax to call the `cassandraTable` method on the Spark context.

**Note:** Starting in DSE 5.1, the entry point for Spark applications is the [SparkSession object \(page 185\)](#). Using the Spark context directly is deprecated and may be removed in future releases.

Access the deprecated context object, call `spark.sparkContext`.

```
val sc = spark.sparkContext
```

To get a Spark RDD that represents a database table, load data from a table into Spark using the sc-dot (sc.) syntax to call the `cassandraTable` method on the Spark context, where `sc` represents the Spark API `SparkContext` class.

```
sc.cassandraTable("keyspace", "table name")
```

By default, the DSE Spark shell creates an `sc` object. The Spark context can be manually retrieved from the `SparkSession` object in the Spark shell by calling `spark.sparkContext`.

```
val sc = spark.sparkContext()
```

Data is mapped into Scala objects and DataStax Enterprise returns a `CassandraRDD[CassandraRow]`. To use the Spark API for creating an application that runs outside DataStax Enterprise, import `com.datastax.spark.connector.SparkContextCassandraFunctions`.

The following example shows how to load a table into Spark and read the table from Spark.

1. Create this keyspace and table in using `cqlsh`. Use the Analytics datacenter to create the keyspace.

```

CREATE KEYSPACE test WITH REPLICATION = {'class' :
 'NetworkTopologyStrategy', 'Analytics' : 1};

CREATE TABLE test.words (word text PRIMARY KEY, count int);

```

This example assumes you start a single-node cluster in [Spark mode \(page 179\)](#).

**2.** Load data into the words table.

```

INSERT INTO test.words (word, count) VALUES ('foo', 10);
INSERT INTO test.words (word, count) VALUES ('bar', 20);

```

**3.** Assuming you started the node in Spark mode, start the Spark shell. Do not use `sudo` to start the shell.

```
bin/dse spark
```

The Welcome to Spark output and prompt appears.

**4.** Use the `:showSchema` command to view the user keyspaces and tables.

```
:showSchema
```

Information about all user keyspaces appears.

```

=====
Keyspace: HiveMetaStore
=====
Table: MetaStore

- key : String (partition key column)
- entity : String (clustering column)
- value : java.nio.ByteBuffer

=====
Keyspace: test
=====
Table: words

- word : String (partition key column)
- count : Int

```

```
:showSchema test
```

```

=====
Keyspace: test
=====
Table: words

- word : String (partition key column)
- count : Int

```

```
:showSchema test words

=====
Keyspace: test
=====
Table: words

- word : String (partition key column)
- count : Int
```

5. Get information about only the `test` keyspace.

```
:showSchema test

=====
Keyspace: test
=====
Table: words

- word : String (partition key column)
- count : Int
```

6. Get information about the `words` table.

```
:showSchema test words

=====
Keyspace: test
=====
Table: words

- word : String (partition key column)
- count : Int
```

7. Define a base RDD to point to the data in the `test.words` table.

```
val rdd = sc.cassandraTable("test", "words")

rdd:
com.datastax.spark.connector.rdd.CassandraRDD[com.datastax.spark.connector.CassandraRow] = CassandraRDD[0] at RDD at CassandraRDD.scala:47
```

The RDD is returned in the `rdd` value. To read the table, use this command.

```
rdd.toArray.foreach(println)

CassandraRow{word: bar, count: 20}
CassandraRow{word: foo, count: 10}
```

Now, you can use methods on the returned RDD to query the `test.words` table.

## Python support for loading cassandraTables

Python supports loading `cassandraTables` from a Spark streaming context and saving a `DStream` to the database.

### Reading column values

You can read columns in a table using the `get` methods of the `CassandraRow` object. The `get` methods access individual column values by column name or column index. Type conversions are applied on the fly. Use `getOption` variants when you expect to receive null values.

Continuing with the previous example, follow these steps to access individual column values.

1. Store the first item of the RDD in the `firstRow` value.

```
val firstRow = rdd.first

firstRow: com.datastax.spark.connector.CassandraRow =
CassandraRow{word: foo, count: 10}
```

2. Get the column names.

```
rdd.columnNames

res3: com.datastax.spark.connector.ColumnSelector = AllColumns
```

3. Use a generic `get` to query the table by passing the return type directly.

```
firstRow.get[Int]("count")

res4: Int = 10

firstRow.get[Long]("count")

res5: Long = 10

firstRow.get[BigInt]("count")

res6: BigInt = 10

firstRow.get[java.math.BigInteger]("count")

res7: java.math.BigInteger = 10

firstRow.get[Option[Int]]("count")

res8: Option[Int] = Some(10)
```

```
firstRow.get[Option[BigInt]]("count")
res9: Option[BigInt] = Some(10)
```

## Reading collections

You can read collection columns in a table using the get methods of the `CassandraRow` object. The get methods access the collection column and returns a corresponding Scala collection.

Assuming you set up the `test` keyspace earlier, follow these steps to access a collection.

1. In the `test` keyspace, set up a collection set using `cqlsh`.

```
CREATE TABLE test.users (
 username text PRIMARY KEY, emails SETtext);

INSERT INTO test.users (username, emails)
 VALUES ('someone', {'someone@email.com', 's@email.com'});
```

2. If Spark is not running, start the Spark shell. Do not use `sudo` to start the shell.

```
bin/dse spark
```

The Welcome to Spark output and prompt appears.

3. Define a `CassandraRDD[CassandraRow]` to access the collection set.

```
val row = sc.cassandraTable("test", "users").toArray.apply(0)

row: com.datastax.spark.connector.CassandraRow =
 CassandraRow{username: someone,
 emails: {s@email.com,someone@email.com}}
```

4. Query the collection set from Spark.

```
row.getList[String]("emails")

res2: Vector[String] = Vector(s@email.com, someone@email.com)

row.get[List[String]]("emails")

res3: List[String] = List(s@email.com, someone@email.com)

row.get[Seq[String]]("emails")

res4: Seq[String] = List(s@email.com, someone@email.com)

row.get[IndexedSeq[String]]("emails")

res5: IndexedSeq[String] = Vector(s@email.com, someone@email.com)
```

```

row.get[Set[String]]("emails")

res6: Set[String] = Set(s@email.com, someone@email.com)

row.get[String]("emails")

res7: String = {s@email.com, someone@email.com}

```

## Restricting the number of fetched columns

For performance reasons, you should not fetch columns you don't need. You can achieve this with the `select` method:

To restrict the number of fetched columns:

```

val row = sc.cassandraTable("test",
 "users").select("username").toArray

row: Array[com.datastax.spark.connector.CassandraRow] =
 Array(CassandraRow{username: someone})

```

## Mapping rows to tuples and case classes

Instead of mapping your rows to objects of the `CassandraRow` class, you can directly unwrap column values into tuples of the desired type.

To map rows to tuples:

```

sc.cassandraTable[(String, Int)]("test", "words").select("word",
 "count").toArray

res9: Array[(String, Int)] = Array((bar,20), (foo,10))

sc.cassandraTable[(Int, String)]("test", "words").select("count",
 "word").toArray

res10: Array[(Int, String)] = Array((20,bar), (10,foo))

```

Define a `case` class with properties of the same name as the columns. For multi-word column identifiers, separate each word using an underscore when creating the columns, and use [camel case](#) abbreviation on the Scala side.

To map rows to `case` classes:

```

case class WordCount(word: String, count: Int)

defined class WordCount

sc.cassandraTable[WordCount]("test", "words").toArray

res14: Array[WordCount] = Array(WordCount(bar,20), WordCount(foo,20))

```

You can name columns using these conventions:

- Use the underscore convention and lowercase letters. (Recommended)
- Use the camel case convention, exactly the same as properties in Scala.

The following examples show valid column names.

**Table 3: Recommended naming convention**

Database column name	Scala property name
count	count
column_1	column1
user_name	userName
user_address	UserAddress

**Table 4: Alternative naming convention**

Database column name	Scala property name
count	count
column1	column1
userName	userName
UserAddress	UserAddress

### Mapping rows to objects with a user-defined function

Invoke `as` on the `CassandraRDD` to map every row to an object of a different type. Contrary to `map`, `as` expects a function having the same number of arguments as the number of columns to be fetched. Invoking `as` in this way performs type conversions. Using `as` to directly create objects of a particular type eliminates the need to create `CassandraRow` objects and also decreases garbage collection pressure.

To map columns using a user-defined function:

```
val table = sc.cassandraTable("test", "words")

table:
 com.datastax.spark.connector.rdd.CassandraRDD[com.datastax.spark.connector.
CassandraRow] = CassandraRDD[9] at RDD at CassandraRDD.scala:47

val total = table.select("count").as((c: Int) => c).sum

total: Double = 30.0

val frequencies = table.select("word", "count").as((w: String, c: Int)
 => (w, c / total)).toArray
```

```
frequencies: Array[(String, Double)] = Array((bar,0.6666666666666666),
(foo,0.3333333333333333))
```

## Filtering rows on the server

To filter rows, you can use the filter transformation provided by Spark. Filter transformation fetches all rows from the database first and then filters them in Spark. Some CPU cycles are wasted serializing and de-serializing objects excluded from the result. To avoid this overhead, `CassandraRDD` has a method that passes an arbitrary CQL condition to filter the row set on the server.

This example shows how to use Spark to filter rows on the server.

1. [Download](#) and unzip the CQL commands for this example. The commands in this file perform the following tasks:

- Create a `cars` table in the `test` keyspace.
- Index the color column.
- Insert some data into the table

2. Run the `test_cars.cql` file using `cqlsh` or DevCenter. For example using `cqlsh`:

```
cqlsh -f test_cars.cql
```

3. Filter the rows using Spark:

```
sc.cassandraTable("test", "cars").select("id",
 "model").where("color = ?", "black").toArray.foreach(println)
```

```
CassandraRow{id: AS-8888, model: Aston Martin DB9 Volante}
CassandraRow{id: KF-334L, model: Ford Mondeo}
CassandraRow{id: MT-8787, model: Hyundai x35}
CassandraRow{id: MZ-1038, model: Mazda CX-9}
CassandraRow{id: DG-2222, model: Dodge Avenger}
CassandraRow{id: DG-8897, model: Dodge Charger}
CassandraRow{id: BT-3920, model: Bentley Continental GT}
CassandraRow{id: IN-9964, model: Infinity FX}
```

```
sc.cassandraTable("test", "cars").select("id",
 "model").where("color = ?", "silver").toArray.foreach(println)
```

```
CassandraRow{id: FR-8877, model: Ferrari FF}
CassandraRow{id: FR-8877, model: Ferrari FF}
CassandraRow{id: HD-1828, model: Honda Accord}
CassandraRow{id: WX-2234, model: Toyota Yaris}
```

## Controlling automatic direct join optimizations in queries

DSE can optimize join queries to directly lookup data in the database without performing a Spark shuffle, which uses a full table scan.

DSE can optimize join queries to directly lookup data in the database without performing a Spark shuffle, which uses a full table scan.

By default, this optimization is turned on. Direct joins are used when:

```
(table size * direct_join_size_ratio) > size of keys
```

The value of `direct_join_size_ratio` should be between 0 and 1. By default, this value is 0.9. The `direct_join_size_ratio` setting can be set when creating the reference to the database table or in the Spark Session.

```
spark.conf.set("direct_join_size_ratio", 0.2)
```

You can permanently enable or disable this optimization by setting the `direct_join_setting` option. Valid settings for `direct_join_setting` are:

- `on` to permanently enable the optimization
- `off` to permanently disable the optimization
- `auto` (the default value) to let DSE determine when to enable it according to the criteria from the `direct_join_size_ratio` setting

You can programmatically enable or disable `direct_join_setting` by calling the `directJoin` function.

```
import org.apache.spark.sql.cassandra.CassandraSourceRelation._
import org.apache.spark.sql.cassandra._
val table = spark.read.cassandraFormat("tab", "ks").load
spark
.range(1L,100000L)
.withColumn("id", concat(lit("Store "), 'id))
.join(table.directJoin(AlwaysOff), 'id === 'store)
```

The `directJoin` function can be set to `AlwaysOn` to permanently enable the optimization, `AlwaysOff` to permanently disable the optimization, or `Automatic` to let DSE determine when to use the optimization according to the formula for `direct_join_size_ratio` described earlier.

Most users should not change the `direct_join_setting` option. In most cases the direct join should be faster than a full table scan. If the calculation is producing less than optimal results adjust the threshold for automatic joins, or turn the optimization off.

## Accessing the Spark session and context for applications running outside of DSE Analytics

You can optionally create session and context objects for applications that are run outside of the DSE Analytics environment.

You can optionally create session and context objects for applications that are run outside of the DSE Analytics environment. This is for advanced use cases where applications do not use `dse spark-submit` for handling the classpath and configuration settings.

All classpath and JAR distribution must be handled by the application. The application classpath must include the output of the `dse spark-classpath` command.

```
dse spark-classpath
```

## Using the Builder API to create a DSE Spark session

To create a DSE Spark session outside of the DSE Analytics application environment, use the `DseConfiguration` class and the `enableDseSupport` method when creating a Spark session.

```
import org.apache.spark.sql.SparkSession
import com.datastax.spark.connector.DseConfiguration._
val spark = SparkSession.builder
 .appName("Datastax Scala example")
 .master("dse://127.0.0.1?")
 .config("spark.jars", "target/scala-2.11/writeread_2.11-0.1.jar")
 .enableHiveSupport()
 .enableDseSupport()
 .getOrCreate()
```

## Creating a Spark Context

When creating a Spark Context object, use the `DseConfiguration` class and call the `enableDseSupport` method when creating the `SparkConfiguration` instance. In Scala:

```
import com.datastax.spark.connector.DseConfiguration._
new SparkConf().enableDseSupport()
```

In Java:

```
SparkConf rawConf = new SparkConf();
SparkConf conf = DseConfiguration.enableDseSupport(rawConf);
```

## Saving RDD data to DSE

With DataStax Enterprise, you can save almost any RDD to the database. Before you use the RDD in a standalone application, import `com.datastax.spark.connector`.

With DataStax Enterprise, you can save almost any RDD to the database. Unless you do not provide a custom mapping, the object class of the RDD must be a tuple or have property names corresponding to table column names. To save the RDD, call the `saveToCassandra` method with a keyspace name, table name, and optionally, a list of columns. Before attempting to use the RDD in a standalone application, import `com.datastax.spark.connector`.

You can also use the [DataFrames API \(page 242\)](#) to manipulate data within Spark.

## Saving a collection of tuples

The following example shows how to save a collection of tuples to the database.

```
val collection = sc.parallelize(Seq(("cat", 30), ("fox", 40)))
```

```
collection: org.apache.spark.rdd.RDD[(String, Int)] =
 ParallelCollectionRDD[6] at parallelize at console:22
```

```
collection.saveToCassandra("test", "words", SomeColumns("word",
 "count"))
```

At the last Scala prompt in this example, no output means that the data was saved to the database.

In cqlsh, query the words table to select all the contents.

```
SELECT * FROM test.words;
```

word	count
bar	20
foo	10
cat	30
fox	40

(4 rows)

## Saving a collection of case class objects to the database

The following example shows how to save a collection of case class objects.

```
case class WordCount(word: String, count: Long)
val collection = sc.parallelize(Seq(WordCount("dog", 50),
 WordCount("cow", 60)))
collection.saveToCassandra("test", "words", SomeColumns("word",
 "count"))
```

In cqlsh, query the words table to select all the contents.

```
SELECT * FROM test.words;
```

word	count
bar	20
foo	10
cat	30
fox	40
dog	50
cow	60

## Using non-default property-name to column-name mappings

Mapping rows to tuples and case classes work out-of-the box, but in some cases, you might need more control over database-Scala mapping. For example, Java classes are likely to use the JavaBeans naming convention, where accessors are named with *get*, *is* or *set* prefixes. To customize column-property mappings, put an

appropriate `ColumnMapper[YourClass]` implicit object in scope. Define such an object in a companion object of the class being mapped. The `ColumnMapper` affects both loading and saving data. DataStax Enterprise includes a few `ColumnMapper` implementations.

## Working with JavaBeans

To work with Java classes, use `JavaBeanColumnMapper`. Make sure objects are serializable; otherwise Spark cannot send them over the network. The following example shows how to use the `JavaBeanColumnMapper`.

To use JavaBeans style accessors:

```
:paste
// Entering paste mode (ctrl-D to finish)
```

Paste this import command and class definition:

```
import com.datastax.spark.connector.mapper.JavaBeanColumnMapper
class WordCount extends Serializable {
 private var _word: String = ""
 private var _count: Int = 0
 def setWord(word: String) { _word = word }
 def setCount(count: Int) { _count = count }
 override def toString = _word + ":" + _count
}
object WordCount {
 implicit object Mapper extends JavaBeanColumnMapper[WordCount]
}
```

Enter CTRL D to exit paste mode. The output is:

```
// Exiting paste mode, now interpreting.

import com.datastax.spark.connector.mapper.JavaBeanColumnMapper
defined class WordCount
defined module WordCount
```

Query the `WordCount` object.

```
sc.cassandraTable[WordCount]("test", "words").toArray
```

To save the data, you need to define getters.

## Manually specifying a property-name to column-name relationship

If for some reason you want to associate a property with a column of a different name, pass a column translation map to the `DefaultColumnMapper` or `JavaBeanColumnMapper`.

To change column names:

```
:paste
// Entering paste mode (ctrl-D to finish)
```

```
import com.datastax.spark.connector.mapper.DefaultColumnMapper
case class WordCount(w: String, c: Int)
object WordCount { implicit object Mapper extends
DefaultColumnMapper[WordCount](Map("w" -> "word", "c" -> "count")) }
```

Enter CTRL D.

```
// Exiting paste mode, now interpreting.

import com.datastax.spark.connector.mapper.DefaultColumnMapper
defined class WordCount
defined module WordCount
```

Continue entering these commands:

```
sc.cassandraTable[WordCount]("test", "words").toArray
sc.parallelize(Seq(WordCount("bar", 20), WordCount("foo", 40))).saveToCassandra("test",
"words", SomeColumns("word", "count"))
```

## Writing custom ColumnMappers

To define column mappings for your classes, create an appropriate implicit object implementing `ColumnMapper[YourClass]` trait.

## Spark supported types

Spark supported CQL types are mapped to Scala types.

This table maps [CQL types](#) to Scala types. All CQL types are supported by the DataStax Enterprise Spark integration. Other type conversions might work, but cause loss of precision or not work for all values. Most types are convertible to strings. You can convert strings that conform to the CQL standard to numbers, dates, addresses or UUIDs. You can convert maps to or from sequences of key-value tuples.

**Table 5: Supported types**

CQL Type	Scala Type
ascii	String
bigint	Long
blob	ByteBuffer, Array
boolean	Boolean
counter	Long
decimal	BigDecimal, java.math.BigDecimal
double	Double
float	Float
inet	java.net.InetAddress
int	Int

CQL Type	Scala Type
list	Vector, List, Iterable, Seq, IndexedSeq, java.util.List
map	Map, TreeMap, java.util.HashMap
set	Set, TreeSet, java.util.HashSet
text, varchar	String
timestamp	Long, java.util.Date, java.sql.Date, org.joda.time.DateTime
timeuuid	java.util.UUID
uuid	java.util.UUID
varint	BigInt, java.math.BigInteger
nullable values	Option

## Loading external HDFS data into the database using Spark

Hadoop HDFS data can be accessed from DataStax Enterprise Analytics nodes and saved to database tables using Spark.

This task demonstrates how to access Hadoop data and save it to the database using Spark on DSE Analytics nodes.

To simplify accessing the Hadoop data, it uses WebHDFS, a REST-based server for interacting with a Hadoop cluster. WebHDFS handles redirect requests to the data nodes, so every DSE Analytics node needs to be able to route to every HDFS node using the Hadoop node's hostname.

These instructions use example weather data, but the principles can be applied to any kind of Hadoop data that can be stored in the database.

### Prerequisites:

You will need:

- A working Hadoop installation with HDFS and WebHDFS enabled and running. You will need the hostname of the machine on which Hadoop is running, and the cluster must be accessible from the DSE Analytics nodes in your DataStax Enterprise cluster.
- A running DataStax Enterprise cluster with DSE Analytics nodes enabled.
- Git installed on a DSE Analytics node.

1. Clone the GitHub repository containing the test data.

```
git clone https://github.com/brianmhess/DSE-Spark-HDFS.git
```

2. Load the maximum temperature test data into the Hadoop cluster using WebHDFS.

In this example, the Hadoop node has a hostname of `hadoopNode.example.com`. Replace it with the hostname of a node in your Hadoop cluster.

```
dse hadoop fs -mkdir webhdfs://hadoopNode.example.com:50070/user/guest/data &&
dse hadoop fs -copyFromLocal data/sftmax.csv webhdfs://hadoopNode:50070/user/guest/data/sftmax.csv
```

3. Create the keyspace and table and load the minimum temperature test data using cqlsh.

```
CREATE KEYSPACE IF NOT EXISTS spark_ex2 WITH REPLICATION =
 { 'class':'SimpleStrategy', 'replication_factor':1}
DROP TABLE IF EXISTS spark_ex2.sftmin
CREATE TABLE IF NOT EXISTS spark_ex2.sftmin(location TEXT, year INT, month INT, day INT, tmin DOUBLE, datestring TEXT, PRIMARY KEY ((location), year, month, day)) WITH CLUSTERING ORDER BY (year DESC, month DESC, day DESC)
COPY spark_ex2.sftmin(location, year, month, day, tmin, datestring) FROM 'data/sftmin.csv'
```

4. Ensure that we can access the HDFS data by interacting with the data using dse hadoop fs.

The following command counts the number of lines of HDFS data.

```
dse hadoop fs -cat webhdfs://hadoopNode.example.com:50070/user/guest/data/sftmax.csv | wc -l
```

You should see output similar to the following:

```
16/05/10 11:21:51 INFO snitch.Workload: Setting my workload to Cassandra 3606
```

5. Start the Spark console and connect to the DataStax Enterprise cluster.

```
dse spark
```

Import the Spark Cassandra connector and create the session.

```
import com.datastax.spark.connector.cql.CassandraConnector
val connector = CassandraConnector(csc.conf)
val session = connector.openSession()
```

6. Create the table to store the maximum temperature data.

```
session.execute(s"DROP TABLE IF EXISTS spark_ex2.sftmax")
session.execute(s"CREATE TABLE IF NOT EXISTS
 spark_ex2.sftmax(location TEXT, year INT, month INT, day INT,
 tmax DOUBLE, datestring TEXT, PRIMARY KEY ((location), year,
 month, day)) WITH CLUSTERING ORDER BY (year DESC, month DESC, day
 DESC) ")
```

7. Create a Spark RDD from the HDFS maximum temperature data and save it to the table.

First create a case class representing the maximum temperature sensor data:

```
case class Tmax(location: String, year: Int, month: Int, day: Int, tmax: Double, datestring: String)
```

Read the data into an RDD.

```
val tmax_raw = sc.textFile("webhdfs://sandbox.hortonworks.com:50070/user/guest/data/sftmax.csv")
```

Transform the data so each record in the RDD is an instance of the `Tmax` case class.

```
val tmax_c10 = tmax_raw.map(x=>x.split(",")).map(x => Tmax(x(0), x(1).toInt, x(2).toInt, x(3).toInt, x(4).toDouble, x(5)))
```

Count the case class instances to make sure it matches the number of records.

```
tmax_c10.count
res11: Long = 3606
```

Save the case class instances to the database.

```
tmax_c10.saveToCassandra("spark_ex2", "sftmax")
```

## 8. Verify the records match by counting the rows using CQL.

```
session.execute("SELECT COUNT(*) FROM
 spark_ex2.sftmax").all.get(0).getLong(0)
res23: Long = 3606
```

## 9. Join the maximum and minimum data into a new table.

Create a `Tmin` case class to store the minimum temperature sensor data.

```
case class Tmin(location: String, year: Int, month: Int, day: Int, tmin: Double, datestring: String)
val tmin_raw = sc.cassandraTable("spark_ex2", "sftmin")
val tmin_c10 = tmin_raw.map(x => Tmin(x.getString("location"),
 x.getInt("year"), x.getInt("month"), x.getInt("day"),
 x.getDouble("tmin"), x.getString("datestring")))
```

In order to join RDDs, they need to be `PairRDDs`, with the first element in the pair being the join key.

```
val tmin_pair = tmin_c10.map(x=>(x.datestring,x))
val tmax_pair = tmax_c10.map(x=>(x.datestring,x))
```

Create a `THiLoDelta` case class to store the difference between the maximum and minimum temperatures.

```
case class THiLoDelta(location: String, year: Int, month: Int,
day: Int, hi: Double, low: Double, delta: Double, datestring:
String)
```

Join the data using the `join` operation on the `PairRDDs`. Convert the joined data to the `THiLoDelta` case class.

```
val tdelta_join1 = tmax_pair1.join(tmin_pair1)
val tdelta_c10 = tdelta_join1.map(x => THiLoDelta(x._2._1._1,
x._2._1._2, x._2._1._3, x._2._1._4, x._2._1._5, x._2._2._5,
x._2._1._5 - x._2._2._5, x._1))
```

Create a new table within Spark using CQL to store the temperature difference data.

```
session.execute(s"DROP TABLE IF EXISTS spark_ex2.sftdelta")
session.execute(s"CREATE TABLE IF NOT EXISTS
spark_ex2.sftdelta(location TEXT, year INT, month INT, day INT,
hi DOUBLE, low DOUBLE, delta DOUBLE, datestring TEXT, PRIMARY
KEY ((location), year, month, day)) WITH CLUSTERING ORDER BY
(year DESC, month DESC, day DESC)")
```

Save the temperature difference data to the table.

```
tdelta_c10.saveToCassandra("spark_ex2", "sftdelta")
```

## Monitoring Spark with the web interface

A Spark web interface is bundled with DataStax Enterprise. The Spark web interface facilitates monitoring, debugging, and managing Spark.

A web interface, bundled with DataStax Enterprise, facilitates monitoring, debugging, and managing Spark.

### Using the Spark web interface

To use the Spark web interface enter the [listen IP address \(page 91\)](#) of any Spark node in a browser followed by port number 7080 (configured in the [spark-env.sh configuration file \(page 213\)](#)). Starting in DSE 5.1, all Spark nodes within an Analytics datacenter will redirect to the current Spark Master.

If the Spark Master is not available, the UI will keep polling for the Spark Master every 10 seconds until the Master is available.

The Spark web interface can be [secured using SSL](#). SSL encryption of the web interface is enabled by default when client encryption is enabled.

If authentication is enabled, and plain authentication is available, you will be prompted for authentication credentials when accessing the web UI. We recommend using SSL with authentication.

**Note:** Kerberos authentication is not supported in the Spark web UI. If authentication is enabled and either LDAP or Internal authentication is not available, the Spark web UI will not be accessible. If this occurs, disable

authentication for the Spark web UI only by removing the `spark.ui.filters` setting in `spark-daemon-defaults.conf` located in the Spark configuration directory.

DSE SSL encryption and authentication only apply to the Spark Master and Worker UIs, not the Spark Driver UI. To use encryption and authentication with the Driver UI, refer to the [Spark security documentation](#).

**Spark 2.2.0**

**Spark Master at spark://10.200.179.153:7077**

URL: `spark://10.200.179.153:7077`  
REST URL: `spark://10.200.179.153:6066 (cluster mode)`

Alive Workers: 2  
Total Cores: 12 / 12  
◦ default: 12 / 12  
Total Memory: 2.0 GB / 26.4 GB  
◦ default: 2.0 GB / 26.4 GB  
Applications: 1 Running, 1 Completed  
Drivers: 0 Running, 0 Completed  
Status: ALIVE

**Workers**

Worker Id	Address	State	Cores	Memory
<a href="#">worker-20180105193611-10.200.179.153-58829</a>	10.200.179.153:58829	ALIVE	Total: 6 / 6 • default: 6 / 6	Total: 1024.0 MB / 13.2 GB • default: 1024.0 MB / 13.2 GB
<a href="#">worker-20180105193616-10.200.179.152-51071</a>	10.200.179.152:51071	ALIVE	Total: 6 / 6 • default: 6 / 6	Total: 1024.0 MB / 13.2 GB • default: 1024.0 MB / 13.2 GB

**Running Applications**

Application ID	Name	Workpool	Cores	Memory per Executor	Submitted Time	User	State	Duration
<a href="#">app-20180111194527-0001</a> (kill)	Spark shell	default	12	1024.0 MB	2018/01/11 19:45:27	anonymous	RUNNING	19 s

**Completed Applications**

Application ID	Name	Workpool	Cores	Memory per Executor	Submitted Time	User	State	Duration
<a href="#">app-20180105193837-0000</a>	SparkSQL::10.200.179.152	default	0	1024.0 MB	2018/01/05 19:38:37	anonymous	FINISHED	1.0 min

The UI includes information on the number of cores and amount of memory available to Spark in total and in each work pool, and similar information for each Spark worker. The applications list the associated work pool.

See the Spark documentation for information on [using the Spark web UI](#).

## Authorization in the Spark web UI

When authorization is enabled and an authenticated user accesses the web UI, what they can see and do is controlled by their permissions. This allows administrators to control who has permission to view specific application logs, view the executors for the application, kill the application, and list all applications. Viewing and modifying applications can be configured per datacenter, work pool, or application.

See [Using authorization with Spark \(page 214\)](#) for details on granting permissions.

## Displaying fully qualified domain names in the web UI

To display fully qualified domain names (FQDNs) in the Spark web UI, set the `SPARK_PUBLIC_DNS` variable in `spark-env.sh` on each Analytics node.

Set `SPARK_PUBLIC_DNS` to the FQDN of the node if you have SSL enabled for the web UI.

## Filtering properties in the Spark Driver UI

The Spark Driver UI has an Environment tab that lists the Spark configuration and system properties used by Spark. This can include sensitive information like passwords and security tokens. DSE Spark filters these properties and mask their values with sequences

of asterisks. The `spark.redaction.regex` filter is configured as a regular expression that by default includes all properties that contain the string "secret", "token", or "password" as well as all system properties. To modify the filter, edit the `spark.redaction.regex` property in `spark-defaults.conf` in the Spark configuration directory.

## Getting started with the Spark Cassandra Connector Java API

The Spark Cassandra Connector Java API allows you to create Java applications that use Spark to analyze database data.

The Spark Cassandra Connector Java API allows you to create Java applications that use Spark to analyze database data. See the Spark Cassandra Connector [Java Doc](#) on GitHub. See the [component versions \(page 17\)](#) for the latest version of the Spark Cassandra Connector used by DataStax Enterprise.

### Using the Java API in SBT build files

Add the following library dependency to the `build.sbt` or other SBT build file.

```
val dseVersion = "6.0.0"
// Please make sure that following DSE version matches your DSE cluster
// version.
// SBT 0.13.13 or greater required because of a dependency resolution
// bug
libraryDependencies += "com.datastax.dse" % "dse-spark-dependencies" %
dseVersion % "provided"
```

For example project templates, see <https://github.com/datastax/SparkBuildExamples>.

### Using the Java API in Maven build files

Add the following dependencies to the `pom.xml` file:

```
<dependency>
 <groupId>com.datastax.dse</groupId>
 <artifactId>dse-spark-dependencies</artifactId>
 <version>${dse.version}</version>
 <scope>provided</scope>
</dependency>
```

Then add the DataStax repository:

```
<repository>
 <id>DataStax-Repo</id>
 <url>https://repo.datastax.com/public-repos/</url>
</repository>
```

For example project templates, see <https://github.com/datastax/SparkBuildExamples>

## Accessing database data in Scala applications

To perform Spark actions on table data, you first obtain a `RDD` object. To create the `RDD` object, create a Spark configuration object, which is then used to create a Spark context object.

```
import com.datastax.spark.connector._
val conf = new SparkConf(true)
.set("spark.cassandra.connection.host", "127.0.0.1")
val sc = new SparkContext("dse://127.0.0.1:7077", "test", conf)
val rdd = sc.cassandraTable("my_keyspace", "my_table")
```

To save data to the database in Scala applications, use the `saveToCassandra` method, passing in the keyspace, table, and mapping information.

```
val collection = sc.parallelize(Seq(("key3", 3), ("key4", 4)))
collection.saveToCassandra("my_keyspace", "my_table",
 SomeColumns("key", "value"))
```

To perform DSE Graph queries in a Scala application, you can cast a `CassandraConnector` session to a `com.datastax.driver.dse.DseSession` and then run graph statements using the `executeGraph` method.

```
val session = CassandraConnector(sc.getConf).withSessionDo(session =>
 session.asInstanceOf[DseSession])
session.executeGraph(graph statement)
```

## Accessing database data in Java applications

To perform Spark actions on table data, you first obtain a `CassandraJavaRDD` object, a subclass of the `JavaRDD` class. The `CassandraJavaRDD` is the Java language equivalent of the `CassandraRDD` object used in Scala applications.

To create the `CassandraJavaRDD` object, create a Spark configuration object, which is then used to create a Spark context object.

```
SparkConf conf = new SparkConf()
.setAppName("My application");
SparkContext sc = new SparkContext(conf);
```

Use the static methods of the

`com.datastax.spark.connector.japi.CassandraJavaUtil` class to get and manipulate `CassandraJavaRDD` instances. To get a new `CassandraJavaRDD` instance, call one of the `javaFunctions` methods in `CassandraJavaUtil`, pass in a context object, and then call the `cassandraTable` method and pass in the keyspace, table name, and mapping class.

```
JavaRDD<String> cassandraRdd = CassandraJavaUtil.javaFunctions(sc)
 .cassandraTable("my_keyspace",
 "my_table", .mapColumnTo(String.class))
 .select("my_column");
```

## Mapping column data to Java types

You can specify the Java type of a single column from a table row by specifying the type in when creating the `CassandraJavaRDD<T>` instance and calling the `mapColumnTo` method and passing in the type. Then call the `select` method to set the column name.

```
JavaRDD<Integer> cassandraRdd = CassandraJavaUtil.javaFunctions(sc)
 .cassandraTable("my_keyspace",
 "my_table", .mapColumnTo(Integer.class))
 .select("column1");
```

JavaBeans classes can be mapped using the `mapRowTo` method. The JavaBeans property names should correspond to the column names following the default mapping rules. For example, the `firstName` property will map by default to the `first_name` column name.

```
JavaRDD<Person> personRdd = CassandraJavaUtil.javaFunctions(sc)
 .cassandraTable("my_keyspace", "my_table",
 mapRowTo(Person.class));
```

`CassandraJavaPairRDD<T, T>` instances are extensions of the `JavaPairRDD` class, and have mapping readers for rows and columns similar to the previous examples. These pair RDDs typically are used for key/value pairs, where the first type is the key and the second type is the value.

When mapping a single column for both the key and the value, call `mapColumnTo` and specify the key and value types, then the `select` method and pass in the key and value column names.

```
CassandraJavaPairRDD<Integer, String> pairRdd =
CassandraJavaUtil.javaFunctions(sc)
 .cassandraTable("my_keyspace", "my_table",
 mapColumnTo(Integer.class), mapColumnTo(String.class))
 .select("id", "first_name");
```

Use the `mapRowTo` method to map row data to a Java type. For example, to create a pair RDD instance with the primary key and then a JavaBeans object:

```
CassandraJavaPairRDD<Integer, Person> idPersonRdd =
CassandraJavaUtil.javaFunctions(sc)
 .cassandraTable("my_keyspace", "my_table",
 mapColumnTo(Integer.class), mapRowTo(Person.class))
 .select("id", "first_name", "last_name", "birthdate", "email");
```

## Saving data to the database in Java applications

To save data from an RDD to the database call the `writerBuilder` method on the `CassandraJavaRDD` instance, passing in the keyspace, table name, and optionally type mapping information for the column or row.

```
CassandraJavaUtil.javaFunctions(personRdd)
```

```
.writerBuilder("my_keyspace", "my_table",
mapToRow(Person.class)).saveToCassandra();
```

## Using DSE Spark with third party tools and integrations

The `dse exec` command sets the required environment variables required to run third-party tools that integrate with Spark.

The `dse exec` command (page 793) sets the required environment variables required to run third-party tools that integrate with Spark.

```
dse exec command
```

### Livy integration

Download and install Livy<https://github.com/cloudera/livy> on a DSE node. By default Livy runs Spark in local mode. Before starting Livy create a configuration file by copying the `conf/livy.conf.template` to `conf/livy.conf`, then uncomment or add the following two properties:

```
livy.spark.master = dse:///
livy.repl.enable-hive-context = true
```

To launch Livy:

```
dse exec livy-server
```

### Zeppelin integration

Download and install Zeppelin on a DSE node. To launch Zeppelin server:

```
dse exec zeppelin.sh
```

By default Zeppelin runs Spark in local mode. Update the master property to `dse:///` in the Spark session in the Interpreters configuration page. No configuration file changes are required to run Zeppelin.

### RStudio integration

Download and [install R](#) (page 245) on all DSE Analytics nodes, install RStudio desktop on one of the nodes, then run RStudio:

```
dse exec rstudio
```

In the RStudio session start a Spark session:

```
library(SparkR, lib.loc = c(file.path(Sys.getenv("SPARK_HOME")), "R",
"lib"))
sparkR.session()
```

**Note:** These instructions are for RStudio desktop, not RStudio Server. In multiuser environments, we recommend using [AlwaysOn SQL \(page 245\)](#) and [JDBC \(page 252\)](#) connections rather than SparkR.

## Configuring Spark

Configuring Spark includes setting Spark properties for DataStax Enterprise and the database, enabling Spark apps, and setting permissions.

Configuring Spark for DataStax Enterprise includes:

### Configuring Spark nodes

Modify the settings for Spark nodes security, performance, and logging.

Modify the settings for Spark nodes security, performance, and logging.

To manage Spark performance and operations:

- [Set environment variables \(page 208\)](#)
- [Protect Spark directories \(page 209\)](#)
- [Grant access to default Spark directories \(page 209\)](#)
- [Secure Spark nodes \(page 210\)](#)
- [Configure Spark memory and cores \(page 210\)](#)
- [Configure Spark logging options \(page 221\)](#)

### Set environment variables

DataStax recommends using the default values of Spark environment variables unless you need to increase the memory settings due to an `OutOfMemoryError` condition or garbage collection taking too long. Use the [Spark memory \(page 131\)](#) configuration options in the `dse.yaml` and `spark-env.sh` files.

You can set a user-specific `SPARK_HOME` directory if you also set `ALLOW_SPARK_HOME=true` in your environment before starting DSE.

For example, on Debian or Ubuntu using a package installation:

```
export SPARK_HOME=$HOME/spark &&
export ALLOW_SPARK_HOME=true &&
sudo service dse start
```

The temporary directory for shuffle data, RDDs, and other ephemeral Spark data can be configured for both the locally running driver and for the Spark server processes managed by DSE (Spark Master, Workers, shuffle service, executor and driver running in cluster mode).

For the locally running Spark driver, the `SPARK_LOCAL_DIRS` environment variable can be customized in the user environment or in `spark-env.sh`. By default, it is set to a `.spark` directory in the system temporary directory. For example, on Ubuntu it is `/tmp/.spark`.

For all other Spark server processes, the `SPARK_EXECUTOR_DIRS` environment variable can be customized in the user environment or in `spark-env.sh`. By default it is set to `/var/lib/spark/rdd`.

**Note:** The default `SPARK_LOCAL_DIRS` and `SPARK_EXECUTOR_DIRS` environment variable values differ from non-DSE Spark.

To configure worker cleanup, modify the `SPARK_WORKER_OPTS` environment variable and add the [cleanup properties](#). The `SPARK_WORKER_OPTS` environment variable can be set in the user environment or in `spark-env.sh`. For example, the following enables worker cleanup, sets the cleanup interval to 30 minutes (i.e. 1800 seconds), and sets the length of time application worker directories will be retained to 7 days (i.e. 604800 seconds).

```
export SPARK_WORKER_OPTS="$SPARK_WORKER_OPTS \
-Dspark.worker.cleanup.enabled=true \
-Dspark.worker.cleanup.interval=1800 \
-Dspark.worker.cleanup.appDataTtl=604800"
```

## Protect Spark directories

After you start up a Spark cluster, DataStax Enterprise creates a Spark work directory for each Spark Worker on worker nodes. A worker node can have more than one worker, configured by the `SPARK_WORKER_INSTANCES` option in `spark-env.sh`. If `SPARK_WORKER_INSTANCES` is undefined, a single worker is started. The work directory contains the standard output and standard error of executors and other application specific data stored by Spark Worker and executors; the directory is writable only by the DSE user.

By default, the Spark parent work directory is located in `/var/lib/spark/work`, with each worker in a subdirectory named `worker-number`, where the number starts at 0. To change the parent worker directory, configure `SPARK_WORKER_DIR` in the `spark-env.sh` file.

The Spark RDD directory is the directory where RDDs are placed when executors decide to spill them to disk. This directory might contain the data from the database or the results of running Spark applications. If the data in the directory is confidential, prevent access by unauthorized users. The RDD directory might contain a significant amount of data, so configure its location on a fast disk. The directory is writable only by the `cassandra` user. The default location of the Spark RDD directory is `/var/lib/spark/rdd`. The directory should be located on a fast disk. To change the RDD directory, configure `SPARK_LOCAL_DIRS` in the `spark-env.sh` file.

## Grant access to default Spark directories

Before starting up nodes on a tarball installation, you need permission to access the default Spark directory locations: `/var/lib/spark` and `/var/log/spark`. Change ownership of these directories as follows:

```
sudo mkdir -p /var/lib/spark/rdd; sudo chmod a+w /var/lib/spark/rdd;
sudo chown -R $USER:$GROUP /var/lib/spark/rdd &&
sudo mkdir -p /var/log/spark; sudo chown -R $USER:$GROUP /var/log/
spark
```

In multiple datacenter clusters, use a virtual datacenter to isolate Spark jobs. Running Spark jobs consume resources that can affect latency and throughput.

DataStax Enterprise supports the use of virtual nodes (vnodes) with Spark.

## Secure Spark nodes

### Client-to-node SSL

Ensure that the truststore entries in `cassandra.yaml` are present as described in [Client-to-node encryption](#), even when client authentication is not enabled.

### Enabling security and authentication

Security is enabled using the `spark_security_enabled` option in `dse.yaml`. Setting it to `enabled` turns on authentication between the Spark Master and Worker nodes, and allows you to enable encryption. To encrypt Spark connections for all components except the web UI, enable `spark_security_encryption_enabled`. The length of the shared secret used to secure Spark components is set using the `spark_shared_secret_bit_length` option, with a default value of 256 bits. These options are described in [DSE Analytics options \(page 131\)](#). For production clusters, enable these authentication and encryption. Doing so does not significantly affect performance.

### Authentication and Spark applications

If authentication is enabled, users need to be authenticated in order to submit an application.

### Authorization and Spark applications

If DSE authorization is enabled, users needs permission to submit an application. Additionally, the user submitting the application automatically receives permission to manage the application, which can optionally be extended to other users.

### Database credentials for the Spark SQL Thrift server

In the `hive-site.xml` file, configure authentication credentials for the Spark SQL Thrift server. Ensure that you use the `hive-site.xml` file in the Spark directory:

- Package installations: `/etc/dse/spark/hive-site.xml`
- Tarball installations: `installation_location/resources/spark/conf/hive-site.xml`

### Kerberos with Spark

With Kerberos authentication, the Spark launcher connects to DSE with Kerberos credentials and requests DSE to generate a delegation token. The Spark driver and executors use the delegation token to connect to the cluster. For valid authentication, the delegation token must be renewed periodically. For security reasons, the user who is authenticated with the token should not be able to renew it. Therefore, delegation tokens have two associated users: token owner and token renewer.

The token renewer is none so that only a DSE internal process can renew it. When the application is submitted, DSE automatically renews delegation tokens that are associated with Spark application. When the application is unregistered (finished), the delegation token renewal is stopped and the token is cancelled.

Set Kerberos options, see [Defining a Kerberos scheme](#).

## Configure Spark memory and cores

Spark memory options affect different components of the Spark ecosystem:

### Spark History server and the Spark Thrift server memory

The **SPARK\_DAEMON\_MEMORY** option configures the memory that is used by the Spark SQL Thrift server and history-server. Add or change this setting in the `spark-env.sh` file on nodes that run these server applications.

### Spark Worker memory

The `memory_total` option ([page 133](#)) in `resource_manager_options.worker_options` section of `dse.yaml` configures the total system memory that you can assign to all executors that are run by the work pools on the particular node. The default work pool will use all of this memory if no other work pools are defined. If you define additional work pools, you can set the total amount of memory by setting the `memory` option ([page 133](#)) in the work pool definition.

### Application executor memory

You can configure the amount of memory that each executor can consume for the application. Spark uses a 512MB default. Use either the **spark.executor.memory** option, described in ["Spark Available Properties"](#), or the `--executor-memory mem` argument to the `dse spark` command ([page 227](#)).

### Application memory

You can configure additional Java options that are applied by the worker when spawning an executor for the application. Use the **spark.executor.extraJavaOptions** property, described in [Spark 1.6.2 Available Properties](#). For example:

```
spark.executor.extraJavaOptions -XX:+PrintGCDetails -Dkey=value -Dnumbers="one two three"
```

### Core management

You can manage the number of cores by configuring these options.

- Spark Worker cores

The `cores_total` option ([page 132](#)) in the `resource_manager_options.worker_options` section of `dse.yaml` configures the total number of system cores available to Spark Workers for executors. If no work pools are defined in the `resource_manager_options.workpools` section ([page 133](#)) of `dse.yaml` the default work pool will use all the cores defined by `cores_total`. If additional work pools are defined, the default work pool will use the cores available after allocating the cores defined by the work pools.

A single executor can borrow more than one core from the worker. The number of cores used by the executor relates to the number of parallel tasks the executor might perform. The number of cores offered by the cluster is the sum of cores offered by all the workers in the cluster.

- Application cores

In the Spark configuration object of your application, you configure the number of application cores that the application requests from the cluster using either the **spark.cores.max** configuration property or the `--total-executor-cores cores` argument to the `dse spark` command ([page 227](#)).

See the [Spark documentation](#) for details about memory and core allocation.

DataStax Enterprise can control the memory and cores offered by particular Spark Workers in semi-automatic fashion. The `resource_manager_options.worker_options` section in the `dse.yaml` file has options to configure the fraction of system resources that are made available to Spark Workers and any defined work pools. The available resources are calculated in the following way:

- Spark Worker memory =  $\text{memory\_total} * (\text{total system memory} - \text{memory assigned to DSE})$
- Spark Worker cores =  $\text{cores\_total} * \text{total system cores}$

The range of the values is 0.01 to 1. If the range is not specified, the default value 0.7 is used.

The lowest values you can assign to a named work pool's memory and cores are 64 MB and 1 core, respectively. If the results are lower, no exception is thrown and the values are automatically limited.

The following example shows a work pool named `workpool1` with 1 core and 512 MB of RAM assigned to it. The remaining resources calculated from the values in `worker_options` are assigned to the default work pool.

```
resource_manager_options:
 worker_options:
 cores_total: 0.7
 memory_total: 0.7

 workpools:
 - name: workpool1
 cores: 1
 memory: 512M
```

## Running Spark clusters in cloud environments

If you are using a cloud infrastructure provider like Amazon EC2, you must explicitly open the ports for publicly routable IP addresses in your cluster. If you do not, the Spark workers will not be able to find the Spark Master.

One work-around is to set the `prefer_local` setting in your `cassandra-rackdc.properties` snitch setup file to true:

```
Uncomment the following line to make this snitch prefer the internal
ip when possible, as the Ec2MultiRegionSnitch does.
prefer_local=true
```

This tells the cluster to communicate only on private IP addresses within the datacenter rather than the public routable IP addresses.

## Configuring the number of retries to retrieve Spark configuration

When Spark fetches configuration settings from DSE, it will not fail immediately if it cannot retrieve the configuration data, but will retry 5 times by default, with increasing delay between retries. The number of retries can be set in the Spark configuration, by modifying

the `spark.dse.configuration.fetch.retries` configuration property when calling the [dse spark command \(page 227\)](#), or in `spark-defaults.conf`.

## Disabling continuous paging

Continuous paging streams bulk amounts of records from DSE to the DataStax Java Driver used by DSE Spark. By default, continuous paging in queries is enabled. To disable it, set the `spark.dse.continuous.paging.enabled` setting to `false` when starting the Spark SQL shell or in `spark-defaults.conf`. For example:

```
dse spark-sql --conf spark.dse.continuous.paging.enabled=false
```

**Note:** Using continuous paging can potentially improve performance up to 3 times, though the improvement will depend on the data and the queries. Some factors that impact the performance improvement are the number of executor JVMs per node and the number of columns included in the query. Greater performance gains were observed with fewer executor JVMs per node and more columns selected.

## Configuring the Spark web interface ports

By default the Spark web UI runs on port 7080. To change the port number, do the following:

1. Open the `spark-env.sh` file in a text editor.
2. Set the `SPARK_MASTER_WEBUI_PORT` variable to the new port number. For example, to set it to port 7082:

```
export SPARK_MASTER_WEBUI_PORT=7082
```

3. Repeat these steps for each Analytics node in your cluster.
4. Restart the nodes in the cluster.

## Enabling Graphite Metrics in DSE Spark

Users can add third party JARs to Spark nodes by adding them to the Spark lib directory on each node and restart the cluster. Add the Graphite Metrics JARs to this directory to enable metrics in DSE Spark.

The default location of the Spark lib directory depends on the type of installation:

- Package installations: `/usr/share/dse/spark/lib`
- Tarball installations: `/var/lib/spark`

To add the Graphite JARs to Spark in a package installation, copy them to the Spark lib directory:

```
cp metrics-graphite-3.1.2.jar /usr/share/dse/spark/lib/ &&
cp metrics-json-3.1.2.jar /usr/share/dse/spark/lib/
```

## Setting Spark properties for the driver and executor

Additional [Spark properties](#) for the Spark driver and executors are set in `spark-defaults.conf`. For example, to enable Spark's `commons-crypto` encryption library:

```
spark.network.crypto.enabled true
```

## Using authorization with Spark

Set permissions on roles to allow Spark applications to be started, stopped, managed, and viewed.

Set permissions on roles to allow Spark applications to be started, stopped, managed, and viewed. To configure the permissions for a particular role, modify the `WORKPOOL` and `SUBMISSION` database objects by issuing CQL commands.

There are two kinds of authorization permissions which apply to Spark. Work pool permissions control the ability to submit or view a Spark application to DSE. Submission permissions control the ability to view or manage a particular application. If authentication and authorization are enabled for the [Spark web UI \(page 202\)](#), these permissions control what the authenticated user is allowed to view and modify.

All the following instructions assume you are issuing the CQL commands as a database superuser. In order to issue the following CQL commands as a regular database user, the user needs to have permission to use the DSE resource manager RPC:

```
GRANT ALL ON REMOTE OBJECT DseResourceManager TO role;
```

Each DSE Analytics user needs to have permission to use the client tools RPC:

```
GRANT ALL ON REMOTE OBJECT DseClientTool TO role;
```

## Authorizing roles to start Spark applications

The `CREATE` permission allows roles to start Spark applications on a work pool.

The following CQL command grants permission to submit a Spark application to any Analytics datacenter.

```
GRANT CREATE ON ANY WORKPOOL
TO role;
```

The following CQL command grants permission to submit a Spark application to a particular work pool in an Analytics datacenter.

```
GRANT CREATE ON WORKPOOL datacenter_name.workpool_name
TO role;
```

You can use a wildcard for `workpool_name` so it applies to all work pools in the datacenter:

```
GRANT CREATE ON WORKPOOL datacenter_name.*
TO role;
```

**Note:** You must specify a work pool name or wildcard when specifying a datacenter. In DSE versions prior to 6.0, you could specify the datacenter name only, but omitting the work pool name or wildcard will result in a syntax error.

There are similar revoke commands:

```
REVOKE CREATE ON ANY WORKPOOL FROM role
```

```
REVOKE CREATE ON WORKPOOL datacenter_name.workpool_name FROM role
```

When an application is submitted, the user who submits that application is automatically granted permission to manage and remove the application. You may also grant the ability to manage the application to another user or role.

Use the `REVOKE` command to remove permissions:

```
REVOKE CREATE ON ANY WORKPOOL FROM role;
```

## Authorizing roles to stop or manage Spark applications

Setting the `MODIFY` permission on the `SUBMISSION` object controls the ability to modify or stop a Spark application.

The following CQL command grants permission to manage any submission in any work pool to the specified role.

```
GRANT MODIFY ON ANY SUBMISSION TO role;
```

The following CQL command grants permission to manage any submission in a specified datacenter.

```
GRANT MODIFY ON ANY SUBMISSION
IN WORKPOOL datacenter_name
TO role;
```

The following CQL command grant permission to manage a submission identified by the provided id in a given data center's work pool.

```
GRANT MODIFY ON SUBMISSION id
IN WORKPOOL datacenter_name.workpool_name
TO role;
```

The ID is a string that is either the Spark application ID or the ID of the Spark driver running in cluster mode.

Use the `REVOKE` command to remove permissions:

```
REVOKE MODIFY ON SUBMISSION id
IN WORKPOOL datacenter_name.workpool_name
FROM role;
```

## Authorizing roles to browse Spark application information in the Spark web UI

The `DESCRIBE` permission allows roles to browse applications in the Spark web UI. The permissions can be set at the work pool or application level.

The following CQL command allows a role to view all applications in any Analytics datacenter.

```
GRANT DESCRIBE ON ANY WORKPOOL TO role;
```

The following CQL command limits the role's ability to view applications to a specific work pool in an Analytics datacenter.

```
GRANT DESCRIBE ON WORKPOOL datacenter_name.workpool_name
TO role;
```

The following CQL command allows a role to view all submissions, including executors, in any work pool to the specified role.

```
GRANT DESCRIBE ON ANY SUBMISSION TO role;
```

You can limit viewing to a specific datacenter: .

```
GRANT DESCRIBE ON ANY SUBMISSION
IN WORKPOOL datacenter_name.workpool_name
TO role;
```

You can further limit a role to only viewing the executors for a single application in a datacenter:

```
GRANT DESCRIBE ON SUBMISSION id
IN WORKPOOL datacenter_name.datacenter_name
TO role;
```

Use the REVOKE command to remove permissions:

```
REVOKE DESCRIBE ON ANY SUBMISSION
IN WORKPOOL datacenter_name.workpool_name
FROM role
```

## Spark server configuration

The spark-daemon-defaults.conf file configures DSE Spark Masters and Workers.

The spark-daemon-defaults.conf file configures DSE Spark Masters and Workers.

**Table 6: Spark server configuration properties**

Option	Default value	Description
dse.spark.application.timeout	30	The duration in seconds after which the application will be considered dead if no heartbeat is received.
spark.dseshuffle.sasl.port	7447	The port number on which a shuffle service for SASL secured applications is started. Bound to the listen_address in cassandra.yaml.

Option	Default value	Description
spark.dseShuffle.noSasl.port	7437	The port number on which a shuffle service for unsecured applications is started. Bound to the <code>listen_address</code> in <code>cassandra.yaml</code> .

By default Spark executor logs, which log the majority of your Spark Application output, are redirected to standard output. The output is managed by Spark Workers. Configure logging by adding `spark.executor.logs.rolling.*` properties to `spark-daemon-defaults.conf` file.

```
spark.executor.logs.rolling.maxRetainedFiles 3
spark.executor.logs.rolling.strategy size
spark.executor.logs.rolling.maxSize 50000
```

Additional [Spark properties](#) that affect the master and driver can be added to `spark-daemon-defaults.conf`. For example, to enable Spark's `commons-crypto` encryption library:

```
spark.network.crypto.enabled true
```

## Automatic Spark Master election

Spark Master elections are automatically managed.

Spark Master elections are automatically managed, and do not require any manual configuration.

DSE Analytics datacenters communicate with each other to elect one of the nodes as the Spark Master and another as the reserve Master. The Master keeps track of each Spark Worker and application, storing the information in a system table. If the Spark Master node fails, the reserve Master takes over and a new reserve Master is elected from the remaining Analytics nodes.

Each Analytics datacenter elects its own master.

For `dsetool` commands and options, see [dsetool \(page 804\)](#).

## Determining the Spark Master address

You do not need to specify the Master address when configuring or using Spark with DSE Analytics. Configuring applications with a [valid URL \(page 182\)](#) is sufficient for DSE to connect to the Master node and run the application. The following commands give information about the Spark configuration of DSE:

- To view the URL used to configure Spark applications:

```
dse client-tool spark master-address

dse://10.200.181.62:9042?
connection.local_dc=Analytics;connection.host=10.200.181.63;
```

- To view the current address of the Spark Master in this datacenter:

```
dse client-tool spark leader-address
```

```
10.200.181.62
```

- Workloads for [Spark Master \(page 176\)](#) are flagged as Workload: Analytics(SM).

```
dsetool ring
```

Address	DC		Rack	Workload
Graph	Status	State	Load	Owns
Token				Health [0,1]
10.200.181.62	Analytics		rack1	Analytics (SM)
-9223372036854775808	no	Up	Normal 111.91 KiB	? 0.10

- Query the `dse_leases.leases` table to list all the masters from each data center with Analytics nodes:

```
select * from dse_leases.leases ;
```

name	dc	duration_ms	epoch	holder
Leader/master/6.0	Analytics	30000	805254	
10.200.176.42				
Leader/master/6.0	SearchGraphAnalytics	30000	1300800	
10.200.176.45				
Leader/master/6.0	SearchAnalytics	30000	7	
10.200.176.44				

## Ensure that the replication factor is configured correctly for the `dse_leases` keyspace

If the `dse_leases` keyspace is not properly replicated, the Spark Master might not be elected.

**Important:** Every time you add a new datacenter, you **must** manually increase the replication factor of the `dse_leases` keyspace for the new DSE Analytics datacenter. If DataStax Enterprise or Spark security options are enabled on the cluster, you must also increase the replication factor for the `dse_security` keyspace across all logical datacenters.

The initial node in a multi datacenter has a replication factor of 1 for the `dse_leases` keyspace. For new datacenters, the first node is created with the `dse_leases` keyspace with an replication factor of 1 for that datacenter. However, any datacenters that you add have a replication factor of 0 and require configuration before you start DSE Analytics

nodes. You must change the replication factor of the `dse_leases` keyspace for multiple analytics datacenters. See [Setting the replication factor for analytics keyspaces \(page 171\)](#).

## Monitoring the lease subsystem

All changes to lease holders are recorded in the `dse_leases.logs` table. Most of the time, you do not want to enable logging.

1. To turn on logging, ensure that the [lease\\_metrics\\_options \(page 119\)](#) is enabled in the dse.yaml file:

```
lease_metrics_options:
 enabled:true
 ttl_seconds: 604800
```

- ## 2. Look at the `dse_leases.logs` table:

```
select * from dse_leases.logs ;
```

name			monitor	at
	new_holder	old_holder		
Leader/master/6.0	dc1	10.200.180.44	2018-05-17	
00:45:02.971000+0000	10.200.180.44			
Leader/master/6.0	dc1	10.200.180.49	2018-05-17	
02:37:07.381000+0000	10.200.180.49			

3. When the `lease_metrics_option` is enabled, you can examine the acquire, renew, resolve, and disable operations. Most of the time, these operations should complete in 100 ms or less:

```
select * from dse_perf.leases ;
```

```

Leader/master/6.0 | dc1 | 10.200.180.44 |
0 | 0 | 0 | 0 | 0 |
0 | 0 | 0 | 24 | 0 |
100 | 100 | 0 | 26 | 0 |
8 | 26 | 26 | 0 | 0 |
| True | 2018-05-03 19:30:38.395000+0000
Leader/master/6.0 | dc1 | 10.200.180.49 |
0 | 0 | 0 | 0 | 0 |
0 | 0 | 0 | 0 | 0 |
0 | 0 | 0 | 32 | 0 |
10 | 32 | 32 | 0 | 0 |
| True | 2018-05-03 19:30:55.656000+0000

```

- If the log warnings and errors do not contain relevant information, edit the `logback.xml` file and add:

```
<logger name="com.datastax.bdp.leasemanager" level="DEBUG">
```

- Restart the node for the debugging settings to take effect.

## Troubleshooting

Perform these various lease holder troubleshooting activities before you contact [DataStax Support](#).

### Verify the workload status

Run the `dsetool ring` command:

```
dsetool ring
```

If the replication factor is inadequate or if the replicas are down, the output of the `dsetool ring` command contains a warning:

Address	DC		Rack	Workload
Graph	Status	State	Load	Owns
Token				Health [0,1]
0				
10.200.178.232	SearchGraphAnalytics	rack1		
SearchAnalytics	yes	Up	Normal	153.04 KiB
		-9223372036854775808		?
0.00				
10.200.178.230	SearchGraphAnalytics	rack1		
SearchAnalytics(SM)	yes	Up	Normal	92.98 KiB
		0		?
0.000				

If the automatic Job Tracker or Spark Master election fails, verify that [an appropriate replication factor is set for the `dse\_leases` keyspace \(page 171\)](#).

**Use cqlsh commands to verify the replication factor of the analytics keyspaces**

1. Describe the `dse_leases` keyspace:

```
DESCRIBE KEYSPACE dse_leases;

CREATE KEYSPACE dse_leases WITH replication =
{'class': 'NetworkTopologyStrategy', 'Analytics1': '1'}
AND durable_writes = true;
```

2. Increase the replication factor of the `dse_leases` keyspace:

```
ALTER KEYSPACE dse_leases WITH replication =
{'class': 'NetworkTopologyStrategy', 'Analytics1': '3',
'Analytics2':'3'}
;
```

3. Run `nodetool repair`.

## Configuring Spark logging options

Configure Spark logging options.

You can configure Spark logging options for the Spark logs.

### Log directories

The Spark logging directory is the directory where the Spark components store individual log files. DataStax Enterprise places logs in the following locations:

#### Executor logs

- `SPARK_WORKER_DIR/worker-n/application_id/executor_id/stderr`
- `SPARK_WORKER_DIR/worker-n/application_id/executor_id/stdout`

#### Spark Master/Worker logs

Spark Master: the global `system.log`

Spark Worker: `SPARK_WORKER_LOG_DIR/worker-n/worker.log`

The default `SPARK_WORKER_LOG_DIR` location is `/var/log/spark/worker`.

#### Default log directory for Spark SQL Thrift server

The default log directory for starting the Spark SQL Thrift server is `$HOME/spark-thrift-server`.

#### Spark Shell and application logs

Spark Shell and application logs are output to the console.

#### SparkR shell log

The default location for the SparkR shell is `$HOME/.sparkR.log`

#### Log configuration file

Log configuration files are located in the [same directory \(page 208\)](#) as `spark-env.sh`.

To configure Spark logging options:

1. Configure logging options, such as log levels, in the following files:

<b>Executors</b>	logback-spark-executor.xml
<b>Spark Master</b>	logback.xml
<b>Spark Worker</b>	logback-spark-server.xml
<b>Spark Driver (Spark Shell, Spark applications)</b>	logback-spark.xml
<b>SparkR</b>	logback-sparkR.xml

2. If you want to enable rolling logging for Spark executors, add the following options to `spark-daemon-defaults.conf`.

Enable rolling logging with 3 log files retained before deletion. The log files are broken up by size with a maximum size of 50,000 bytes.

```
spark.executor.logs.rolling.maxRetainedFiles 3
spark.executor.logs.rolling.strategy size
spark.executor.logs.rolling.maxSize 50000
```

The default location of the Spark configuration files depends on the type of installation:

- **Package installations:** `/etc/dse/spark/`
- **Tarball installations:** `installation_location/resources/spark/conf`

3. Configure a safe communication channel to access the Spark user interface.

**Note:** When user credentials are specified in plain text on the `dse` command line, like `dse -u username -p password`, the credentials are present in the logs of Spark workers when the driver is run in cluster mode.

The Spark Master, Spark Worker, executor, and driver logs might include sensitive information. Sensitive information includes passwords and digest authentication tokens for [Kerberos guidelines](#) mode that are passed in the command line or Spark configuration. DataStax recommends using only safe communication channels like VPN and SSH to access the Spark user interface.

**Tip:** You can provide authentication credentials in several ways, see [Credentials for authentication](#).

## Running Spark processes as separate users

Spark processes can be configured to run as separate operating system users.

Spark processes can be configured to run as separate operating system users.

By default, processes started by DSE are run as the same OS user who started the DSE server process. This is called the DSE service user. One consequence of this is that all

applications that are run on the cluster can access DSE data and configuration files, and access files of other applications.

You can delegate running Spark applications to runner processes and users by changing options in `dse.yaml`.

### **Overview of the `run_as` process runner**

The `run_as` process runner allows you to run Spark applications as a different OS user than the DSE service user. When this feature is enabled and configured:

- All simultaneously running applications deployed by a single DSE service user will be run as a single OS user.
- Applications deployed by different DSE service users will be run by different OS users.
- All applications will be run as a different OS user than the DSE service user.

This allows you to prevent an application from accessing DSE server private files, and prevent one application from accessing the private files of another application.

### **How the `run_as` process runner works**

DSE uses `sudo` to run Spark applications components (drivers and executors) as specific OS users. DSE doesn't link a DSE service user with a particular OS user. Instead, a configurable number of spare user accounts or *slots* are used. When a request to run an executor or a driver is received, DSE finds an unused slot, and locks it for that application. Until the application is finished, all of that application's processes run as that slot user. When the application completes, the slot user will be released and will be available to other applications.

Since the number of slots is limited, a single slot is shared among all the simultaneously running applications run by the same DSE service user. Such a slot is released once all the applications of that user are removed. When there is not enough slots to run an application, an error is logged and DSE will try to run the executor or driver on a different node. DSE does not limit the number of slots you can configure. If you need to run more applications simultaneously, create more slot users.

Slots assignment is done on a per node basis. Executors of a single application may run as different slot users on different DSE nodes. When DSE is run on a fat node, different DSE instances running within the same OS should be configured with different sets of slot users. If they use the same slot users, a single OS user may run the applications of two different DSE service users.

When a slot is released, all directories which are normally managed by Spark for the application are removed. If the application doesn't finish, but all executors are done on a node, and a slot user is about to be released, all the application files are modified so that their ownership is changed to the DSE service user with owner-only permission. When a new executor for this application is run on this node, the application files are reassigned back to the slot user assigned to that application.

### **Configuring the `run_as` process runner**

The administrator needs to prepare slot users in the OS before configuring DSE. The `run_as` process runner requires:

- Each slot user has its own primary group, which name is the same as the name of slot user. This is typically the default behaviour of the OS. For example, the `slot1` user's primary group is `slot1`.
- The DSE service user is a member of each slot's primary group. For example, if the DSE service user is `cassandra`, the `cassandra` user is a member of the `slot1` group.
- The DSE service user is a member of a group with the same name as the service user. For example, if the DSE service user is `cassandra`, the `cassandra` user is a member of the `cassandra` group.
- `sudo` is configured so that the DSE service user can execute any command as any slot user without providing a password.

Override the `umask` setting to `007` for slot users so that files created by sub-processes will not be accessible by anyone else by default, and DSE configuration files are not visible to slot users.

You may further secure the DSE server environment by modifying the OS's `limits.conf` file to set exact disk space quotas for each slot user.

After adding the slot users and groups and configuring the OS, modify the `dse.yaml` file. In the `spark_process_runner` section enable the `run_as` process runner and set the list of slot users on each node.

```
spark_process_runner:
 # Allowed options are: default, run_as
 runner_type: run_as

 run_as_runner_options:
 user_slots:
 - slot1
 - slot2
```

## Example configuration for `run_as` process runner

In this example, two slot users, `slot1` and `slot2` will be created and configured with DSE. The default DSE service user of `cassandra` is used.

1. Create the slot users.

```
sudo useradd -r -s /bin/false slot1 &&
sudo useradd -r -s /bin/false slot2
```

2. Add the slot users to the DSE service user's group.

```
sudo usermod -a -G slot1,slot2 cassandra
```

3. Make sure the DSE service user is a member of a group with the same name as the service user. For example, if the DSE service user is `cassandra`:

```
groups cassandra
```

```
cassandra : cassandra
```

4. Log out and back in again to make the group changes take effect.
5. Modify the `sudoers` file with the slot users.

```
Runas_Alias SLOTS = slot1, slot2
Defaults>SLOTS umask=007
Defaults>SLOTS umask_override
cassandra ALL=(SLOTS) NOPASSWD: ALL
```

6. Modify `dse.yaml` to enable the `run_as` process runner and add the new runners.

```
Configure the way how the driver and executor processes are
created and managed.
spark_process_runner:
 # Allowed options are: default, run_as
 runner_type: run_as

 # RunAs runner uses sudo to start Spark drivers and executors. A
 # set of predefined fake users, called slots, is used
 # for this purpose. All drivers and executors owned by some DSE
 # user are run as some slot user x. At the same time
 # drivers and executors of any other DSE user use different
 # slots.
 run_as_runner_options:
 user_slots:
 - slot1
 - slot2
```

## Configuring the Spark history server

Load the event logs from Spark jobs that were run with event logging enabled.

The Spark history server provides a way to load the event logs from Spark jobs that were run with event logging enabled. The Spark history server works only when files were not flushed before the Spark Master attempted to build a history user interface.

To enable the Spark history server:

1. Create a directory for event logs in the DSEFS file system:

```
dse fs 'mkdir -p /spark/events'
```

2. On each node in the cluster, edit the `spark-defaults.conf` file to enable event logging and specify the directory for event logs:

```
#Turns on logging for applications submitted from this machine
spark.eventLog.dir dsefs:///spark/events
spark.eventLog.enabled true
#Sets the logging directory for the history server
spark.history.fs.logDirectory dsefs:///spark/events
Optional property that changes permissions set to event log
files
spark.eventLog.permissions=777
```

3. Start the Spark history server on one of the nodes in the cluster:

The Spark history server is a front-end application that displays logging data from all nodes in the Spark cluster. It can be started from any node in the cluster.

If you've enabled authentication set the authentication method and credentials in a properties file and pass it to the `dse` command. For example, for basic authentication:

```
spark.hadoop.com.datastax.bdp.fs.client.authentication.basic.username=role
name
spark.hadoop.com.datastax.bdp.fs.client.authentication.basic.password=password
```

If you set the event log location in `spark-defaults.xml`, set the `spark.history.fs.logDirectory` property in your properties file.

```
spark.history.fs.logDirectory=dsefs:///spark/events
```

```
dse spark-history-server start
```

With a properties file:

```
dse spark-history-server start --properties-file properties file
```

The history server is started and can be viewed by opening a browser to `http://node hostname:18080`.

**Note:** The Spark Master web UI does not show the historical logs. To work around this known issue, access the history from port 18080.

- When event logging is enabled, the default behavior is for all logs to be saved, which causes the storage to grow over time. To enable automated cleanup edit `spark-defaults.conf` and edit the following options:

```
spark.history.fs.cleaner.enabled true
spark.history.fs.cleaner.interval 1d
spark.history.fs.cleaner.maxAge 7d
```

For these settings, automated cleanup is enabled, the cleanup is performed daily, and logs older than seven days are deleted.

## Setting Spark Cassandra Connector-specific properties

Use the Spark Cassandra Connector options to configure DataStax Enterprise Spark.

Spark integration uses the Spark Cassandra Connector under the hood. You can use the configuration options defined in that project to configure DataStax Enterprise Spark. Spark recognizes system properties that have the `spark.` prefix and adds the properties to the configuration object implicitly upon creation. You can avoid adding system properties to the configuration object by passing `false` for the `loadDefaults` parameter in the `SparkConf` constructor.

The full list of parameters is included in the [Spark Cassandra Connector documentation](#).

You pass settings for Spark, Spark Shell, and other DataStax Enterprise Spark built-in applications using the intermediate application `spark-submit`, described in [Spark documentation](#).

## Configuring the Spark shell

Pass Spark configuration arguments using the following syntax:

```
dse spark [submission_arguments] [application_arguments]
```

where `submission_arguments` are:

- `--properties-file path_to_properties_file`

The location of the properties file that has the configuration settings. By default, Spark loads the settings from `spark-defaults.conf`.

- `--executor-memory memory`

How much memory to allocate on each machine for the application. You can provide the memory argument in JVM format using either the k, m, or g suffix.

- `--total-executor-cores cores`

The total number of cores the application uses

- `--conf name=value`

An arbitrary Spark option to the Spark configuration prefixed by `spark`.

- `--help`

Shows a help message that displays all options except DataStax Enterprise Spark shell options.

- `--jars <additional-jars>`

A comma-separated list of paths to additional JAR files.

- `--verbose`

Displays which arguments are recognized as Spark configuration options and which arguments are forwarded to the Spark shell.

Spark shell application arguments:

- `-i file`

Runs a script from the specified file.

## Configuring Spark applications

You pass the Spark submission arguments using the following syntax:

```
dse spark-submit [submission_arguments] application_file
[application_arguments]
```

All `submission_arguments` ([page 227](#)) and these additional `spark-submit` `submission_arguments`:

- `--class class_name`  
The full name of the application main class.
- `--name name`  
The application name as displayed in the Spark web application.
- `--py-files files`  
A comma-separated list of the .zip, .egg, or .py files that are set on PYTHONPATH for Python applications.
- `--files files`  
A comma-separated list of files that are distributed among the executors and available for the application.

In general, Spark submission arguments are translated into system properties – `Dname=value` and other VM parameters like `classpath`. The application arguments are passed directly to the application.

## Property list

When you run `dse spark-submit` on a node in your Analytics cluster, all the following properties are set automatically, and the Spark Master is automatically detected. Only set the following properties if you need to override the automatically managed properties.

### **spark.cassandra.connection.native.port**

Default = 9042. Port for native client protocol connections.

### **spark.cassandra.connection.rpc.port**

Default = 9160. Port for thrift connections.

### **spark.cassandra.connection.host**

The host name or IP address to which the Thrift RPC service and native transport is bound. The `native_transport_address` property in the `cassandra.yaml`, which is `localhost` by default, determines the default value of this property.

You can explicitly set the Spark Master [address \(page 182\)](#) using the `--master master address` parameter to `dse spark-submit`.

```
dse spark-submit --master master address application JAR file
```

For example, if the Spark node is at 10.0.0.2:

```
dse spark-submit --master dse://10.0.0.2? myApplication.jar
```

The following properties can be overridden for performance or availability:

## Read properties

### **spark.cassandra.input.split.size**

Default = 100000. Approximate number of rows in a single Spark partition. The higher the value, the fewer Spark tasks are created. Increasing the value too much may limit the parallelism level.

### **spark.cassandra.input.fetch.size\_in\_rows**

Default = 1000. Number of rows being fetched per round-trip to the database. Increasing this value increases memory consumption. Decreasing the value increases the number of round-trips. In earlier releases, this property was **spark.cassandra.input.page.row.size**.

#### **spark.cassandra.input.consistency.level**

Default = LOCAL\_ONE. Consistency level to use when reading.

### Write properties

You can set the following properties in `SparkConf` to fine tune the saving process.

#### **spark.cassandra.output.batch.size.bytes**

Default = auto. Number of bytes per single batch. The default, auto, means the connector adjusts the number of bytes based on the amount of data.

#### **spark.cassandra.output.consistency.level**

Default = LOCAL\_ONE. Consistency level to use when writing.

#### **spark.cassandra.output.concurrent.writes**

Default = 100. Maximum number of batches executed in parallel by a single Spark task.

#### **spark.cassandra.output.batch.size.rows**

Default = 64K. The maximum total size of the batch in bytes.

See the [Spark Cassandra Connector documentation](#) for details on additional, low-level properties.

## Creating a DSE Analytics Solo datacenter

DSE Analytics Solo datacenters do not store any database or search data, but are strictly used for analytics processing. They are used in conjunction with one or more datacenters that contain database data.

DSE Analytics Solo datacenters do not store any database or search data, but are strictly used for analytics processing. They are used in conjunction with one or more datacenters that contain database data.

### Creating a DSE Analytics Solo datacenter within an existing DSE cluster

In this example scenario, there is an existing datacenter, DC1 which has existing database data. Create a new DSE Analytics Solo datacenter, DC2, which does not store any data but will perform analytics jobs using the database data from DC1.

- Make sure all keyspaces in the DC1 datacenter use `NetworkTopologyStrategy`. If necessary, alter the keyspace.

```
ALTER KEYSPACE mykeyspace
WITH REPLICATION = { 'class' = 'NetworkTopologyStrategy', 'DC1' :
 3 };
```

- Add nodes to a new datacenter named DC2, then [enable Analytics on those nodes \(page 867\)](#).
- Configure the `dse_leases` and `dse_analytics` keyspaces to replicate to both DC1 and DC2. For example:

```
ALTER KEYSPACE dse_leases
WITH REPLICATION = { 'class' = 'NetworkTopologyStrategy', 'DC1' :
 3, 'DC2' : 3 };
```

- When submitting Spark applications specify the --master URL with the name or IP address of a node in the DC2 datacenter, and set the spark.cassandra.connection.local\_dc configuration option to DC1.

```
dse spark-submit --master "dse://?connection.local_dc=DC2"
--class com.datastax.dse.demo.loss.Spark10DayLoss --conf
"spark.cassandra.connection.local_dc=DC1" portfolio.jar
```

The Spark workers read the data from the DC1.

## Accessing an external DSE transactional cluster from a DSE Analytics Solo cluster

To access an external DSE transactional cluster, explicitly set the connection to the transactional cluster when creating RDDs or Datasets within the application.

In the following examples, the external DSE transactional cluster has a node running on 10.10.0.2.

To create an RDD from the transactional cluster's data:

```
import com.datastax.spark.connector._
import com.datastax.spark.connector.cql._
import org.apache.spark.SparkContext

def analyticsSoloExternalDataExample (sc: SparkContext) = {
 val connectorToTransactionalCluster =
 CassandraConnector(sc.getConf.set("spark.cassandra.connection.host",
 "10.10.0.2"))

 val rddFromTransactionalCluster = {
 // Sets connectorToTransactionalCluster as default connection for
 // everything in this code block
 implicit val c = connectorToTransactionalCluster
 // get the data from the test.words table
 sc.cassandraTable("test","words")
 }
}
```

Creating a Dataset from the transactional :

```
import org.apache.spark.sql.cassandra._
import com.datastax.spark.connector.cql.CassandraConnectorConf

// set params for the particular cluster
spark.setCassandraConf("TransactionalCluster",
 CassandraConnectorConf.ConnectionHostParam.option("10.10.0.2"))

val df = spark
 .read
```

```
.format("org.apache.spark.sql.cassandra")
.options(Map("table" -> "words", "keyspace" -> "test"))
.load()
```

When you submit the application to the DSE Analytics Solo cluster, it will retrieve the data from the external DSE transactional cluster.

## Spark JVMs and memory management

Spark jobs running on DataStax Enterprise are divided among several different JVM processes.

Spark jobs running on DataStax Enterprise are divided among several different JVM processes, each with different memory requirements.

### DataStax Enterprise and Spark Master JVMs

The Spark Master runs in the same process as DataStax Enterprise, but its memory usage is negligible. The only way Spark could cause an `OutOfMemoryError` in DataStax Enterprise is indirectly by executing queries that fill the client request queue. For example, if it ran a query with a high limit and paging was disabled or it used a very large batch to update or insert data in a table. This is controlled by `MAX_HEAP_SIZE` in `cassandra-env.sh`. If you see an `OutOfMemoryError` in `system.log`, you should treat it as a standard `OutOfMemoryError` and follow the usual troubleshooting steps.

### Spark executor JVMs

The Spark executor is where Spark performs transformations and actions on the RDDs and is usually where a Spark-related `OutOfMemoryError` would occur. An `OutOfMemoryError` in an executor will show up in the `stderr` log for the currently executing application (usually in `/var/lib/spark`). There are several configuration settings that control executor memory and they interact in complicated ways.

- The `memory_total` option in the `resource_manager_options.worker_options` section of `dse.yaml` defines the maximum fraction of system memory to give *all* executors for *all* applications running on a particular node. It uses the following formula:

$$\text{memory\_total} * (\text{total system memory} - \text{memory assigned to DataStax Enterprise})$$

- `spark.executor.memory` is a system property that controls how much executor memory a *specific* application gets. It must be less than or equal to the calculated value of `memory_total`. It can be specified in the constructor for the `SparkContext` in the driver application, or via `--conf spark.executor.memory` or `--executor-memory` command line options when submitting the job using `spark-submit`.

### The client driver JVM

The driver is the client program for the Spark job. Normally it shouldn't need very large amounts of memory because most of the data should be processed within the executor. If it does need more than a few gigabytes, your application may be using an anti-pattern like pulling all of the data in an RDD into a local data structure by using `collect` or `take`. Generally you should never use `collect` in production code and if you use `take`, you

should be only taking a few records. If the driver runs out of memory, you will see the `OutOfMemoryError` in the driver `stderr` or wherever it's been configured to log. This is controlled one of two places:

- `SPARK_DRIVER_MEMORY` in `spark-env.sh`
- `spark.driver.memory` system property which can be specified via `--conf spark.driver.memory` or `--driver-memory` command line options when submitting the job using `spark-submit`. This *cannot* be specified in the `SparkContext` constructor because by that point, the driver has already started.

## Spark worker JVMs

The worker is a watchdog process that spawns the executor, and should never need its heap size increased. The worker's heap size is controlled by `SPARK_DAEMON_MEMORY` in `spark-env.sh`. `SPARK_DAEMON_MEMORY` also affects the heap size of the Spark SQL thrift server.

## Using Spark modules with DataStax Enterprise

Spark Streaming, Spark SQL, and MLlib are modules that extend the capabilities of Spark.

### Getting started with Spark Streaming

Spark Streaming allows you to consume live data streams from sources, including Akka, Kafka, and Twitter. This data can then be analyzed by Spark applications, and the data can be stored in the database. This example uses Scala.

[Spark Streaming](#) allows you to consume live data streams from sources, including Akka, Kafka, and Twitter. This data can then be analyzed by Spark applications, and the data can be stored in the database.

You use Spark Streaming by creating an `org.apache.spark.streaming.StreamingContext` instance based on your Spark configuration. You then create a `DStream` instance, or a *discretized stream*, an object that represents an input stream. `DStream` objects are created by calling one of the methods of `StreamingContext`, or using a utility class from external libraries to connect to other sources like Twitter.

The data you consume and analyze is saved to the database by calling one of the `saveToCassandra` methods on the stream object, passing in the keyspace name, the table name, and optionally the column names and batch size.

**Note:** Spark Streaming applications require synchronized clocks to operate correctly. See [Synchronize clocks \(page 62\)](#).

The following Scala example demonstrates how to connect to a text input stream at a particular IP address and port, count the words in the stream, and save the results to the database.

1. Import the streaming context objects.

```
import org.apache.spark.streaming._
```

2. Create a new `StreamingContext` object based on an existing `SparkConf` configuration object, specifying the interval in which streaming data will be divided into batches by passing in a batch duration.

```
val sparkConf =
val ssc = new StreamingContext(sc, Seconds(1)) // Uses the context
automatically created by the spark shell
```

Spark allows you to specify the batch duration in milliseconds, seconds, and minutes.

3. Import the database-specific functions for `StreamingContext`, `DStream`, and `RDD` objects.

```
import com.datastax.spark.connector.streaming._
```

4. Create the `DStream` object that will connect to the IP and port of the service providing the data stream.

```
val lines = ssc.socketTextStream(server IP address, server port number)
```

5. Count the words in each batch and save the data to the table.

```
val words = lines.flatMap(_.split(" "))
val pairs = words.map(word => (word, 1))
val wordCounts = pairs.reduceByKey(_ + _)
 .saveToCassandra("streaming_test", "words_table",
SomeColumns("word", "count"))
```

6. Start the computation.

```
ssc.start()
ssc.awaitTermination()
```

In the following example, you start a service using the `nc` utility that repeats strings, then consume the output of that service using Spark Streaming.

Using `cqlsh`, start by creating a target keyspace and table for streaming to write into.

```
CREATE KEYSPACE IF NOT EXISTS streaming_test
WITH REPLICATION = {'class': 'SimpleStrategy',
'replication_factor': 1 };

CREATE TABLE IF NOT EXISTS streaming_test.words_table
(word TEXT PRIMARY KEY, count COUNTER);
```

In a terminal window, enter the following command to start the service:

```
nc -lk 9999
one two two three three three four four four four someword
```

In a different terminal start a Spark shell.

```
dse spark
```

In the Spark shell enter the following:

```
import org.apache.spark.streaming._
import com.datastax.spark.connector.streaming._

val ssc = new StreamingContext(sc, Seconds(1))
val lines = ssc.socketTextStream("localhost", 9999)
val words = lines.flatMap(_.split(" "))
val pairs = words.map(word => (word, 1))

val wordCounts = pairs.reduceByKey(_ + _)
wordCounts.saveToCassandra("streaming_test", "words_table",
 SomeColumns("word", "count"))
wordCounts.print()
ssc.start()
ssc.awaitTermination()
exit()
```

Using `cqlsh` connect to the `streaming_test` keyspace and run a query to show the results.

```
cqlsh -k streaming_test
```

```
select * from words_table;
```

word	count
three	3
one	1
two	2
four	4
someword	1

### What's next:

Run the [http\\_receiver demo \(page 270\)](#). See the [Spark Streaming Programming Guide](#) for more information, API documentation, and examples on Spark Streaming.

## Creating a Spark Structured Streaming sink using DSE

A Spark Structured Streaming sink pulls data into DSE.

Spark Structured Streaming is a high-level API for streaming applications. DSE supports Structured Streaming for storing data into DSE.

The following Scala example shows how to store data from a streaming source to DSE using the `cassandraFormat` method.

```
val query = source.writeStream
```

```
.option("checkpointLocation", checkpointDir.toString)
.cassandraFormat("table name", "keyspace name")
.outputMode(OutputMode.Update)
.start()
```

This example sets the `OutputMode` to `Update`, described in the [Spark API documentation](#).

The `cassandraFormat` method is equivalent to calling the `format` method and in `org.apache.spark.sql.cassandra`.

```
val query = source.writeStream
.option("checkpointLocation", checkpointDir.toString)
.format("org.apache.spark.sql.cassandra")
.option("keyspace", ks)
.option("table", "kv")
.outputMode(OutputMode.Update)
.start()
```

## Using Spark SQL to query data

Spark SQL allows you to execute Spark queries using a variation of the SQL language.

Spark SQL allows you to execute Spark queries using a variation of the SQL language. Spark SQL includes APIs for returning Spark `Datasets` in Scala and Java, and interactively using a SQL shell.

### Spark SQL basics

In DSE, Spark SQL allows you to perform relational queries over data stored in DSE clusters, and executed using Spark. Spark SQL is a unified relational query language for traversing over distributed collections of data, and supports a variation of the SQL language used in relational databases. Spark SQL is intended as a replacement for Shark and Hive, including the ability to run SQL queries over Spark data sets. You can use traditional Spark applications in conjunction with Spark SQL queries to analyze large data sets.

The `SparkSession` class and its subclasses are the entry point for running relational queries in Spark.

`DataFrames` are Spark `Datasets` organized into named columns, and are similar to tables in a traditional relational database. You can create `DataFrame` instances from any Spark data source, like CSV files, Spark RDDs, or, for DSE, tables in the database. In DSE, when you access a Spark SQL table from the data in DSE transactional cluster, it registers that table to the Hive metastore so SQL queries can be run against it.

**Note:** Any tables you create or destroy, and any table data you delete, in a Spark SQL session will not be reflected in the underlying DSE database, but only in that session's metastore.

### Starting the Spark SQL shell

The Spark SQL shell allows you to interactively perform Spark SQL queries. To start the shell, run `dse spark-sql`:

```
dse spark-sql
```

The Spark SQL shell in DSE automatically creates a Spark session and connects to the [Spark SQL Thrift server \(page 243\)](#) to handle the underlying JDBC connections.

## Spark SQL limitations

- You cannot load data from one file system to a table in a different file system.

```
CREATE TABLE IF NOT EXISTS test (id INT, color STRING) PARTITIONED
 BY (ds STRING);
LOAD DATA INPATH 'hdfs2://localhost/colors.txt' OVERWRITE INTO
 TABLE test PARTITION (ds = '2008-08-15');
```

The first line creates a table on the default file system. The second line attempts to load data into that table from a path on a different file system, and will fail.

## Querying database data using Spark SQL in Scala

You can execute Spark SQL queries in Scala by starting the Spark shell. When you start Spark, DataStax Enterprise creates a Spark session instance to allow you to run Spark SQL queries against database tables.

When you start Spark, DataStax Enterprise creates a Spark session instance to allow you to run Spark SQL queries against database tables. The session object is named `spark` and is an instance of `org.apache.spark.sql.SparkSession`. Use the `sql` method to execute the query.

1. Start the Spark shell.

```
dse spark
```

2. Use the `sql` method to pass in the query, storing the result in a variable.

```
val results = spark.sql("SELECT * from my_keyspace_name.my_table")
```

3. Use the returned data.

```
results.show()
```

	id	description
de2d0de1-4d70-11e...	thing	
db7e4191-4d70-11e...	another	
d576ad50-4d70-11e...	yet another	

## Querying database data using Spark SQL in Java

You can execute Spark SQL queries in Java applications that traverse over tables. Java applications that query table data using Spark SQL require a Spark session instance.

Java applications that query table data using Spark SQL first need an instance of `org.apache.spark.sql.SparkSession`.

The Spark session object is used to connect to DataStax Enterprise.

Create the Spark session instance using the [builder interface](#):

```
SparkSession spark = SparkSession
 .builder()
 .appName("My application name")
 .config("option name", "option value")
 .master("dse://1.1.1.1?connection.host=1.1.2.2,1.1.3.3")
 .getOrCreate();
```

After the Spark session instance is created, you can use it to create a `DataFrame` instance from the query. Queries are executed by calling the `SparkSession.sql` method.

```
DataFrame employees = spark.sql("SELECT * FROM company.employees");
employees.registerTempTable("employees");
DataFrame managers = spark.sql("SELECT name FROM employees WHERE role
== 'Manager'");
```

The returned `DataFrame` object supports the standard Spark operations.

```
employees.collect();
```

## Querying DSE Graph vertices and edges with Spark SQL

Spark SQL can query DSE Graph vertex and edge tables.

Spark SQL can query DSE Graph vertex and edge tables. The `dse_graph` database holds the vertex and edge tables for each graph. The naming format for the tables is `graph_name_vertices` and `graph_name_edges`. For example, if you have a graph named `gods`, the vertices and edges are accessible in Spark SQL in the `dse_graph.gods_vertices` and `dse_graph.gods_edges` tables.

```
select * from dse_graph.gods_vertices;
```

If you have properties that are spelled the same but with different capitalizations (for example, `id` and `Id`), start Spark SQL with the `--conf spark.sql.caseSensitive=true` option.

### Prerequisites:

Start your cluster with [both Graph and Spark enabled \(page 867\)](#).

1. Start the Spark SQL shell.

```
dse spark-sql
```

2. Query the vertices and edges using `SELECT` statements.

```
USE dse_graph;
SELECT * FROM gods_vertices where name = 'Zeus';
```

### 3. Join the vertices and edges in a query.

Vertices are identified by `id` columns. Edge tables have `src` and `dst` columns that identify the from and to vertices, respectively. A join can be used to traverse the graph. For example to find all vertex ids that are reached by the out edges:

```
SELECT gods_edges.dst FROM gods_vertices JOIN gods_edges ON
 gods_vertices.id = gods_edges.src;
```

**What's next:** The same steps work from the Spark shell using `spark.sql()` to run the query statements, or using the [JDBC \(page 252\)](#)/[ODBC \(page 252\)](#) driver and the [Spark SQL Thrift Server \(page 243\)](#).

## Using Spark predicate push down in Spark SQL queries

Spark predicate push down to database allows for better optimized Spark SQL queries.

Spark predicate push down to database allows for better optimized Spark SQL queries. By using the `EXPLAIN` command in Spark SQL, queries can be analyzed to see if the predicates need to be cast to the correct data type.

When creating Spark SQL queries that use comparison operators, making sure the predicates are pushed down to the database correctly is critical to retrieving the correct data with the best performance. This is particularly true for string representations of predicate data.

For example, given a CQL table with the following schema:

```
CREATE TABLE test.common (
 year int,
 birthday timestamp,
 userid uuid,
 likes text,
 name text,
 PRIMARY KEY (year, birthday, userid)
)
```

Suppose you want to write a query that selects all entries where the birthday is earlier than a given date:

```
select * from test.common where birthday < '2001-1-1';
```

Use the `EXPLAIN` command to see the query plan:

```
EXPLAIN SELECT * FROM test.common WHERE birthday < '2001-1-1';

== Physical Plan ==
*Filter (cast(birthday#1 as string) < 2001-1-1)
+- *Scan org.apache.spark.sql.cassandra.CassandraSourceRelation
 [year#0,birthday#1,userid#2,likes#3,name#4] ReadSchema:
 struct<year:int,birthday:timestamp,userid:string,likes:string,name:string>
Time taken: 0.72 seconds, Fetched 1 row(s)
```

Note that the `Filter` directive is treating the `birthday` column, a CQL `TIMESTAMP`, as a string. Using the less than operator for string comparison is not what is intended for the query.

To push down the correct predicate for this query, use the `cast` method to specify that `birthday` is a `TIMESTAMP`.

```
EXPLAIN SELECT * FROM test.common WHERE birthday < cast('2001-1-1' as
TIMESTAMP);

== Physical Plan ==
*Scan org.apache.spark.sql.cassandra.CassandraSourceRelation
[year#0,birthday#1,userid#2,likes#3,name#4]
PushedFilters: [*LessThan(birthday,2001-01-01 00:00:00.0)],
ReadSchema:
struct<year:int,birthday:timestamp,userid:string,likes:string,name:string>
Time taken: 0.034 seconds, Fetched 1 row(s)
```

Note the `PushedFilters` indicating that the `LessThan` predicate will be pushed down for the column data in `birthday`.

## Supported syntax of Spark SQL

Spark SQL supports a subset of the SQL-92 language.

### Syntax:

The following syntax defines a `SELECT` query.

```
SELECT [DISTINCT] [column names] | [wildcard]
FROM [keyspace name.]table name
[JOIN clause table name ON join condition]
[WHERE condition]
[GROUP BY column name]
[HAVING conditions]
[ORDER BY column names [ASC | DSC]]
```

A `SELECT` query using joins has the following syntax.

```
SELECT statement
FROM statement
[JOIN | INNER JOIN | LEFT JOIN | LEFT SEMI JOIN | LEFT OUTER JOIN |
RIGHT JOIN | RIGHT OUTER JOIN | FULL JOIN | FULL OUTER JOIN]
ON join condition
```

Several select clauses can be combined in a `UNION`, `INTERSECT`, or `EXCEPT` query.

```
SELECT statement 1
[UNION | UNION ALL | UNION DISTINCT | INTERSECT | EXCEPT]
SELECT statement 2
```

**Note:** Select queries run on new columns return ' ', or empty results, instead of None.

### Syntax:

The following syntax defines an `INSERT` query.

```
INSERT [OVERWRITE] INTO [keyspace name.]table name [(columns)]
VALUES values
```

### Syntax:

The following syntax defines a `CACHE TABLE` query.

```
CACHE TABLE table name [AS table alias]
```

You can remove a table from the cache using a `UNCACHE TABLE` query.

```
UNCACHE TABLE table name
```

## Keywords in Spark SQL

The following keywords are reserved in Spark SQL.

ALL  
AND  
AS  
ASC  
APPROXIMATE  
AVG  
BETWEEN  
BY  
CACHE  
CAST  
COUNT  
DESC  
DISTINCT  
FALSE  
FIRST  
LAST  
FROM  
FULL  
GROUP  
HAVING  
IF  
IN  
INNER  
INSERT  
INTO

```
IS
JOIN
LEFT
LIMIT
MAX
MIN
NOT
NULL
ON
OR
OVERWRITE
LIKE
RLIKE
UPPER
LOWER
REGEXP
ORDER
OUTER
RIGHT
SELECT
SEMI
STRING
SUM
TABLE
TIMESTAMP
TRUE
UNCACHE
UNION
WHERE
INTERSECT
EXCEPT
SUBSTR
SUBSTRING
SQRT
ABS
```

## Inserting data into tables with static columns using Spark SQL

Static columns are mapped to different columns in Spark SQL and require special handling.

Static columns are mapped to different columns in Spark SQL and require special handling. Spark SQL Thrift servers use Hive. When you run an insert query, you must pass data to those columns.

To work around the different columns, set `cql3.output.query` in the insertion Hive table properties to limit the columns that are being inserted. In Spark SQL, alter the external table to configure the prepared statement as the value of the Hive CQL output query. For example, this prepared statement takes values that are inserted into columns a and b in

`mytable` and maps these values to columns `b` and `a`, respectively, for insertion into the new row.

```
spark-sql> ALTER TABLE mytable SET TBLPROPERTIES ('cql3.output.query'
= 'update
 mykeyspace.mytable set b = ? where a = ?');
spark-sql> ALTER TABLE mytable SET SERDEPROPERTIES
('cql3.update.columns' =
 'b,a');
```

## Running HiveQL queries using Spark SQL

Spark SQL supports queries that are written using HiveQL, a SQL-like language that produces queries that are converted to Spark jobs.

Spark SQL supports queries written using HiveQL, a SQL-like language that produces queries that are converted to Spark jobs. HiveQL is more mature and supports more complex queries than Spark SQL. To construct a HiveQL query, first create a new `HiveContext` instance, and then submit the queries by calling the `sql` method on the `HiveContext` instance.

See the [Hive Language Manual](#) for the full syntax of HiveQL.

**Note:** Creating indexes with `DEFERRED REBUILD` is not supported in Spark SQL.

1. Start the Spark shell.

```
bin/dse spark
```

2. Use the provided `HiveContext` instance `sqlContext` to create a new query in HiveQL by calling the `sql` method on the `sqlContext` object..

```
val results = sqlContext.sql("SELECT * FROM
my_keyspace.my_table")
```

## Using the DataFrames API

The Spark DataFrame API encapsulates data sources, including DataStax Enterprise data, organized into named columns.

The Spark [DataFrames API](#) encapsulates data sources, including DataStax Enterprise data, organized into named columns.

The Spark Cassandra Connector provides an integrated `DataSource` to simplify creating `DataFrames`. For more technical details, see the [Spark Cassandra Connector documentation](#) that is maintained by DataStax and the [Cassandra and PySpark DataFrames](#) post.

## Examples of using the DataFrames API

This Python example shows using the `DataFrames API` to read from the table `ks.kv` and insert into a different table `ks.othertable`.

```
table1 = spark.read.format("org.apache.spark.sql.cassandra")
```

```

.options(table="kv", keyspace="ks")
.load()
table1.write.format("org.apache.spark.sql.cassandra")
.options(table="othertable", keyspace = "ks")
.save(mode = "append")

```

Using the DSE Spark console, the following Scala example shows how to create a DataFrame object from one table and save it to another.

```

val table1 = spark.read.format("org.apache.spark.sql.cassandra")
 .options(Map("table" -> "words", "keyspace" -> "test"))
 .load()
table1.createCassandraTable("test", "otherwords", partitionKeyColumns
 = Some(Seq("word")), clusteringKeyColumns = Some(Seq("count")))
table1.write.cassandraFormat("otherwords", "test").save()

```

The write operation uses one of the helper methods, `cassandraFormat`, included in the Spark Cassandra Connector. This is a simplified way of setting the format and options for a standard DataFrame operation. The following command is equivalent to write operation using `cassandraFormat`:

```

table1.write.format("org.apache.spark.sql.cassandra")
 .options(Map("table" -> "othertable", "keyspace" -> "test"))
 .save()

```

## Using the Spark SQL Thriftserver

The Spark SQL Thriftserver uses a JDBC and an ODBC interface for client connections to DSE.

The Spark SQL Thriftserver uses [JDBC and ODBC interfaces](#) for client connections to the database.

The [AlwaysOn SQL \(page 245\)](#) service is a high-availability service built on top of the Spark SQL Thriftserver. The Spark SQL Thriftserver is started manually on a single node in an Analytics datacenter, and will not failover to another node. Both AlwaysOn SQL and the Spark SQL Thriftserver provide JDBC and ODBC interfaces to DSE, and share many configuration settings.

1. If you are using Kerberos authentication, in the `hive-site.xml` file, configure your authentication credentials for the Spark SQL Thrift server.

```

<property>
 <name>hive.server2.authentication.kerberos.principal</name>
 <value>thriftserver/_HOST@EXAMPLE.COM</value>
</property>

<property>
 <name>hive.server2.authentication.kerberos.keytab</name>
 <value>/etc/dse/dse.keytab</value>
</property>

```

Ensure that you use the `hive-site.xml` file in the Spark directory:

- **Package installations:** /etc/dse/spark/hive-site.xml
  - **Tarball installations:** *installation\_location/resources/spark/conf/hive-site.xml*
2. Start DataStax Enterprise with Spark enabled as a [service \(page 867\)](#) or in a [standalone \(page 870\)](#) installation.
3. Start the server by entering the `dse spark-sql-thriftserver start` command as a user with permissions to write to the Spark directories.

To override the default settings for the server, pass in the configuration property using the **--hiveconf** option. See the [HiveServer2 documentation](#) for a complete list of configuration properties.

```
dse spark-sql-thriftserver start
```

By default, the server listens on port 10000 on the localhost interface on the node from which it was started. You can specify the server to start on a specific port. For example, to start the server on port 10001, use the **--hiveconf hive.server2.thrift.port=10001** option.

```
dse spark-sql-thriftserver start --hiveconf
hive.server2.thrift.port=10001
```

You can configure the port and bind address permanently in `resources/spark/conf/spark-env.sh`:

```
export HIVE_SERVER2_THRIFT_PORT=10001
export HIVE_SERVER2_THRIFT_BIND_HOST=1.1.1.1
```

You can specify general Spark configuration settings by using the **--conf** option.

```
dse spark-sql-thrift-server start --conf spark.cores.max=4
```

4. Use DataFrames to read and write large volumes of data. For example, to create the `table_a_cass_df` table that uses a DataFrame while referencing `table_a`:

```
CREATE TABLE table_a_cass_df using org.apache.spark.sql.cassandra
OPTIONS (table "table_a", keyspace "ks")
```

**Note:** With DataFrames, compatibility issues exist with `UUID` and `Inet` types when inserting data with the JDBC driver.

5. Use the [Spark Cassandra Connector](#) tuning parameters to optimize reads and writes.
6. To stop the server, enter the `dse spark-sql-thriftserver stop` command.

```
dse spark-sql-thriftserver stop
```

## What's next:

You can now connect your application by using the [Simba JDBC driver \(page 252\)](#) to the server at the URI: `jdbc:hive2://hostname:port number`, using the Simba ODBC driver for [Windows \(page 253\)](#) or [Linux \(page 254\)](#), or use `dse beeline (page 254)`.

## Using SparkR with DataStax Enterprise

Apache SparkR is a front-end for the R programming language for creating analytics applications. DataStax Enterprise integrates SparkR to support creating data frames from DSE data.

Apache SparkR is a front-end for the R programming language for creating analytics applications. DataStax Enterprise integrates SparkR to support creating data frames from DSE data.

SparkR support in DSE requires you to first install R on the client machines on which you will be using SparkR. To use R user defined functions and distributed functions the same version of R should be installed on all the nodes in the Analytics cluster. DSE SparkR is built against R version 3.1.1. Many Linux distributions by default install older versions of R.

For example, on Debian and Ubuntu clients:

```
sudo sh -c 'echo "deb http://cran.rstudio.com/bin/linux/ubuntu trusty/" >> /etc/apt/sources.list' &&
gpg --keyserver keyserver.ubuntu.com --recv-key E084DAB9 &&
gpg -a --export E084DAB9 | sudo apt-key add - &&
sudo apt-get update &&
sudo apt-get install r-base
```

On RedHat and CentOS clients:

```
sudo yum install R
```

## Starting SparkR

Start the SparkR shell using the [dse command \(page 795\)](#) to automatically set the Spark session within R.

1. Start the R shell using the `dse` command.

```
dse sparkR
```

## Using AlwaysOn SQL service

AlwaysOn SQL is a high availability service that responds to SQL queries from JDBC and ODBC applications.

AlwaysOn SQL is a high availability service that responds to SQL queries from JDBC and ODBC applications. It is built on top of the [Spark SQL Thriftserver \(page 243\)](#), but provides failover and caching between instances so there is no single point of failure. AlwaysOn SQL provides enhanced security, leveraging the same user management as the rest of DSE, executing queries to the underlying database as the user authenticated to AlwaysOn SQL.

Lifecycle Manager allows you to [enable and configure AlwaysOn SQL](#) in managed clusters.

When AlwaysOn SQL is enabled within an Analytics datacenter, all nodes within the datacenter must have AlwaysOn SQL enabled. Use [dsetool ring \(page 848\)](#) to find which nodes in the datacenter are Analytics nodes. The JDK with JPS tool is recommended for AlwaysOn SQL. By default, AlwaysOn SQL is disabled.

**Note:** AlwaysOn SQL is not supported when using [DSE Multi-Instance](#) or other deployments with multiple DSE instances on the same server.

The `dse client-tool alwayson-sql` command controls the server. The command works on the local datacenter unless you specify the datacenter with the `--dc` option:

```
dse client-tool alwayson-sql --dc datacenter_name command
```

## Checklist for enabling AlwaysOn SQL

In order to run AlwaysOn SQL, you must have:

- A running datacenter with [DSE Analytics nodes enabled](#).
- [Enabled AlwaysOn SQL \(page 246\)](#) on every Analytics node in the datacenter.
- Modified the [replication factor for all Analytics nodes \(page 171\)](#), if necessary.
- Set the `native_transport_address` in `cassandra.yaml` to an IP address accessible to the AlwaysOn SQL clients.
- Configured AlwaysOn SQL for [security \(page 250\)](#), if authentication is enabled.

## Enabling AlwaysOn SQL

Set `enabled` to `true` in the [AlwaysOn SQL options \(page 135\)](#) in `dse.yaml`.

## Configuring AlwaysOn SQL

The `alwayson_sql_options` section in `dse.yaml`, described in detail at [AlwaysOn SQL options \(page 135\)](#), has options for setting the ports, timeout values, log location, and other Spark or Hive configuration settings. Additional configuration options are located in `spark-alwayson-sql.conf`.

AlwaysOn SQL binds to the `native_transport_address` in `cassandra.yaml`.

If you have changed some configuration settings in `dse.yaml` while AlwaysOn SQL is running, you can have the server pick up the new configuration by entering:

```
dse client-tool alwayson-sql reconfig
```

The following settings can be changed using `reconfig`:

- `reserve_port_wait_time_ms`
- `alwayson_sql_status_check_wait_time_ms`
- `log_dsefs_dir`
- `runner_max_errors`

Changing other options requires a `restart`, except for `enabled`. Disabling AlwaysOn SQL requires [restarting DSE \(page 867\)](#).

The `spark-alwayson-sql.conf` file contains Spark and Hive settings as properties. When AlwaysOn SQL is started, `spark-alwayson-sql.conf` is scanned for Spark properties, similar

to other Spark applications started with `dse spark-submit`. Properties that begin with `spark.hive` are submitted as properties using `--hiveconf`, removing the `spark.` prefix.

For example, if `spark-alwayson-sql.conf` has the following setting:

```
spark.hive.server2.table.type.mapping CLASSIC
```

That setting will be converted to `--hiveconf`  
`hive.server2.table.type.mapping=CLASSIC` when AlwaysOn SQL is started.

## Configuring AlwaysOnSQL in a DSE Analytics Solo datacenter

If AlwaysOn SQL is used in a DSE Analytics Solo datacenter, modify `spark-alwayson-sql.conf` to configure Spark with the DSE Analytics Solo datacenters. In the following example, the transactional datacenter name is `dc0` and the DSE Analytics Solo datacenter is `dc1`.

Under `spark.master` set the Spark URI to the connect to the DSE Analytics Solo datacenter.

```
spark.master=dse://?connection.local_dc=dc1
```

Add the `spark.cassandra.connection.local_dc` property to `spark-alwayson-sql.conf` and set it to the name of the transactional datacenter.

```
spark.cassandra.connection.local_dc=dc0
```

## Starting and stopping AlwaysOn SQL

If you have enabled AlwaysOn SQL, it will start when the cluster is started. You only need to explicitly start the server if it has been stopped, for example for a configuration change.

To start AlwaysOn SQL service:

```
dse client-tool alwayson-sql start
```

To start the server on a specific datacenter, specify the datacenter name with the `--dc` option:

```
dse client-tool alwayson-sql --dc dc-west start
```

To completely stop AlwaysOn SQL service:

```
dse client-tool alwayson-sql stop
```

The server must be manually started after issuing a `stop` command.

To restart a running server:

```
dse client-tool alwayson-sql restart
```

## Checking the status of AlwaysOn SQL

To find the status of AlwaysOn SQL issue a `status` command using `dse-client-tool`.

```
dse client-tool alwayson-sql status
```

You can also view the status in a web browser by going to `http://node name or IP address:AlwaysOn SQL web UI port`. By default, the port is 9077. For example, if 10.10.10.1 is the IP address of an Analytics node with AlwaysOn SQL enabled, navigate to `http://10.10.10.1:9077`.

The returned status is one of:

- RUNNING: the server is running and ready to accept client requests.
- STOPPED\_AUTO\_RESTART: the server is being started but is not yet ready to accept client requests.
- STOPPED\_MANUAL\_RESTART: the server was stopped with either a `stop` or `restart` command. If the server was issued a `restart` command, the status will be changed to `STOPPED_AUTO_RESTART` as the server starts again.
- STARTING: the server is actively starting up but is not yet ready to accept client requests.

## Caching tables within Spark SQL queries

To increase performance, you can specify tables to be cached into RAM using the `CACHE TABLE` directive. Permanent cached tables will be recached on server restart.

You can cache an existing table by issuing a `CACHE TABLE` Spark SQL command through a client:

```
CACHE TABLE keyspace_name.table_name;
```

```
CACHE TABLE keyspace_name.table_name AS select statement;
```

The temporary cache table is only valid for the session in which it was created, and will not be recreated on server restart.

Create a permanent cache table using the `CREATE CACHE TABLE` directive and a `SELECT` query:

```
CREATE CACHE TABLE keyspace_name.table_name AS select_statement;
```

Cached tables can be destroyed using the `UNCACHE TABLE` and `CLEAR CACHE` directives.

```
UNCACHE TABLE keyspace_name.table_name;
```

The `CLEAR CACHE` directive removes all cached tables.

```
CLEAR CACHE;
```

## Enabling SSL for AlwaysOn SQL

Communication with the AlwaysOn SQL can be encrypted using SSL.

Communication between the driver and AlwaysOn SQL can be encrypted using SSL.

The following instructions give an example of how to set up SSL with a self-signed keystore and truststore.

1. Ensure [client-to-node encryption](#) is enabled and configured correctly.
2. If the SSL keystore and truststore used for AlwaysOn SQL differ from the keystore and truststore configured in `cassandra.yaml`, add the required settings to enable SSL to the `hive-site.xml` configuration file.

**Note:** By default the SSL settings in `cassandra.yaml` will be used with AlwaysOn SQL.

```
<property>
 <name>hive.server2.thrift.bind.host</name>
 <value>hostname</value>
</property>
<property>
 <name>hive.server2.use.SSL</name>
 <value>true</value>
</property>
<property>
 <name>hive.server2.keystore.path</name>
 <value>path to keystore/keystore.jks</value>
</property>
<property>
 <name>hive.server2.keystore.password</name>
 <value>keystore password</value>
</property>
```

3. Start or restart the AlwaysOn SQL service.

**Note:** Changes in the `hive-site.xml` configuration file only require a restart of AlwaysOn SQL service, not DSE.

```
dse client-tool alwayson-sql start
```

4. Test the connection with Beeline.

```
dse beeline

beeline> !connect jdbc:hive2://hostname:10000/
default;ssl=true;sslTrustStore=path to truststore/
truststore.jks;trustStorePassword=truststore password
```

**Note:** The JDBC URL for the Simba JDBC Driver is:

```
jdbc:spark://hostname:10000/default;SSL=1;SSLTrustStore=path to
truststore/truststore.jks;SSLTrustStorePwd=truststore password
```

## Using authentication with AlwaysOn SQL

AlwaysOn SQL can be configured to use DSE authentication.

AlwaysOn SQL can be configured to use DSE authentication.

When [DSE authentication is enabled](#), modify the `hive-site.xml` configuration file to enable JDBC authentication. DSE supports configurations for password authentication and Kerberos authentication. The `hive-site.xml` file has sections with preconfigured settings to use no authentication (the default), password authentication, or Kerberos authentication. Uncomment the preferred authentication mechanism, then restart AlwaysOn SQL.

**Note:** DSE supports multiple authentication mechanisms, but AlwaysOn SQL only supports one mechanism per datacenter.

AlwaysOn SQL supports [DSE proxy authentication](#). The user who executes the queries is the user who authenticated using JDBC. If AlwaysOn SQL was started by user Amy, and then Bob begins a JDBC session, the queries are executed by Amy on behalf of Bob. Amy must have permissions to execute these queries on behalf of Bob.

To enable authentication in AlwaysOn SQL [alwayson\\_sql\\_options \(page 135\)](#), follow these steps.

1. Create the [auth\\_user \(page 136\)](#) role specified in [AlwaysOn SQL options \(page 135\)](#) and grant the following permissions to the role.

```
CREATE ROLE alwayson_sql WITH LOGIN=true; // role name
matches auth_user (page 136)

// Required if scheme_permissions (page 106) true
GRANT EXECUTE ON ALL AUTHENTICATION SCHEMES TO alwayson_sql;

// Spark RPC settings
GRANT ALL PERMISSIONS ON REMOTE OBJECT DseResourceManager TO
alwayson_sql;
GRANT ALL PERMISSIONS ON REMOTE OBJECT DseClientTool TO
alwayson_sql;
GRANT ALL PERMISSIONS ON REMOTE OBJECT AlwaysOnSqlRoutingPRC to
alwayson_sql;

// Spark and DSE required table access
GRANT SELECT ON system.size_estimates TO alwayson_sql;
GRANT SELECT, MODIFY ON "HiveMetaStore".sparkmetastore TO
alwayson_sql;
GRANT SELECT, MODIFY ON dse_analytics.alwayson_sql_cache_table TO
alwayson_sql;
GRANT SELECT, MODIFY ON dse_analytics.alwayson_sql_info TO
alwayson_sql;
```

```
// Permissions to create and change applications
GRANT CREATE, DESCRIBE ON ANY WORKPOOL TO alwayson_sql;
GRANT MODIFY, DESCRIBE ON ANY SUBMISSION TO alwayson_sql;
```

See [Setting up DSE Spark application permissions](#) for more details.

- Set up a role that matches the full Kerberos principal name for each user.

```
CREATE ROLE 'user_name/example.com@EXAMPLE.COM'
WITH LOGIN = true;
```

**Note:** Grant permissions on all keyspaces and/or tables contains data the user needs to access.

- Grant permissions to access keyspaces and tables to the user role.

```
GRANT SELECT ON KEYSPACE keyspace_name
TO 'user_name/example.com@EXAMPLE.COM';
```

- Allow the AlwaysOn SQL role (`auth_user`) to execute commands with the user role.

```
GRANT PROXY.EXECUTE
ON ROLE 'user_name/example.com@EXAMPLE.COM'
TO alwayson_sql;
```

- Open the `hive-site.xml` configuration file in an editor.

- Uncomment and modify the authentication mechanism used in `hive-site.xml`.

- If password authentication is used, [enable password authentication in DSE](#).
- If Kerberos authentication is to be used, Kerberos does not need to be enabled in DSE. AlwaysOn SQL must have its own service principal and keytab.
- The user must have [login permissions](#) in DSE in order to login through JDBC to AlwaysOn SQL.

This example shows how to enable Kerberos authentication. Modify the Kerberos domain and path to the keytab file.

```
<!-- Start of: configuration for authenticating JDBC users with
Kerberos -->
<property>
 <name>hive.server2.enable.doAs</name>
 <value>true</value>
</property>

<property>
 <name>hive.server2.authentication</name>
 <value>KERBEROS</value>
</property>

<property>
 <name>hive.server2.authentication.kerberos.principal</name>
```

```
<value>hiveserver2/_HOST@KERBEROS DOMAIN</value>
</property>

<property>
<name>hive.server2.authentication.kerberos.keytab</name>
<value>path to hiveserver2.keytab</value>
</property>
<!-- End of: configuration for authenticating JDBC users with
Kerberos -->
```

## 7. Restart AlwaysOn SQL.

```
dse client-tool alwayson-sql restart
```

## Accessing AlwaysOn SQL with the Simba JDBC driver

The Simba JDBC driver allows you to access AlwaysOn SQL.

The Simba JDBC Driver for Spark provides a standard JDBC interface to the information stored in DataStax Enterprise with AlwaysOn SQL running.

Your DSE license includes a license to use the Simba drivers.

### Prerequisites:

You must have a running [DSE Analytics cluster with Spark enabled \(page 867\)](#), and have [enabled AlwaysOn SQL \(page 246\)](#).

1. Download the Simba JDBC Driver for Apache Spark from the [DataStax Drivers Download page](#).
2. Expand the ZIP file containing the driver.
3. In your JDBC application, configure the following details:
  - a. Add `SparkJDBC41.jar` and the rest of the JAR files included in the ZIP file in your classpath.
  - b. The JDBC driver class is `com.simba.spark.jdbc41.Driver` and the JDBC data source is `com.simba.spark.jdbc41.DataSource`.
  - c. Set the connection URL to `jdbc:spark://<hostname>:<port>` where `<hostname>` is the hostname of the node on which AlwaysOn SQL is running, and `<port>` is the port number on which AlwaysOn SQL is listening.  
`jdbc:spark://node1.example.com:10000`

4. For more details, refer to the included documentation in the Simba driver download ZIP.

## Simba ODBC Driver for Apache Spark (Windows)

The Simba ODBC Driver for Spark allows you to connect to AlwaysOn SQL from Windows.

The Simba ODBC Driver for Spark provides Windows users access to the information stored in DataStax Enterprise clusters with a running AlwaysOn SQL. This driver allows you to access the data stored on your DataStax Enterprise Spark nodes using business

intelligence (BI) tools, such as Tableau and Microsoft Excel. The driver is compliant with the latest ODBC 3.52 specification and automatically translates any SQL-92 query into Spark SQL.

Your DSE license includes a license to use the Simba drivers.

#### **Prerequisites:**

To use the Simba ODBC Driver for Spark you must have:

- One of the following operating systems:
    - # Windows 7 SP1
    - # Windows 8 or 8.1
    - # Windows Server 2008 R2 SP1
    - # Windows Server 2012 and Windows Server 2012 R2
  - A running [DSE Analytics cluster with Spark enabled \(page 867\)](#), and [enabled AlwaysOn SQL \(page 246\)](#).
1. Download the appropriate Simba ODBC Driver for Apache Spark (Windows 32- or 64-bit) from the [DataStax Drivers Download page](#).
  2. Double-click the downloaded installer and follow the installation wizard.
  3. Refer to the *Simba ODBC Driver for Spark Installation Guide* which is installed at **Start > Program Files > Simba Spark ODBC Driver**.

## **Configuring the Spark ODBC Driver (Windows)**

Adding a Simba ODBC Driver for Apache Spark data source to Windows.

Configure an ODBC data source for use by ODBC applications, including business intelligence (BI) tools like Tableau or Microsoft Excel.

1. Choose either the 32 bit or 64 bit ODBC driver.
  - a. For the 32-bit driver, click **Start > Program Files > Simba Spark ODBC Driver > 32 bit ODBC Data Source Administrator**.
  - b. For the 64-bit driver, click **Start > Program Files > Simba Spark ODBC Driver > 64 bit ODBC Data Source Administrator**.
2. Click the **Drivers** tab to verify that the Simba Spark ODBC Driver is present.
3. Create either a User or System DSN (data source name) for your ODBC tool connection.
  - a. Click the **User DSN** or **System DSN** tab.
  - b. Click **Add > Simba Spark ODBC Driver > Finish**.
  - c. In **Simba Spark ODBC Driver DSN Setup**, enter the following:

Data Source Name	The name for your DSN.
Description	Optional longer description of your DSN.
Spark Server Type	SparkThriftServer (Spark 1.1 and later)
Host(s)	IP or hostname of your AlwaysOn SQL service.
Port	Listening port for AlwaysOn SQL(default 10000)

Database	Specify <code>default</code> to load all tables into the default database. Or pick a specific keyspace.
Auth Mechanism	User Name
User Name	leave blank

d. Click **Test**.

The test results should indicate a successful connection.

4. For advanced configuration options, refer to the *Simba ODBC Driver for Spark Installation Guide* which is installed at **Start > Program Files > Simba Spark ODBC Driver**.

**What's next:** Use the newly created data source in ODBC applications like Tableau and Microsoft Excel.

After the ODBC query is transmitted to the Spark SQL Thrift server, the appropriate Spark jobs are executed, then the data is returned via ODBC to the application.

To troubleshoot or understand the queries being executed at AlwaysOn SQL, open a web browser to the Spark Master web interface (`http://node_name:4040`) on the DSE cluster, click on the Thrift server application, then view the SQL tab.

## Simba ODBC Driver for Apache Spark (Linux)

The Simba ODBC Driver for Spark allows you to connect to AlwaysOn SQL from Linux.

The Simba ODBC Driver for Spark provides Linux users access to the information stored in DataStax Enterprise clusters with a running AlwaysOn SQL. The driver is compliant with the latest ODBC 3.52 specification and automatically translates any SQL-92 query into Spark SQL.

Your DSE license includes a license to use the Simba drivers.

### Prerequisites:

To use the Simba ODBC Driver for Spark you must have:

- A running [DSE Analytics cluster with Spark enabled \(page 867\)](#), and [enabled AlwaysOn SQL \(page 246\)](#).
1. Download the appropriate Simba ODBC Driver for Apache Spark (Linux 32- or 64-bit) from the [DataStax Drivers Download page](#).
  2. Expand the downloaded file into a suitable location.

```
mkdir simba-odbc &&
cd simba-odbc &&
tar xvf version.tar.gz
```

3. Refer to the included *Spark ODBC Install and Configuration Guide* (PDF format) for detailed usage and configuration information.

## Connecting to AlwaysOn SQL server using Beeline

Use Spark Beeline to test AlwaysOn SQL.

You can use Shark Beeline to test [AlwaysOn SQL \(page 245\)](#).

1. [Start AlwaysOn SQL \(page 247\)](#).
2. Start the Beeline shell.

```
dse beeline
```

3. Connect to the server using the JDBC URI for your server.

```
beeline> !connect jdbc:hive2://localhost:10000
```

4. Connect to a keyspace and run a query from the Beeline shell.

```
0: jdbc:hive2://localhost:10000> use test;
0: jdbc:hive2://localhost:10000> select * from test;
```

## Accessing DataStax Enterprise data from external Spark clusters

Information on accessing data in DataStax Enterprise clusters from external Spark clusters, or Bring Your Own Spark (BYOS).

DataStax Enterprise works with external Spark clusters in a bring-your-own-Spark (BYOS) model.

### Overview of BYOS support in DataStax Enterprise

DataStax Enterprise provides a JAR and configuration files for connecting to DataStax Enterprise clusters from external Spark clusters.

BYOS support in DataStax Enterprise consists of a JAR file and a generated configuration file that provides all the necessary classes and configuration settings for connecting to a particular DataStax Enterprise cluster from an external Spark cluster. To specify a different classpath to accommodate applications originally written for open source Apache Spark, specify the [-framework \(page 797\)](#) option with `dse spark` commands.

All DSE resources, including DSEFS file locations, can be accessed from the external Spark cluster.

BYOS is tested against the version of Spark integrated into DSE (described in the [DataStax Enterprise 6.0 release notes \(page 17\)](#)) and the following Spark distributions:

- Hortonworks Data Platform (HDP) 2.5
- Cloudera CDH 5.10

### Generating the BYOS configuration file

The `byos.properties` file contains configuration settings to connect to a particular DataStax Enterprise cluster.

The `byos.properties` file is used to connect to a DataStax Enterprise cluster from a Spark cluster. The configuration file contains connection information about the DataStax Enterprise cluster. This file must be generated on a node in the DataStax Enterprise cluster. You can specify an arbitrary name for the generated configuration

file. The `byos.properties` name is used throughout the documentation to refer to this configuration file.

1. Connect to a node in your DataStax Enterprise cluster.
2. Generate the `byos.properties` file using the `dse client-tool` command.

```
dse client-tool configuration byos-export ~/byos.properties
```

This will generate the `byos.properties` file in your home directory. See [dse client-tool \(page 799\)](#) for more information on the options for `dse client-tool`.

#### What's next:

The `byos.properties` file can be copied to a node in the external Spark cluster and used with the Spark shell, as described in [Connecting to DataStax Enterprise using the Spark shell on an external Spark cluster \(page 256\)](#).

## Connecting to DataStax Enterprise using the Spark shell on an external Spark cluster

Use the Spark shell on an external Spark cluster to connect to DataStax Enterprise.

Use the [generated `byos.properties` configuration file \(page 255\)](#) and the `byos-version.jar` from a DataStax Enterprise node to connect to the DataStax Enterprise cluster from the Spark shell on an external Spark cluster.

#### Prerequisites:

You must generate the `byos.properties` on a node in your DataStax Enterprise cluster.

1. Copy the `byos.properties` file you previously generated from the DataStax Enterprise node to the local Spark node.

```
scp user@dsenode1.example.com:~/byos.properties .
```

If you are using Kerberos authentication, specify the `--generate-token` and `--token-renewer <username>` options when generating `byos.properties`, as described in [dse client-tool configuration byos-export](#).

2. Copy the `byos-version.jar` file from the `clients` directory from a node in your DataStax Enterprise cluster to the local Spark node.

The `byos-version.jar` file location depends on the type of installation.

```
scp user@dsenode1.example.com:/usr/share/dse/clients/dse-byos_2.10-5.0.1-5.0.0-all.jar byos-5.0.jar
```

3. Merge external Spark properties into `byos.properties`.

```
cat ${SPARK_HOME}/conf/spark-defaults.conf >> byos.properties
```

4. If you are using Kerberos authentication, set up a CRON job or other task scheduler to periodically call `dse client-tool cassandra renew-token <token>` where `<token>` is the encoded token string in `byos.properties`.
5. Start the Spark shell using the `byos.properties` and `byos-version.jar` file.

```
spark-shell --jars byos-5.0.jar --properties-file
byos.properties
```

## Generating Spark SQL schema files

Generate Spark SQL schema files for use with Spark SQL on external Spark clusters.

Spark SQL can import schema files generated by DataStax Enterprise.

1. Export the schema file using `dse client-tool`.

```
dse client-tool --use-server-config spark sql-schema --all >
output.sql
```

2. Copy the schema to an external Spark node.

```
scp output.sql user@sparknode1.example.com:
```

3. On a Spark node, import the schema using Spark.

```
spark-sql --jars byos-5.1.jar --properties-file byos.properties -f
output.sql
```

## Starting Spark SQL Thrift Server with Kerberos

Starting Spark SQL Thrift Server with Kerberos and BYOS.

Spark SQL Thrift Server is a long running service and must be configured to start with a keytab file if Kerberos is enabled. The user principal must be added to DSE, and Spark SQL Thrift Server restarted with the generated BYOS configuration file and `byos-version.jar`.

### Prerequisites:

These instructions are for the Spark SQL Thrift Server included in HortonWorks 2.4. The Hadoop Spark SQL Thrift Server principal is `hive/_HOST@REALM`.

1. Create the principal on the DSE node using `cqlsh`.

```
create user hive/spark_sql_thrift_server_host@REALM;
```

2. Login as the `hive` user on the Spark SQL Thrift Server host.
3. Create a `~/.java.login.config` file with a JAAS Kerberos configuration.
4. Merge the existing Spark SQL Thrift Server configuration properties with the generated BYOS configuration file into a new file.

```
cat /usr/hdp/current/spark-thriftserver/conf/spark-thrift-
sparkconf.conf byos.properties > custom-sparkconf.conf
```

5. Start Spark SQL Thrift Server with the custom configuration file and `byos-version.jar`.

```
/usr/hdp/2.4.2.0-258/spark/sbin/start-thriftserver.sh --jars
byos-version.jar --properties-file custom-sparkconf.conf
```

## 6. Connect using the Beeline client.

```
beeline -u 'jdbc:hive2://hostname:port/default;principal=hive/
_HOST@REALM'
```

### What's next:

Generated [SQL schema \(page 257\)](#) files can be passed to `beeline` with the `-f` option to generate a mapping for DSE tables so both Hadoop and DataStax Enterprise tables will be available through the service for queries.

## Using the Spark Jobserver

DSE includes Spark Jobserver, a REST interface for submitting and managing Spark jobs.

DataStax Enterprise includes a bundled copy of the open-source [Spark Jobserver](#), an optional component for submitting and managing Spark jobs, Spark contexts, and JARs on DSE Analytics clusters. Refer to the [Components \(page 17\)](#) in the release notes to find the version of the Spark Jobserver included in this version of DSE.

Valid [spark-submit options \(page 227\)](#) are supported and can be applied to the Spark Jobserver. To use the Jobserver:

- Start the job server:

```
dse spark-jobserver start [any_spark_submit_options]
```

- Stop the job server:

```
dse spark-jobserver stop
```

The default location of the Spark Jobserver depends on the type of installation:

- Package installations: `/usr/share/dse/spark/spark-jobserver`
- Tarball installations: `installation_location/resources/spark/spark-jobserver`

All the uploaded JARs, temporary files, and log files are created in the user's `$HOME/.spark-jobserver` directory, first created when starting Spark Jobserver.

Beneficial use cases for the Spark Jobserver include sharing cached data, repeated queries of cached data, and faster job starts.

### Note:

Running multiple `SparkContext` instances in a single JVM is not recommended. Therefore it is not recommended to create a new `SparkContext` for each submitted job in a single Spark Jobserver instance. We recommend one of the two following Spark Jobserver usages.

- [Persistent Context Mode](#): a single pre-created `SparkContext` shared by all jobs.
- [Context per JVM](#): each job has its own `SparkContext` in a separate JVM.

By default, the H2 database is used for storing Spark Jobserver related metadata. In this setup, using Context per JVM requires additional configuration. See the [Spark Jobserver docs](#) for details.

**Note:** In Context per JVM mode, job results must not contain instances of classes that are not present in the Spark Jobserver classpath. Problems with returning unknown (to server) types can be recognized by following log line:

```
Association with remote system [akka.tcp://
JobServer@127.0.0.1:45153]
has failed, address is now gated for [5000] ms.
Reason: [<unknown type name is placed here>]
```

Please consult Spark Jobserver docs to see configuration details.

For an example of how to create and submit an application through the Spark Jobserver, see the `spark-jobserver` demo included with DSE.

The default location of the `demos` directory depends on the type of installation:

- Package installations: `/usr/share/dse/demos`
- Tarball installations: `installation_location/demos`

## Enabling SSL communication with Jobserver

To enable SSL encryption when connecting to Jobserver, you must have a server certificate, and a truststore containing the certificate. Add the following configuration section to the `dse.conf` file in the Spark Jobserver directory.

```
spray.can.server {
 ssl-encryption = on
 keystore = "path to keystore"
 keystorePW = "keystore password"
}
```

The default location of the Spark Jobserver depends on the type of installation:

- Package installations: `/usr/share/dse/spark/spark-jobserver`
- Tarball installations: `installation_location/resources/spark/spark-jobserver`

Restart the Jobserver after saving the configuration changes.

## Spark examples

DataStax Enterprise includes Spark example applications that demonstrate different Spark features.

DataStax Enterprise includes Spark example applications that demonstrate different Spark features.

## Portfolio Manager demo using Spark

The Portfolio Manager demo runs an application that is based on a financial use case. You run scripts that create a portfolio of stocks.

The Portfolio Manager demo runs an application that is based on a financial use case. You run scripts that create a portfolio of stocks. On the OLTP (online transaction processing) side, each portfolio contains a list of stocks, the number of shares purchased, and the purchase price. The demo's pricer utility simulates real-time stock data. Each portfolio gets updated based on its overall value and the percentage of gain or loss compared to the purchase price. The utility also generates 100 days of historical market data (the end-of-day price) for each stock. On the DSE OLAP (online analytical processing) side, a Spark job calculates the greatest historical 10 day loss period for each portfolio, which is an indicator of the risk associated with a portfolio. This information is then fed back into the real-time application to allow customers to better gauge their potential losses.

To run the demo:

**Note:** DataStax Demos do not work with LDAP or internal authorization (username/password) enabled.

### 1. Install a node

**Note:** If using a tarball installation, the Portfolio Manager demo is installed as part of the normal installation. If using a package install, you must include the command for installing the demos.

- Default Interface: localhost (127.0.0.1) You must use this IP for the demo.

### 2. Start DataStax Enterprise as DSE Analytics node:

- For [package \(page 868\)](#) installations:

- a. In /etc/default/dse, set:

```
SPARK_ENABLED=1
```

- b. Start the node:

```
sudo service dse start
```

- For [tarball \(page 870\)](#) installations:

```
installation_location/bin/dse cassandra -k ## Starts node in
analytics mode
```

### 3. Go to the Portfolio Manager demo directory.

The default location of the Portfolio Manager demo depends on the type of installation:

- **Package installations:** /usr/share/dse/demos/portfolio\_manager
- **Tarball installations:** *installation\_location*/demos/portfolio\_manager

**4.** Run the bin/pricer utility to generate stock data for the application:

- To see all of the available options for this utility:

```
bin/pricer --help
```

- Start the pricer utility:

```
bin/pricer -o INSERT_PRICES &&
bin/pricer -o UPDATE_PORTFOLIOS &&
bin/pricer -o INSERT_HISTORICAL_PRICES -n 100
```

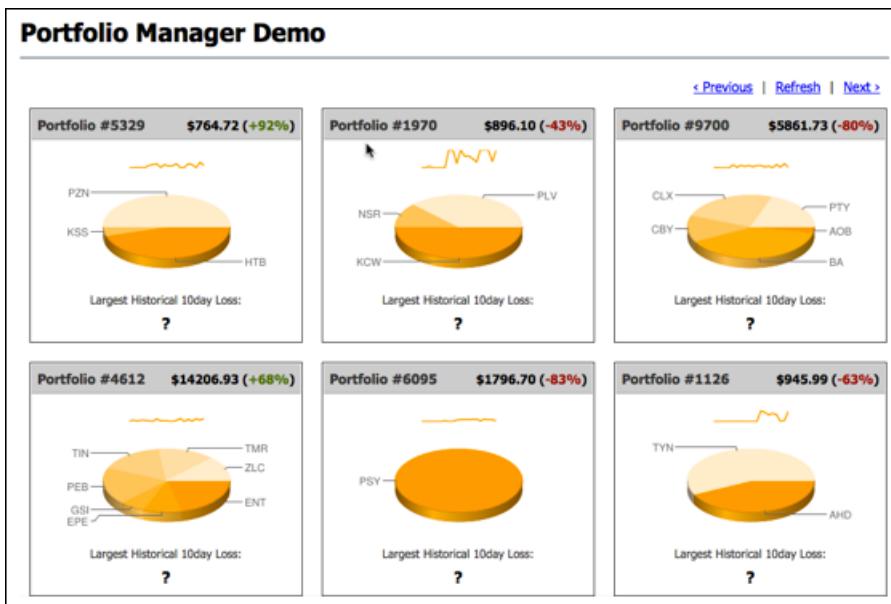
The pricer utility takes several minutes to run.

**5.** Start the web service:

```
cd website &&
sudo ./start
```

**6.** Open a browser and go to <http://localhost:8983/portfolio>.

The real-time Portfolio Manager demo application is displayed.



**7.** Open another terminal.

**8.** Run the Spark SQL job in the `10-day-loss.q` file.

```
dse spark-sql -f 10-day-loss.q
```

**9.** Run the equivalent Spark Scala job in the `10-day-loss.sh` script.

The Spark application takes several minutes to run.

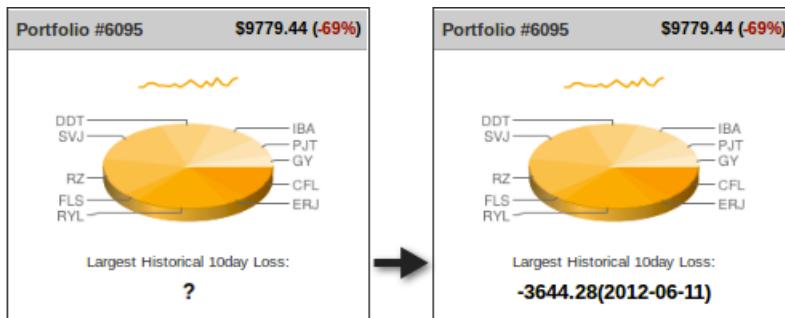
```
./10-day-loss.sh
```

10. Run the equivalent Spark Java job in the `10-day-loss-java.sh` script.

```
./10-day-loss-java.sh
```

11. After the job completes, refresh the **Portfolio Manager** web page.

The results of the Largest Historical 10 day Loss for each portfolio are displayed.



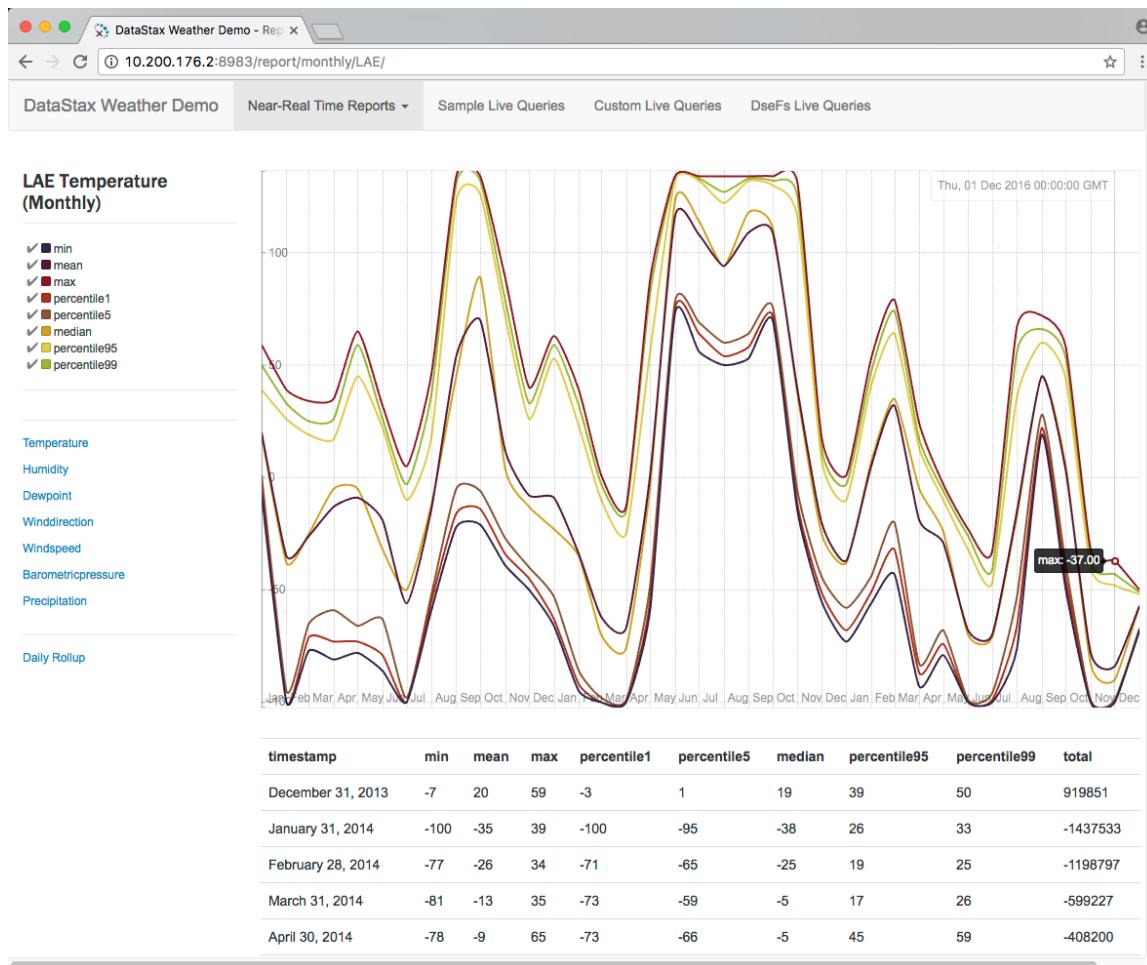
#### What's next:

The Scala and Java source code for the demo are in the `src` directory.

### Running the Weather Sensor demo

The Weather Sensor demo compares how long it takes to run Spark SQL queries against aggregated data for a number of weather sensors in various cities.

Using the Weather Sensor demo, you can compare how long it takes to run Spark SQL queries against aggregated data for a number of weather sensors in various cities. For example, you can view reports using different metrics, such as temperature or humidity, and get a daily roll up.



You run customize Spark SQL queries using different metrics and different dates. In addition to querying CQL tables, you time Spark SQL queries against data in DataStax Enterprise File System (DSEFS).

**Note:** DataStax Demos do not work with LDAP or internal authorization (username/password) enabled.

## Prerequisites

Before running the demo, install the following source code and tools if you do not already have them:

- Python 2.7:

# Debian and Ubuntu

```
sudo apt-get install python2.7-dev
```

# RedHat or CentOS

```
sudo yum install python27
```

- # Mac OS X already has Python 2.7 installed.
- pip installer tool:
  - # Debian and Ubuntu

```
sudo apt-get install python-pip
```
  - # RedHat or CentOS

```
sudo yum install python-pip
```
  - # Mac OS X

```
sudo easy_install pip
```
- The libsasl2-dev package:
  - # Debian and Ubuntu

```
sudo apt-get install libsasl2-dev
```
  - # RedHat or CentOS

```
sudo yum install cyrus-sasl-lib
```
- The required Python packages:
  - # All platforms

```
sudo pip install pyhs2 six flask cassandra-driver
```

If you installed DataStax Enterprise using a tarball, set the PATH environment variable to the DataStax Enterprise installation /bin directory.

```
export PATH=$PATH:$installation_location/bin
```

## Start DataStax Enterprise and import data

You start DataStax Enterprise in Spark mode, and then run a script that creates the schema for weather sensor data model. The script also imports aggregated data from CSV files into DSE tables. The script uses the hadoop fs command to put the CSV files into the DSEFS.

1. Start DataStax Enterprise in [Spark mode \(page 179\)](#).
2. Run the create-and-load CQL script in the `demos/weather_sensors/resources` directory. On Linux, for example:

```
cd $installation_location/demos/weather_sensors &&
bin/create-and-load
```

The default location of the `demos` directory depends on the type of installation:

- Package installations: `/usr/share/dse/demos`

- Tarball installations: `installation_location/demos`

The output confirms that the script imported the data into CQL and copied files to DSEFS.

```
.
.
10 rows imported in 0.019 seconds.
2590 rows imported in 2.211 seconds.
76790 rows imported in 33.522 seconds.
+ echo 'Copy csv files to Hadoop...'
Copy csv files to Hadoop...
+ dse hadoop fs -mkdir /datastax/demos/weather_sensors/
```

If an error occurs, set the `PATH` as described in [Prerequisites \(page 263\)](#), and retry.

## Starting the Spark SQL Thrift server

You start the Spark SQL Thrift server on a specific port to avoid conflicts. Start using your local user account. Do not use `sudo`.

1. Start the Spark SQL Thrift server on port 5588. On Linux, for example:

```
cd installation_location &&
dse spark-sql-thriftserver start --hiveconf
hive.server2.thrift.port=5588
```

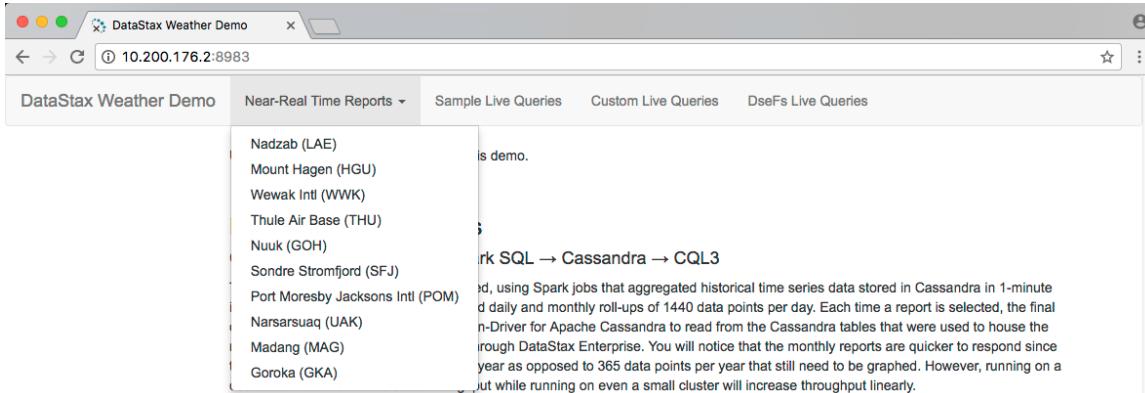
## Start the web app and query the data

1. Open another terminal and start the Python service that controls the web interface:

```
cd installation_location/demos/weather_sensors &&
python web/weather.py
```

2. Open a browser and go to the following URL: `http://localhost:8983/`

The weather sensors app appears. Select **Near Real-Time Reports** on the horizontal menu. A drop-down listing weather stations appears:



3. Select a weather station from the drop-down, view the graph, and select different metrics from the vertical menu on the left side of the page.
4. On the horizontal menu, click **Sample Live Queries**, then select a sample script. Click the **Spark SQL** button, then click Submit.

The time spent loading results using Spark appears.

The screenshot shows a web browser window titled "DataStax Weather Demo - Sam" with the URL "10.200.176.2:8983/sample\_queries/". The page has a navigation bar with tabs: "DataStax Weather Demo", "Near-Real Time Reports", "Sample Live Queries" (which is selected), "Custom Live Queries", and "DseFs Live Queries".

In the "Sample Live Queries" section, there is a "Select Sample Script" dropdown menu with several options:

- Find correlation of median temperatures between two locations on a monthly scale
- Find correlation of median temperatures between two locations on a daily scale**
- Find correlation between temperature and humidity for the same location on a monthly scale
- Find correlation between temperature and humidity for the same location on a daily scale
- Find correlation between temperature and humidity for the same location on a daily scale for "GKA"
- Find correlation of multiple metrics between two locations on a monthly scale

Below the dropdown is a "Spark SQL Query" text area containing the following code:

```
SELECT a.stationid AS station_a,
b.stationid AS station_b,
CORR(a.median, b.median) AS corr_temperature
FROM weathercql.daily a JOIN weathercql.daily b
ON (a.date = b.date) AND (a.metric = b.metric)
WHERE (a.stationid > b.stationid) AND (a.metric = 'temperature') AND (b.metric = 'temperature')
GROUP BY a.stationid, b.stationid, a.metric;
```

At the bottom of the query area is a "Submit Query" button.

Below the query area, a message states "Time spent loading: 13.9667 seconds for 45 records".

The results are displayed in a table:

0	1	2
SFJ	MAG	0.28281157041213123
SFJ	GOH	-0.06949801991458818
WWK	UAK	0.14294221419658784
LAE	GKA	-0.06787906592935311
UAK	GOH	-0.4155584922684909

5. From the horizontal menu, click Custom Live Queries. Click a Week Day, and then a metric, such as Wind Direction. Click Recalculate Query. The query reflects the selections you made.
6. From the horizontal menu, click DSEFS Live Queries. Click Submit query. The time spent loading results from DSEFS using Spark SQL appears.

The screenshot shows a browser window titled "DataStax Weather Demo - Dse" with the URL "10.200.176.2:8983/dsefs\_queries/". The tab bar includes "DataStax Weather Demo", "Near-Real Time Reports", "Sample Live Queries", "Custom Live Queries", and "DseFs Live Queries" (which is selected). The main content area contains a "Spark SQL Query" input box with the following code:

```

SELECT c.stationid AS stationid, c.month AS month, CORR(c.mediantemp, d.mediantemp) AS corr_temperature
FROM
(SELECT stationid, month(date) AS month, median AS mediantemp FROM weathercql.monthly WHERE metric = 'temperature') c
JOIN
(SELECT stationid, month(date) AS month, median AS mediantemp FROM weatherdls.monthly WHERE metric = 'temperature') d
ON (c.stationid = d.stationid) AND (c.month = d.month)
GROUP BY c.stationid, c.month
ORDER BY stationid, month;

```

Below the query box is a "Submit Query" button. A message below the button states "Time spent loading: 5.5457 seconds for 120 records". The results are displayed in a table with three columns labeled 0, 1, and 2. The data rows are as follows:

0	1	2
GKA	1	-9.069376370392622e-18
GKA	2	0
GKA	3	1.971972718068318e-17
GKA	4	2.151613645452582e-17
GKA	5	-2.3502082701455015e-17
GKA	6	-5.711514694703702e-17
GKA	7	-4.8529647397205636e-17
GKA	8	-2.1035315044944424e-17
GKA	9	-1.427598505958721e-17

## Clean up

To remove all generated data, run the following commands:

```
cd installation_location/demos/weather_sensors &&
bin/cleanup
```

To remove the keyspace from the cluster, run the following command:

```
cqlsh -e "DROP KEYSPACE weathercql;"
```

## Running the Wikipedia demo with SearchAnalytics

The Wikipedia Solr demo can be run on a SearchAnalytics node to retrieve Spark RDDs using search queries.

The following instructions describe how to use search queries in the Spark console on SearchAnalytics nodes using the Wikipedia demo.

### Prerequisites:

You must have created a new SearchAnalytics datacenter as described in the [single datacenter deployment scenario](#).

**1.** Start the node or nodes in SearchAnalytics mode.

- Package installations: See [Starting DataStax Enterprise as a service \(page 867\)](#).
- Package installations: See [Starting DataStax Enterprise as a stand-alone process \(page 870\)](#).

**2.** Ensure that the cluster is running correctly by running `dsetool ring`. The node type should be `SearchAnalytics`.

Package installations: `dsetool ring`

Tarball installations: `installation_location/bin/dsetool ring`

**3.** In a terminal, go to the Wikipedia demo directory.

The default wikipedia demo location depends on the type of installation:

- Package installations: `/usr/share/dse/demos/wikipedia`
- Tarball installations: `installation_location/demos/wikipedia`

```
cd /usr/share/dse/demos/wikipedia
```

**4.** Add the schema by running the `1-add-schema.sh` script.

```
./1-add-schema.sh
```

**5.** Create the search indexes.

```
./2-index.sh
```

**6.** Start the Spark console.

```
dse spark
```

**7.** Create an RDD based on the `wiki.solr` table.

```
scala> val table = sc.cassandraTable("wiki","solr")
```

```
table:
com.datastax.spark.connector.rdd.CassandraTableScanRDD[com.datastax.spark.connector.
= CassandraTableScanRDD[0] at RDD at CassandraRDD.scala:15
```

**8.** Run a query using the title Solr index and collect the results.

```
scala> val result =
table.select("id","title").where("solr_query='title:Boroph*'").collect
```

Equivalent JSON query:

```
where("solr_query='{"q": "title:Boroph*"}'")
```

```

result:
 Array[com.datastax.spark.connector.CassandraRow] = Array(
 CassandraRow{id: 23729958, title: Borophagus parvus},
 CassandraRow{id: 23730195, title: Borophagus dudleyi},
 CassandraRow{id: 23730528, title: Borophagus hilli},
 CassandraRow{id: 23730810, title: Borophagus
diversidens},
 CassandraRow{id: 23730974, title: Borophagus
littoralis},
 CassandraRow{id: 23731282, title: Borophagus orc},
 CassandraRow{id: 23731616, title: Borophagus pugnator},
 CassandraRow{id: 23732450, title: Borophagus secundus})

```

### What's next:

For details on using search query syntax in CQL, see [CQL queries](#).

## Running the Spark MLlib demo application

The Spark MLlib demo application demonstrates how to run machine-learning analytic jobs using Spark and DataStax Enterprise.

The Spark MLlib demo application demonstrates how to run machine-learning analytic jobs using Spark and DataStax Enterprise. The demo solves the classic iris flower classification problem, using the [iris flower data set](#). The application will use the iris flower data set to build a Naive Bayes classifier that will recognize a flower based on four feature measurements.

### Prerequisites:

We strongly recommend that you install the BLAS library on your machines before running Spark MLlib jobs. For instructions on installing the BLAS library on your platform, see <https://github.com/fommil/netlib-java/blob/master/README.md#machine-optimised-system-libraries>.

The BLAS library is not distributed with DSE due to licensing restrictions, but improves MLlib performance significantly.

You must have the Gradle build tool installed to build the demo. See <https://gradle.org/> for details on installing Gradle on your OS.

1. Start the nodes in Analytics mode.

- Package installations: See [Starting DataStax Enterprise as a service \(page 867\)](#).
- Tarball installations: See [Starting DataStax Enterprise as a stand-alone process \(page 870\)](#).

2. In a terminal, go to the `spark-mlib` directory located in the Spark demo directory.

The default location of the Spark demo depends on the type of installation:

- Package installations: `/usr/share/dse/demos/portfolio_manager`
- Tarball installations: `installation_location/demos/portfolio_manager`

3. Build the application using the `gradle` build tool.

```
gradle
```

#### 4. Use spark-submit to submit the application JAR.

The Spark MLlib demo application reads the *Spark demo directory/spark-mllib/iris.csv* file on each node. This file must be accessible in the same location on each node. If some nodes do not have the same local file path, set up a shared network location accessible to all the nodes in the cluster.

To run the application where each node has access to the same local location of *iris.csv*.

```
dse spark-submit NaiveBayesDemo.jar
```

To specify a shared location of *iris.csv*:

```
dse spark-submit NaiveBayesDemo.jar /mnt/shared/iris.csv
```

## Running the http\_receiver demo

The *http\_receiver* demo uses Spark Streaming to save data to DSE.

The *http\_receiver* demo uses Spark Streaming to save data to DSE. It is located in the *http-receivers* directory in the *demos* directory.

The default location of the *demos* directory depends on the type of installation:

- Package installations: */usr/share/dse/demos*
- Tarball installations: *installation\_location/demos*

See the *README.txt* file for instructions on running *http\_receivers*.

## Importing a text file into a table

This example shows how to use Spark to import a local or DSEFS based text file into an existing table.

This example shows how to use Spark to import a local or DSEFS based text file into an existing table. You use the *saveToCassandra* method present in the Spark RDDs to save an arbitrary RDD to the database.

#### 1. Create a keyspace and a table in the database. For example, use *cqlsh*.

```
CREATE KEYSPACE int_ks WITH replication =
 {'class': 'NetworkTopologyStrategy', 'Analytics':1};
USE int_ks;
CREATE TABLE int_compound (pkey int, ckey1 int, data1 int ,
 PRIMARY KEY (pkey,ckey1));
```

#### 2. Insert data into the table

```
INSERT INTO int_compound (pkey, ckey1, data1) VALUES (1, 2, 3);
INSERT INTO int_compound (pkey, ckey1, data1) VALUES (2, 3, 4);
INSERT INTO int_compound (pkey, ckey1, data1) VALUES (3, 4, 5);
INSERT INTO int_compound (pkey, ckey1, data1) VALUES (4, 5, 1);
```

```
INSERT INTO int_compound (pkey, ckey1, data1) VALUES (5, 1, 2);
```

3. Create a text file named `normalfill.csv` that contains this data.

```
6,7,8
7,8,6
8,6,7
```

4. Put the CSV file into DSEFS. For example:

```
dse hadoop fs -put mypath/normalfill.csv /
```

5. Start the Spark shell.
6. Verify that Spark can access the `int_ks` keyspace:

```
:showSchema int_ks
```

```
=====
Keyspace: int_ks
=====
Table: int_compound

- pkey : Int (partition key column)
- ckey1 : Int (clustering column)
- data1 : Int
```

`int_ks` appears in the list of keyspaces.

7. Read in the file, splitting it on the comma delimiter. Transform each element into an Integer.

```
val normalfill = sc.textFile("/normalfill.csv").map(line =>
 line.split(",").map(_.toInt));
```

```
normalfill: org.apache.spark.rdd.RDD[Array[Int]] = MappedRDD[2] at
map at console:22
```

Alternatively, read in the file from the local file system.

```
val file = sc.textFile("file:///local-path/normalfill.csv")
```

```
file: org.apache.spark.rdd.RDD[String] = MappedRDD[4] at textFile
at console:22
```

8. Check that Spark can find and read the CSV file.

```
normalfill.take(1);
```

```
res2: Array[Array[Int]] = Array(Array(6, 7, 8))
```

9. Save the new data to the database.

```
normalfill.map(line => (line(0), line(1),
 line(2))).saveToCassandra(
 "int_ks", "int_compound", Seq("pkey", "ckey1", "data1"))
```

The step produces no output.

**10.** Check that the data was saved using `cqlsh`.

```
SELECT * FROM int_ks.int_compound;
```

pkey	ckey1	data1
5	1	2
1	2	3
8	6	7
2	3	4
4	5	1
7	8	6
6	7	8
3	4	5

(8 rows)

## Running spark-submit job with internal authentication

Example of running a spark-submit job with internal authentication.

This example shows how to run a `spark-submit` job with internal authentication.

When you use `dse spark-submit` to submit a Spark job, the Spark Master URL and the Spark database connection URL are set automatically. Use the Spark session builder API to set the application name. For example:

```
SparkSession spark = SparkSession
 .builder()
 .appName("Datastax Java example")
 .getOrCreate();
```

1. Clone the example source files from [github](#).

```
git clone https://github.com/datastax/SparkBuildExamples.git
```

2. Select your preferred language and build system. For example for Java and Maven:

```
cd SparkBuildExamples/java/maven/dse
```

3. Build the package with Maven:

```
mvn package
```

4. Create your authentication credentials.

Authentication credentials can be provided in several ways, see [Providing credentials from DSE tools](#).

5. Use `spark-submit` to run the application. The following example assumes you've set your authentication credentials in an environment variable or config file.

```
dse spark-submit --class com.datastax.spark.example.WriteRead ./target/writeRead-0.1.jar
```

## DSEFS (DataStax Enterprise file system)

DSEFS (DataStax Enterprise file system) is the default distributed file system on DSE Analytics nodes.

DSEFS is the default distributed file system on DSE Analytics nodes.

### About DSEFS

DSEFS (DataStax Enterprise file system) is a distributed file system within DataStax Enterprise.

DSEFS (DataStax Enterprise file system) is a fault-tolerant, general-purpose, distributed file system within DataStax Enterprise. It is designed for use cases that need to leverage a distributed file system for data ingestion, data staging, and state management for Spark Streaming applications (such as checkpointing or write-ahead logging). DSEFS is similar to HDFS, but avoids the deployment complexity and single point of failure typical of HDFS. DSEFS is HDFS-compatible and is designed to work in place of HDFS in Spark and other systems.

DSEFS is the default distributed file system in DataStax Enterprise, and is automatically enabled on all analytics nodes.

DSEFS stores file metadata (such as file path, ownership, permissions) and file contents separately:

- Metadata is stored in the database.
- File data blocks are stored locally on each node and are replicated onto multiples nodes.

The redundancy factor is set at the DSEFS directory or file level, which is more granular than the replication factor that is set at the keyspace level in the database.

For performance on production clusters, store the DSEFS data on physical devices that are separate from the database. For development and testing you may store DSEFS data on the same physical device as the database.

### Deployment overview

- The DSEFS server runs in the same JVM as DataStax Enterprise. Similar to the database, there is no master node. All nodes running DSEFS are equal.
- A single DSEFS cannot span multiple datacenters. To deploy DSEFS in multiple datacenters, you can create a separate instance of DSEFS for each datacenter.

- You can use different keyspaces to [configure multiple DSEFS file systems \(page 278\)](#) in a single datacenter.
- For optimal performance, locate the local DSEFS data on a different physical drive than the database.
- Encryption is not supported. Use operating system access controls to protect the local DSEFS data directories. Other [limitations \(page 277\)](#) apply.
- DSEFS uses the [LOCAL\\_QUORUM](#) consistency level to store file metadata. DSEFS will always try to write each data block to replicated node locations, and even if a write fails, it will retry to another node before acknowledging the write. DSEFS writes are very similar to the ALL consistency level, but with additional failover to provide high-availability. DSEFS reads are similar to the ONE consistency level.

## Enabling DSEFS

Steps to enable DSEFS on DataStax Enterprise nodes.

DSEFS is automatically enabled on analytics nodes, and disabled on non-analytics nodes. You can enable the DSEFS service on any node in a DataStax Enterprise cluster. Nodes within the same datacenter with DSEFS enabled will join together to behave as a DSEFS cluster.

On each node:

1. In the dse.yaml file, set the properties for the DSE File System options:

```
dsefs_options:
 enabled:
 keyspace_name: dsefs
 work_dir: /var/lib/dsefs
 public_port: 5598
 private_port: 5599
 data_directories:
 - dir: /var/lib/dsefs/data
 storage_weight: 1.0
 min_free_space: 5368709120
```

- a. Enable DSEFS:

```
 enabled: true
```

If enabled is blank or commented out, DSEFS starts only if the node is configured to run analytics workloads.

- b. Define the keyspace for storing the DSEFS metadata:

```
 keyspace_name: dsefs
```

You can optionally [configure multiple DSEFS file systems \(page 278\)](#) in a single datacenter.

- c. Define the work directory for storing the DSEFS metadata for the local node. The work directory should not be shared with other DSEFS nodes:

```
work_dir: /var/lib/dsefs
```

- d. Define the public port on which DSEFS listens for clients:

```
public_port: 5598
```

**Note:** DataStax recommends that all nodes in the cluster have the same value. Firewalls must [open this port](#) to trusted clients. The service on this port is bound to the [native\\_transport\\_address \(page 91\)](#).

- e. Define the private port for DSEFS inter-node communication:

```
private_port: 5599
```

**Caution:** Do not open this port to firewalls; this private port must be not visible from outside of the cluster.

- f. Set the data directories where the file data blocks are stored locally on each node.

```
data_directories:
 - dir: /var/lib/dsefs/data
```

If you use the default `/var/lib/dsefs/data` data directory, verify that the directory exists and that you have root access. Otherwise, you can define your own directory location, change the ownership of the directory, or both:

```
sudo mkdir -p /var/lib/dsefs/data; sudo chown -R $USER: $GROUP /var/lib/dsefs/data
```

Ensure that the data directory is writeable by the DataStax Enterprise user. Put the data directories on different physical devices than the database. Using multiple data directories on JBOD improves performance and capacity.

- g. For each data directory, set the weighting factor to specify how much data to place in this directory, relative to other directories in the cluster. This soft constraint determines how DSEFS distributes the data. For example, a directory with a value of 3.0 receives about three times more data than a directory with a value of 1.0.

```
data_directories:
 - dir: /var/lib/dsefs/data
 storage_weight: 1.0
```

- h. For each data directory, define the reserved space, in bytes, to not use for storing file data blocks. See [min\\_free\\_space \(page 138\)](#).

```
data_directories:
 - dir: /var/lib/dsefs/data
 storage_weight: 1.0
```

```
min_free_space: 5368709120
```

2. Restart the node.
3. Repeat steps for the remaining nodes.
4. With guidance from [DataStax Support](#), you can tune advanced DSEFS properties:

```
service_startup_timeout_ms: 30000
service_close_timeout_ms: 600000
server_close_timeout_ms: 2147483647 # Integer.MAX_VALUE
compression_frame_max_size: 1048576
query_cache_size: 2048
query_cache_expire_after_ms: 2000
gossip_options:
round_delay_ms: 2000
startup_delay_ms: 5000
shutdown_delay_ms: 10000
rest_options:
request_timeout_ms: 330000
connection_open_timeout_ms: 55000
client_close_timeout_ms: 60000
server_request_timeout_ms: 300000
idle_connection_timeout_ms: 60000
internode_idle_connection_timeout_ms: 120000
core_max_concurrent_connections_per_host: 8
transaction_options:
transaction_timeout_ms: 3000
conflict_retry_delay_ms: 200
conflict_retry_count: 40
execution_retry_delay_ms: 1000
execution_retry_count: 3
block_allocator_options:
overflow_margin_mb: 1024
overflow_factor: 1.05
```

5. Continue with [using DSEFS \(page 277\)](#).

## Disabling DSEFS

Steps to disable DSEFS and remove metadata and data.

To disable DSEFS and remove metadata and data:

1. Remove all directories and files from the DSEFS file system:

```
dse fs rm -r filepath
```

2. Wait a while for all nodes to perform the delete operations.
3. Verify that all DSEFS data directories where the file data blocks are stored locally on each node are empty.

These data directories are configured in dse.yaml. Your directories are probably different from this default `data_directories` value:

```
data_directories:
```

```
- dir: /var/lib/dsefs/data
```

4. Disable the DSEFS entries in all `dse.yaml` files on all nodes.
5. Restart DataStax Enterprise.
6. Truncate all of the tables in the `dsefs` keyspace.

Do not remove the `dsefs` keyspace. If you inadvertently removed the `dsefs` keyspace, you must specify a different keyspace name in `dse.yaml` or create an empty `dsefs` keyspace (this empty `dsefs` keyspace will be populated with tables during DSEFS start up).

## Using DSEFS

Steps to use DSEFS, configure data replication, and other functions, including setting the Kafka log retention.

You must configure data replication. You can optionally [configure \(page 274\)](#) multiple DSEFS file systems in a datacenter, and perform other functions, including setting the Kafka log retention.

DSEFS does not span datacenters. Create a separate DSEFS instance in each datacenter, as described in the steps below.

### DSEFS limitations

Know these limitations when you configure and tune DSEFS. The following functionality and features are not supported:

- Encryption.  
Use operating system access controls to protect the local DSEFS data directories.
  - File system consistency checks (`fsck`) and file repair have only limited support.  
Running `fsck` will re-replicate blocks that were under-replicated because a node was taken out of a cluster.
  - File repair.
  - Forced rebalancing, although the cluster will eventually reach balance.
  - Checksum.
  - Automatic backups.
  - Multi-datacenter replication.
  - Symbolic links (soft links, symlinks) and hardlinks.
  - Snapshots.
1. Configure replication for the metadata and the data blocks.

You must set the replication factor appropriately to prevent data loss in the case of node failure. Replication factors must be set for both the metadata and the data blocks. The replication factor of 3 for data blocks is suitable for most use-cases.

- a. Globally: set replication for the metadata in the `dsefs` keyspace that is stored in the database.

For example, use a CQL statement to configure a replication factor of 3 on the Analytics datacenter using `NetworkTopologyStrategy`:

```
ALTER KEYSPACE dsefs
WITH REPLICATION = {
 'class': 'NetworkTopologyStrategy',
 'Analytics': '3'};
```

**Note:** Datacenter names are case-sensitive. Verify the case of the using utility, such as [dsetool status \(page 852\)](#).

- b.** Run `nodetool repair` on the DSEFS keyspace.

```
nodetool repair dsefs
```

- c.** Locally: set the replication factor on a specific DSEFS file or directory where the data blocks are stored.

For example, use the command line:

```
dse fs mkdir -n 4 newdirectory
```

When a replication factor (RF) is not specified, the RF is inherited from the parent directory.

2. If you have multiple Analytics datacenters, you must configure each DSEFS file system to replicate within its own datacenter:

- a.** In the `dse.yaml` file, specify a separate DSEFS keyspace for each logical datacenter.

For example, on a cluster with logical datacenters DC1 and DC2.

On each node in DC1:

```
dsefs_options:
 ...
 keyspace_name: dsefs1
```

On each node in DC2:

```
dsefs_options:
 ...
 keyspace_name: dsefs2
```

- b.** Restart the nodes.
- c.** Alter the keyspace replication to exist only on the specific datacenters.

On DC1:

```
ALTER KEYSPACE dsefs1
WITH REPLICATION = {
 'class': 'NetworkTopologyStrategy',
 'DC1': '3'};
```

On DC2:

```
ALTER KEYSPACE dsefs2
WITH REPLICATION = {
 'class': 'NetworkTopologyStrategy',
 'DC2': '3'};
```

- d. Run `nodetool repair` on the DSEFS keyspace.

```
nodetool repair dsefs
```

For example, in a cluster with multiple datacenters, the keyspace names `dsefs1` and `dsefs2` define separate file systems in each datacenter.

3. When bouncing a streaming application, verify the [Kafka log configuration](#) (especially `log.retention.check.interval.ms` and `policies.log.retention.bytes`). Ensure the Kafka log retention policy is robust enough to handle the length of time expected to bring the application and consumers back up.

For example, if the log retention policy is too conservative and deletes or rolls are logged very frequently to save disk space, the users are likely to encounter issues when attempting to recover from a checkpoint that references offsets that are no longer maintained by the Kafka logs.

## DSEFS command line tool

Options and command arguments for the DSE File System (DSEFS).

The DSEFS functionality supports operations including uploading, downloading, moving, and deleting files, creating directories, and verifying the DSEFS status.

DSEFS commands are available only in the logical datacenter. DSEFS works with secured and unsecured clusters, see [DSEFS authentication \(page 291\)](#).

You can interact with the DSEFS file system in several modes: interactive command line shell, as part of `dse` commands, or with a REST API.

### Interactive DSEFS command line shell

To use the interactive command line shell:

Action	Command line
Launch DSEFS shell	<pre>dse fs dsefs / &gt;</pre> <p>The DSEFS prompt shows the current working directory on DSEFS. The current local working directory that you launch DSEFS from is the default directory that is used for searching local files.</p>

Action	Command line
Launch DSEFS shell with precedence given to the specified hosts	<pre>dse fs --prefer-contact-points -h 10.0.0.2,10.0.0.5</pre> <p>The <code>--prefer-contact-points</code> is used in conjunction with the <code>-h</code> option to give precedence to the specified hosts, regardless of proximity, when issuing DSEFS commands. As long as the specified hosts are available, DSEFS will not switch to other DSEFS nodes in the cluster.</p> <p>Without the <code>--prefer-contact-points</code> option, DSEFS will switch to the closest available DSEFS node automatically, even if the <code>-h</code> option is used to specify contact points.</p>
View entire DSEFS command list	<pre>dsefs / &gt; help</pre>
View help for any DSEFS command	<pre>dsefs / &gt; help dsefs_command</pre>
Add a comment to a DSEFS shell command	<p>Use the <code>#</code> character. Everything after the <code>#</code> character will be ignored.</p> <pre>dsefs / &gt; get archive.tgz local_archive.tgz #retrieve the archive</pre>
Exit DSEFS shell	Press Ctrl+D or type <code>exit</code>

## Configuring DSEFS shell logging

The default location of the DSEFS shell log file `.dsefs-shell.log` is the user home directory. The default log level is INFO. To configure DSEFS shell logging, edit the `installation_location/resources/dse/conf/logback-dsefs-shell.xml` file.

## Using with the `dse` command line

Precede the DSEFS command with `dse fs`:

```
dse [dse_auth_credentials] fs dsefs_command [options]
```

For example, to list the file system status and disk space usage in human-readable format:

```
dse -u user1 -p mypassword fs df -h
```

Optional command arguments are enclosed in square brackets. For example, `[dse_auth_credentials]` and `[-R]`

Variable values are italicized. For example, `directory` and `[subcommand]`.

## Working with the local file system in the DSEFS shell

You can refer to files in the local file system by prefixing paths with `file::`. For example the following command will list files in the system root directory:

```
dsefs dsefs://127.0.0.1:5598/ > ls file:/
```

```

bin cdrom dev home lib32 lost+found mnt proc run srv tmp
var initrd.img.old vmlinuz.old
boot data etc lib lib64 media opt root sbin sys usr
initrd.img vmlinuz

```

If you need to perform many subsequent operations on the local file system, first change the current working directory to `file:` or any local file system path:

```

dsefs dsefs://127.0.0.1:5598/ > cd file:
dsefs file:/home/user1/path/to/local/files > ls
conf src target build.sbt
dsefs file:/home/user1/path/to/local/files > cd ..
dsefs file:/home/user1/path/to/local >

```

DSEFS shell remembers the last working directory of each file system separately. To go back to the previous DSEFS directory, enter:

```

dsefs file:/home/user1/path/to/local/files > cd dsefs:
dsefs dsefs://127.0.0.1:5598/ >

```

To go back again to the previous local directory:

```

dsefs dsefs://127.0.0.1:5598/ > cd file:
dsefs file:/home/user1/path/to/local/files >

```

To refer to a path relative to the last working directory of the file system, prefix a relative path with either `dsefs:` or `file:.` The following session will create a directory `new_directory` in the directory `/home/user1`:

```

dsefs dsefs://127.0.0.1:5598/ > cd file:/home/user1
dsefs file:/home/user1 > cd dsefs:
dsefs dsefs://127.0.0.1:5598/ > mkdir file:new_directory
dsefs dsefs://127.0.0.1:5598/ > realpath file:new_directory
file:/home/user1/new_directory
dsefs dsefs://127.0.0.1:5598/ > stat file:new_directory
DIRECTORY file:/home/user1/new_directory:
Owner user1
Group user1
Permission rwxr-xr-x
Created 2017-01-15 13:10:06+0200
Modified 2017-01-15 13:10:06+0200
Accessed 2017-01-15 13:10:06+0200
Size 4096

```

To copy a file between two different file systems, you can also use the `cp` command with explicit file system prefixes in the paths:

```

dsefs file:/home/user1/test > cp dsefs:archive.tgz another-archive-
copy.tgz
dsefs file:/home/user1/test > ls
another-archive-copy.tgz archive-copy.tgz archive.tgz

```

## Authentication

For `dse dse_auth_credentials` you can provide user credentials in several ways, see [Providing credentials from DSE tools](#). For authentication with DSEFS, see [DSEFS authentication \(page 291\)](#).

## Wildcard support

Some DSEFS commands support wildcard pattern expansion in the path argument. Path arguments containing wildcards are expanded before method invocation into a set of paths matching the wildcard pattern, then the given method is invoked for each expanded path.

For example in the following directory tree:

```
dirA
|--dirB
|--file1
|--file2
```

Giving the `stat dirA/*` command would be transparently translated into three invocations: `stat dirA/dirB`, `stat dirA/file1`, and `stat dirA/file2`.

DSEFS supports the following wildcard patterns:

- `*` matches any files system entry (file or directory) name, as in the example of `stat dirA/*`.
- `?` matches any single character in the file system entry name. For example `stat dirA/dir?` matches `dirA/dirB`.
- `[]` matches any characters enclosed within the brackets. For example `stat dirA/file[0123]` matches `dirA/file1` and `dirA/file2`.
- `{}` matches any sequence of characters enclosed within the brackets and separated with `..`. For example `stat dirA/{dirB,file2}` matches `dirA/dirB` and `dirA/file2`.

There are no limitations on the number of wildcard patterns in a single path.

For authentication with DSEFS, see [DSEFS authentication \(page 291\)](#).

## Executing multiple commands

DSEFS can execute multiple commands on one line. Use quotes around the commands and arguments. Each command will be executed separately by DSEFS.

```
dse fs 'cat file1 file2 file3 file4' 'ls dir1'
```

## Forcing synchronization

Before confirming writing a file, DSEFS by default forces all blocks of the file to be written to the storage devices. This behavior can be controlled with `--no-force-sync` and `--force-fsync` flags when creating files or directories in the DSEFS shell with `mkdir`, `put`, and `cp` commands. The force/no-force behavior is inherited from the parent directory, if not specified. For example, if a directory is created with `--no-force-sync`, then all files are created with `--no-force-sync` unless `--force-fsync` is explicitly set during file creation.

Turning off forced synchronization improves latency and performance at a cost of durability. For example, if a power loss occurs before writing the data to the storage device, you may

lose data. Turn off forced synchronization only if you have a reliable backup power supply in your datacenter and failure of all replicas is unlikely, or if you can afford losing file data.

The Hadoop `SYNC_BLOCK` flag has the same effect as `--force-sync` in DSEFS. The Hadoop `LAZY_PERSIST` flag has the same effect as `--no-force-sync` in DSEFS.

## DSEFS command options

The following DSEFS commands and arguments are supported:

DSEFS command	Description and command arguments
<code>append source destination</code>	Append a local file to a remote file. <ul style="list-style-type: none"> <li><code>source</code> is the path to the local file to read data from.</li> <li><code>destination</code> is the path to the remote file to append the file to.</li> </ul>
<code>cat file_or_files</code>	Concatenate files and print on the standard output. <ul style="list-style-type: none"> <li><code>file_or_files</code> is the file or files in DSEFS to print to standard output. Separate files with a space. The path may contain wildcards.</li> </ul>
<code>cd directory</code>	Change the remote working directory in DSEFS. <ul style="list-style-type: none"> <li><code>directory</code> is the remote directory to change to. The path may contain wildcards.</li> <li><code>..</code> is the parent directory.</li> </ul> The DSEFS prompt identifies the current working directory in DSEFS: <ul style="list-style-type: none"> <li><code>dsefs / &gt;</code> is the default directory</li> <li><code>dsefs /dir2 &gt;</code> is the current working directory <code>dir2</code></li> </ul>
<code>chgrp group path</code>	Change file or directory group ownership. <ul style="list-style-type: none"> <li><code>group</code> the new group name.</li> <li><code>path</code> the file or directory whose group will be changed. The path may contain wildcards.</li> </ul>
<code>chmod octal_permission mode path</code>	Change the permissions of a file or directory. <ul style="list-style-type: none"> <li><code>octal permission mode</code> octal representation of permission mode for owner, group, and others.</li> <li><code>path</code> the file or directory whose permissions will be changed. The path may contain wildcards.</li> </ul>
<code>chown [options] path</code>	Change files or directories ownership and/or group ownership. <ul style="list-style-type: none"> <li><code>-u, --user username</code> the new owner username.</li> <li><code>-g, --group group</code> the new group owner name.</li> <li><code>path</code> the file or directory whose ownership will be changed. The path may contain wildcards.</li> </ul>

DSEFS command	Description and command arguments
<code>cp [options] source destination</code>	<p>Copies a file within a file system or between two file systems. If the destination path points to a different file system than DSEFS, the block size and redundancy options are ignored.</p> <ul style="list-style-type: none"> <li>• <code>-o, --overwrite</code> overwrite the destination file if it exists.</li> <li>• <code>-b, --block-size value</code> The preferred block size in bytes.</li> <li>• <code>--force-fsync</code> forces synchronization of the file data to the storage devices, regardless of the value of the <code>--force-sync</code> flag of its parent directory. The value of the <code>--force-sync</code> flag of a file or directory can be displayed with <code>stat</code>. This option applies to a directory's files, not the directory itself, similar to behavior of block size or compression options. It can be applied during directory creation so that files in the directory will inherit the option.</li> <li>• <code>--no-force-fsync</code> causes the blocks of data to be written only to the buffers of the operating system, and syncing them to the storage device is then left to the operating system. Using <code>--no-force-fsync</code> improves latency and performance at the cost of durability. If a power loss occurs before writing the data to the storage device, you may lose data. Use <code>--no-force-fsync</code> only if you have a reliable backup power supply in your datacenter and failure of all replicas is unlikely, or if you can afford losing file data.</li> </ul> <p>Using the <code>--no-force-fsync</code> flag on a directory will by default make any new files created in this directory as if the <code>--no-force-fsync</code> flag was given when creating the files.</p> <ul style="list-style-type: none"> <li>• <code>-n, --redundancy-factor num_nodes</code> is how many replicas of the file data to create in DSEFS. This redundancy factor is similar to the replication factor in the database keyspaces, but is more granular. Set this value to one number greater than the number of nodes that are allowed to fail before data loss occurs. For example, set this value to 3 to allow 2 nodes to fail. For simple replication, you can use a value that is equivalent to the replication factor.</li> <li>• <code>source</code> the source file to be copied. The path may contain wildcards.</li> <li>• <code>destination</code> the destination file to be created.</li> </ul>
<code>df [options]</code>	<p>List the DSEFS file system status and disk space usage.</p> <ul style="list-style-type: none"> <li>• <code>-h</code> to list the sizes in human-readable format. Sizes are rounded to three significant places and presented using units:           <ul style="list-style-type: none"> <li># K (for a kilobyte = 1024 bytes),</li> <li># M (for a megabyte = 1024K),</li> <li># G (for a gigabyte = 1024M),</li> <li># T (for a terabyte = 1024G)</li> </ul> </li> </ul> <p>Without this option sizes are printed in bytes.</p>

DSEFS command	Description and command arguments
echo	<p>Outputs a line of text to display the quoted strings it is being passed as a single argument:</p> <pre>echo "some text"</pre> <p>Output a line of text to display the strings it is being passed as multiple arguments:</p> <pre>echo Getting a file</pre>
exit	Exit the DSEFS shell client. You can also type Ctrl+D to exit the shell.
fsck	Perform a file system consistency check and repair file system errors.
get <i>source destination</i>	<p>Get a file from the DSEFS remote file system and copy the file to the local file system.</p> <ul style="list-style-type: none"> <li><i>source</i> is the path to the DSEFS remote file to copy.</li> <li><i>destination</i> is the path to the local file to create.</li> </ul>
ls [ <i>options</i> ] [ <i>file_system_entry_or_entries</i> ]	<p>List the DSEFS file system entries (files or directories) in the current working directory.</p> <ul style="list-style-type: none"> <li>-R to list subdirectories recursively.</li> <li>-l to use a long listing format with metadata.</li> <li>-h to list the sizes in human-readable format. Sizes are rounded to three significant places and presented using units:           <ul style="list-style-type: none"> <li># K (for a kilobyte = 1024 bytes),</li> <li># M (for a megabyte = 1024K),</li> <li># G (for a gigabyte = 1024M),</li> <li># T (for a terabyte = 1024G)</li> </ul> </li> </ul> <p>Without this option sizes are printed in bytes.</p> <ul style="list-style-type: none"> <li>-1 limits the number of printed columns to one, so one file is printed per line. This allows the output to more easily be parsed by external tools.</li> <li><i>file_system_entry_or_entries</i> is the directory or directories to list the contents of. The path may contain wildcards.</li> </ul>

DSEFS command	Description and command arguments
<code>mkdir [options] dir_or_dirs</code>	<p>Make a new directory or directories.</p> <ul style="list-style-type: none"> <li>• <code>-p</code> to make parent directories as needed.</li> <li>• <code>-b bytes</code> is the preferred block size for files stored in this directory.</li> <li>• <code>--force-fsync</code> forces synchronization of the file data to the storage devices, regardless of the value of the <code>--force-sync</code> flag of its parent directory. The value of the <code>--force-sync</code> flag of a file or directory can be displayed with <code>stat</code>. This option applies to a directory's files, not the directory itself, similar to behavior of block size or compression options. It can be applied during directory creation so that files in the directory will inherit the option.</li> <li>• <code>--no-force-fsync</code> causes the blocks of data to be written only to the buffers of the operating system, and syncing them to the storage device is then left to the operating system. Using <code>--no-force-fsync</code> improves latency and performance at the cost of durability. If a power loss occurs before writing the data to the storage device, you may lose data. Use <code>--no-force-fsync</code> only if you have a reliable backup power supply in your datacenter and failure of all replicas is unlikely, or if you can afford losing file data.</li> </ul> <p>Using the <code>--no-force-fsync</code> flag on a directory will by default make any new files created in this directory as if the <code>--no-force-fsync</code> flag was given when creating the files.</p> <ul style="list-style-type: none"> <li>• <code>-c, --compression-encoder value</code> the encoder name to use for compression. DSE ships with the <code>lz4</code> compression encoder.</li> <li>• <code>-n, --redundancy-factor num_nodes</code> is how many replicas of the file data to create in DSEFS. This redundancy factor is similar to the replication factor in the database keyspaces, but is more granular. Set this value to one number greater than the number of nodes that are allowed to fail before data loss occurs. For example, set this value to 3 to allow 2 nodes to fail. For simple replication, you can use a value that is equivalent to the replication factor.</li> <li>• <code>-m, --permission-mode value</code> octal representation of permission mode for owner, group and others.</li> <li>• <code>dir_or_dirs</code> is the directory or directories to create.</li> </ul>
<code>mv source destination</code>	<p>Move or rename a file or directory.</p> <ul style="list-style-type: none"> <li>• <code>source</code> is the path to the DSEFS file system entry to be moved. The path may contain wildcards.</li> <li>• The destination path on DSEFS: <ul style="list-style-type: none"> <li># <code>destination</code> is the full destination path, including the name of the file or directory being moved.</li> <li># <code>destination/</code> is the full destination path. If the destination ends with a slash (/) the original file or directory name will be retained.</li> </ul> </li> </ul>

DSEFS command	Description and command arguments
<code>put [options] source destination</code>	<p>Copy a local file to the DSEFS.</p> <ul style="list-style-type: none"> <li>• <code>-o, --overwrite</code> to overwrite the destination file if it exists.</li> <li>• <code>-b, --block-size bytes</code> is the preferred block size in bytes.</li> <li>• <code>--force-fsync</code> forces synchronization of the file data to the storage devices, regardless of the value of the <code>--force-sync</code> flag of its parent directory. The value of the <code>--force-sync</code> flag of a file or directory can be displayed with <code>stat</code>. This option applies to a directory's files, not the directory itself, similar to behavior of block size or compression options. It can be applied during directory creation so that files in the directory will inherit the option.</li> <li>• <code>--no-force-fsync</code> causes the blocks of data to be written only to the buffers of the operating system, and syncing them to the storage device is then left to the operating system. Using <code>--no-force-fsync</code> improves latency and performance at the cost of durability. If a power loss occurs before writing the data to the storage device, you may lose data. Use <code>--no-force-fsync</code> only if you have a reliable backup power supply in your datacenter and failure of all replicas is unlikely, or if you can afford losing file data.</li> </ul> <p>Using the <code>--no-force-fsync</code> flag on a directory will by default make any new files created in this directory as if the <code>--no-force-fsync</code> flag was given when creating the files.</p> <ul style="list-style-type: none"> <li>• <code>-c, --compression-encoder value</code> the encoder name to use for compression. DSE ships with the <code>lz4</code> compression encoder.</li> <li>• <code>-n, --redundancy-factor num_nodes</code> is how many replicas of the file data to create in DSEFS. This redundancy factor is similar to the replication factor in the database keyspaces, but is more granular. Set this value to one number greater than the number of nodes that are allowed to fail before data loss occurs. For example, set this value to 3 to allow 2 nodes to fail. For simple replication, you can use a value that is equivalent to the replication factor.</li> <li>• <code>-f, --compression-frame-size value</code> the preferred frame size in bytes. Frame is a subject of compression. The bigger the frame the bigger the chance for high compression ratio. For most cases the default value of 131072 bytes is sufficient.</li> <li>• <code>-m, --permission-mode value</code> octal representation of permission mode for owner, group and others.</li> <li>• <code>source</code> is the path to the local source file.</li> <li>• <code>destination</code> is the path to the destination file to be created on DSEFS.</li> </ul>
<code>pwd [path]</code>	<p>Print the working directory of the current file system or specified path.</p> <ul style="list-style-type: none"> <li>• <code>path</code> the current working directory of the file system at the root of the path.</li> </ul>

DSEFS command	Description and command arguments
<code>realpath [options] path</code>	<p>Print the resolved absolute path for a specified file or directory.</p> <ul style="list-style-type: none"> <li>• <code>-e, --canonicalize-existing</code> all components of the path must exist.</li> <li>• <code>-m, --canonicalize-missing</code> no path components need to exist or be a directory.</li> <li>• <code>path</code> the path to resolve.</li> </ul>
<code>rename path name</code>	<p>Rename a file or directory in the current location.</p> <ul style="list-style-type: none"> <li>• <code>path</code> is the path to the file system entry to be renamed.</li> <li>• <code>name</code> is the new name of the file system entry.</li> </ul>
<code>rm [-r] path</code>	<p>Remove files or directories.</p> <ul style="list-style-type: none"> <li>• <code>-r</code> specifies to recursively remove the files or directories.</li> <li>• <code>-v</code> explain what is being done.</li> <li>• <code>path</code> is the path to the file system entry to be removed. The path may contain wildcards.</li> </ul>
<code>rmdir path</code>	<p>Remove an empty directory or directories.</p> <ul style="list-style-type: none"> <li>• <code>path</code> is the path to the directory to be removed.</li> </ul>
<code>stat file_or_dir [-v]</code>	<p>Display the file system entry status.</p> <ul style="list-style-type: none"> <li>• <code>file_or_dir</code> is the file system. The path may contain wildcards.</li> <li>• <code>-v</code> to print verbose detailed information about the file status.</li> </ul>
<code>truncate file</code>	<p>Truncate a file to 0 bytes. Useful for retaining the metadata for the file.</p> <ul style="list-style-type: none"> <li>• <code>file</code> is the file to truncate.</li> </ul>
<code>umount [-f] locations</code>	<p>Unmount file system storage locations.</p> <ul style="list-style-type: none"> <li>• <code>-f</code> to force unmounting, even if the location is unavailable.</li> <li>• <code>locations</code> is the UUID (Universal Unique Identifier) of UUIDs of the locations to unmount. Get the UUID from the <code>df</code> command.</li> </ul>

## Removing a DSEFS node

When removing a node running DSEFS from a DSE cluster, additional steps are needed to ensure proper correctness within the DSEFS data set.

1. From a node in the same datacenter as the node to be removed, start the DSEFS shell.

```
dse fs
```

2. Show the current DSEFS nodes with the `df` command.

```
dsefs > df
```

Location		Status	DC	Rack
Host	Address	Port	Directory	Used
Free	Reserved			
144e587c-11b1-4d74-80f7-dc5e0c744aca		up	GraphAnalytics	rack1
node1.example.com	10.200.179.38	5598	/var/lib/dsefs/data	0
29289783296	5368709120			
98ca0435-fb36-4344-b5b1-8d776d35c7d6		up	GraphAnalytics	rack1
node2.example.com	10.200.179.39	5598	/var/lib/dsefs/data	0
29302099968	5368709120			

3. Find the node to be removed in the list and note the UUID value for it under the Location column.
4. If the node is up, unmount it from DSEFS with the command `umount UUID`.
5. If the node is not up (for example, after a hardware failure), force unmount it from DSEFS with the command `umount -f UUID`.
6. Continue with the normal steps for [removing a node](#).

## Examples

Using the DSEFS shell, these commands put the local `bluefile` to the remote DSEFS `greenfile`:

```
dsefs / > ls -l
dsefs / > put file:/bluefile greenfile
```

To view the new file in the DSEFS directory:

```
dsefs / > ls -l
Type Permission Owner Group Length Modified Name
file rwxrwxrwx none none 17 2016-05-11 09:34:26+0000
greenfile
```

Using the `dse` command, these commands create the `test2` directory and upload the local `README.md` file to the new DSEFS directory.

```
dse fs "mkdir /test2" &&
dse fs "put README.md /test2/README.md"
```

To view the new directory listing:

```
dse fs "ls -l /test2"
Type Permission Owner Group Length Modified Name
file rwxrwxrwx none none 3382 2016-03-07 23:20:34+0000 README.md
```

You can use two or more `dse` commands in a single command line. This is faster because the JVM is launched and connected/disconnected with DSEFS only once. For example:

```
dse fs "mkdir / test2" "put README.md /test/README.md"
```

The following example shows how to use the `--no-force-sync` flag on a directory, and how to check the state of the `--force-sync` flag using `state`. These commands are run from within the DSEFS shell.

```
dsefs> mkdir --no-force-sync /tmp
dsefs> put file:some-file.dat /tmp/file.tmp
dsefs> stat /tmp/file.tmp
FILE dsefs://127.0.0.1:5598/tmp/file.tmp
Owner none
Group none
Permission rwxrwxrwx
Created 2017-03-06 17:54:35+0100
Modified 2017-03-06 17:54:35+0100
Accessed 2017-03-06 17:54:35+0100
Size 1674
Block size 67108864
Redundancy 3
Compressed false
Encrypted false
Forces sync false
Comment
```

## DSEFS compression

DSEFS is able to compress files to save storage space and bandwidth.

DSEFS is able to compress files to save storage space and bandwidth. Compression is performed by DSE during upload upon a user's explicit request. Decompression is transparent. Data is always uncompressed by the server before it is returned to the client.

Compression is performed within block boundaries. The unit of compression—the chunk of data that gets compressed individually—is called a frame and its size can be specified during file upload.

### Encoders

DSEFS is shipped with the lz4 encoder which works out of the box.

### Compression

To compress files use the `-c` or `--compression-encoder` parameter for `put` or `cp` ([page 279](#)) command. The parameter specifies the compression encoder to use for the file that is about to get uploaded.

```
dsefs / > put -c lz4 file /path/to/file
```

The frame size can optionally be set with the `-f`, `--compression-frame-size` option.

The maximum frame size in bytes is set in the `compression_frame_max_size` option in `dse.yaml`. If a user sets the frame size to a value greater than `compression_frame_max_size` when using `put -f` an error will be thrown and the command will fail. Modify the `compression_frame_max_size` setting based on the available memory of the node.

Files that are compressed can be appended in the same way as uncompressed files. If the file is compressed the appended data gets transparently compressed with the file's encoder specified for the initial `put` operation.

Directories can have a default compression encoder specified during directory creation with the `mkdir` ([page 283](#)) command. Newly added files with the `put` command inherit the default compression encoder from containing directory. You can override the default compression encoder with the `c` parameter during `put` operations.

```
dsefs / > mkdir -c lz4 /some/path
```

## Decompression

Decompression is performed automatically for all commands that transport data to the client. There is no need for additional configuration to retrieve the original, decompressed file content.

## Storage space

Enabling compression creates a distinction between the logical and physical file size.

The logical size is the size of a file before uploading it to DSEFS, where it is then compressed. The logical size is shown by the `stat` ([page 279](#)) command under Size.

```
dsefs dsefs://10.0.0.1:5598/ > stat /tmp/wikipedia-sample.bz2
FILE dsefs://10.0.0.1:5598/tmp/wikipedia-sample.bz2:
Owner none
Group none
Permission rwxrwxrwx
Created 2017-04-06 20:06:21+0000
Modified 2017-04-06 20:06:21+0000
Accessed 2017-04-06 20:06:21+0000
Size 7723180
Block size 67108864
Redundancy 3
Compressed true
Encrypted false
Comment
```

The physical size is the actual size of a data stored on the storage device. The physical size is shown by the `df` ([page 279](#)) command and by the `stat -v` command for each block separately, under the Compressed length column.

## Limitations

Truncating compressed files is not possible.

## DSEFS authentication

DSEFS works with secured DataStax Enterprise clusters.

DSEFS works with secured DataStax Enterprise clusters.

## DSEFS authentication with secured clusters

Authentication is required only when it is enabled in the cluster. DSEFS on secured clusters requires the `DseAuthenticator`, see [Configuring DSE Unified Authentication](#). Authentication is off by default.

DSEFS supports authentication using DSE Unified Authentication, and supports all authentication schemes supported by DSE Authenticator, including Kerberos.

DSEFS authentication can secure client to server communication.

## Spark applications

For Spark applications, provide authentication credentials in one of these ways:

- Set with the `dse spark-submit` command using one of the credential options described in [Command line](#).
- Programmatically set the user credentials in the Spark configuration object before the `SparkContext` is created:

```
conf.set("spark.hadoop.com.datastax.bdp.fs.client.authentication.basic.username",
 <user>)
conf.set("spark.hadoop.com.datastax.bdp.fs.client.authentication.basic.password",
 <pass>)
```

If a Kerberos authentication token is in use, you do not need to set any properties in the context object. If you need to explicitly set the token, set the `spark.hadoop.cassandra.auth.token` property.

- When running the Spark Shell, where the `SparkContext` is created at startup, set the properties in the Hadoop configuration object:

```
sc.hadoopConfiguration.set("com.datastax.bdp.fs.client.authentication.basic.username",
 <user>)
sc.hadoopConfiguration.set("com.datastax.bdp.fs.client.authentication.basic.password",
 <pass>)
```

Note the absence of the `spark.hadoop` prefix.

- When running a Spark application or the Spark Shell, provide properties in the `spark-defaults.conf` configuration file:

```
<property>
 <name>com.datastax.bdp.fs.client.authentication.basic.username</name>
 <value>username</value>
</property>
<property>
 <name>com.datastax.bdp.fs.client.authentication.basic.password</name>
 <value>password</value>
</property>
```

Optional: If you want to use this method, but do not have privileges to write to `core-default.xml`, copy this file to any location `path` and set the environment variable to point to the file with:

```
export HADOOP2_CONF_DIR=path
```

## DSEFS shell

Providing authentication credentials while using the DSEFS shell is as easy as in other DSE tools. The DSEFS shell supports different authentication methods listed below in priority order. When more than one method can be used, the one with higher priority is chosen. For example when the `DSE_TOKEN` environment variable is set and the DSEFS shell is started with a username and password set as environment variables in the `$HOME/.dserc` file the provided username and password is used for authentication as it has higher priority.

1. Specifying a username and password [on the command line](#), in the [~/.dserc file](#), or [in environment variables](#).

```
export DSE_USERNAME=username &&
export DSE_PASSWORD=password
```

```
dse fs 'mkdir /dir1'
```

2. Using a [Kerberos delegation token](#). See [dse client-tool cassandra](#) for further information.

```
export DSE_TOKEN=`dse -u token_user -p password client-tool cassandra
generate-token`
```

```
dse fs 'mkdir /dir1'
```

3. Using a [cached Kerberos ticket](#) after authenticating using a tool like `kinit`.

```
kinit username
```

```
dse fs 'mkdir /dir1'
```

4. Using a [Kerberos keytab file](#) and a login configuration file.

If the configuration file is in a non-default location, specify the location using the `java.security.auth.login.config` property in the `DSEFS_SHELL_OPTS` variable:

```
DSEFS_SHELL_OPTS="-Djava.security.auth.login.config=path to login
config file" dse fs
```

## DSEFS authorization

DSEFS authorization verifies user and group permissions on files and directories stored in DSEFS.

DSEFS authorization verifies user and group permissions on files and directories stored in DSEFS.

DSEFS authorization is disabled by default. It requires no configuration, it is automatically enabled along with [DSE authorization \(page 108\)](#).

## Owners, groups, and permissions

In unsecured clusters with DSEFS authentication disabled all newly created files and directories are created with the owner set to `none`, group set to `none`. In unsecured clusters every DSEFS user has full access to every file and directory.

```
dsefs dsefs://127.0.0.1:5598/ > ls -l
Type Permission Owner Group Length Modified
 Name
dir rwxrwxrwx none none - 2016-12-01
15:50:49+0100 some_dir
```

In secured clusters with DSEFS authentication enabled all newly created files and directories are created with owner set the authenticated user's username and group set to authenticated user primary role. See [the CQL roles documentation](#) for detailed information on user roles. File and directory permissions can be specified during creation as a parameter for the `put` and `mkdir` commands. Please use `help put` or `help mkdir` for details.

```
dsefs dsefs://127.0.0.1:5598/ > ls -l
Type Permission Owner Group Length Modified
 Name
dir rwxr-x--- john admin - 2016-12-02
15:52:54+0100 other_dir
```

To change the owner or group of an existing file or directory use `chown` or `chgrp` commands. Please use `help chown` or `help chgrp` for details.

DSEFS by default creates directories with `rwxr-xr-x` (octal 755) permissions and files with `rw-r-r-` (octal 644). To change the permissions of an existing file or directory use the `chmod` command. Please use `help chmod` for details.

## DSEFS superusers

A DSEFS user is a superuser if and only if the user is a database superuser. Superusers are allowed to read and write every file and directory stored in DSEFS. Only superusers are allowed to execute DSEFS maintenance operations like `fsck` and `umount`.

## DSEFS users

User access is verified against:

- Owner permissions if the file or directory owner name is equal to the authenticated user's username.
- Group permissions if the file or directory group belongs to the authenticated user's groups. Groups are mapped from the database's user role names.
- Other permissions if the above conditions are false.

Each [DSEFS command \(page 279\)](#) requires its own set of permissions. For a given path `a/b/c`, `c` is a leaf and `a/b` is a parent path. The following table shows what permissions must be present for the given operation to succeed. R indicates read, W indicates write, and X indicates execute privileges.

**Table 7: Affect of permissions on files by DSEFS command**

Command	Path checked for permissions	Parent path permissions	Leaf permissions
append a/b/c	a/b/c	X	W
cat a/b/c	a/b/c	X	R
cd a/b/c	a/b/c	X	
chgrp	same as in chown for group		
chmod a/b/c	a/b/c	X	The user must be the owner.
chown a/b/c	a/b/c	X	Only superusers can change the owner. To change the group the user needs to be a member of the target group or be a superuser.
cp	same as in get and than put		
expand a/?/c	a/?/c	X	X
get a/b/c	a/b/c	X	R
ls a/b/c	a/b/c	X	RX if c is a directory.
mkdir a/b/c	a/b	X	WX
mv a/b/c d/e/f	a/b and d/e	X	WX
put a/b/c	a/b	X	WX
realpath a/b/c	a/b/c	X	
rename a/b/c d	a/b	X	WX
rm a/b/c	a/b	X	WX
rmdir a/b/c	a/b	X	WX

Command	Path checked for permissions	Parent path permissions	Leaf permissions
stat a/b/c	a/b/c	X	
truncate a/b/c	a/b/c	X	W

## Authorization transitional mode

DSEFS authorization supports [transitional mode \(page 109\)](#) provided by `DSEAuthorizer`. Legacy authorizers, like `TransitionalAuthorizer`, are not supported. DSE will not start if unsupported authorizer is configured and error is reported in log messages.

## Enabling SSL encryption for DSEFS

DSEFS can use SSL encryption.

There are two parts to enabling SSL encryption for DSEFS: node-to-node encryption, and client-to-node encryption. Enabling [node-to-node encryption in DSE](#) automatically enables encrypted communication between DSEFS nodes. DSE nodes with [client-to-node encryption enabled](#) allow SSL connections from the DSEFS shell.

## Configuring the DSEFS shell to use SSL encryption

In most cases, you don't need to add any DSEFS shell settings to connect using SSL. If a `~/.dse/dsefs-shell.yaml` configuration file cannot be found, DSEFS shell will attempt to load server-side configuration and SSL settings from DSE configuration files.

To manually configure SSL, create and edit the DSEFS shell configuration file. The DSEFS shell is configured in the `~/.dse/dsefs-shell.yaml` configuration file. Add the following settings to enable SSL encryption:

```
encryption_options:
 # set to true to enable secure client-server connection
 enabled: true
 # if optional is true, and enabled is true, ssl will be used if
 # possible,
 # but will failover to non ssl
 optional: true
 # path to truststore
 truststore_path: path
 # optional truststore type; default value: JKS
 truststore_type:
 # optional, will be prompted for if doesn't exist
 truststore_password:
 # path to keystore
 keystore_path: path
 # optional truststore type; default value: JKS
 keystore_type:
 # optional, will be prompted for if doesn't exist
 keystore_password:
 # optional protocol name; default value: TLSv1.2
 protocol:
```

```

optional keymanager and trustmanager algorithm; default value:
SunX509
algorithm:
optional list of ciphers
cipher_suites:
set to true to enable checking if the certificate matches endpoint
address
require_endpoint_verification: false

```

The same settings can be given as `dse fs` command-line options, except `keystore_password`, `truststore_password`, and `cipher_suites`. If passwords are not given in the configuration file, they will be prompted for at the DSEFS shell startup. The command line options override settings read from the configuration file.

**Note:** If a non-optional secure connection is established, a `[secure]` flag will appear in the prompt of the DSEFS shell.

## Using the DSEFS REST interface

DSEFS provides a REST interface that implements all the commands from WebHDFS.

DSEFS provides a REST interface that implements the [commands from WebHDFS](#).

The REST interface is enabled on all DSE nodes running DSEFS. It is available at the following base URI: `http://node hostname or IP address:5598/webhdfs/v1`

For example from a terminal using the `curl` command:

```

curl -L -X PUT 'localhost:5598/webhdfs/v1/fs/a/b/c/d/e?op=MKDIRS' &&
curl -L -X PUT -T logfile.txt '127.0.0.1:5598/webhdfs/v1/fs/log?
op=CREATE&overwrite=true&blocksize=50000&rf=1' &&
curl -L -X POST logfile.txt 'localhost:5598/webhdfs/v1/fs/log?op=APPEND'

```

Or from the DSE Spark shell:

```
val rdd1 = sc.textFile("webhdfs://localhost:5598/webhdfs/v1/fs/log")
```

## Programmatic access to DSEFS

DSEFS can be accessed programmatically from an application by obtaining DSEFS's implementation of Hadoop's `FileSystem` interface.

DSEFS can be accessed programmatically from an application by obtaining DSEFS's implementation of Hadoop's `FileSystem` interface.

DSE includes a demo project with simple applications that demonstrate how to acquire, configure, and use this implementation. The demo project demonstrates reading, writing and connecting to a secured DSEFS using the API. The demo is located in the `dsefs` directory under the `demos` directory.

The default location of the `demos` directory depends on the type of installation:

- Package installations: `/usr/share/dse/demos`
- Tarball installations: `installation_location/demos`

The README.md has instructions on building and running the demo applications.

## Hadoop FileSystem interface implemented by DseFileSystem

DseFileSystem has partial support of the Hadoop FileSystem interface.

The DseFileSystem class has partial support of the Hadoop FileSystem interface. The following table outlines which methods have been implemented.

**Table 8: Methods of Hadoop FileSystem interface implemented by DseFileSystem**

Method	Status	Comment
getScheme()	#	since 5.0.12, 5.1.6
getURI()	#	
getName()	#	default, deprecated
getDefaultPort()	#	since 5.0.12, 5.1.6
makeQualified(Path)	#	default
getDelegationToken(String)	#	returns null
addDelegationTokens(String, Credentials)	#	
collectDelegationTokens(...)	#	
getChildFileSystems()	#	default, returns null
getFileBlockLocations(FileStatus, long, long)	#	
getFileBlockLocations(Path, long, long)	#	
getServerDefaults()	#	default, deprecated
getServerDefaults(Path)	#	default
resolvePath(Path)	#	default
open	#	all variants, buffer size not supported
create	#	all variants, checksum options, progress reporting and APPEND, NEW_BLOCK flags not supported
createNonRecursive	#	all variants

Method	Status	Comment
createNewFile	#	default
append	#	all variants, progress reporting not supported
concat	#	since 5.0.12, 5.1.6
getReplication(Path)	#	
setReplication(Path, short)	#	does nothing
rename	#	
truncate(Path, long)	#	since 5.0.12, 5.1.6
delete(Path)	#	
delete(Path, boolean)	#	
deleteOnExit(Path)	#	default
cancelDeleteOnExit(Path)	#	default
exists(Path)	#	
isDirectory(Path)	#	
isFile(Path)	#	
getLength(Path)	#	
getContentSummary(Path)	#	default
listStatus	#	all variants
listCorruptFileBlocks(Path)	#	throws UnsupportedOperationException
globStatus	#	default
listLocatedStatus	#	default
listStatusIterator	#	default
listFiles	#	default
getHomeDirectory()	#	default

Method	Status	Comment
getWorkingDirectory()	#	
setWorkingDirectory()	#	
getInitialWorkingDirectory()	#	default, returns null
mkdirs	#	
copyFromLocalFile	#	default
moveFromLocalFile	#	default
copyToLocalFile	#	default
moveToLocalFile	#	default
startLocalOutput	#	default
close	#	
getUsed	#	default, slow
getBlockSize	#	
getDefaultBlockSize()	#	since 5.0.12, 5.1.6
getDefaultBlockSize(Path)	#	since 5.0.12, 5.1.6
getDefaultReplication()	#	since 5.0.12, 5.1.6
getDefaultReplication(Path)	#	since 5.0.12, 5.1.6
getFileStatus(Path)	#	
access(Path, FsAction)	#	default
createSymLink	#	throws UnsupportedOperationException
getFileLinkStatus	#	default, same as getFileStatus
supportsSymLinks	#	returns false
getLinkTarget	#	throws UnsupportedOperationException
resolveLink	#	throws UnsupportedOperationException
getFileChecksum	#	returns null

Method	Status	Comment
setVerifyChecksum	#	does nothing
setWriteChecksum	#	does nothing
getStatus	#	default, returns incorrect default data
setPermission	#	
setOwner	#	
setTimes	#	does nothing
createSnapshot	#	throws UnsupportedOperationException
renameSnapshot	#	throws UnsupportedOperationException
deleteSnapshot	#	throws UnsupportedOperationException
modifyAclEntries	#	throws UnsupportedOperationException
removeAclEntries	#	throws UnsupportedOperationException
removeDefaultAcl	#	throws UnsupportedOperationException
removeAcl	#	throws UnsupportedOperationException
setAcl	#	throws UnsupportedOperationException
getAclStatus	#	throws UnsupportedOperationException
setXAttr	#	throws UnsupportedOperationException
getXAttr	#	throws UnsupportedOperationException
getXAttrs	#	throws UnsupportedOperationException
listXAttrs	#	throws UnsupportedOperationException

Method	Status	Comment
removeXAttr	#	throws UnsupportedOperationException

## Using JMX to read DSEFS metrics

DSEFS reports status and performance metrics through JMX in domain com.datastax.bdp:type=dsefs.

DSEFS reports status and performance metrics through JMX in the domain com.datastax.bdp:type=dsefs. This page describes the classes exposed in JMX.

### Location

Location metrics provide information about each DSEFS location status. There is one set of Location metrics for each DSEFS location. Every DSE node knows about all locations, so connect to any node to get the full status of the cluster. The following gauges are defined:

#### directory

Path to the directory where DSEFS data is stored. This is a constant value configured in dse.yaml

#### estFreeSpace

Estimated amount of free space on the device where the storage directory is located, in bytes. This value is refreshed periodically, so if you need an up-to-date value, read the BlockStore.freeSpace metric.

#### estUsedSpace

Estimated amount of space used by the contents of the storage directory, in bytes. This value is refreshed periodically, so if you need an up-to-date value, read the BlockStore.usedSpace metric.

#### minFreeSpace

Amount of reserved space in bytes. Configured statically in dse.yaml.

#### privateAddress

IP and port of the endpoint for DSEFS internode communication.

#### publicAddress

IP and port of the endpoint for DSEFS clients.

#### readOnly

Returns true if the location is in read-only mode.

#### status

One of the following values: up, down, unavailable. If the location is up, the location is fully operational and this node will attempt to read or write from it. If the location is down, the location is on a node that has been gracefully shut down by the administrator and no reads or writes will be attempted. If the location is unavailable, this node has problems with communicating with that location, and the real status is unknown. This node will check the status periodically.

#### storageWeight

How much data relative to other locations will be stored in this location. This is a static value configured in dse.yaml

## BlockStore

`BlockStore` metrics report how fast and how much data is being read/written by the data layer of the DSEFS node. They are reported only for the locations managed by the node to which you connect with JMX. In order to get metrics information for all the locations in the cluster, you need to individually connect to all nodes with DSEFS.

### `blocksDeleted`

How many blocks are deleted, in blocks per second.

### `blocksRead`

Read accesses in blocks per second.

### `blocksWritten`

Writes in blocks per second.

### `bytesDeleted`

How fast data is removed, in bytes per second.

### `bytesRead`

How fast data is being read, in bytes per second.

### `bytesWritten`

How fast data is written, in bytes per second.

### `readErrors`

The total count and rate of read errors (rate in errors per second).

### `writeErrors`

The total count and rate of write errors (rate in errors per second).

### `directory`

The path to the storage directory of this location.

### `freeSpace`

How much space is left on the device in bytes.

### `usedSpace`

Estimated amount of space used by this location in bytes.

## RestServer

`RestServer` reports metrics related to the communication layer of DSEFS, separately for internode traffic and clients. Each set of these metrics is identified by a scope of the form: `listen address:listen port`. By default port 5598 is used for clients, and port 5599 is for internode communication.

### `connectionCount`

The current number of open inbound connections.

### `connectionRate`

The total rate and count of connections since the server was started.

### `requestRate`

### `deleteRate`

### `getRate`

### `postRate`

### `putRate`

The total rate and number of requests, respectively: all, DELETE, GET, POST, and PUT requests.

### `downloadBytesRate`

Throughput in bytes per second of the transfer from server to client.

#### **uploadBytesRate**

Throughput in bytes per second of the transfer from client to server.

#### **responseTime**

The time that elapses from receiving the full request body to the moment the server starts sending out the response.

#### **uploadTime**

The time it takes to read the request body from the client.

#### **downloadTime**

The time that it takes to send the response body to the client.

#### **errors**

A counter which is increased every time the service handling the request throws an unexpected error. `errors` is not increased by errors handled by the service logic. For example, file not found errors do not increment `errors`.

## **CassandraClient**

`CassandraClient` reports metrics related to the communication layer between DSEFS and the database.

#### **responseTime**

Tracks the response times of database queries.

#### **errors**

A counter increased by query execution errors (for example, timeout errors).

## DSE Search

DSE Search allows you to quickly find data and provide a modern search experience for your users, helping you create features like product catalogs, document repositories, ad-hoc reporting engines, and more.

DSE Search allows you to quickly find data and provide a modern search experience for your users, helping you create features like product catalogs, document repositories, ad-hoc reporting engines, and more.

Because DataStax Enterprise is a cohesive data management platform so other workloads such as [DSE Graph \(page 363\)](#), [DSE Analytics and Search integration \(page 172\)](#), and [DSE Analytics \(page 170\)](#) can take full advantage of the indexing and query capabilities of DSE Search.

## About DSE Search

DSE Search simplifies using search applications for data stored in a database.

DSE Search is an integral part of the always-on DataStax Enterprise (DSE) data platform. DSE Search simplifies using search applications for data stored in a database. DSE Search allows you to quickly find data and provide a modern search experience for your users, helping you create features like product catalogs, document repositories, ad-hoc reporting engines, and more.

Because DataStax Enterprise is a cohesive data management platform so other workloads such as [DSE Graph \(page 363\)](#), [DSE Analytics and Search integration \(page 172\)](#), and

DSE Analytics ([page 170](#)) can take full advantage of the indexing and query capabilities of DSE Search.

DSE Search manages search indexes with a persistent store.

The benefits of running enterprise search functions through DataStax Enterprise and DSE Search include:

- DSE Search is backed by a scalable database.
- A persistent store for search indexes.
- A fully fault-tolerant, no-single-point-of-failure search architecture across multiple datacenters.
- Add search capacity just like you add capacity in the DSE database.
- Set up replication for DSE Search nodes the same way as other nodes by [creating a keyspace](#) or [changing the replication factor](#) of a keyspace to optimize performance.
- DSE Search has two indexing modes: Near-real-time (NRT) and live indexing, also called real-time (RT) indexing. [Configure](#) and tune DSE Search for maximum indexing throughput.
- Near real-time query capabilities.
- TDE encryption of DSE Search data, including search indexes and commit logs. See [Encrypting Search indexes](#).
- CQL index management [commands](#) simplify search index management.
- Local node (optional) management of search indexing resources with [dsetool \(page 318\)](#) commands.
- Read/write to any DSE Search node and automatically index stored data.
- Examine and aggregate real-time data using CQL.
- Fault-tolerant [queries](#), efficient [deep paging](#), and advanced search node resiliency.
- [Virtual nodes \(vnodes\) \(page 168\)](#) support.
- Set the [location](#) of the search index.
- Native CQL queries leverage search indexes for a wide array of CQL query functionality and indexing support.
- Using CQL, DSE Search supports partial document updates that enable you to modify existing information while maintaining a lower transaction cost.
- Supports indexing and querying of advanced data types, including [tuples](#) and [user-defined types \(UDT\)](#).
- DSE Search is built with a production-certified version of Apache Solr™. DSE Search uses some Solr tools and APIs, the implementation does not guarantee that Solr tools and APIs work as expected. Be sure to review all [unsupported features for DSE Search \(page 306\)](#).

See the DataStax blog post [What's New for Search in DSE 6](#).

## DSE Search versus Open Source Apache Solr™

Differences between DSE Search and Open Source Solr (OSS).

By virtue of its integration into DataStax Enterprise, differences exist between DSE Search and Open Source Solr (OSS).

## Major differences

Capability	DSE Search	OS Solr	Description
Includes a database	yes	no	For OSS, create an interface to add a database.
Indexes real-time data	yes	no	Ingests real-time data and automatically indexes the data.
Provides an intuitive way to update data	yes	no	CQL for loading and updating data.
Supports data distribution	yes	yes [1]	Transparently distributes real-time, analytics, and search data to multiple nodes in a cluster.
Balances loads on nodes/shards	yes	no	Unlike Solr and Solr Cloud, DSE Search <a href="#">loads can be efficiently rebalanced</a> .
Spans indexes over multiple datacenters	yes	no	A DSE cluster can have more than one datacenter for different types of nodes.
Makes durable updates to data	yes	no	Updates are durable and written to the commit log for all updates.
Automatically reindexes search data	yes	no	OSS requires the client to reingest everything to reindex data in Solr.
Upgrades of Apache Lucene® preserve data	yes	no	DataStax integrates Lucene upgrades periodically and data is preserved when you upgrade DSE.
Supports timeAllowed queries with deep paging.	yes	no	OSS Solr does not support using timeAllowed queries with deep paging.

[1] OSS requires using Zookeeper.

## Feature differences

DSE Search supports [limiting queries by time](#) by using the [Solr timeAllowed](#) parameter.

DSE Search differs from native Solr:

- If the timeAllowed is exceeded, an exception is thrown.
- If the timeAllowed is exceeded, and the additional shards.tolerant parameter is set to true, the application returns the partial results collected so far.

When partial results are returned, the CQL custom payload contains the DSESearch.isPartialResults key.

## Unsupported features for DSE Search

Unsupported Apache Cassandra, Apache Solr, and other features.

Unsupported features include Apache Cassandra™ and Apache Solr™ features.

## Apache Solr™ and Apache Lucene® limitations

Apache Solr and Lucene limitations apply to DSE Search. For example:

- The 2 billion records per node limitation as described in [Lucene limitations](#).
- The 1024 maxBoolean clause limit in [SOLR-4586](#).
- Solr field name policy applies to the indexed field names:
  - # Every field must have a `name`.
  - # Field names must consist of alphanumeric or underscore characters only.
  - # Fields cannot start with a digit.
  - # Names with both leading and trailing underscores (for example, `_version_`) are reserved.

**Note:** Non-compliant field names are not supported from all components.  
Backward compatibility is not guaranteed.

- Limitations and known Apache Solr issues apply to DSE Search queries. For example: incorrect [SORT](#) results for tokenized text fields.

## Unsupported Apache Cassandra features

These imitations apply to DSE Search:

- Column aliases are not supported in `solr_query` queries.
- Continuous paging.
- [Static columns](#)
- Counter columns
- Super columns
- Thrift-compatible tables with column comparators other than UTF-8 or ASCII.
- PER PARTITION clause is not supported for DSE Search `solr_query` queries.
- Indexing frozen maps is not supported. However, indexing frozen sets and lists of native and user-defined (tuple/UDT) element types is supported.
- Using DSE Search with newly created COMPACT STORAGE tables is deprecated.

## Unsupported Apache Solr™ features

These imitations apply to DSE Search:

- DSE Search does not support [Solr Managed Resources](#).
- Solr schema fields that are both dynamic and multiValued only for CQL-based search indexes.
- The deprecated `replaceFields` request parameters on document updates for CQL-based search indexes. Instead, use the [suggested procedure](#) for inserting/updating data.
- Block joins based on the Lucene `BlockJoinQuery` in search indexes and CQL tables.
- Schemaless mode.
- Partial schema updates through the REST API after search indexes are changed.

For example, to update individual fields of a schema using the REST API to add a new field to a schema, you must change the `schema.xml` file, upload it again, and reload the core (same for copy fields).

- org.apache.solr.spelling.IndexBasedSpellChecker and org.apache.solr.spelling.FileBasedSpellChecker  
Instead use org.apache.solr.spelling.DirectSolrSpellChecker for spell checking.
- The commitWithin parameter.
- The SolrCloud CloudSolrServer feature of SolrJ for endpoint discovery and round-robin load balancing.
- The DSE Search configurable SolrFilterCache does not support auto-warming.
- DSE Search does not support the duration Cassandra data type.
- SELECT statements with DISTINCT are not supported with solr\_query.
- RealTime Get.
- GetReplicationHandler: Store & Restore.
- useDocValuesAsStored in schema fields and as a query request parameter.
- Solr Graph queries.
- Solr SQLStreaming aggregations.
- Data import handler.
- Tuple/UDT subfield sorting and faceting.
- The dataDir parameter in `solrconfig.xml`.

## Deprecated Solr features

- The Tika functionality that is bundled with Apache Solr is deprecated. Instead, use the stand-alone Apache Tika project.

## Other deprecated features

The following features that were previously available for use with DSE Search are deprecated and no longer supported.

- The DSE custom URP implementation is deprecated. Use the [field input/output \(page 354\)](#) (FIT) transformer API instead.

## Other unsupported features

- JBOD mode.
- The Solr updatelog is not supported in DSE Search.

The commit log replaces the Solr updatelog. Consequently, features that require the updateLog are not supported. Instead of using [atomic updates](#), partial document updates are available by running the update with CQL.

- CQL Solr queries do not support native functions or column aliases as selectors.
- RamDirectoryFactory or other non-persistent DirectoryFactory implementations.

- Tuple and UDT limitations apply.

## Configuring DSE Search

DSE Search logging, search index configuration, reference information and so on.

### DSE Search reference

Reference information for DSE Search.

Reference information for DSE Search.

### Search index config

Reference information to change query behavior for search indexes.

Reference information to change query behavior for search indexes:

- DataStax recommends CQL [CREATE SEARCH INDEX](#) and [ALTER SEARCH INDEX CONFIG](#) commands.
- [dsetool](#) ([page 318](#)) commands can also be used to manage search indexes.

### Changing search index config

To create and make changes to the search index config, follow these basic steps:

1. Create a search index. For example:

```
CREATE SEARCH INDEX ON demo.health_data;
```

2. Alter the search index. For example:

```
ALTER SEARCH INDEX CONFIG ON demo.health_data SET autoCommitTime = 30000;
```

3. Optionally view the XML of the pending search index. For example:

```
DESCRIBE PENDING SEARCH INDEX CONFIG ON demo.health_data;
```

4. Make the pending changes active. For example:

```
RELOAD SEARCH INDEX ON demo.health_data;
```

### Sample search index config

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<config>
 <abortOnConfigurationError>${solr.abortOnConfigurationError:true}</
 abortOnConfigurationError>
 <luceneMatchVersion>LUCENE_6_0_0</luceneMatchVersion>
 <dseTypeMappingVersion>2</dseTypeMappingVersion>
 <directoryFactory class="solr.StandardDirectoryFactory"
 name="DirectoryFactory" />
 <indexConfig>
 <rt>false</rt>
```

```
<rtOffheapPostings>true</rtOffheapPostings>
<useCompoundFile>false</useCompoundFile>
<ramBufferSizeMB>512</ramBufferSizeMB>
<mergeFactor>10</mergeFactor>
<reopenReaders>true</reopenReaders>
<deletionPolicy class="solr.SolrDeletionPolicy">
 <str name="maxCommitsToKeep">1</str>
 <str name="maxOptimizedCommitsToKeep">0</str>
</deletionPolicy>
<infoStream file="INFOSTREAM.txt">false</infoStream>
</indexConfig>
<jmx/>
<updateHandler class="solr.DirectUpdateHandler2">
 <autoSoftCommit>
 <maxTime>10000</maxTime>
 </autoSoftCommit>
</updateHandler>
<query>
 <maxBooleanClauses>1024</maxBooleanClauses>
 <filterCache class="solr.SolrFilterCache" highWaterMarkMB="2048"
lowWaterMarkMB="1024"/>
 <enableLazyFieldLoading>true</enableLazyFieldLoading>
 <useColdSearcher>true</useColdSearcher>
 <maxWarmingSearchers>16</maxWarmingSearchers>
</query>
<requestDispatcher handleSelect="true">
 <requestParsers enableRemoteStreaming="true"
multipartUploadLimitInKB="2048000"/>
 <httpCaching never304="true"/>
</requestDispatcher>
<requestHandler class="solr.SearchHandler" default="true"
name="search">
 <lst name="defaults">
 <int name="rows">10</int>
 </lst>
</requestHandler>
<requestHandler
class="com.datastax.bdp.search.solr.handler.component.CqlSearchHandler"
name="solr_query">
 <lst name="defaults">
 <int name="rows">10</int>
 </lst>
</requestHandler>
<requestHandler class="solr.UpdateRequestHandler" name="/update"/>
<requestHandler class="solr.UpdateRequestHandler" name="/update/csv"
startup="lazy"/>
<requestHandler class="solr.UpdateRequestHandler" name="/update/json"
startup="lazy"/>
<requestHandler class="solr.FieldAnalysisRequestHandler" name="/
analysis/field" startup="lazy"/>
<requestHandler class="solr.DocumentAnalysisRequestHandler" name="/
analysis/document" startup="lazy"/>
<requestHandler class="solr.admin.AdminHandlers" name="/admin//>
<requestHandler class="solr.PingRequestHandler" name="/admin/ping">
```

```

<lst name="invariants">
 <str name="qt">search</str>
 <str name="q">solrpinglequery</str>
</lst>
<lst name="defaults">
 <str name="echoParams">all</str>
</lst>
</requestHandler>
<requestHandler class="solr.DumpRequestHandler" name="/debug/dump">
 <lst name="defaults">
 <str name="echoParams">explicit</str>
 <str name="echoHandler">true</str>
 </lst>
</requestHandler>
<admin>
 <defaultQuery>*:*</defaultQuery>
</admin>
</config>

```

For CQL index management, use configuration element shortcuts with CQL commands.

Configuration elements are listed alphabetically by shortcut. The XML element is shown with the element start tag. An ellipsis indicates that other elements or attributes are not shown.

### **autoCommitTime**

Defines the time interval between updates to the search index with the most recent data after an INSERT, UPDATE, or DELETE. By default, changes are automatically committed every 10000 milliseconds. To change the time interval between updates:

1. Set auto commit time on the pending search index:

```
ALTER SEARCH INDEX CONFIG ON wiki.solr SET autoCommitTime = 30000;
```

2. You can view the pending search config:

```
DESCRIBE PENDING SEARCH INDEX CONFIG on wiki.solr;
```

The resulting XML shows the maximum time between updates is 30000 milliseconds:

```

<updateHandler class="solr.DirectUpdateHandler2">
 <autoSoftCommit>
 <maxTime>30000</maxTime>
 </autoSoftCommit>
</updateHandler>

```

3. To make the pending changes active, reload the search index:

```
RELOAD SEARCH INDEX ON wiki.solr;
```

See [Configuring and tuning indexing performance](#).

### **defaultQueryField**

Name of the default field to query. Default not set. To set the field to use when no field is specified by the query, see [Setting up default query field](#).

### **directoryFactory**

The directory factory to use for search indexes. Encryption is enabled per search index. To enable [encryption for a search index](#), change the class for directoryFactory to EncryptedFSDirectoryFactory.

1. Enable encryption on the pending search index:

```
ALTER SEARCH INDEX CONFIG ON wiki.solr SET directoryFactory =
EncryptedFSDirectoryFactory;
```

2. You can view the pending search config:

```
DESCRIBE PENDING SEARCH INDEX CONFIG on wiki.solr;
```

The resulting XML shows that encryption is enabled:

```
<directoryFactory class="solr.EncryptedFSDirectoryFactory"
name="DirectoryFactory"/>
```

3. To make the pending changes active, reload the search index:

```
RELOAD SEARCH INDEX ON wiki.solr;
```

Even though additional properties are available to tune encryption, DataStax recommends using the default settings.

### **filterCacheLowWaterMark**

Default is 1024 MB. See below.

### **filterCacheHighWaterMark**

Default is 2048 MB.

The DSE Search configurable filter cache reliably bounds the filter cache memory usage for a search index. This implementation contrasts with the default Solr implementation which defines bounds for filter cache usage per segment.

SolrFilterCache bounding works by evicting cache entries after the configured per search index (per core) high watermark is reached, and stopping after the configured lower watermark is reached.

#### **Note:**

- The filter cache is cleared when the search index is reloaded.
- SolrFilterCache does not support auto-warming.

SolrFilterCache defaults to offheap. In general, the larger the index is, then the larger the filter cache should be. A good default is 1 to 2 GB. If the index is 1 billion docs per node, then set to 4 to 5 GB.

1. To change cache eviction for a large index, set the low and high values one at a time:

```
ALTER SEARCH INDEX CONFIG ON solr.wiki SET
filterCacheHighWaterMark = 5000;
```

```
ALTER SEARCH INDEX CONFIG ON solr.wiki SET
filterCacheLowWaterMark = 2000;
```

**2.** View the pending search index config:

```
<query>
...
<filterCache class="solr.SolrFilterCache"
highWaterMarkMB="5000" lowWaterMarkMB="2000"/>
...
</query>
```

**3.** To make the pending changes active, reload the search index:

```
RELOAD SEARCH INDEX ON wiki.solr;
```

### **mergeFactor**

When a new segment causes the number of lowest-level segments to exceed the merge factor value, then those segments are merged together to form a single large segment. When the merge factor is 10, each merge results in the creation of a single segment that is about ten times larger than each of its ten constituents. When there are 10 of these larger segments, then they in turn are merged into an even larger single segment. Default is 10.

**1.** To change the number of segments to merge at one time:

```
ALTER SEARCH INDEX CONFIG ON solr.wiki SET mergeFactor = 5;
```

**2.** View the pending search index config:

```
<indexConfig>
...
<mergeFactor>10</mergeFactor>
...
</indexConfig>
```

**3.** To make the pending changes active, reload the search index:

```
RELOAD SEARCH INDEX ON wiki.solr;
```

### **mergeMaxThreadCount**

Must configure with mergeMaxMergeCount. The number of concurrent merges that Lucene can perform for the Solr core. The default mergeScheduler settings are set automatically. Do not adjust this setting. Default:  $\frac{1}{2}$  the number of tpc\_cores

**Note:** Exception is ... DEBUG ... ^^

### **mergeMaxMergeCount**

Must configure with mergeMaxThreadCount. The number of pending merges (active and in the backlog) that can accumulate before segment merging starts to block/throttle incoming writes. The default mergeScheduler settings are set automatically. Do not adjust this setting. Default: double the mergeMaxThreadCount

#### **ramBufferSize**

The index RAM buffer size in megabytes (MB). The RAM buffer holds uncommitted documents. A larger RAM buffer reduces flushes. Segments are also larger when flushed. Fewer flushes reduces I/O pressure which is ideal for higher write workload scenarios. Default is 512.

For example, adjust the ramBufferSize when you configure live indexing:

```
ALTER SEARCH INDEX CONFIG ON wiki.solr SET autoCommitTime = 100;
ALTER SEARCH INDEX CONFIG ON wiki.solr SET realtime = true;
ALTER SEARCH INDEX CONFIG ON wiki.solr SET ramBufferSize = 2048;
RELOAD SEARCH INDEX ON wiki.solr ;
```

#### **realtime**

Enables live indexing to increase indexing throughput. Enable live indexing on only one node per cluster. Live indexing, also called real-time (RT) indexing, supports searching directly against the Lucene RAM buffer and more frequent, cheaper soft-commits, which provide earlier visibility to newly indexed data.

Live indexing requires a larger RAM buffer and more memory usage than an otherwise equivalent NRT setup. See [Tuning RT indexing](#).

## **Configuration elements without shortcuts**

To specify configuration elements that do not have shortcuts, you can specify the XML path to the setting and separate child elements using a period.

#### **deleteApplicationStrategy**

Controls how to retrieve deleted documents when deletes are being applied. Seek exact is the safe default most people should choose, but if you are looking for a little extra performance you can try seek ceiling.

Valid case-insensitive values are:

- seekexact
  - Uses bloom filters to avoid reading from most segments. Use when memory is limited and the unique key field data does not fit into memory.
- seekceiling
  - More performant when documents are deleted/inserted into the database with sequential keys, because this strategy can stop reading from segments when it is known that terms can no longer appear.

#### **mergePolicyFactory**

The AutoExpungeDeletesTieredMergePolicy custom merge policy is based on TieredMergePolicy. This policy cleans up the large segments by merging them when deletes reach the percentage threshold. A single auto expunge merge occurs at a time. Use for large indexes that are not merging the largest segments

due to deletes. To determine whether this merge setting is appropriate for your workflow, view the segments on the Solr [Segment Info](#) screen.

When set, the XML is described as:

```
<indexConfig>
 <mergePolicyFactory
 class="org.apache.solr.index.AutoExpungeDeletesTieredMergePolicyFactory">
 <int name="maxMergedSegmentMB">1005</int>
 <int name="forceMergeDeletesPctAllowed">25</int>
 <bool name="mergeSingleSegments">true</bool>
 </mergePolicyFactory>
</indexConfig>
```

To extend TieredMergePolicy to support automatic removal of deletes:

1. To enable automatic removal of deletes, set the custom policy:

```
ALTER SEARCH INDEX CONFIG ON wiki.solr SET
 indexConfig.mergePolicyFactory[@class='org.apache.solr.index.AutoExpungeDeletesTieredMergePolicyFactory']
 = true;
```

2. Set the maximum segment size in MB:

```
ALTER SEARCH INDEX CONFIG ON wiki.solr SET
 indexConfig.mergePolicyFactory[@class='org.apache.solr.index.AutoExpungeDeletesTieredMergePolicyFactory']
 = 1005;
```

3. Set the percentage threshold for deleting from the large segments:

```
ALTER SEARCH INDEX CONFIG ON wiki.solr SET
 indexConfig.mergePolicyFactory[@class='org.apache.solr.index.AutoExpungeDeletesTieredMergePolicyFactory']
 = 25;
```

If mergeFactor is in the existing index config, you must drop it from the search index before you alter the table to support automatic removal of deletes:

```
ALTER SEARCH INDEX CONFIG ON wiki.solr DROP
 indexConfig.mergePolicyFactory;
```

### **parallelDeleteTasks**

Regulates how many tasks are created to apply deletes during soft/hard commit in parallel. Supported for RT and NRT indexing. Specify a positive number greater than 0. The default value is the number of available processors.

Leave parallelDeleteTasks at the default value, except when issues occur with write load when running a mixed read/write workload. If writes occasionally spike in utilization and negatively impact your read performance, then set this value lower.

## **Search index schema**

Reference information about the schema defines the relationship between data in a table and a search index.

Search index schema reference information to use for creating and altering a search index schema:

- DataStax recommends CQL [CREATE SEARCH INDEX](#) and [ALTER SEARCH INDEX SCHEMA](#) commands.
- [dsetool](#) ([page 318](#)) commands can also be used to manage search indexes.

The schema defines the relationship between data in a table and a search index. See [Creating a search index with default values](#) and [Quick Start for CQL index management](#) for details and examples.

A sample search index schema XML:

### Sample XML

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<schema name="autoSolrSchema" version="1.5">
 <types>
 <fieldType class="org.apache.solr.schema.TextField"
name="TextField">
 <analyzer>
 <tokenizer class="solr.StandardTokenizerFactory"/>
 <filter class="solr.LowerCaseFilterFactory" />
 </analyzer>
 </fieldType>
 <fieldType class="org.apache.solr.schema.TrieIntField"
name="TrieIntField"/>
 </types>
 <fields>
 <field indexed="true" multiValued="false" name="grade_completed"
type="TextField"/>
 <field indexed="true" multiValued="false"
name="diagnosed_thyroid_disease" type="TextField"/>
 <field indexed="true" multiValued="false" name="pets"
type="TextField"/>
 <field indexed="true" multiValued="false" name="secondary_smoke"
type="TextField"/>
 <field indexed="true" multiValued="false" name="diagnosed_lupus"
type="TextField"/>
 <field indexed="true" multiValued="false" name="gender"
type="TextField"/>
 <field indexed="true" multiValued="false" name="birthplace"
type="TextField"/>
 <field docValues="true" indexed="true" multiValued="false"
name="income_group" type="TrieIntField"/>
 <field indexed="true" multiValued="false" name="marital_status"
type="TextField"/>
 <field docValues="true" indexed="true" multiValued="false"
name="age_months" type="TrieIntField"/>
 <field indexed="true" multiValued="false" name="bird"
type="TextField"/>
 <field indexed="true" multiValued="false" name="hay_fever"
type="TextField"/>
```

```

<field indexed="true" multiValued="false"
name="diagnosed_hay_fever" type="TextField"/>
<field indexed="true" multiValued="false"
name="routine_medical_coverage" type="TextField"/>
<field indexed="true" multiValued="false"
name="annual_income_20000" type="TextField"/>
<field indexed="true" multiValued="false" name="exam_status"
type="TextField"/>
<field indexed="true" multiValued="false" name="other_pet"
type="TextField"/>
<field indexed="true" multiValued="false" name="diagnosed_stroke"
type="TextField"/>
<field indexed="true" multiValued="false" name="employer_paid_plan"
type="TextField"/>
<field docValues="true" indexed="true" multiValued="false"
name="family_sequence" type="TrieIntegerField"/>
<field indexed="true" multiValued="false"
name="diagnosed_cataracts" type="TextField"/>
<field indexed="true" multiValued="false"
name="major_medical_coverage" type="TextField"/>
<field indexed="true" multiValued="false" name="diagnosed_gout"
type="TextField"/>
<field indexed="true" multiValued="false" name="age_unit"
type="TextField"/>
<field indexed="true" multiValued="false" name="goiter"
type="TextField"/>
<field indexed="true" multiValued="false" name="chronic_bronchitis"
type="TextField"/>
<field indexed="true" multiValued="false" name="county"
type="TextField"/>
<field docValues="true" indexed="true" multiValued="false"
name="num_smokers" type="TrieIntegerField"/>
<field indexed="true" multiValued="false" name="screening_month"
type="TextField"/>
<field indexed="true" multiValued="false"
name="diagnosed_emphysema" type="TextField"/>
<field indexed="true" multiValued="false"
name="diagnosed_other_cancer" type="TextField"/>
<field docValues="true" indexed="true" multiValued="false"
name="id" type="TrieIntegerField"/>
<field indexed="true" multiValued="false" name="dental_coverage"
type="TextField"/>
<field indexed="true" multiValued="false" name="health_status"
type="TextField"/>
<field docValues="true" indexed="true" multiValued="false"
name="monthly_income_total" type="TrieIntegerField"/>
<field indexed="true" multiValued="false" name="fish"
type="TextField"/>
<field indexed="true" multiValued="false" name="dog"
type="TextField"/>
<field indexed="true" multiValued="false" name="asthma"
type="TextField"/>
<field indexed="true" multiValued="false" name="ethnicity"
type="TextField"/>
```

```
<field docValues="true" indexed="true" multiValued="false"
name="age" type="TrieIntField"/>
<field indexed="true" multiValued="false" name="diagnosed_asthma"
type="TextField"/>
<field indexed="true" multiValued="false" name="race_ethnicity"
type="TextField"/>
<field indexed="true" multiValued="false"
name="diagnosed_congestive_heart_failure" type="TextField"/>
<field docValues="true" indexed="true" multiValued="false"
name="family_size" type="TrieIntField"/>
<field indexed="true" multiValued="false" name="race"
type="TextField"/>
<field indexed="true" multiValued="false" name="thyroid_disease"
type="TextField"/>
<field indexed="true" multiValued="false" name="bronchitis"
type="TextField"/>
<field docValues="true" indexed="true" multiValued="false"
name="household_size" type="TrieIntField"/>
<field indexed="true" multiValued="false" name="cat"
type="TextField"/>
<field indexed="true" multiValued="false" name="diagnosed_goiter"
type="TextField"/>
<field indexed="true" multiValued="false"
name="diagnosed_skin_cancer" type="TextField"/>
<field indexed="true" multiValued="false" name="fips"
type="TextField"/>
</fields>
<uniqueKey>(id,age)</uniqueKey>
</schema>
```

## dsetool search index commands

A list of the dsetool commands for search index management.

### dsetool commands for DSE Search

The dsetool commands for DSE Search provide search index management.

- [dsetool create\\_core \(page 809\)](#)
- [dsetool core\\_indexing\\_status \(page 807\)](#)
- [dsetool get\\_core\\_config \(page 816\)](#)
- [dsetool get\\_core\\_schema \(page 818\)](#)
- [dsetool index\\_checks \(experimental\) \(page 821\)](#)
- [dsetool infer\\_solr\\_schema \(page 823\)](#)
- [dsetool list\\_index\\_files \(page 826\)](#)
- [dsetool read\\_resource \(page 843\)](#)
- [dsetool rebuild\\_indexes \(page 844\)](#)
- [dsetool reload\\_core \(page 846\)](#)
- [dsetool stop\\_core\\_reindex \(page 853\)](#)
- [dsetool unload\\_core \(page 858\)](#)
- [dsetool upgrade\\_index\\_files \(page 859\)](#)

- [dsetool write\\_resource \(page 861\)](#)

DataStax recommends using [CQL](#) commands to manage search indexes.

## Configuration properties

Reference information for DSE Search configuration properties.

Reference information for DSE Search configuration properties.

- Data location in `cassandra.yaml` ([page 319](#))
- Scheduler settings in `dse.yaml` ([page 319](#))
- Indexing settings in `dse.yaml` ([page 320](#))
- Safety thresholds in `cassandra.yaml` ([page 321](#))
- Inter-node communication in `dse.yaml` ([page 321](#))
- Query options in `dse.yaml` ([page 322](#))
- Client connections in `dse.yaml` ([page 322](#))
- Performance in `cassandra.yaml` ([page 322](#))
- Performance in `dse.yaml` ([page 323](#))

### Data location in `cassandra.yaml`

See [Set the location of search indexes](#).

#### `data_file_directories`

The directory where table data is stored on disk. The database distributes data evenly across the location, subject to the granularity of the configured compaction strategy. If not set, the directory is `$DSE_HOME/data/data`.

**Tip:** For production, DataStax recommends [RAID 0 and SSDs](#).

Default: `- /var/lib/cassandra/data`

### Scheduler settings in `dse.yaml`

Configuration options to control the scheduling and execution of indexing checks.

#### `ttl_index_rebuild_options`

Section of options to control the schedulers in charge of querying for and removing expired records, and the execution of the checks.

#### `fix_rate_period`

Time interval to check for expired data in seconds.

Default: 300

#### `initial_delay`

The number of seconds to delay the first TTL check to speed up start-up time.

Default: 20

#### `max_docs_per_batch`

The maximum number of documents to check and delete per batch by the TTL rebuild thread. All documents determined to be expired are deleted from the index during each check, to avoid memory pressure, their unique keys are retrieved and deletes issued in batches.

Default: 4096

#### `thread_pool_size`

The maximum number of cores that can execute TTL cleanup concurrently. Set the `thread_pool_size` to manage system resource consumption and prevent many search cores from executing simultaneous TTL deletes.

Default: 1

## Indexing settings in `dse.yaml`

### `solr_resource_upload_limit_mb`

Option to disable or configure the maximum file size of the search index config or schema. Resource files can be uploaded, but the search index config and schema are stored internally in the database after upload.

- 0 - disable resource uploading
- upload size - The maximum upload size limit in megabytes (MB) for a DSE Search resource file (search index config or schema).

Default: 10

### `flush_max_time_per_core`

The maximum time, in minutes, to wait for the flushing of asynchronous index updates that occurs at DSE Search commit time or at flush time. Expert level knowledge is required to change this value. Always set the value reasonably high to ensure flushing completes successfully to fully sync DSE Search indexes with the database data. If the configured value is exceeded, index updates are only partially committed and the commit log is not truncated which can undermine data durability.

**Note:** When a timeout occurs, it usually means this node is being overloaded and cannot flush in a timely manner. Live indexing increases the time to flush asynchronous index updates.

Default: commented out (5)

### `load_max_time_per_core`

The maximum time, in minutes, to wait for each DSE Search index to load on startup or create/reload operations. This advanced option should be changed only if exceptions happen during search index loading. When not set, the default is 5 minutes.

Default: commented out (5)

### `enable_index_disk_failure_policy`

Whether to apply the configured disk failure policy if `IOExceptions` occur during index update operations.

- true - apply the configured Cassandra disk failure policy to index write failures
- false - do not apply the disk failure policy

When not set, the default is false.

Default: commented out (`false`)

### `solr_data_dir`

The directory to store index data. By default, each DSE Search index is saved in `solrconfig_data_dir/keyspace_name.table_name`, or as specified by the `dse.solr.data.dir` system property.

**Tip:** See [Managing the location of DSE Search data](#).

Default: commented out (/MyDir)

#### **solr\_field\_cache\_enabled**

The Apache Lucene® field cache is deprecated. Instead, for fields that are sorted, faceted, or grouped by, set docValues="true" on the field in the search index schema. Then reload the search index and reindex. When not set, the default is false.

Default: commented out (false)

#### **async\_bootstrap\_reindex**

For DSE Search, configure whether to asynchronously reindex bootstrapped data. Default: false

- If enabled, the node joins the ring immediately after bootstrap and reindexing occurs asynchronously. Do not wait for post-bootstrap reindexing so that the node is not marked down.
- If disabled, the node joins the ring after reindexing the bootstrapped data.

### **Safety thresholds**

Configure safety thresholds and fault tolerance for DSE Search with options in dse.yaml and cassandra.yaml.

#### **Safety thresholds in cassandra.yaml**

Configuration options include:

#### **read\_request\_timeout\_in\_ms**

Default: 5000. How long the coordinator waits for read operations to complete before timing it out.

### **Security in dse.yaml**

Security options for DSE Search. See [DSE Search security checklist](#).

#### **solr\_encryption\_options**

Settings to tune encryption of search indexes.

#### **decryption\_cache\_offheap\_allocation**

Whether to allocate shared DSE Search decryption cache off JVM heap.

- true - allocate shared DSE Search decryption cache off JVM heap
- false - do not allocate shared DSE Search decryption cache off JVM heap

When not set, the default is true.

Default: commented out (true)

#### **decryption\_cache\_size\_in\_mb**

The maximum size of shared DSE Search decryption cache in megabytes (MB).

Default: commented out (256)

#### **http\_principal**

The http\_principal is used by the Tomcat application container to run DSE Search. The Tomcat web server uses the GSSAPI mechanism (SPNEGO) to negotiate the GSSAPI security mechanism (Kerberos). Set *REALM* to the name of your Kerberos realm. In the Kerberos principal, *REALM* must be uppercase.

### **Inter-node communication in dse.yaml**

Inter-node communication between DSE Search nodes.

### **shard\_transport\_options**

Fault tolerance option for inter-node communication between DSE Search nodes.

### **netty\_client\_request\_timeout**

Timeout behavior during distributed queries. The internal timeout for all search queries to prevent long running queries. The client request timeout is the maximum cumulative time (in milliseconds) that a distributed search request will wait idly for shard responses.

Default: 60000 (1 minute)

### **Query options in dse.yaml**

Options for CQL Solr queries.

#### **cql\_solr\_query\_paging**

- driver - Respects driver paging settings. Specifies to use [Solr pagination](#) (cursors) only when the driver uses pagination. Enabled automatically for DSE SearchAnalytics workloads.
- off - Paging is off. Ignore driver paging settings for [CQL queries](#) and use normal Solr paging unless:
  - # The current workload is an analytics workload, including SearchAnalytics. SearchAnalytics nodes always use driver paging settings.
  - # The cqlsh query parameter paging is set to driver.

Even when `cql_solr_query_paging: off`, paging is dynamically enabled with the `"paging": "driver"` parameter in [JSON queries](#).

When not set, the default is off.

Default: commented out (`off`)

#### **cql\_solr\_query\_row\_timeout**

The maximum time in milliseconds to wait for each row to be read from the database during CQL Solr queries.

Default: commented out (`10000` 10 seconds)

### **Client connections in dse.yaml**

The default IP address that the HTTP and Solr Admin interface uses to access DSE Search. See [Changing Tomcat web server settings](#).

#### **native\_transport\_address**

When left blank, uses the configured hostname of the node. Unlike the `listen_address`, this value can be set to 0.0.0.0, but you must set the `native_transport_broadcast_address` to a value other than 0.0.0.0.

**Note:** Set `native_transport_address` OR `native_transport_interface`, not both.

Default: localhost

### **Performance in cassandra.yaml**

Decreasing the memtable space to make room for Solr caches might improve performance. See [Changing the stack size and memtable space](#).

#### **memtable\_heap\_space\_in\_mb**

The amount of on-heap memory allocated for memtables. The database uses the total of this amount and the value of memtable\_offheap\_space\_in\_mb to set a threshold for automatic memtable flush.

**Tip:** See [memtable\\_cleanup\\_threshold \(page 80\)](#) and [Tuning the Java heap](#).

Default: *calculated 1/4 of heap size (2048)*

### Performance in dse.yaml

Node routing options.

### node\_health\_options

Node health options are always enabled.

### refresh\_rate\_ms

Default: 60000

### uptime\_ramp\_up\_period\_seconds

The amount of continuous uptime required for the node's uptime score to advance the [node health score](#) from 0 to 1 (full health), assuming there are no recent dropped mutations. The health score is a composite score based on dropped mutations and uptime.

**Tip:** If a node is repairing after a period of downtime, you might want to increase the uptime period to the expected repair time.

Default: commented out (10800 3 hours)

### dropped\_mutation\_window\_minutes

The historic time window over which the rate of dropped mutations affect the node health score.

Default: 30

## Viewing search index schema and config

How to view search index schema and config resources.

Search index schema and config are stored internally in the database. When you modify a search index schema or config, the changes are *pending*.

Use the [RELOAD SEARCH INDEX](#) command to apply the pending changes to the active (in use) search index.

DataStax recommends using CQL to view the pending or active (in use) schema or config.

### CQL shell DESCRIBE command

Use the CQL shell command DESCRIBE SEARCH INDEX to view the active and pending schema and config.

Show the active index config for wiki.solr:

```
DESCRIBE ACTIVE SEARCH INDEX CONFIG ON demo.health_data;
```

The results are shown in XML:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<config>
```

```
<abortOnConfigurationError>${solr.abortOnConfigurationError:true}</
abortOnConfigurationError>
<luceneMatchVersion>LUCENE_6_0_0</luceneMatchVersion>
<dseTypeMappingVersion>2</dseTypeMappingVersion>
<directoryFactory class="solr.StandardDirectoryFactory"
name="DirectoryFactory"/>
<indexConfig>
<rt>false</rt>
<rtOffheapPostings>true</rtOffheapPostings>
<useCompoundFile>false</useCompoundFile>
<ramBufferSizeMB>512</ramBufferSizeMB>
<mergeFactor>10</mergeFactor>
<reopenReaders>true</reopenReaders>
<deletionPolicy class="solr.SolrDeletionPolicy">
 <str name="maxCommitsToKeep">1</str>
 <str name="maxOptimizedCommitsToKeep">0</str>
</deletionPolicy>
<infoStream file="INFOSTREAM.txt">false</infoStream>
</indexConfig>
<jmx/>
<updateHandler class="solr.DirectUpdateHandler2">
 <autoSoftCommit>
 <maxTime>10000</maxTime>
 </autoSoftCommit>
</updateHandler>
<query>
 <maxBooleanClauses>1024</maxBooleanClauses>
 <filterCache class="solr.SolrFilterCache" highWaterMarkMB="2048"
lowWaterMarkMB="1024"/>
 <enableLazyFieldLoading>true</enableLazyFieldLoading>
 <useColdSearcher>true</useColdSearcher>
 <maxWarmingSearchers>16</maxWarmingSearchers>
</query>
<requestDispatcher handleSelect="true">
 <requestParsers enableRemoteStreaming="true"
multipartUploadLimitInKB="2048000"/>
 <httpCaching never304="true" />
</requestDispatcher>
<requestHandler class="solr.SearchHandler" default="true"
name="search">
 <lst name="defaults">
 <int name="rows">10</int>
 </lst>
</requestHandler>
<requestHandler
class="com.datastax.bdp.search.solr.handler.component.CqlSearchHandler"
name="solr_query">
 <lst name="defaults">
 <int name="rows">10</int>
 </lst>
</requestHandler>
<requestHandler class="solr.UpdateRequestHandler" name="/update" />
<requestHandler class="solr.UpdateRequestHandler" name="/update/csv"
startup="lazy" />
```

```

<requestHandler class="solr.UpdateRequestHandler" name="/update/json"
startup="lazy"/>
<requestHandler class="solr.FieldAnalysisRequestHandler" name="/analysis/field" startup="lazy"/>
<requestHandler class="solr.DocumentAnalysisRequestHandler" name="/analysis/document" startup="lazy"/>
<requestHandler class="solr.admin.AdminHandlers" name="/admin/" />
<requestHandler class="solr.PingRequestHandler" name="/admin/ping">
 <lst name="invariants">
 <str name="qt">search</str>
 <str name="q">solrpingleader</str>
 </lst>
 <lst name="defaults">
 <str name="echoParams">all</str>
 </lst>
</requestHandler>
<requestHandler class="solr.DumpRequestHandler" name="/debug/dump">
 <lst name="defaults">
 <str name="echoParams">explicit</str>
 <str name="echoHandler">true</str>
 </lst>
</requestHandler>
<admin>
 <defaultQuery>*:*</defaultQuery>
</admin>
</config>
```

Show the pending index config:

View the pending search index config or schema before it is active. For example, to view the pending index schema for demo.health\_data:

```
DESCRIBE PENDING SEARCH INDEX SCHEMA ON demo.health_data;
```

The results are shown in XML:

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<schema name="autoSolrSchema" version="1.5">
 <types>
 <fieldType class="org.apache.solr.schema.TextField"
name="TextField">
 <analyzer>
 <tokenizer class="solr.StandardTokenizerFactory"/>
 <filter class="solr.LowerCaseFilterFactory"/>
 </analyzer>
 </fieldType>
 <fieldType class="org.apache.solr.schema.TrieIntField"
name="TrieIntField"/>
 </types>
 <fields>
 <field indexed="true" multiValued="false" name="grade_completed"
type="TextField"/>
 <field indexed="true" multiValued="false"
name="diagnosed_thyroid_disease" type="TextField"/>
```

```
<field indexed="true" multiValued="false" name="pets"
type="TextField"/>
<field indexed="true" multiValued="false" name="secondary_smoke"
type="TextField"/>
<field indexed="true" multiValued="false" name="diagnosed_lupus"
type="TextField"/>
<field indexed="true" multiValued="false" name="gender"
type="TextField"/>
<field indexed="true" multiValued="false" name="birthplace"
type="TextField"/>
<field docValues="true" indexed="true" multiValued="false"
name="income_group" type="TrieIntegerField"/>
<field indexed="true" multiValued="false" name="marital_status"
type="TextField"/>
<field docValues="true" indexed="true" multiValued="false"
name="age_months" type="TrieIntegerField"/>
<field indexed="true" multiValued="false" name="bird"
type="TextField"/>
<field indexed="true" multiValued="false" name="hay_fever"
type="TextField"/>
<field indexed="true" multiValued="false" name="diagnosed_hay_fever"
type="TextField"/>
<field indexed="true" multiValued="false"
name="routine_medical_coverage" type="TextField"/>
<field indexed="true" multiValued="false" name="annual_income_20000"
type="TextField"/>
<field indexed="true" multiValued="false" name="exam_status"
type="TextField"/>
<field indexed="true" multiValued="false" name="other_pet"
type="TextField"/>
<field indexed="true" multiValued="false" name="diagnosed_stroke"
type="TextField"/>
<field indexed="true" multiValued="false" name="employer_paid_plan"
type="TextField"/>
<field docValues="true" indexed="true" multiValued="false"
name="family_sequence" type="TrieIntegerField"/>
<field indexed="true" multiValued="false" name="diagnosed_cataracts"
type="TextField"/>
<field indexed="true" multiValued="false"
name="major_medical_coverage" type="TextField"/>
<field indexed="true" multiValued="false" name="diagnosed_gout"
type="TextField"/>
<field indexed="true" multiValued="false" name="age_unit"
type="TextField"/>
<field indexed="true" multiValued="false" name="goiter"
type="TextField"/>
<field indexed="true" multiValued="false" name="chronic_bronchitis"
type="TextField"/>
<field indexed="true" multiValued="false" name="county"
type="TextField"/>
<field docValues="true" indexed="true" multiValued="false"
name="num_smokers" type="TrieIntegerField"/>
<field indexed="true" multiValued="false" name="screening_month"
type="TextField"/>
```

```

<field indexed="true" multiValued="false" name="diagnosed_emphysema"
type="TextField"/>
<field indexed="true" multiValued="false"
name="diagnosed_other_cancer" type="TextField"/>
<field docValues="true" indexed="true" multiValued="false" name="id"
type="TrieIntField"/>
<field indexed="true" multiValued="false" name="dental_coverage"
type="TextField"/>
<field indexed="true" multiValued="false" name="health_status"
type="TextField"/>
<field docValues="true" indexed="true" multiValued="false"
name="monthly_income_total" type="TrieIntField"/>
<field indexed="true" multiValued="false" name="fish"
type="TextField"/>
<field indexed="true" multiValued="false" name="dog"
type="TextField"/>
<field indexed="true" multiValued="false" name="asthma"
type="TextField"/>
<field indexed="true" multiValued="false" name="ethnicity"
type="TextField"/>
<field docValues="true" indexed="true" multiValued="false"
name="age" type="TrieIntField"/>
<field indexed="true" multiValued="false" name="diagnosed_asthma"
type="TextField"/>
<field indexed="true" multiValued="false" name="race_ethnicity"
type="TextField"/>
<field indexed="true" multiValued="false"
name="diagnosed_congestive_heart_failure" type="TextField"/>
<field docValues="true" indexed="true" multiValued="false"
name="family_size" type="TrieIntField"/>
<field indexed="true" multiValued="false" name="race"
type="TextField"/>
<field indexed="true" multiValued="false" name="thyroid_disease"
type="TextField"/>
<field indexed="true" multiValued="false" name="bronchitis"
type="TextField"/>
<field docValues="true" indexed="true" multiValued="false"
name="household_size" type="TrieIntField"/>
<field indexed="true" multiValued="false" name="cat"
type="TextField"/>
<field indexed="true" multiValued="false" name="diagnosed_goiter"
type="TextField"/>
<field indexed="true" multiValued="false"
name="diagnosed_skin_cancer" type="TextField"/>
<field indexed="true" multiValued="false" name="fips"
type="TextField"/>
</fields>
<uniqueKey>(id,age)</uniqueKey>
</schema>
```

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<schema name="autoSolrSchema" version="1.5">
<types>
```

```
<fieldType class="org.apache.solr.schema.TextField"
name="TextField">
 <analyzer>
 <tokenizer class="solr.StandardTokenizerFactory"/>
 <filter class="solr.LowerCaseFilterFactory"/>
 </analyzer>
</fieldType>
<fieldType class="org.apache.solr.schema.TrieIntField"
name="TrieIntField"/>
</types>
<fields>
 <field indexed="true" multiValued="false" name="grade_completed"
type="TextField"/>
 <field indexed="true" multiValued="false"
name="diagnosed_thyroid_disease" type="TextField"/>
 <field indexed="true" multiValued="false" name="pets"
type="TextField"/>
 <field indexed="true" multiValued="false" name="secondary_smoke"
type="TextField"/>
 <field indexed="true" multiValued="false" name="diagnosed_lupus"
type="TextField"/>
 <field indexed="true" multiValued="false" name="gender"
type="TextField"/>
 <field indexed="true" multiValued="false" name="birthplace"
type="TextField"/>
 <field docValues="true" indexed="true" multiValued="false"
name="income_group" type="TrieIntField"/>
 <field indexed="true" multiValued="false" name="marital_status"
type="TextField"/>
 <field docValues="true" indexed="true" multiValued="false"
name="age_months" type="TrieIntField"/>
 <field indexed="true" multiValued="false" name="bird"
type="TextField"/>
 <field indexed="true" multiValued="false" name="hay_fever"
type="TextField"/>
 <field indexed="true" multiValued="false" name="diagnosed_hay_fever"
type="TextField"/>
 <field indexed="true" multiValued="false"
name="routine_medical_coverage" type="TextField"/>
 <field indexed="true" multiValued="false" name="annual_income_20000"
type="TextField"/>
 <field indexed="true" multiValued="false" name="exam_status"
type="TextField"/>
 <field indexed="true" multiValued="false" name="other_pet"
type="TextField"/>
 <field indexed="true" multiValued="false" name="diagnosed_stroke"
type="TextField"/>
 <field indexed="true" multiValued="false" name="employer_paid_plan"
type="TextField"/>
 <field docValues="true" indexed="true" multiValued="false"
name="family_sequence" type="TrieIntField"/>
 <field indexed="true" multiValued="false" name="diagnosed_cataracts"
type="TextField"/>
```

```

<field indexed="true" multiValued="false"
name="major_medical_coverage" type="TextField"/>
<field indexed="true" multiValued="false" name="diagnosed_gout"
type="TextField"/>
<field indexed="true" multiValued="false" name="age_unit"
type="TextField"/>
<field indexed="true" multiValued="false" name="goiter"
type="TextField"/>
<field indexed="true" multiValued="false" name="chronic_bronchitis"
type="TextField"/>
<field indexed="true" multiValued="false" name="county"
type="TextField"/>
<field docValues="true" indexed="true" multiValued="false"
name="num_smokers" type="TrieIntField"/>
<field indexed="true" multiValued="false" name="screening_month"
type="TextField"/>
<field indexed="true" multiValued="false" name="diagnosed_emphysema"
type="TextField"/>
<field indexed="true" multiValued="false"
name="diagnosed_other_cancer" type="TextField"/>
<field docValues="true" indexed="true" multiValued="false" name="id"
type="TrieIntField"/>
<field indexed="true" multiValued="false" name="dental_coverage"
type="TextField"/>
<field indexed="true" multiValued="false" name="health_status"
type="TextField"/>
<field docValues="true" indexed="true" multiValued="false"
name="monthly_income_total" type="TrieIntField"/>
<field indexed="true" multiValued="false" name="fish"
type="TextField"/>
<field indexed="true" multiValued="false" name="dog"
type="TextField"/>
<field indexed="true" multiValued="false" name="asthma"
type="TextField"/>
<field indexed="true" multiValued="false" name="ethnicity"
type="TextField"/>
<field docValues="true" indexed="true" multiValued="false"
name="age" type="TrieIntField"/>
<field indexed="true" multiValued="false" name="diagnosed_asthma"
type="TextField"/>
<field indexed="true" multiValued="false" name="race_ethnicity"
type="TextField"/>
<field indexed="true" multiValued="false"
name="diagnosed_congestive_heart_failure" type="TextField"/>
<field docValues="true" indexed="true" multiValued="false"
name="family_size" type="TrieIntField"/>
<field indexed="true" multiValued="false" name="race"
type="TextField"/>
<field indexed="true" multiValued="false" name="thyroid_disease"
type="TextField"/>
<field indexed="true" multiValued="false" name="bronchitis"
type="TextField"/>
<field docValues="true" indexed="true" multiValued="false"
name="household_size" type="TrieIntField"/>
```

```
<field indexed="true" multiValued="false" name="cat"
type="TextField"/>
<field indexed="true" multiValued="false" name="diagnosed_goiter"
type="TextField"/>
<field indexed="true" multiValued="false"
name="diagnosed_skin_cancer" type="TextField"/>
<field indexed="true" multiValued="false" name="fips"
type="TextField"/>
</fields>
<uniqueKey>(id,age)</uniqueKey>
</schema>
```

## Alternate ways to view

Other ways to view the search index schema and config in XML:

- dsetool

View the pending (uploaded) or active (in use) schema or config.

```
dsetool get_core_config (page 816)
dsetool get_core_schema (page 818)
```

- Solr Admin

View only the last uploaded (pending) resource.

## Customizing the search index schema

Customizing the search schema that defines the relationship between data in a table and a search index.

A search schema defines the relationship between data in a table and a search index. The schema identifies the columns to index and maps column names to Apache Solr™ [types](#).

### Schema defaults

DSE Search automatically maps the CQL column type to the corresponding Solr field type, defines the field type analyzer and filtering classes, and sets the DocValue.

**Tip:** If required, modify the schema using the CQL-Solr type compatibility matrix.

### Table and schema definition

Fields with `indexed="true"` are indexed and stored as secondary files in Lucene so that the fields are searchable. The indexed fields are stored in the database, not in Lucene, with the exception of copy fields. Copy field destinations are not stored in the database.

### Sample schema

The following example from [Querying CQL collections](#) uses a simple primary key. The schema version attribute is the Solr version number for the schema syntax and semantics. In this example, `version="1.5"`.

```
<schema name="my_search_demo" version="1.5">
<types>
```

```

<fieldType class="solr.StrField" multiValued="true"
name="StrCollectionField"/>
<fieldType name="string" class="solr.StrField"/>
<fieldType name="text" class="solr.TextField"/>
<fieldType class="solr.TextField" name="textcollection"
multiValued="true">
 <analyzer>
 <tokenizer class="solr.StandardTokenizerFactory"/>
 </analyzer>
</fieldType>
</types>
<fields>
 <field name="id" type="string" indexed="true"/>
 <field name="quotes" type="textcollection" indexed="true"/>
 <field name="name" type="text" indexed="true"/>
 <field name="title" type="text" indexed="true"/>
</fields>
<defaultSearchField>quotes</defaultSearchField>
<uniqueKey>id</uniqueKey>
</schema>

```

DSE Search indexes the id, quotes, name, and title fields.

## Mapping CQL primary keys and Solr unique keys

DSE Search supports [CQL tables](#) using simple or [compound primary keys](#).

If the field is a compound primary key or [Defining a multi-column partition key](#) column in the database, the unique key value is enclosed parentheses. The schema for this kind of table requires a different syntax than the simple primary key:

- List each compound primary key column that appears in the CQL table in the schema as a field, just like any other column.
- Declare the unique key using the key columns enclosed in parentheses.
- Order the keys in the uniqueKey element as the keys are ordered in the CQL table.
- When using composite partition keys, do not include the extra set of parentheses in the uniqueKey.

## Changing auto-generated search index settings

Customize the default settings for auto-generated search indexes using dsetool.

Using dsetool, you can customize the default settings for auto-generated search indexes by providing a YAML-formatted file with these options:

### **auto\_soft\_commit\_max\_time:ms**

The maximum auto soft commit time in milliseconds.

### **default\_query\_field:field**

The query field to use when no field is specified in queries.

### **distributed=( true | false )**

Whether to distribute and apply the operation to all nodes in the local datacenter.

- True applies the operation to all nodes in the local datacenter.

- False applies the operation only to the node it was sent to. False works only when recovery=true.

Default: true

**Warning:** Distributing a re-index to an entire datacenter degrades performance severely in that datacenter.

#### **enable\_string\_copy\_fields:( true | false )**

Whether to generate non-stored string copy fields for non-key text fields, so that you can have text both tokenized or non tokenized.

Default: false

#### **exclude\_columns: col1, col2, col3, ...**

A comma-separated (CSV) list of columns to exclude.

#### **generate\_DocValues\_for\_fields:( \* | field1, field2, ... )**

The fields to automatically configure DocValues in the generated search index schema. Specify "\*" to add all possible fields:

```
generate_DocValues_for_fields: '*'
```

or specify a comma-separated list of fields, for example:

```
generate_DocValues_for_fields: uuidfield, bigintfield
```

Due to [SOLR-7264](#), setting docValues to true on a boolean field in the Solr schema does not work. A workaround for boolean docValues is to use 0 and 1 with a TriIntField.

#### **generateResources=( true | false )**

Whether to automatically generate search index resources based on the existing CQL table metadata. Cannot be used with schema= and solrconfig=.

Valid values:

- true - Automatically generate search index schema and configuration resources if resources do not already exist. If resources exist,
- false - Default. Do not automatically generate search index resources.

#### **include\_columns=col1, col2, col3, ...**

A comma-separated (CSV) list of columns to include. Empty = includes all columns.

#### **index\_merge\_factor:segments**

How many segments of equal size to build before merging them into a single segment.

#### **index\_ram\_buffer\_size=MB**

The index ram buffer size in megabytes (MB).

#### **lenient=( true | false )**

Ignore non-supported type columns and continue to generate resources, instead of erroring out when non-supported type columns are encountered. Default: false

#### **resource\_generation\_profiles**

To minimize index size, specify a CSV list of profiles to apply while generating resources.

**Table 9: Resource generation profiles**

Profile name	Description
spaceSavingAll	Applies all options: spaceSavingNoTextfield, spaceSavingNoJoin, and spaceSavingSlowTriePrecision.
spaceSavingNoTextfield	No TextFields. Use StringField instead.
spaceSavingNoJoin	Do not index a hidden primary key field. Prevents joins across cores.
spaceSavingSlowTriePrecision	Set trie fields precisionStep to '0', allowing for greater space saving but slower querying.

**Note:** Using spaceSavings profiles disables auto generation of DocValues.

For example:

```
resource_generation_profiles: spaceSavingNoTextfield,
spaceSavingSlowTriePrecision
```

#### rt=true

Whether to enable live indexing to increase indexing throughput. Enable live indexing on only one search index per cluster.

```
rt=true
```

## CQL index management command examples

For example:

```
CREATE SEARCH INDEX CONFIG ON wiki.solr SET
defaultQueryField='last_name';
```

See [About search commands](#).

## Using dsetool

Customize the search index config with YAML-formatted files

Create a `config.yaml` file that lists the following options to customize the config and schema files:

```
default_query_field: name
auto_soft_commit_max_time: 1000
generate_DocValues_for_fields: '*'
enable_string_copy_fields: false
```

Use the `dsetool` command to generate the search index with these options to customize the config and schema generation. Use `coreOptions` to specify the `config.yaml` file:

```
dsetool create_core demo.health_data coreOptions=config.yaml
```

Customize the search index with options inline

Use the dsetool command to generate the search index and customize the schema generation. Use coreOptions to turn on live indexing (also called RT):

```
dsetool create_core udt_ks.users generateResources=true reindex=true
coreOptions=rt.yaml
```

You can verify that DSE Search created the solrconfig and schema by reading core resources using dsetool.

Enable encryption for a new search index

Specify the class for directoryFactory to `solr.EncryptedFSDirectoryFactory` with `coreOptionsInline`:

```
dsetool create_core keyspace_name.table_name generateResources=true
coreOptionsInline="directory_factory_class:solr.EncryptedFSDirectoryFactory"
```

## Configuring additional search components

You can add search components and a handler for the search components. For example, add spell check.

To configure additional search components, add the search component and define it in the handler.

For example, to add the Java spelling checking package JaSpell:

```
<searchComponent class="solr.SpellCheckComponent"
name="suggest_jaspell">
 <lst name="spellchecker">
 <str name="name">suggest</str>
 <str name="classname">org.apache.solr.spelling.suggest.Suggester</
str>
 <str
name="lookupImpl">org.apache.solr.spelling.suggest.jaspell.JaspellLookup</
str>
 <str name="field">suggest</str>
 <str name="storeDir">suggest</str>
 <str name="buildOnCommit">true</str>
 <float name="threshold">0.0</float>
 </lst>
</searchComponent>
```

Configure the parameters for the request handler:

```
<requestHandler class="org.apache.solr.handler.component.SearchHandler"
name="/suggest">
 <lst name="defaults">
 <str name="spellcheck">true</str>
 <str name="spellcheck.dictionary">suggest</str>
 <str name="spellcheck.collate">true</str>
 <str name="spellcheck.extendedResults">true</str>
 </lst>
 <arr name="last-components">
 <str>suggest_jaspell</str>
```

```
</arr>
</requestHandler>
```

## DSE Search operations

You can run DSE Search on one or more nodes. Typical operations including configuration of nodes, policies, query routing, balancing loads, and communications.

You can run DSE Search on one or more nodes. Typical operations including configuration of nodes, policies, query routing, balancing loads, and communications.

### DSE Search initial data migration

Best practices and guidelines for loading data into DSE Search.

Best practices and guidelines for loading data into DSE Search.

When you initially load data into DataStax Enterprise (DSE) resource contention requires planning to ensure performance.

- DSE is performant when writing data.
- Apache Solr™ is resource intensive when creating a search index.

These two activities compete for resources, so proper resource allocation is critical to maximize efficiency for initial data load.

### Recommendations

- For maximum throughput, store the [search index data](#) and DataStax Enterprise (Cassandra) data on separate physical disks.  
If you are unable to use separate disks, DataStax recommends that SSDs have a minimum of 500 MB/s read/write speeds (bandwidth).
- Enable OpsCenter 6.1 repair service.

Also see [memory recommendations](#) in the planning guide.

### Initial bulk loading

DataStax recommends following this high-level procedure:

1. Install DSE and configure nodes for search workloads.
2. Use the CQL [CREATE SEARCH INDEX](#) command to create search indexes.
3. [Tune the index](#) for maximum indexing throughput.
4. Load data into the database using best practices for data loading. For example, load data with the driver with the consistency level at LOCAL\_ONE (CL.LOCAL\_ONE) and a sufficiently high write timeout.

After data loading is completed, there might be lag time because indexing is asynchronous.

5. Verify the indexing [QueueSize \(page 346\)](#) with the IndexPool MBean. After the index queue size has receded, run this CQL query to verify that the number of records is as expected:

```
SELECT count(*) FROM ks.table WHERE solr_query = '*:*';
```

New data is automatically indexed.

## Troubleshooting

If the record count does not stabilize:

- If dropped mutations exist in the [nodetool tpstats](#) output for some nodes, and OpsCenter repair service is not enabled, run [manual repair](#) on those nodes.
- If dropped mutations do not exist, check the system.log and the [Solr validation log](#) for indexing errors.

## Verifying indexing status

Steps to check the indexing status using dsetool, the Core Admin, or logs.

You can check the indexing status using dsetool, the Core Admin, or the logs.

## Examples

To view the indexing status for the local node:

```
dsetool core_indexing_status demo.health_data
```

The results are displayed:

```
[demo.health_data]: INDEXING, 38% complete, ETA 452303 milliseconds (7 minutes 32 seconds)
```

To view the indexing status for a search index on a specified node:

```
dsetool -h 200.192.10.11 core_indexing_status demo.health_data
```

To view indexing status of all search indexes in the data center:

```
dsetool -h 200.192.10.11 core_indexing_status --all
```

## Checking the indexing status using the Core Admin

To check the indexing status, open the Solr Admin and click **Core Admin**.

## Checking the indexing status using the logs

You can also check the logs to get the indexing status. For example, you can check information about the plugin initializer:

```
INDEXING / REINDEXING -
INFO SolrSecondaryIndex plugin initializer. 2013-08-26 19:25:43,347
 SolrSecondaryIndex.java (line 403) Reindexing 439171 keys for core
 wiki.solr
```

Or you can check the SecondaryIndexManager.java information:

```
INFO Thread-38 2013-08-26 19:31:28,498 SecondaryIndexManager.java
 (line 136) Submitting index build of wiki.solr for data in
 SSTableReader(path='/mnt/cassandra/data/wiki/solr/wiki-solr-ic-5-
 Data.db'), SSTableReader(path='/mnt/cassandra/data/wiki/solr/wiki-solr-
 ic-6-Data.db')

FINISH INDEXING -
INFO Thread-38 2013-08-26 19:38:10,701 SecondaryIndexManager.java (line
 156) Index build of wiki.solr complete
```

## Restoring a search node from backup

Steps to restore a search node from backup.

Reload data and rebuild the indexes as we load data.

1. Use the DataStax Enterprise [restore steps](#) with indexing enabled and let the data write as data is coming in.

Use the OpsCenter [Backup Service](#).

2. Follow the steps in [DSE Search initial data migration \(page 335\)](#).

## Monitoring DSE Search

Information about metrics MBeans for DSE Search.

DataStax Enterprise exposes a number of statistics and management operations via Java Management Extensions (JMX). JMX is a Java technology that supplies tools for managing and monitoring Java applications and services. Any statistic or operation that a Java application has exposed as an MBean can then be monitored or manipulated using JMX.

JMX is a Java technology that supplies tools for managing and monitoring Java applications and services. Any statistic or operation that a Java application has exposed as an MBean can then be monitored or manipulated using JMX.

Metrics MBeans are used for troubleshooting, tuning performance and consistency issues.

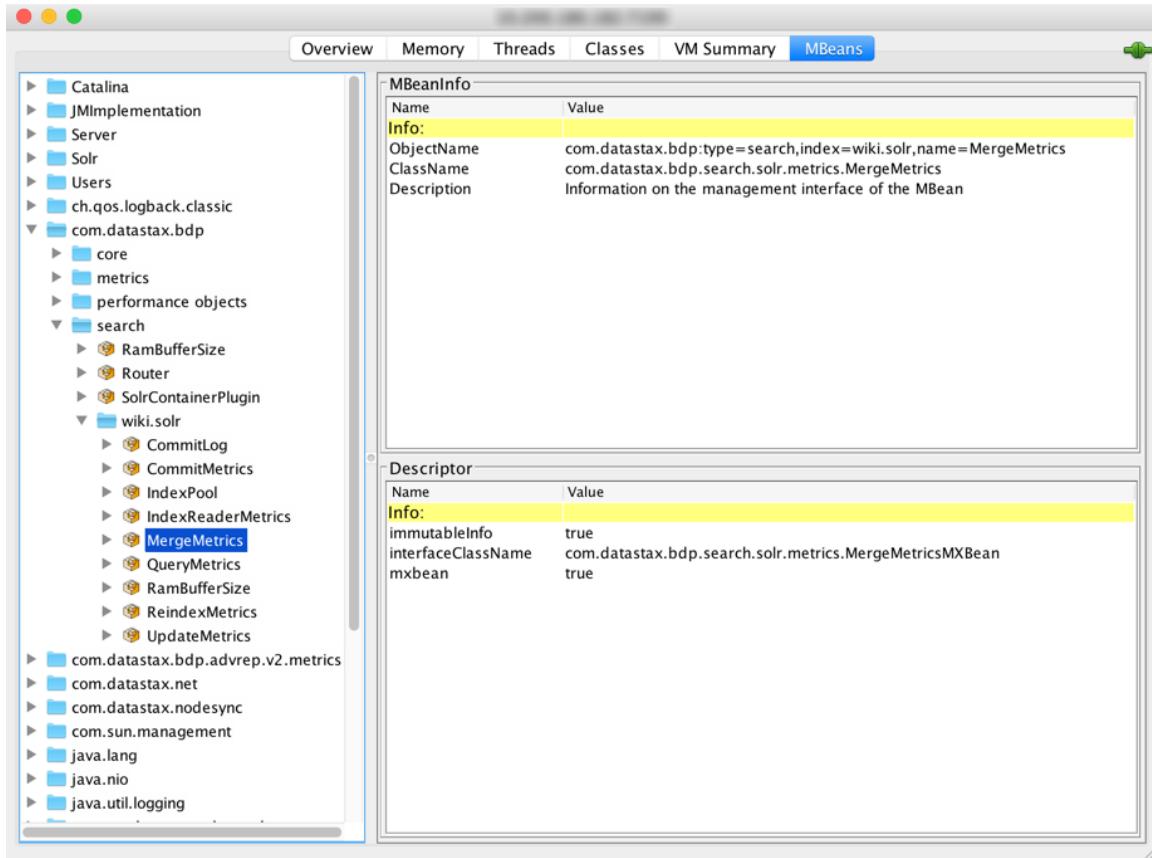
Use the [IndexPool JMX MBean \(page 345\)](#) to view the progress of indexing tasks.

The following paths identify the MBeans:

```
type=search,index=search_index,name=CommitMetrics (page 338)
type=search,index=search_index,name=MergeMetrics (page 346)
type=search,index=search_index,name=QueryMetrics (page 345)
type=search,index=search_index,name=UpdateMetrics (page 349)
type=search,index=search_index,name=IndexPool (page 345)
```

where search\_index is the name of the search index that is referenced by the metrics.

For example, the following figure shows the com.datastax.bdp merge metrics MBean in JConsole. The wiki.solr search index under search is expanded.



## Using MBeans to evaluate performance

Example steps to use the MBeans on Linux to obtain information about performance on a DSE Search node.

To use the MBeans on Linux to obtain information about performance on a DSE Search node:

1. Start a single DSE Search node.
2. Find the process ID (PID) of the DSE Search node:

```
pgrep -f dse
```

```
368
668
45706
```

**Tip:** To verify the PID:

```
pgrep -f cassandra
```

```
pgrep -f cassandra
45706
```

PID 45706 is the correct PID for dse and cassandra.

3. Start JConsole using the PID of the DSE Search node:

```
jconsole 45706
```

4. In JConsole, connect to a DSE Search node. For example, connect to the Local Process com.datastax.bdp.DseModule.

## MBeans search demo

Using MBeans to evaluate performance for search demo.

Use MBeans to evaluate performance for the search demo.

1. Complete the [steps to use the MBeans \(page 338\)](#) on Linux to obtain information about performance.
2. Change to the `demos` directory.

The default location of the `demos` directory depends on the type of installation:

- Package installations: `/usr/share/dse/demos`
- Tarball installations: `installation_location/demos`

3. Make `demos/solr_stress` your current directory.
4. Execute this script to create the schema:

```
pushd resources/schema &&
./create-schema.sh
```

where the script options are:

### CQL table creation options

`--ssl` use SSL for table creation over cqlsh

### Solr HTTP options

`-e CA_CERT_FILE` use HTTPS with the provided CA certificate  
`-E CLIENT_CERT_FILE` use the provided client certificate  
`-h HOST` hostname or IP for Solr HTTP requests  
`-a` enable Kerberos  
`-u USERNAME` Kerberos username  
`-p PASSWORD` Kerberos password

The script creates the schema and posts the `solrconfig.xml` and `schema.xml` files to these locations:

- `http://localhost:8983/solr/resource/demo.solr/solrconfig.xml`
- `http://localhost:8983/solr/resource/demo.solr/schema.xml`

The script then creates the search index by posting to the following location:

- `http://localhost:8983/solr/admin/cores?action=CREATE&name=demo.solr`

You can override the script defaults by specifying command line parameters:

```
-x schemafile.xml -t tableCreationFile.cql -r solrCofgFile.xml -
k solrCore
```

**5. Execute this script to run the benchmark:**

```
./run-benchmark.sh [--clients=clients_count] [--loops=loops_count] [--fetch=fetch_size] [--solrCore=solr_core] [--testData=test_data_file] [--url=url1,url2,url3,...] [--qps=qps] [--stats=true|false] [--seed=seed_value]
```

where the script options are:

**--clients**

The number of client threads to create.

Default: 1

**--loops**

The number of times the commands list gets executed if running sequentially or the number of commands to run if running randomly.

Default: 1

**--fetch**

Fetch size for CQL pagination (disabled by default). Only the first page is retrieved.

**--solrCore**

Search index name to run the benchmark against.

**--testData**

Name of the file that contains the test data.

**--seed**

Value to set the random generator seed to.

**--qps**

Maximum number of queries per second allowed.

**--stats**

Specifies whether to gather statics during runtime and create a csv file with the recorded values.

Default: false

**--url**

A comma delimited list of servers to run the benchmark against. For example: --url=http://localhost:8983,http://192.168.10.45:8983,http://192.168.10.46:8983

Default: http://localhost:8983

The demo creates a Search index named demo.solr and indexes 50,000 documents.

Example CQL commands:

```
./run-benchmark.sh --url=http://localhost:8983 --
testData=resources/testCqlQuery.txt --solrCore=demo.solr
```

```
./run-benchmark.sh --url=http://localhost:8983 --
testData=resources/testCqlWrite.txt --solrCore=demo.solr
```

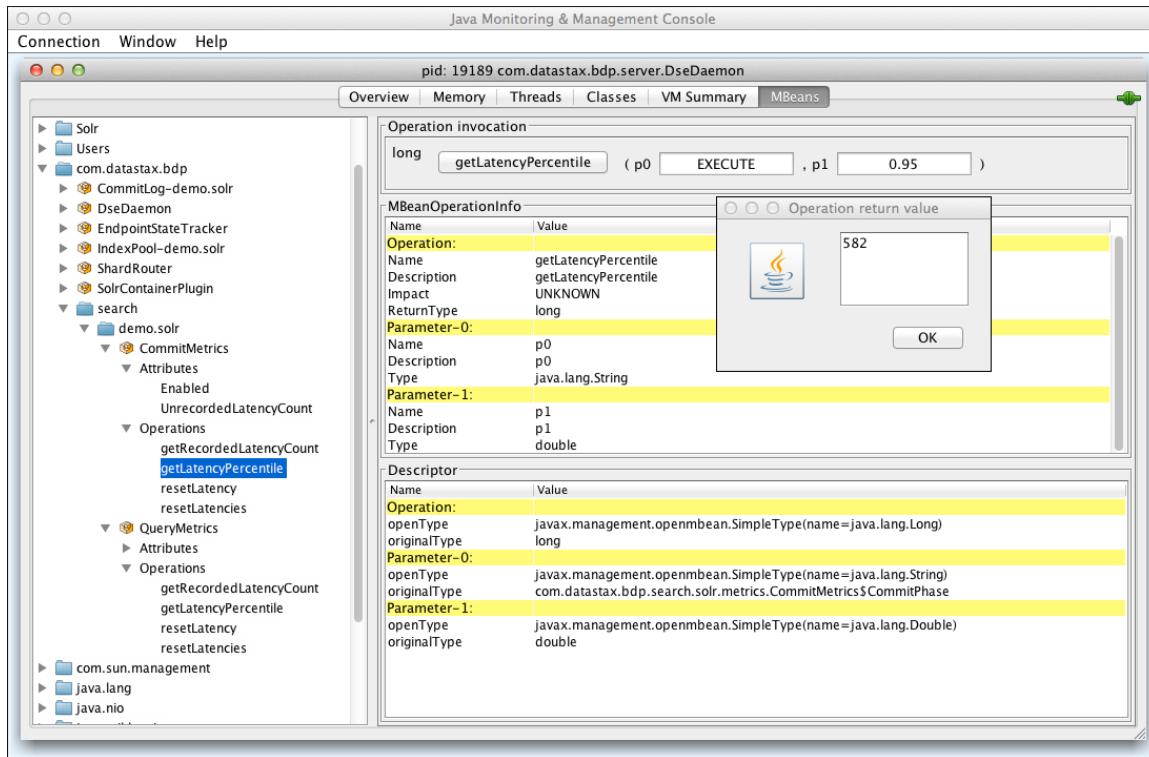
See `/demos/solr_stress/README.txt` for execution modes and sample script commands.

- In JConsole, expand `com.datastax.bdp#search#demo.solr` to view the MBeans.

The CommitMetrics and QueryMetrics MBeans are present.

- In JConsole, in **Search#demo.solr#CommitMetrics#Operations#  
getLatencyPercentile**, type `EXECUTE` in the `p0` text entry box and `0.95` in the `p1` text entry box. Click the **getLatencyPercentile** button.

The Operation return value, 582 microseconds, appears:



## Commit metrics MBean

The commit metrics MBean used for troubleshooting index performance and resolving data consistency issues that are caused by asynchronous commits between different index replicas.

The commit metrics MBean used for troubleshooting index performance as well as data consistency issues caused by asynchronous commits between different index replicas. This MBean is also useful for fine-tuning indexing **back pressure**. The commit metrics MBean records the amount of time that is spent to execute two **main phases** ([page 342](#)) of a commit operation on the index.

### Main operational phases

The main phases of a commit operation on the index are:

#### FLUSH

Comprising the time spent by flushing the async indexing queue.

## EXECUTE

Comprising the time spent by actually executing the commit on the index.

Commit metrics MBean operations use the FLUSH and EXECUTE commit phase names.

## Commit metrics MBean set operations

The commit metrics MBean measures latency in microseconds. You can set these commit metrics MBean operations.

- `setEnabled(boolean enabled)`  
Enables/disables metrics recording. Enabled by default.
- `resetLatency(String phase)`  
Resets latency metrics for the given commit phase.
- `resetLatencies()`  
Resets all latency metrics.

## Commit metrics MBean get operations

The commit metrics MBean measures latency in microseconds. You can get these commit metrics MBean operations:

- `isEnabled()`  
Checks that metrics recording is enabled.
- `getLatencyPercentile(String phase, double percentile)`  
Gets a commit latency percentile by its phase.
- `getRecordedLatencyCount(String phase)`  
Gets the total count of recorded latency metrics by its commit phase.
- `getUnrecordedLatencyCount()`  
Gets the total count of unrecorded latency values due to exceeding the maximum tracked latency, which is 10 minutes.

## EndpointStateTracker MBean

Observes state of other nodes using the gossip. Use to blacklist a node and other operations that include node health, workload, and status.

The EndpointStateTracker MBean is identified by the following path:

```
com.datastax.bdp:name=EndpointStateTracker,type=core,name=EndpointStateTracker
```

This MBean has an attribute to blacklist a node and operations that include node health, workload, and status.

## Attributes

### Blacklisted

Boolean attribute to remove a node from the list of searchable nodes while it's being diagnosed, repaired, reindexed, and verified as healthy.

Sets blacklisted status that is gossiped around the cluster and used during the replica selection phase of distributed search queries.

- true - forcibly rank this node below active nodes for distributed search queries
- false - make this node eligible for selection during distributed search queries.

## **ServerID**

String that identifies the server ID of a local node.

## **Operations**

The arguments for the operations are strings for the IP address, except where noted.

- **getNodeHealth**  
Gets the node health for a given IP address.
- **getWorkloads**  
Gets the workload type of a remote endpoint. Persists between restarts.
- **getDatacenter**  
Gets the datacenter for the given endpoint, basing on the information from the Gossiper or information saved in the Cassandra system table. Persists between restarts.
- **getActiveStatus**  
Gets active status for the given endpoint. A node is active if the server and required plugins are all started. Computed at runtime.
- **getServerId**  
Gets the DSE multi-instance server ID for a remote endpoint. Persists between restarts.
- **getCoreIndexingStatus**  
Gets the dynamic indexing status (INDEXING, FINISHED, or FAILED) of the search index of a given endpoint. Computed at runtime.
- **getRing**  
Takes a single argument, the keyspace. Returns information about every node in the cluster. Computed at runtime.
- **getIsGraphServer**  
Returns true if graph is enabled for the given endpoint. Computed at runtime.
- **vnodesEnabled**  
Returns true if vnodes are enabled. Computed at runtime.
- **getBlacklistedStatus**

Is node removed from node from the list of searchable nodes. Persists between restarts.

**Note:** The gossip state is persisted locally. Set the Blacklisted attribute to remove the blacklisting status.

You can also use the nodetool sjk command to blacklist a node.

## IndexPool MBean

Exposes metrics around the progress of indexing tasks. Useful for controlling task submission and flush.

The IndexPool MBean exposes metrics around the progress of indexing tasks as they move through the pipeline, and provides a mechanism to tweak the flushing, concurrency, and back-pressure behavior of a core indexing thread pool.

The index pool MBean is useful for controlling task submission and flush with these properties:

### Configurable concurrency

The maximum number of concurrent workers is predefined at construction time. The actual concurrency can be dynamically configured between 1 (synchronous execution) and the given max concurrency.

### Flow control via back pressure

To reduce memory consumption in case of fast producers, back pressure throttles incoming tasks. Back-pressure is applied directly as a result of the size of the global RAM buffer.

## Path

The index pool MBean is identified by the following path:

```
com.datastax.bdp.search.keyspace_name.table_name.IndexPool
```

where:

- search is the Mbean type
- keyspace\_name.table\_name is the search index (core) that the metrics reference
- IndexPool is the MBean name

For example:

### IndexPool MBean attributes that you can modify

The attributes are effective only until the node is restarted. To make the change permanent, you must change the corresponding option in dse.yaml.

#### FlushMaxTime

The maximum time, in milliseconds, to wait before flushing asynchronous index updates, which occurs at DSE Search commit time or at database flush time. In dse.yaml (in minutes): [flush\\_max\\_time\\_per\\_core \(page 122\)](#).

#### Concurrency

The maximum number of concurrent asynchronous indexing threads.

## IndexPool MBean view-only attributes

You can get the following MBean operations:

### BackPressurePauseNanos

Get the average back pressure pause.

### IncomingRate

Get the 1-minute rate of ingested tasks per second.

### MaxConcurrency

Get the predefined max concurrency level.

### OutgoingRate

Get the 1-minute rate of processed tasks per second.

### ProcessedTasks

Get the total number of processed tasks for all workers.

### QueueSize

Get the current size of each processing queues.

### TaskProcessingTimeNanos

Get the last processing time for all workers. Could be 0 in case the clock resolution is too coarse.

### Throughput

The 1-minute rate of work throughput per second.

### TotalQueueSize

Get the total size of all processing queues.

## Merge metrics MBean

The merge metrics MBean used for tuning merge operations.

The merge metrics MBean tracks the time that Apache Solr™/Apache Lucene® spend on merging segments that accumulate on disk. Segments are files that store new documents and are a self-contained index. When data is deleted, Lucene does not remove it, but instead marks documents as deleted. For example, during the merging process, Lucene copies the data from 100 segment files into a single new file. Documents that are marked deleted are not included in the new segment files. Next, Lucene removes the 100 old segment files, and the single new file holds the index on disk.

After segments are written to disk, they are immutable.

In a high throughput environment, a single segment file is rare. Typically, there are several files and Lucene runs the merge metric operation concurrently with inserts and updates of the data using a merge policy and merge schedule.

Merge operations are costly and can impact the performance of CQL queries. A huge merge operation can cause a sudden increase in query execution time.

## Main operational phases

The main phases of a merge operation on the index are:

### INIT

How long it takes to initialize the merge process.

### EXECUTE

How long it takes to execute the merge process.

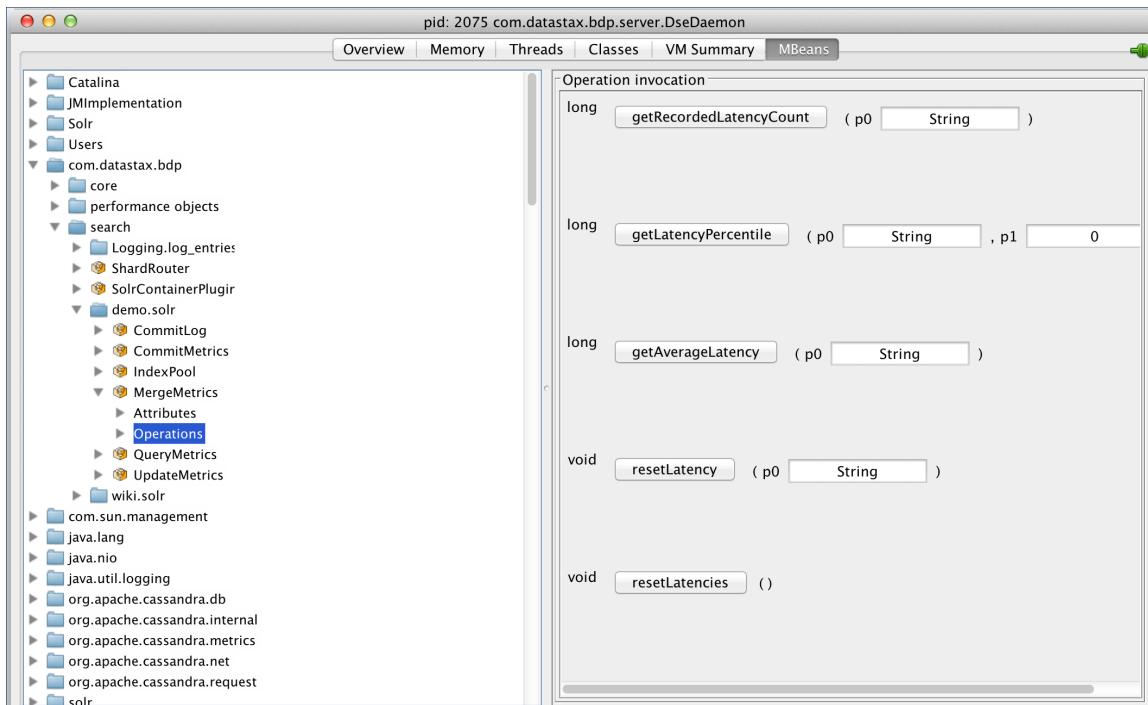
### WARM

How long it takes to warm up segments to speed up cold queries.

WARM time is part of EXECUTE time: EXECUTE time = WARM time + other operations. For example, if the EXECUTE phase is 340 ms, and the WARM phase is 120 ms, then other operations account for the remaining 220 ms.

The merge metrics MBean operations are:

- `getRecordedLatencyCount`
- `getLatencyPercentile`
- `getAverageLatency`
- `resetLatency`
- `resetLatencies`



To get merge metrics, insert one of the phases of the merge operation and select a phase, such as EXECUTE:

## Query metrics MBean

Troubleshoot query performance, tune DSE Search configuration and tune server resources.

Use the query metrics MBean to troubleshoot query performance; tune DSE Search configuration, such as the search index schema and caches; and tune server resources, such as the JVM heap. The query metrics MBean records the amount of time spent to run several main [phases \(page 348\)](#) of a distributed query on the search index.

The query metrics MBean measures latency in microseconds.

To group by query, provide an additional `query.name` parameter. For example, for a search index named `demo.solr` with an indexed field named `type`, use this URL to provide the additional `query.name` parameter:

```
http://localhost:8983/solr/demo.solr/select/?
q=type:1&query.name=myquery
```

All metrics collected under a given query name are recorded and retrieved separately. If a query name is not provided, all metrics are recorded together.

## Main operational phases

The main phases of a distributed query operation are:

### ENQUEUE

Comprises the time spent by a query request waiting for a thread to execute.

### EXECUTE

Comprises the time spent by a single shard to execute the actual index query.

This value is computed on the local node executing the shard query.

### RETRIEVE

Comprises the time spent by a node to retrieve a single row from the database.

This value will be computed on the local node hosting the requested data.

### COORDINATE

Comprises the total amount of time spent by the coordinator node to distribute the query and gather/process results from shards. This value is computed only on query coordinator nodes. Includes RETRIEVE and EXECUTE in the total.

### TOTAL

Comprises the total server-side time for a search query. Includes COORDINATE and ENQUEUE in the total.

## Query metrics MBean set operations

Operations are:

- `setEnabled(boolean enabled)`  
Enables/disables metrics recording. Enabled by default.
- `isEnabled()`  
Checks if metrics recording is enabled.
- `getLatencyPercentile(String phase, String query, double percentile)`  
Gets a query latency percentile by its query name, which is optional and can be null, and phase.
- `getRecordedLatencyCount(String phase, String query)`  
Gets the total count of recorded latency metrics by its query name, which is optional and can be null, and phase.
- `getUnrecordedLatencyCount()`  
Gets the total count of unrecorded latency values due to exceeding the maximum tracked latency, which is 10 minutes.
- `resetLatency(String query)`  
Resets latency metrics for the given query name, which is optional and can be null.

- `resetLatencies()`  
Resets all latency metrics.

## Query metrics MBean attributes

Attributes are:

### Enabled

Indicates whether metrics recording is enabled or disabled.

### EnqueuedRequestCount

Indicates the number of client requests that are currently waiting for a query thread.

If the value of the EnqueuedRequestCount MBean increases, or stabilizes above zero for a prolonged period, then DSE Search has reached a point of maximum throughput, where additional load will only increase latencies.

To access this attribute, use the `-f` option with the `nodetool sjk` command, as shown in the following example.

```
nodetool sjk -p 7199 mx -mg \
-b
com.datastax.bdp:type=search,index=demo.solr,name=QueryMetrics
\
-f EnqueuedRequestCount
```

### UnrecordedLatencyCount

The total count of unrecorded latency values due to exceeding the maximum tracked latency, which is 10 minutes.

## Update metrics MBean

The update metrics MBean is useful for tuning indexing performance.

This MBean records the amount of time spent to execute an index update, split by the following main phases:

### QUEUE

Updated for reindexing only. The time spent by the index update task into the index pool.

### PREPARE

The time spent preparing the actual index update.

### EXECUTE

The time spent to actually execute the index update on Apache Lucene®.

Use the update metrics MBean to tune all factors that impact indexing performance, such as [back pressure](#), indexing threads, RAM buffer size, and merge factor.

## MBean operations

The following MBean operations are provided:

- `setEnabled(boolean enabled)`

Enables/disables metrics recording (enabled by default).

- `isEnabled()`  
Checks if metrics recording is enabled.
- `getLatencyPercentile(String phase, double percentile)`  
Gets a commit latency percentile by its phase.
- `getRecordedLatencyCount(String phase)`  
Gets the total count of recorded latency metrics by its phase.
- `getUnrecordedLatencyCount()`  
Gets the total count of unrecorded latency values, because exceeding the max tracked latency.
- `resetLatency(String phase)`  
Resets latency metrics for the given phase.
- `resetLatencies()`  
Resets all latency metrics.

The maximum tracked latency is 10 minutes. Latency values are in microseconds.

## Uploading the search index schema and config

Steps to upload a search index schema and config.

After generating or changing the search index [schema \(page 315\)](#) and [configuration \(page 309\)](#), use dsetool to upload to a DSE Search node to create a search index. You can also post additional resource files.

You can configure the maximum resource file size or disable resource upload with the [DSE Search resource upload limit \(page 121\)](#) option in `dse.yaml`.

**Note:** Using custom resources is not supported by the CQL [CREATE SEARCH INDEX](#) command.

Index resources are stored internally in the database, not in the file system. The schema and configuration resources are persisted in the `solr_admin.solr_resources` database table.

1. Write the schema:

```
dsetool write_resource keyspace.table name=schema.xml
file=schemaFile.xml
```

2. Post the configuration file:

```
dsetool write_resource keyspace.table name=solrconfig.xml
file=solrconfigFile.xml
```

3. Post any other resources that you might need.

```
dsetool write_resource keyspace.table name=ResourceFile.xml
file=schemaFile.xml
```

You can specify a path for the resource file:

```
dsetool write_resource keyspace.table name=ResourceFile.xml
file=myPath1/myPath2/schemaFile.xml
```

4. To verify the resources after they are posted:

For example:

```
dsetool read_resource keyspace.table name=ResourceFile.xml
file=myPath1/myPath2/schemaFile.xml
```

## Solr interfaces

### Accessing search indexes from Solr Admin UI (deprecated)

Permissions required to view search indexes from the Solr Admin UI in an authorization enabled environment.

When DataStax Enterprise authorization is enabled, access to search indexes (cores) is restricted from the Solr Admin UI. You must grant permissions to roles of Solr Admin UI users for HTTP operations.

Table	Required permissions	Operation
solr_admin.solr_resources	SELECT	Read a resource
solr_admin.solr_resources	MODIFY	Write a resource
core table	ALTER	Stop core reindex
core table	SELECT	Query core and all remaining admin query operations on core
core table	MODIFY	Commit and delete

**Tip:** Permissions are inherited. Granting permissions on a keyspace allows users with that role to access all tables in the keyspace.

## Examples

To grant permission to read resources:

```
GRANT SELECT ON solr_admin.solr_resources
```

```
TO role_name;
```

## Configuring the Solr library path

Workaround for DSE Search failure to find files in directories that are defined by the <lib> property.

The location for library files in DataStax Enterprise is not the same location as open source Apache Solr™. Contrary to the examples shown in the `solrconfig.xml` file that indicate support for relative paths, DSE Search does not support the relative path values that are set for the <lib> property and cannot find files in directories that are defined by the <lib> property. The workaround is to place custom code or Solr contrib modules in the Solr library directories.

The default Solr library path location depends on the type of installation:

- Package installations: `/usr/share/dse/solr/lib`
- Tarball installations: `installation_location/resources/solr/lib`

When the plugin JAR file is not in the directory that is defined by the <lib> property, attempts to deploy custom Solr libraries in DataStax Enterprise fail with `java.lang.ClassNotFoundException` and an error in the `system.log` like this:

```
ERROR [http-8983-exec-5] 2015-12-06 16:32:33,992 CoreContainer.java
 (line 956) Unable to create core: boogle.main
org.apache.solr.common.SolrException: Error loading class
 'com.boogle.search.CustomQParserPlugin'
at org.apache.solr.core.SolrCore.(SolrCore.java:851)
at org.apache.solr.core.SolrCore.(SolrCore.java:640)
at
 com.datastax.bdp.search.solr.core.CassandraCoreContainer.doCreate(CassandraCoreContainer.
at
 com.datastax.bdp.search.solr.core.CassandraCoreContainer.create(CassandraCoreContainer.ja
at
 com.datastax.bdp.search.solr.core.SolrCoreResourceManager.createCore(SolrCoreResourceMana
at
 com.datastax.bdp.search.solr.handler.admin.CassandraCoreAdminHandler.handleCreateAction(C
...
Caused by: org.apache.solr.common.SolrException: Error loading class
 'com.boogle.search.CustomQParserPlugin'
at
 org.apache.solr.core.SolrResourceLoader.findClass(SolrResourceLoader.java:474)
at
 org.apache.solr.core.SolrResourceLoader.findClass(SolrResourceLoader.java:405)
at org.apache.solr.core.SolrCore.createInstance(SolrCore.java:541)
...
Caused by: java.lang.ClassNotFoundException:
 com.boogle.search.CustomQParserPlugin
at java.net.URLClassLoader$1.run(Unknown Source)
at java.net.URLClassLoader$1.run(Unknown Source)
...
```

## Workaround

Using the class in this example with the JAR file name

`com.boogle.search.CustomQParserPlugin-1.0.jar`, follow these steps to get the custom plugin working on all DSE Search nodes.

1. Define the parser in the search index config file:

```
<queryParser name="myCustomQP"
 class="com.boogle.search.CustomQParserPlugin"/>
```

2. Place custom code or Solr contrib modules in the Solr library directories.

3. Deploy the JAR file on all DSE Search nodes in the cluster in the appropriate `lib/` directory.

For example, package installations: `/usr/share/dse/solr/lib/com.boogle.search.CustomQParserPlugin-1.0.jar`

4. Reload the search index with the new configuration.

## Using the Solr HTTP API

Use the Solr HTTP API to query data indexed in DSE Search.

You can use the Solr HTTP API to query data indexed in DSE Search.

**Note:** [Solr restrictions \(page 307\)](#) apply to queries.

HTTP search queries use local/internal reads and do not actuate read repair.

With only the HTTP API, define the default number of rows in the `solrconfig.xml` file:

```
<requestHandler name="search" class="solr.SearchHandler" default="true">
 <lst name="defaults">
 <int name="rows">10</int>
 </lst>
</requestHandler>
```

### Solr HTTP API example

Assuming you performed the [example of using a collection set](#) to find the titles in the `mykeyspace.mysolr` table that begin with the letters succ in XML, use this URL:

```
http://localhost:8983/solr/mykeyspace.mysolr/select?q=%20title
%3Asucc*&fl=title
```

The response is:

```
<response>
<lst name="responseHeader">
 <int name="status">0</int>
 <int name="QTime">2</int>
<lst name="params">
 <str name="fl">title</str>
 <str name="q">title:Succ*</str>
</lst>
```

```
</lst>
<result name="response" numFound="2" start="0">
 <doc>
 <str name="title">Success</str>
 </doc>
 <doc>
 <str name="title">Success</str>
 </doc>
</result>
</response>
```

## Field transformer (FIT)

Steps to use the field input/output transformer API as an option to the input/output transformer support in Solr.

DataStax Enterprise (DSE) supports using a field input/output transformer (FIT) API.

A field input/output transformer, an alternative for handling update requests, is executed later than a URP at indexing time. See the DataStax Developer Blog post [An Introduction to DSE Field Transformers](#).

**Note:** The DSE custom URP implementation is deprecated.

DSE custom URP provided similar functionality to the Solr URP chain, but appeared as a plugin to Solr. The classic URP is invoked when updating a document using HTTP and the custom URP is invoked when updating a table using DSE. If both classic and custom URPs are configured, the classic version is executed first. The custom URP chain and the FIT API work with CQL and HTTP updates.

Examples are provided for using the [field input/output \(page 354\)](#) transformer API and the deprecated [custom URP \(page 357\)](#).

## Field input/output (FIT) transformer API

Steps to use the field input/output transformer API as an option to the input/output transformer support in Apache Solr.

Use the field input/output transformer API as an option to the input/output transformer support in Apache Solr™. [An Introduction to DSE Field Transformers](#) provides details on the transformer classes.

DSE Search includes the released version of a plugin API for Solr updates and a plugin to the CassandraDocumentReader. The plugin API transforms data from the secondary indexing API before data is submitted. The plugin to the CassandraDocumentReader transforms the results data from the database to DSE Search.

Using the API, applications can tweak a Solr Document before it is mapped and indexed according to the `schema.xml`. The API is a counterpart to the input/output transformer support in Solr.

The field input transformer (FIT) requires:

- `name="dse"`
- A trailing Z for date field values

To use the API:

1. Define the plugin in the top level <config> element in the `solrconfig.xml` for a table (search core).

```
<config>
...
<fieldInputTransformer name="dse" class=
 com.datastax.bdp.cassandra.index.solr.functional.
 BinaryFieldInputTransformer">
</fieldInputTransformer>

<fieldOutputTransformer name="dse" class=
 com.datastax.bdp.cassandra.index.solr.functional.
 BinaryFieldOutputTransformer">
</fieldOutputTransformer>
...
</config>
```

2. Write a transformer class something like this [reference implementation \(page 355\)](#) to tweak the data in some way.
3. Export the class to a JAR file. You must place the JAR file in this location:
  - Tarball installations: `install-location/resources/solr/lib`
  - Package installations: `/usr/share/dse/solr/lib`
 The JAR is added to the CLASSPATH automatically.
4. Test your implementation using something like the reference implementation.

## FIT transformer class examples

Field input and output transformer (FIT) class examples.

The DataStax Developer Blog provides an [introduction to DSE Field Transformers](#).

Here are examples of field input and output transformer (FIT) classes.

### Input transformer example

```
package com.datastax.bdp.search.solr.functional;

import java.io.IOException;

import org.apache.commons.codec.binary.Hex;
import org.apache.commons.lang.StringUtils;
import org.apache.lucene.document.Document;
import org.apache.solr.core.SolrCore;
import org.apache.solr.schema.SchemaField;

import com.datastax.bdp.search.solr.FieldOutputTransformer;
import org.apache.solr.schema.IndexSchema;

public class BinaryFieldInputTransformer extends
FieldInputTransformer
{
```

```

@Override
public boolean evaluate(String field)
{
 return field.equals("binary");
}

@Override
public void addFieldToDocument(SolrCore core,
IndexSchema schema,
String key,
Document doc,
SchemaField fieldInfo,
String fieldValue,
DocumentHelper helper)
throws IOException
{
try
{
byte[] raw = Hex.decodeHex(fieldValue.toCharArray());
byte[] decomp = DSP1493Test.decompress(raw);
String str = new String(decomp, "UTF-8");
String[] arr = StringUtils.split(str, ",");
String binary_name = arr[0];
String binary_type = arr[1];
String binary_title = arr[2];

SchemaField binaryNameField =
core.getSchema().getFieldOrNull("binary_name");
SchemaField binaryTypeField =
core.getSchema().getFieldOrNull("binary_type");
SchemaField binaryTitleField =
core.getSchema().getFieldOrNull("binary_title");

helper.addFieldToDocument(core, core.getSchema(), key, doc,
binaryNameField, binary_name);
helper.addFieldToDocument(core, core.getSchema(), key, doc,
binaryTypeField, binary_type);
helper.addFieldToDocument(core, core.getSchema(), key, doc,
binaryTitleField, binary_title);
}
catch (Exception ex)
{
throw new RuntimeException(ex);
}
}
}
}

```

## Output transformer example

```

package com.datastax.bdp.search.solr.functional;

import java.io.IOException;
import org.apache.commons.lang.StringUtils;
import org.apache.lucene.index.FieldInfo;

```

```

import org.apache.lucene.index.StoredFieldVisitor;
import com.datastax.bdp.search.solr.FieldOutputTransformer;

public class BinaryFieldOutputTransformer extends
FieldOutputTransformer
{
@Override
public void binaryField(FieldInfo fieldInfo, byte[] value,
StoredFieldVisitor visitor, DocumentHelper helper) throws
IOException
{
byte[] bytes = DSP1493Test.decompress(value);
String str = new String(bytes, "UTF-8");
String[] arr = StringUtils.split(str, ",");
String binary_name = arr[0];
String binary_type = arr[1];
String binary_title = arr[2];

FieldInfo binary_name_fi = helper.getFieldInfo("binary_name");
FieldInfo binary_type_fi = helper.getFieldInfo("binary_type");
FieldInfo binary_title_fi = helper.getFieldInfo("binary_title");

visitor.stringField(binary_name_fi, binary_name);
visitor.stringField(binary_type_fi, binary_type);
visitor.stringField(binary_title_fi, binary_title);
}
}

```

## Custom URP example (deprecated)

The custom update request processor (URP) extends the Apache Solr™ URP.

DSE Search includes the released version of a plugin API for Solr updates and a plugin to the CassandraDocumentReader. The plugin API transforms data from the secondary indexing API before data is submitted. The plugin to the CassandraDocumentReader transforms the results data from the database to DSE Search.

**Notice:** The DSE custom URP implementation is deprecated. A custom URP is almost always unnecessary. Instead, DataStax recommends using the [field input/output \(page 354\)](#) (FIT) transformer API instead.

Using the API, applications can tweak a search document before it is mapped and indexed according to the index schema.

The field input transformer (FIT) requires a trailing Z for date field values.

To use the API:

1. Configure the custom URP in the solrconfig.xml.

```

<dseUpdateRequestProcessorChain name="dse">
<processor
class="com.datastax.bdp.search.solr.functional.DSEUpdateRequestProcessorFactoryExample"
/>

```

```
</dseUpdateRequestProcessorChain>
```

2. Write a class to use the custom URP that extends the Solr [UpdateRequestProcessor](#).  
For example:

```
package com.datastax.bdp.search.solr.functional;

import java.io.IOException;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.datastax.bdp.search.solr.handler.update.CassandraAddUpdateCommand;
import com.datastax.bdp.search.solr.handler.update.CassandraCommitUpdateCommand;
import org.apache.solr.update.AddUpdateCommand;
import org.apache.solr.update.CommitUpdateCommand;
import org.apache.solr.update.processor.UpdateRequestProcessor;

public class TestUpdateRequestProcessor extends
 UpdateRequestProcessor
{
 protected final Logger logger =
 LoggerFactory.getLogger(TestUpdateRequestProcessor.class);

 public TestUpdateRequestProcessor(UpdateRequestProcessor next)
 {
 super(next);
 }

 public void processAdd(AddUpdateCommand cmd) throws IOException
 {
 if (cmd instanceof CassandraAddUpdateCommand)
 {
 logger.info("Processing Cassandra-actuated document
update.");
 }
 else
 {
 logger.info("Processing HTTP-based document update.");
 }

 super.processAdd(cmd);
 }

 public void processCommit(CommitUpdateCommand cmd) throws
IOException
 {
 if (cmd instanceof CassandraCommitUpdateCommand)
 {
 logger.info("Processing DSE-actuated commit.");
 }
 }
}
```

```

 else
 {
 logger.info("Processing client-actuated commit.");
 }
 super.processCommit(cmd);
 }
}

```

**3. Export the class to a JAR, and place the JAR in this location:**

- Tarball installations: *install-location/resources/solr/lib*
- Package installations: */usr/share/dse/solr/lib*

The JAR is added to the CLASSPATH automatically.

**4. Test your implementation. For example:**

```

package com.datastax.bdp.search.solr.functional;

import
com.datastax.bdp.search.solr.handler.update.DSEUpdateProcessorFactory;
import org.apache.solr.core.SolrCore;
import org.apache.solr.update.processor.UpdateRequestProcessor;

public class DSEUpdateRequestProcessorFactoryExample extends
DSEUpdateProcessorFactory
{
 SolrCore core;

 public DSEUpdateRequestProcessorFactoryExample(SolrCore core) {
 this.core = core;
 }

 public UpdateRequestProcessor getInstance(
 UpdateRequestProcessor next)
 {
 return new TestUpdateRequestProcessor(next);
 }
}

```

## Interface for custom field types

The CustomFieldType interface marks Apache Solr custom field types and provides their actual stored field type.

DSE Search implements a CustomFieldType interface that marks Apache Solr™ custom field types and provides their actual stored field type. The custom field type stores an integer trie field as a string representing a comma separated list of integer values. When indexed the string is split into its integer values, each one indexed as a trie integer field. This class effectively implements a multi-valued field based on its string representation.

A CustomFieldType can override this method to provide the FieldType for the binary response writer to look at when it determines whether to call the field's `toObject()`. This

allows the binary response writer, for instance, to return `java.util.Date` in place of text for a `CustomFieldType` that extends `TrieDateField`.

To ensure that custom field types control their serialized value, use:

```
public Class<? extends FieldType> getKnownType()
{
 return getClass();
}
```

See the example reference implementation.

To use the `CustomFieldType` interface:

1. Implement a custom field type class something like the following reference implementation.
2. Export the class to a JAR, and place the JAR in this location:

- Package installations: `/usr/share/dse/solr/libusr/share/dse`
- Tarball installations: `install-location/resources/solr/libinstallation_location/resources/dse/lib`

The JAR is added to the CLASSPATH automatically.

## Reference implementation

Here is an example of a custom field type class:

```
package com.datastax.bdp.search.solr.functional;

import com.datastax.bdp.search.solr.CustomFieldType;
import java.util.ArrayList;
import java.util.List;
import org.apache.lucene.index.IndexableField;
import org.apache.solr.schema.FieldType;
import org.apache.solr.schema.SchemaField;
import org.apache.solr.schema.StrField;
import org.apache.solr.schema.TrieField;

public class CustomTestField extends TrieField implements
CustomFieldType
{
 public CustomTestField()
 {
 this.type = TrieField.TrieTypes.INTEGER;
 }

 @Override
 public FieldType getStoredFieldType()
 {
 return new StrField();
 }

 @Override
```

```

public boolean multiValuedFieldCache()
{
 return true;
}

@Override
public ListIndexableField createFields(SchemaField sf, Object
value)
{
 String[] values = ((String) value).split(" ");
 ListIndexableField fields = new ArrayListIndexableField();
 for (String v : values)
 {
 fields.add(createField(sf, v));
 }
 return fields;
}

@Override
public String toInternal(String value)
{
 return value;
}

@Override
public String toExternal(IndexableField f)
{
 return f.stringValue();
}

public Class<? extends FieldType> getKnownType()
{
 return TrieField.class;
}
}

```

## Deleting by query

Delete data based on search criteria, and delete by query best practices.

Delete by query no longer accepts wildcard queries, including queries that match all documents (for example, <delete><query>\* : \*</query></delete>). Instead, use the CQL TRUNCATE command.

Delete by query supports deleting data based on search criteria. After you issue a delete by query, documents start getting deleted immediately and deletions continue until all documents are removed. For example, you can delete the data that you inserted using this command:

```
curl http://localhost:8983/solr/mykeyspace.mysolr/update --data
'<delete><query>color:red</query></delete>' -H 'Content-type:text/xml;
charset=utf-8'
```

Using `&allowPartialDeletes` parameter set to false (default) prevents deletes if a node is down. Using `&allowPartialDeletes` set to true causes the delete to fail if a node is down and the delete does not meet a consistency level of quorum. Delete by queries using `*:*` are an exception to these rules. These queries issue a truncate, which requires all nodes to be up in order to succeed.

## Best practices

DataStax recommends that queries for delete-by-query operations touch columns that are **not** updated. For example, a column that is not updated is one of the elements of a compound primary key.

### Delete by query problem example

The following workflow demonstrates that not following this best practice is problematic:

- When a search coordinator receives a delete-by-query request, the request is forwarded to every node in the search datacenter.
- At each search node, the query is run locally to identify the candidates for deletion, and then the LOCAL\_ONE consistency level deletes the queries for each of those candidates.
- When those database deletes are perceived at the appropriate nodes across the cluster, the records are deleted from the search index.

For example, in a certificates table, each certificate has a date of issue that is a timestamp. When a certificate is renewed, the new issue date is written to the row, and that write is propagated to all replicas. In this example, let's assume that one replica misses it. If you run a periodic delete-by-query that removes all of the certificates with issue dates older than a specified date, unintended consequences occur when the replica that just missed the write with the "certificate renewal" matches the delete query. The certificate is deleted across the entire cluster, on all datacenters making that delete unrecoverable.

## HTTP API SolrJ and other Solr clients

Apache Solr clients work with DataStax Enterprise. DataStax has extended SolrJ to protect internal Solr communication and HTTP access using SSL.

Apache Solr™ clients work with DataStax Enterprise. If you have an existing Solr application, you can create a schema, then import your data and query using your existing Solr tools. The [Wikipedia demo](#) is built and queried using SolrJ. The query is done using pure Ajax. No DataStax Enterprise API is used for the demo.

DataStax has extended SolrJ to protect internal Solr communication and HTTP access using SSL. You can also use SolrJ to change the consistency level of the write in the database on the client side.

## DSE Graph

Documentation for developers and administrators on installing, configuring, and using the features and capabilities of DSE Graph.

DataStax Enterprise Graph is the first graph database fast enough to power customer facing applications. It is capable of scaling to massive datasets and executing both transactional and analytical workloads. DSE Graph incorporates all of the enterprise-class functionality found in DataStax Enterprise, including [advanced security protection](#), built-in [analytics \(page 170\)](#) and enterprise [search \(page 304\)](#) functionality, and [visual management, monitoring](#), and development tools including [DataStax Studio \(page 874\)](#).

## About DSE Graph

Documentation for developers and administrators on installing, configuring, and using the features and capabilities of DSE Graph.

DataStax Enterprise Graph is the first graph database fast enough to power customer facing applications. It is capable of scaling to massive datasets and executing both transactional and analytical workloads. DSE Graph incorporates all of the enterprise-class functionality found in DataStax Enterprise, including [advanced security protection](#), built-in [analytics \(page 170\)](#) and enterprise [search \(page 304\)](#) functionality, and [visual management, monitoring](#), and development tools including [DataStax Studio \(page 874\)](#).

### What is a graph database?

A [graph database](#) is a database that uses graph structures to store data along with the data's relationships. Graph databases use a data model that is as simple as a whiteboard drawing. Graph databases employ vertices, edges, and properties as described in [Data modeling \(page 423\)](#).

### What is DSE Graph?

The built-for-scale architecture of the DSE database means that it is capable of handling petabytes of information and thousands of concurrent users and operations per second. DSE Graph is built on top of the DSE database, a component of DataStax Enterprise. DSE Graph provides the following benefits:

<b>Support for large graphs</b>	Graphs stored in DSE Graph scale with the number of machines in the cluster because the DSE database provides the distributed storage layer. Graphs can contain hundreds of millions ( $10^8$ ) of vertices and billions ( $10^9$ ) of edges.
---------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<b>Support for very many concurrent transactions and operational graph processing (OLTP)</b>	The transactional capacity of DSE Graph scales with the number of machines in the cluster and answers complex traversal queries on huge graphs in milliseconds.
<b>Support for global graph analytics and batch graph processing (OLAP)</b>	Support for global graph analytics and batch graph processing (OLAP) through the Spark framework.
<b>Integration with DSE Search</b>	Integrates with DSE Search for efficient indexing.
<b>Support for geographic, numeric range, and full text search</b>	Support for geographic, numeric range, and full text search for vertices and edges on large graphs.
<b>Native support for Apache TinkerPop</b>	Native support for the popular property graph data model exposed by <a href="#">Apache TinkerPop</a> .
<b>Native support for the Gremlin query language</b>	Native support for the graph traversal language <a href="#">Gremlin</a> .
<b>Integration of the Gremlin Server</b>	Integration with the <a href="#">Gremlin</a> graph server.
<b>Performance tuning options</b>	Numerous graph-level configurations provide options for tuning performance.
<b>Vertex-centric indexes provide optimal querying</b>	Vertex-centric indexes provide vertex-level querying to alleviate issues with the <a href="#">super node problem</a> .
<b>Optimized disk representation</b>	Provides an optimized disk representation to allow for efficient use of storage and speed of access.

## How does DSE Graph differ from Titan?

DSE Graph has higher performance than Titan for the following reasons:

- Specifically engineered for the DSE database. DSE Graph is designed to take advantage of the DSE database's features.

- Optimized storage for graph data. DSE Graph partitions the adjacency list of high-degree vertices, storing and efficient querying of graph data with highly-skewed degree distributions.
- Dedicated index structures that make queries faster.
- Optimized distributed queries. DSE Graph intelligently routes queries to the cluster nodes most suitable for handling each query. This routing achieves higher degrees of data locality and requires moving less data around the cluster. In Titan, all query executions are local on the coordinator, which pull in all data from other cluster instances.

In addition, DSE Graph takes advantage of features of DSE:

- Certified for production environments
- Advanced security features
- Integrated with Enterprise Search and Analytics
- Visual management and monitoring with OpsCenter
- Visual development with DataStax Studio
- Graph support in certified DataStax drivers
- No ETL or synchronization

## How is DSE Graph different from other graph databases?

DSE Graph utilizes the DSE database as a storage backend, so the graph database is distributed, always available, and has a scale-out architecture. The data in a DSE Graph is automatically partitioned across all the nodes in a cluster. Additionally, DSE Graph has built-in support for analytics for OLAP analysis and search on graph data. Finally, all DSE components use advanced security options, so [DSE Graph can be secured for sensitive data](#).

## What is Apache TinkerPop?

[Apache TinkerPop](#) is an open source project that provides an abstraction framework used to interact with DSE Graph as well as other graph databases.

## What is Gremlin?

[Gremlin](#) is the primary interface into DSE Graph. Gremlin is a [graph traversal](#) language and [virtual machine](#) developed by [Apache TinkerPop](#). Gremlin is a [functional language](#) that enables Gremlin to naturally support [imperative](#) and [declarative](#) querying.

## How do I interact with DSE Graph?

The most basic way to interact with DSE Graph is using the Gremlin console `dse gremlin-console`. [Using the Gremlin console \(page 624\)](#), you can create graph database schemas, insert and query data, plus query the database for metadata using graph traversals.

Complex traversals are simple to define with Gremlin compared to SQL. If you prefer a graphical tool, use [DataStax Studio \(page 874\)](#). For production, DataStax supplies a number of [drivers](#) in various programming languages, which pass Gremlin statements to DSE Graph: Java, Python, C#, C/C++, Node.js, and Ruby.

[DSE OpsCenter](#) provides monitoring capability.

## How can I move data to and from DSE Graph?

Use a variety of methods to insert data:

- The [DSE Graph Loader \(page 471\)](#) provides a command line utility that loads data from CSV, JSON, text files, Gryo files, and queries from JDBC-compatible databases.
- Gremlin scripts and commands in [DataStax Studio \(page 368\)](#) and the [Gremlin console \(page 624\)](#).
- [GraphSON \(page 465\)](#) files are JSON files that can exchange graph data and metadata.
- [GraphML \(page 465\)](#) is a standard for exchanging graph data. It can exchange vertex and edge information, but metadata is limited.
- [Gryo \(page 465\)](#) is a Kryo variation, enabling the exchange of binary data.

**Important:** Best practices start with data modeling before inserting data. The paradigm shift between relational and graph databases requires careful analysis of data and data modeling before importing and querying data in a graph database. [DSE Graph data modeling \(page 423\)](#) provides information and examples.

## What tools come with DSE Graph?

DSE Graph comes bundled with a number of tools:

- [DataStax Studio \(page 874\)](#), a web-based notebook for running Gremlin commands and visualizing graphs
- Gremlin Console, a shell for exploring DSE Graph
- Gremlin Server to serve remote queries
- DSE OpsCenter, a monitoring and administrative tool
- Integration with [DataStax Enterprise \(DSE\) Search \(page 304\)](#) and [DSE Analytics \(page 170\)](#)

## What kind of hardware or cloud environment do I need to run DSE Graph?

DSE Graph runs on commodity hardware with common specifications like other DataStax Enterprise offerings. See [Planning a cluster deployment](#).

## DSE Graph Terminology

Explain terminology specific to DSE Graph.

This terminology is specific to DSE Graph.

### adjacency list

A collection of unordered lists used to represent a finite graph. Each list describes the set of neighbors of a vertex in the graph.

### adjacent vertex

A vertex directly attached to another vertex by an edge.

### directed graph

A set of vertices and a set of arcs (ordered pairs of vertices). In DSE Graph, the terminology "arcs" is not used, and edges are directional.

### edge

A connection between vertices. Edges can be unordered (no directional orientation) or ordered (directional). An edge can also be described as an object that has a vertex at its tail and head.

### **element**

An element is a vertex, edge, or property.

### **global index**

An index structure over the entire graph.

### **graph**

A collection of vertices and edges.

### **graph degree**

The largest vertex degree of the graph.

### **graph partitioning**

A process that consists of dividing a graph into components, such that the components are of about the same size and there are few connections between the components.

### **graph traversal**

An algorithmic walk across the elements of a graph according to the referential structure explicit within the graph data structure.

### **incident edge**

An edge incident to a particular vertex, meaning that the edge and vertex touch.

### **index**

An index is a data structure that allows for the fast retrieval of elements by a particular key-value pair.

### **meta-property**

A property that describes some attribute of another property.

### **order**

The magnitude of the number of edges to the number of vertices.

### **partitioned vertex**

Used for vertices that have a very large number of edges, a partitioned vertex consists of a portion of a vertex's data that results from dividing the vertex into smaller components for graph database storage. *Experimental*

### **property**

A key-value pair that describes some attribute of either a vertex or an edge.

Property key is used to describe the key in the key-value pair. All properties are global in DSE Graph, meaning that a property can be used for any vertices. For example, "name" can be used for all vertices in a graph.

### **traversal source**

A domain specific language (DSL) that specifies the traversal methods used by a traversal.

### **undirected graph**

A set of vertices and a set of edges (unordered pairs of vertices).

### **vertex-centric index**

A local index structure built per vertex.

### **vertex**

A vertex is the fundamental unit of which graphs are formed. A vertex can also be described as an object that has incoming and outgoing edges.

### **vertex degree**

The number of edges incident to a vertex.

## DSE Graph QuickStart

DSE Graph QuickStart using DataStax Studio or Gremlin console.

### QuickStart Introduction

QuickStart Introduction

Graph databases are useful for discovering simple and complex relationships between objects. Relationships are fundamental to how objects interact with one another and their environment. Graph databases perfectly represent the relationships between objects.

Graph databases consist of three elements:

#### **vertex**

A vertex is an object, such as a person, location, automobile, recipe, or anything else you can think of as nouns.

#### **edge**

An edge defines the relationship between two vertices. A person can create software, or an author can write a book. Typically an edge is equivalent to a verb.

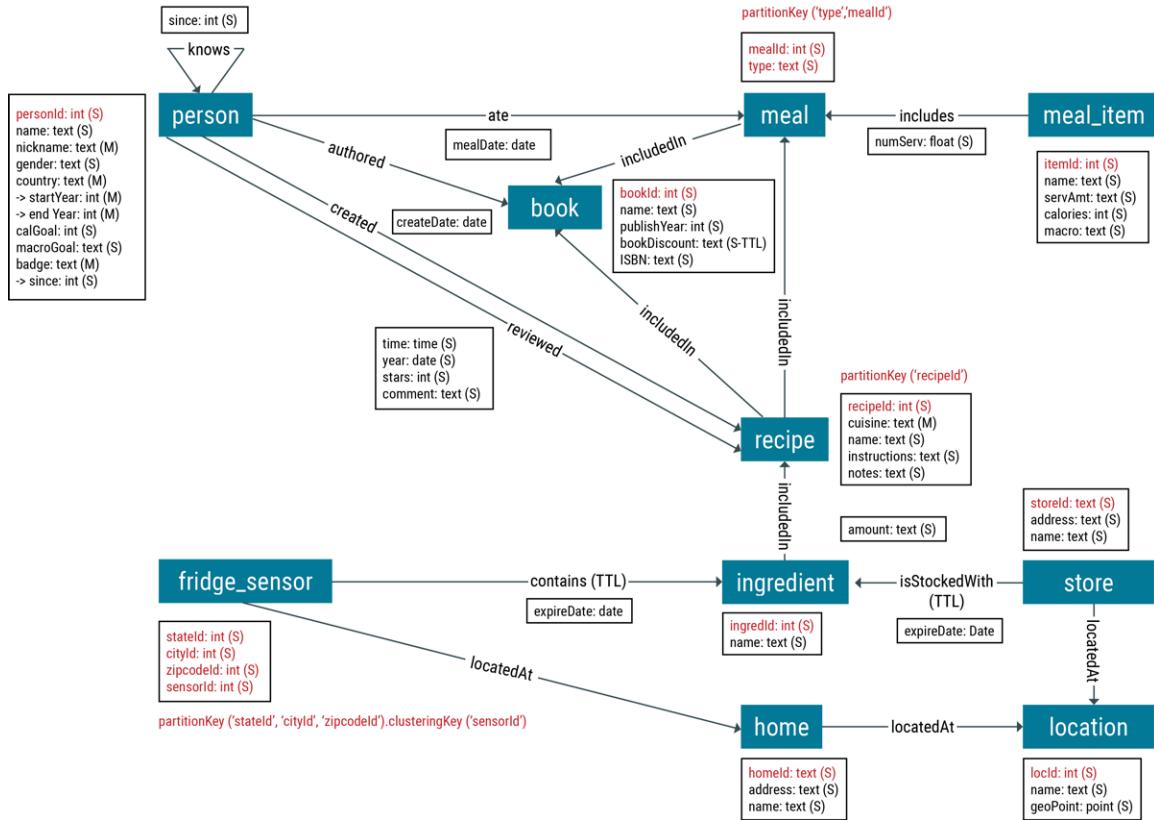
#### **property**

A key-value pair that describes some attribute of either a vertex or an edge. A property key is used to describe the key in the key-value pair. All properties are global in DSE Graph, meaning that a property can be used for any vertices. For example, "name" can be used for all vertices in a graph.

Vertices, edges, and properties can have properties; for this reason, DSE Graph is classified as a **property graph**. The properties for elements are an important element of storing and querying information in a property graph.

Property graphs are typically quite large, although the nature of querying the graph varies depending on whether the graph has large numbers of vertices, edges, or both vertices and edges. To get started with graph database concepts, a *toy* graph is used for simplicity. The example used here explores the world of food.

Figure 3: Recipe Toy Graph



Elements are labeled to distinguish the type of vertices and edges in a graph database using **vertex labels** and **edge labels**. A vertex labeled *person* holds information about an author or reviewer or someone who ate a meal. An edge between an *person* and a *book* is labeled *authored*. Specifying appropriate labels is an important step in [graph data modeling \(page 423\)](#).

Vertices and edges generally have properties. For instance, a *person* vertex can have properties *name* and *gender*. Edges can also have properties. A *created* edge can have a *createDate* property that identifies when the adjoining *recipe* vertex was created.

Information in a graph database is retrieved using **graph traversals**. **Graph traversals walk** a graph with a single or series of **traversal steps** from a defined starting point and filter each step until returning a result.

To retrieve information using graph traversals, you must first insert data. The steps listed in this section allow you to gain a rudimentary understanding of DSE Graph with a minimum amount of configuration and schema creation.

## QuickStart Installation

Install DataStax Enterprise and DataStax Studio.

1. [Install DataStax Enterprise](#).
2. Start DataStax Enterprise with [DSE Graph enabled \(page 867\)](#).

3. Start either [DataStax Studio \(page 370\)](#) or [Gremlin console \(page 370\)](#):

- a. [Install DataStax Studio and start Studio \(page 876\)](#).
- b. Start the Gremlin Console.

```
bin/dse gremlin-console
```

```
\, , /
(o o)
----o00o-(3)-o00o----plugin activated:
tinkerpop.tinkergraph
plugin activated: tinkerpop.server
plugin activated: tinkerpop.utilities
==>Connected - localhost/127.0.0.1:8182-[4edf75f9-ed27-4add-
a350-172abe37f701]
==>Set remote timeout to 2147483647ms
==>All scripts will now be sent to Gremlin Server
- [localhost/127.0.0.1:8182]-[4edf75f9-ed27-4add-
a350-172abe37f701] - type ':remote console' to return to
local mode
gremlin>
```

Gremlin console sends all commands typed at the prompt to the Gremlin Server that will process the commands. DSE Graph runs a Gremlin Server `tinkerpop.server` on each DSE node. Gremlin console automatically connects to the Gremlin Server.

The Gremlin console runs in *remote* mode automatically, processing commands on the Gremlin server. The Gremlin console by default opens a session to run commands on the remote server. The Gremlin console can be switched to run commands locally using:

```
:remote console
```

All commands will need to be submitted remotely once this command is run. Using the command again will switch the context back to the Gremlin server.

## QuickStart Configuration

Configure DSE Graph to run QuickStart.

1. Create a Studio notebook and configure a graph for the QuickStart. If you are using Gremlin console, skip to [this step \(page 371\)](#).
  - a. This tutorial exists as a Studio notebook, *DSE Graph QuickStart*, so that you do not have to create a notebook. However, in Studio, [creating a notebook \(page 899\)](#) is simple. If running Studio on a DSE node, the default connection of localhost works, otherwise [create a connection \(page 877\)](#) for the DSE cluster desired. Each notebook is connected to a particular graph. Multiple notebooks can be connected to the same graph, or multiple notebooks can be created to connect to different graphs.

A connection in Studio defines the graph and assigns a graph traversal *g* for that graph. A graph traversal is the mechanism for visiting each vertex in a graph, based on the filters defined in the graph traversal. To query DSE Graph, the graph traversal *g* must be assigned to a particular graph; Studio manages this assignment with connections.

A blank notebook opens with a single cell. DSE Graph runs a Gremlin Server `tinkerpop.server` on each DataStax Enterprise node. Studio automatically connects to the Gremlin Server, and if it doesn't exist, it creates a graph using the connection information. The *graph* is stored as one graph instance per DSE database keyspace. Once a graph exists, a graph traversal source *g* is configured that allows graph traversals to be executed to query the graph. A graph traversal is bound to a specific traversal source, which by default is the standard OLTP traversal engine. The *graph* commands can add vertices and edges to the database, or get other graph information. The *g* commands can query or add vertices and edges.

- b.** Set the schema mode to **Development** and allow full scans.

```
schema.config().option('graph.schema_mode').set('Development')
schema.config().option('graph.allow_scan').set('true')
```

**Caution:** Development is a more lenient mode that allows schema to be created automatically when adding data, and also allows full scans that can inspect the data with broad graph traversals. Full scans over large graphs will have high read latency, and are not appropriate for production applications. For production, the schema mode should be set to **Production** to require schema prior to inserting data and disallow full scans.

2. Create a graph in Gremlin Console and configure the graph for the QuickStart.

- a.** Create a graph to hold the data. The *system* commands are used to run commands that affect graphs in DSE Graph.

```
system.graph('test').create()
```

```
==>null
```

Once a graph exists, a graph traversal *g* is configured that will allow graph traversals to be executed. Graph traversals are used to query the graph data and return results. A graph traversal is bound to a specific traversal source which is the standard OLTP traversal engine.

- b.** Configure a graph traversal *g* to use the default graph traversal setting, which is `test.g`. This step will also create an implicit `graph` object.

```
:remote config alias g test.g
```

```
==>g=test.g
```

The `graph` commands allow graphs to be written to file, add vertices, properties, or edges to the database, and set or get other graph configuration. The `g` commands create queries to obtain results, and can also add vertices, properties, or edges to the database.

- c. Set the schema mode to **Development** and allow full scans.

```
schema.config().option('graph.schema_mode').set('Development')
schema.config().option('graph.allow_scan').set('true')
```

**Caution:** Development is a more lenient mode that allows schema to be created automatically when adding data, and also allows full scans that can inspect the data with broad graph traversals. Full scans over large graphs will have high read latency, and are not appropriate for production applications. For production, the schema mode should be set to **Production** to require schema prior to inserting data and disallow full scans.

- d. When creating a new graph, to check what graphs already exist, use:

```
system.graphs()

==>test
==>anotherTest
```

## QuickStart Vertex and edge counting

Methods for counting vertices and edges in DSE Graph.

There are different methods for accomplishing vertex and edge counts in DSE Graph. Examples here will show how to use the Gremlin `count()` command either as a transactional or analytical query, and Spark SQL for analytical queries.

A transactional Gremlin query can be used to check the number of vertices that exist in the graph, and are useful for exploring small graphs. However, such a query scans the full graph, traversing every vertex, and should **not** be run on large graphs! If multiple DSE nodes are configured, this traversal step intensively walks all partitions on all nodes in the cluster that have graph data. This method is not appropriate for Production operations.

An analytical Gremlin query can be used to check the number of vertices that exist in any graph, large or small, and are much safer for Production operations. The queries will be written like transactional Gremlin queries, but executed with the analytic Spark engine.

Spark SQL provides another query method for counting vertices in transactional graph traversals. If the [AlwaysOnSQL service \(page 245\)](#) is turned on, Studio uses the JDBC interface to pass queries to DSE Analytics. Two tables, `graphName_vertices` and `graphName_edges`, are automatically generated in the Spark database `dse_graph` for each graph, where `graphName` is replaced with the graph used for the Studio connection assigned to a Studio notebook. These tables can be queried with common Spark SQL commands directly in Studio, or can be explored with the [dse spark-sql \(page 235\)](#) shell.

To learn more about using Spark SQL to query, see the [Using Spark SQL to query data \(page 235\)](#) documentation.

### Transactional Gremlin count()

- Use the traversal step `count()`; the current count will be zero, because no data exists yet. A graph traversal `g` is chained with `V()` to retrieve all vertices and `count()` to compute the number of vertices. Chaining executes sequential traversal steps in the most efficient order.

```
g.V().count()
```



### Analytical Gremlin count()

- To use Gremlin console, configure the traversal to run an analytical query:

```
:remote config alias g test.a
```

where `test.a` denotes that the graph will be used for analytic purposes.

- To use Studio, configure the Run option to "Execute using analytic engine (Spark)" before running the query.
- Use the traversal step `count()`; the current count will be zero, because no data exists yet. A graph traversal `g` is chained with `V()` to retrieve all vertices and `count()` to compute the number of vertices. Chaining executes sequential traversal steps in the most efficient order.

```
g.V().count()
```



## Spark SQL count

- Enable AlwaysOn SQL or start a Spark SQL Thrift server instance.

To use Spark SQL in Studio, enable AlwaysOn SQL service in the `dse.yaml` file by setting the option to `true` and restart DSE:

```
AlwaysOn SQL options
alwayson_sql_options:
 # If it's true, the node is enabled for AlwaysOn SQL. Only
 # can be enabled as a AlwaysOn SQL node
 enabled: true
```

In a Studio cell, select *Spark SQL* in the language menu in a cell and set the database to `dse_graph`.

To use the Spark SQL shell, start the shell:

```
dse spark-sql
```

and navigate to the correct database:

```
USE dse_graph;
```

- Then, in either Studio or the Spark SQL shell, execute the Spark SQL query for finding the vertex count:

```
SELECT count(*) FROM DSE_GRAPH_QUICKSTART_vertices;
```

## Edge counts

- To do an edge count with Gremlin, replace `v()` with `E()`:

```
g.E().count()
```

- To do an edge count with Spark SQL, replace the word `vertices` in the table name with `edges`:

```
SELECT count(*) FROM DSE_GRAPH_QUICKSTART_edges;
```

## QuickStart Simple example

Simple DSE Graph example.

Let's start with a simple example from the recipe data model. The data is composed of two vertices, one person who is an author (*Julia Child*) and one book (*The Art of French Cooking, Vol. 1*) with an edge between them to identify that Julia Child authored that book. Although we could make this graph without schema, and DSE Graph would make a best guess about the data types, we'll supply schema before inserting the graph data.

Next `graph.addVertex` is used to add data for a single vertex. Note the use of label to designate the vertex label. A `g.addV` statement could also be used, as shown in the alternate method.

Run the command and look at the results using the buttons to display the Raw JSON, Table, and Graph views

1. Schema is defined for properties `personId`, `name`, and `gender`. Properties should be created first, before vertex labels. A vertex label `person` identifies a partitionKey `personId` using an user-defined vertex id with a single partitionKey; `personId` is an integer for simplicity in this example. The schema to add the partitionKey and properties are executed with two statements, but could be executed as a single chained statement.

```
schema.propertyKey('personId').Int().single().create()
schema.propertyKey('name').Text().single().create()
schema.propertyKey('gender').Text().single().create()
schema.vertexLabel('person').partitionKey('personId').create()
schema.vertexLabel('person').properties('name', 'gender').add()
```

The user-defined vertex id is used to partition the graph data amongst the cluster's nodes ([more information](#)) ([page 561](#)). User-defined vertex (UDV) ids are strongly recommended, although auto-generated vertex ids are also available, but deprecated in DSE 6.0, with warnings logged when using auto-generated vertex ids. [(add a link here)]([link info](#))

As you will see in the schema for a `book` vertex label, a property key can be reused for different types of information. While properties are "global" in the sense that they can be used with multiple vertex labels, it is important to understand that when specifying a property in a graph traversal, it is always used in conjunction with a vertex label.

2. First, insert a vertex for Julia Child using a `graph.addVertex()` command. The vertex label is `person` and two property key-value pairs are created for `name` and `gender`. Note that a label designates the key for a key-value pair that sets the vertex label.

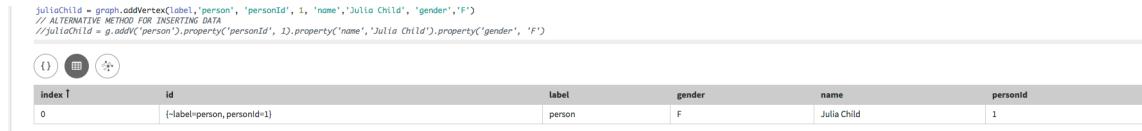
```
juliaChild = graph.addVertex(label, 'person', 'personId', 1,
 'name', 'Julia Child', 'gender', 'F')
```

Note that there is an alternative method of inserting the vertex with a graph traversal `g.addV`:

```
juliaChild = g.addV('person').property('personId',
 1).property('name', 'Julia Child').property('gender', 'F')
```

Performance tests show that the `graph.addVertex()` is faster, but the `g.addV` can be used in applications using [DSE Drivers](#).

The Studio result:



A screenshot of the DataStax Studio interface. At the top, there is a code editor window containing the following Java code:

```
juliaChild = graph.addVertex("label", "person", "personId", 1, "name", "Julia Child", "gender", "F")
// ALTERNATIVE METHOD FOR INSERTING DATA
//juliaChild = g.addV("person").property("personId", 1).property("name", "Julia Child").property("gender", "F")
```

Below the code editor is a table with three rows of data. The first two rows are headers: "index T" and "id". The third row contains the data: "0", "-label=person, personId=1", "person", "F", "Julia Child", and "1".

index T	id	label	gender	name	personId
0	-label=person, personId=1	person	F	Julia Child	1

**Tip:** In Studio, the result can be displayed using different views: Raw JSON, Table, or Graph. Explore the options.

The Gremlin console result:

```
==>v[{:label=>person, :personId=>1}]
```

3. Create the schema for a vertex label `book` that has an user-defined vertex id single partitionKey `bookId` and includes the properties `name`, `publishYear`, and `ISBN`.

```
schema.propertyKey('bookId').Int().single().create()
schema.propertyKey('publishYear').Int().single().create()
schema.propertyKey('ISBN').Text().single().create()
schema.vertexLabel('book').partitionKey('bookId').create()
schema.vertexLabel('book').properties('name', 'publishYear', 'ISBN').add()
```

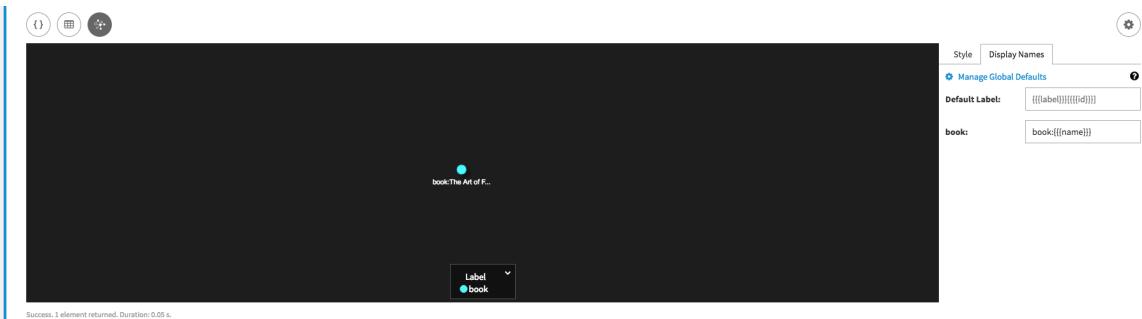
4. Insert a book into the graph:

```
artOfFrenchCookingVolOne = graph.addVertex(label, 'book', 'bookId',
 1001, 'name', 'The Art of French Cooking, Vol. 1', 'year', 1961)
```

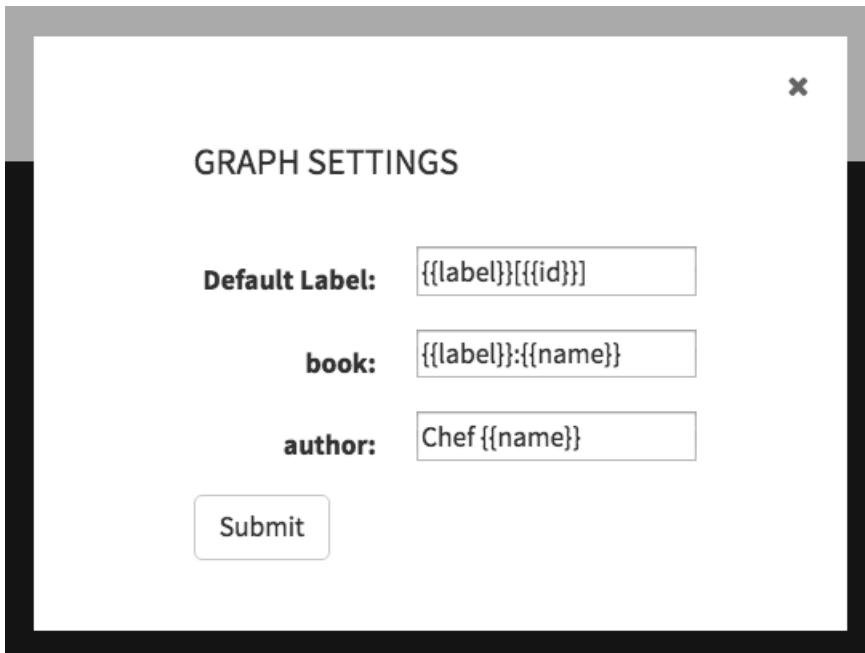
or optionally, the traversal query:

```
artOfFrenchCookingVolOne = g.addV('book').property('bookId',
 1001).property('name', 'The Art of French Cooking, Vol.
 1').property('publishYear', 1961)
```

The Studio result:



As with the author vertex, you can see all the information about the book vertex created. In **Graph view**, use the **Settings button** (the gear) to change the display label for author by entering `Chef {{name}}`. Change the book display label with `{{label}}:{{name}}`. Change the book display label with `{}{{name}}`. To set graph display names more generally, look for “Configure Graph Display Names” under the three bars in the upper lefthand corner of Studio.



The Gremlin console result:

```
==>v[~label=book, bookId=1001]
```

##### 5. Add schema for the edge between the two vertices:

```
schema.edgeLabel('authored').multiple().create()
schema.edgeLabel('authored').connection('person', 'book').add()
```

##### 6. The first query uses a variable `juliaChild` to hold the `person` vertex information, while the second query uses the variable `artOfFrenchCookingVolOne` to hold the `book` vertex information. The third query uses a graph traversal

`g.V(firstVertex).addE(edgeLabel).to(secondVertex)` to create the edge between the author and book vertices.

```
juliaChild = g.V().has('person', 'personId', 1).next()
artOfFrenchCookingVolOne = g.V().has('book', 'bookId', 1001).next()
g.V(juliaChild).addE('authored').to(artOfFrenchCookingVolOne)
```

or the graph alternative:

```
juliaChild.addEdge('authored', artOfFrenchCookingVolOne)
```

Use **Graph view** in Studio to see the relationship. Scroll over elements to display additional information.

The screenshot shows the DataStax Studio interface with the 'Graph view' selected. It displays a graph with two nodes: a green dot representing 'Chef Julia Child' and a blue dot representing 'book: The Art of French Cooking Vol One'. A single edge connects them, labeled 'authored'. A tooltip appears when hovering over the edge, showing the label 'book' and the person 'person'.

The Gremlin console result:

```
==>e[~label=authored, ~out_vertex=~label=person, personId=1,
 ~in_vertex=~label=book, bookId=1001,
 ~local_id=5deac140-0562-11e8-a4a1-4b3271ac7767][~label=person,
 personId=1]-authored->(~label=book, bookId=1001)]
```

7. Ensure that the data inserted for the author is correct by checking with a `has()` step using the vertex label `person` and the property `name = Julia Child`. This graph traversal is a basic starting point for more complex traversals, because it narrows the search of the graph with specific information.

```
g.V().has('person', 'name', 'Julia Child')
```

In Studio, use the **Table view** to look at the results, as it is much more readable than the **Raw JSON view**.

The screenshot shows the DataStax Studio interface with the 'Table view' selected. It displays a single row of data for the vertex 'Julia Child'. The columns are: index (0), id (~label=person, personId=1), label (person), gender (F), name (Julia Child), and personId (1).

index	id	label	gender	name	personId
0	~label=person, personId=1	person	F	Julia Child	1

The vertex information is displayed for the `person` vertex for `Julia Child`. Note the id consists of the label and the user-defined vertex id `personId`.

The Gremlin console result:

```
==>v[{~label=person, personId=1}]
```

8. Another useful traversal is `valueMap()`, which prints the key-value listing of each property value for specified vertices.

```
g.V().hasLabel('person').valueMap()
```

```
{}

{
 "gender": [
 "F"
],
 "name": [
 "Julia Child"
],
 "personId": [
 1
]
}
```

**Caution:** Using `valueMap()` without specifying properties can result in slow query latencies, if a large number of property keys exist for the queried vertex or edge. Specific properties can be specified, such as `valueMap('name')`.

9. Although Spark SQL is used more for analytical queries, simple queries similar to Gremlin can be made, such as querying information about vertices. A query can look for specific columns for a specific vertex label, in this case, a person with the name *Julia Child*. Notice the use of backticks to escape the tilde in the column name `~label` and `name`.

```
SELECT personid, name, gender FROM DSE_GRAPH_QUICKSTART_vertices WHERE
`~label` = 'person' AND `name` = 'Julia Child';
```

index	personid	name	gender
0	1	Julia Child	F

## QuickStart Key features

Key features of DSE Graph.

A *vertex label* **person** specifies the type of vertex, *personId* provides a user-defined vertex id to manage cluster storage of the vertex, and the *property keys* **name** and **gender** display the properties for a person. [Creating vertex labels \(page 451\)](#) explains the id components.

1. A useful traversal is `valueMap()` which prints the key-value listing of each property value for specified vertices.

```
g.V().hasLabel('person').valueMap()
```

```
{}
{
 "gender": [
 "F"
],
 "name": [
 "Julia Child"
],
 "personId": [
 1
]
}
```

**Caution:** Using `valueMap()` without specifying properties can result in slow query latencies, if a large number of property keys exist for the queried vertex or edge. Specific properties can be specified, such as `valueMap('name')`.

2. If only the value of a particular property key is desired, the `values()` traversal step can be used. To get the *name* of all vertices, use:

```
g.V().values('name')
```

The screenshot shows the DataStax Studio interface with the GREMLIN tab selected. The command `g.V().values('name')` is entered in the input field. The results are displayed in a table:

index	↑	value
0		Julia Child
1		The Art of French Cooking, Vol. 1

Below the table, it says "Displaying 1 - 2 of 2" and "2 ELEMENTS RETURNED".

3. Edge information may also be retrieved. The next command filters all edges to find those with an *edge label authored*.

```
g.V().hasLabel('authored')
```

The Raw JSON view of the edge information displays details about the incoming and outgoing vertices as well as edge parameters *id*, *label*, and *type*.



```
g.E().hasLabel('authored')

{ "id": "->label=authored, ~out_vertex={~label=person, personId=1}, ~local_id=d93c7df0-02d5-11e8-a4a1-4b3271ac7767, ~in_vertex={~label=book, bookId=1001}", "label": "authored", "type": "edge", "inV": "person", "outV": "book", "inVLabel": "person", "outVLabel": "book", "inVId": 1, "outVId": 1001 }
```

In Gremlin console:

```
==>e[{~label=authored, ~out_vertex={~label=person, personId=1}, ~in_vertex={~label=book, bookId=1001}, ~local_id=5deac140-0562-11e8-a4a1-4b3271ac7767}] [{~label=person, personId=1} -authored-> {~label=book, bookId=1001}]
```

- Spark SQL can also be used to find information about edges. Notice that the Spark-generated tables display different information than the Gremlin graph query. The traversal step *count()* is useful for counting both the number of vertices and the number of edges. To count edges, use *E()* rather than *V()*. You should have one edge. The same cautions apply about real-time transactional uses in Production - Spark SQL count or OLAP execution, both analytical actions, will be a better choice!

```
SELECT * FROM DSE_GRAPH_QUICKSTART_edges;
```



index	id	src	dst	-label
0	d93c7df0-02d5-11e8-a4a1-4b3271ac7767	person:AAAABAAAAAAE=	book:AAAABAAAAA+k=	authored

## QuickStart Graph schema

Set graph schema.

Before adding more data to the graph, let's stop and talk about schema. Schema defines the possible properties and their data types for the graph. These properties are then used in the definitions of vertex labels and edge labels. The last critical step in schema creation is index creation. Indexes play an important role in making graph traversals efficient and fast. See [creating schema \(page 436\)](#) and [creating indexes \(page 457\)](#) for more information.

First, let's create schema for the property keys. In the next two cells, the first command clears the schema for the previously created vertices and edge. After the schema creation is completed, the next step is to enter data for those elements again in a longer script.

**Note:** DSE Graph has two schema modes, Production and Development.

In Production mode, all schema must be identified before data is entered. In Development mode, schema can be created or modified after data is entered.

**1. Clear the schema:**

```
schema.drop()
```

The screenshot shows the DataStax Studio interface with the Gremlin tab selected. A command line window displays the following text:  
GREMLIN >  
> schema.clear()  
null  
1 element returned. Duration: 1.223 s.

**2. To keep the Spark SQL data synchronized with the graph, drop the Spark SQL tables. The tables will be automatically rebuilt, so that the data will align with the graph schema and data entered later.**

```
DROP TABLE DSE_GRAPH_QUICKSTART_vertices;
DROP TABLE DSE_GRAPH_QUICKSTART_edges;
```

### Property Key schema

**3. Create the new property key schema:**

```
// *****
// PROPERTY KEYS
// *****
// SYNTAX:
// propertyKey('name').
// type().
// [single() | multiple()].
// [ttl].
// [properties(metadata_property)].
// [ifNotExists()].
// [create() | add() | describe() | exists()]
// DEFAULT IS SINGLE CARDINALITY
// *****

// SINGLE CARDINALITY PROPERTY KEY
schema.propertyKey('personId').Int().single().create()
schema.propertyKey('mealId').Int().single().create()
schema.propertyKey('itemId').Int().single().create()
schema.propertyKey('recipeId').Int().single().create()
schema.propertyKey('bookId').Int().single().create()
schema.propertyKey('ingredId').Int().single().create()
schema.propertyKey('homeId').Int().single().create()
schema.propertyKey('storeId').Int().single().create()
schema.propertyKey('locId').Text().single().create()
schema.propertyKey('stateId').Int().single().create()
schema.propertyKey('cityId').Int().single().create()
schema.propertyKey('sensorId').Int().single().create()
schema.propertyKey('name').Text().single().create()
schema.propertyKey('gender').Text().single().create()
```

```

schema.propertyKey('calGoal').Int().single().create()
schema.propertyKey('macroGoal').Text().single().create()
schema.propertyKey('publishYear').Int().single().create()
schema.propertyKey('ISBN').Text().single().create()
schema.propertyKey('bookDiscount').Text().ttl(604800).create()
schema.propertyKey('instructions').Text().single().create()
schema.propertyKey('notes').Text().single().create()
schema.propertyKey('type').Text().single().create()
schema.propertyKey('servAmt').Text().single().create()
schema.propertyKey('macro').Text().single().create()
schema.propertyKey('calories').Int().single().create()
schema.propertyKey('geoPoint').Point().withGeoBounds().create()
schema.propertyKey('address').Text().single().create()
schema.propertyKey('amount').Text().single().create()
// MULTIPLE CARDINALITY PROPERTY KEY
schema.propertyKey('nickname').Text().multiple().create()
schema.propertyKey('cuisine').Text().multiple().create()
// MULTIPLE CARDINALITY PROPERTY KEY WITH SINGLE CARDINALITY META-
PROPERTY
schema.propertyKey('since').Int().single().create() // meta-property
schema.propertyKey('badge').Text().multiple().properties('since').create()
// MULTIPLE CARDINALITY PROPERTY KEY WITH MULTIPLE CARDINALITY META-
PROPERTY
schema.propertyKey('startYear').Int().multiple().create() // meta-
property
schema.propertyKey('endYear').Int().multiple().create() // meta-
property
schema.propertyKey('country').Text().multiple().properties('startYear', 'endYear').creat

// EDGE PROPERTIES
schema.propertyKey('numServ').Int().single().create()
schema.propertyKey('mealDate').Date().single().create()
schema.propertyKey('useDate').Date().single().create()
schema.propertyKey('createDate').Date().single().create()
schema.propertyKey('expireDate').Date().single().create()
schema.propertyKey('stars').Int().single().create()
schema.propertyKey('time').Time().single().create()
schema.propertyKey('year').Date().single().create()
schema.propertyKey('comment').Text().single().create()

```

Each property must be defined with a [data type \(page 740\)](#). DSE Graph data types are aligned with the DSE database data types. By default, properties have single cardinality, but can be defined with multiple cardinality. Multiple cardinality allows more than one value to be assigned to a property.

In addition, properties can have their own properties, or **meta-properties**. Meta-properties can only be nested one deep, and are useful for keying information to an individual property. Notice that property keys can be created with an additional method `ifNotExists()`. This method prevents overwriting a definition that can already exist.

## Vertex label schema

4. After property keys are created, vertex labels can be defined.

```

// *****
// VERTEX LABELS
// *****
// SYNTAX:
// schema.vertexLabel('vertexLabel').
// [partitionKey(propertyKey, [partitionKey(propertyKey)])].
// [clusteringKey(propertyKey)].
// [ttl].
// [properties(property, property)].
// [index].
// [partition()].
// [cache()].
// [ifNotExists()].
// [create() | add() | describe() | exists()]
// *****

// SINGLE-KEY VERTEX ID
schema.vertexLabel('person').partitionKey('personId').create()
schema.vertexLabel('person').properties('name', 'nickname', 'gender', 'calGoal', 'macroGoal')
schema.vertexLabel('book').partitionKey('bookId').create()
schema.vertexLabel('book').properties('name', 'publishYear', 'ISBN', 'bookDiscount').add()
schema.vertexLabel('meal_item').partitionKey('itemId').create()
schema.vertexLabel('meal_item').properties('name', 'servAmt',
 'macro', 'calories').add()
schema.vertexLabel('ingredient').partitionKey('ingredId').create()
schema.vertexLabel('ingredient').properties('name').add()
schema.vertexLabel('home').partitionKey('homeId').create()
schema.vertexLabel('home').properties('name', 'address').add()
schema.vertexLabel('store').partitionKey('storeId').create()
schema.vertexLabel('store').properties('name', 'address').add()
schema.vertexLabel('location').partitionKey('locId').create()
schema.vertexLabel('location').properties('name', 'geoPoint').add()
schema.vertexLabel('recipe').partitionKey('recipeId').create()
schema.vertexLabel('recipe').properties('name', 'cuisine',
 'instructions', 'notes').add()
// MULTIPLE-KEY VERTEX ID
schema.vertexLabel('meal').partitionKey('type', 'mealId').create()
// COMPOSITE KEY VERTEX ID
schema.vertexLabel('fridgeSensor').partitionKey('stateId',
 'cityId').clusteringKey('sensorId').create()
schema.vertexLabel('fridgeSensor').properties('name').add()

```

The schema for vertex labels defines the label *type*, and optionally defines the properties associated with the vertex label. There are two different methods for defining the association of the properties with vertex labels, either during creation, or by adding them after vertex label addition. The *ifNotExists()* method can be used for any schema creation.

*Vertex ids* should be [user-defined \(UDV\) ids \(page 451\)](#), as [auto-generated vertex ids \(page 451\)](#) are deprecated in DSE 6.0. UDV ids are explained in further detail in the documentation, but note that *partition keys* and *clustering keys* may be defined.

DSE Graph limits the number of vertex labels to 200 per graph.

β

### Edge label schema

- After property keys are created, edge labels can be defined.

```
// *****
// EDGE LABELS
// *****
// SYNTAX:
//schema.edgeLabel('edgeLabel').
// [single() | multiple()].
// [connection(outVertex, inVertex)].
// [ttl].
// [properties(property[, property])].
// [ifNotExists()].
// [create() | add() | describe() | exists()]
// DEFAULT IS MULTIPLE CARDINALITY
// *****

schema.edgeLabel('ate').multiple().create()
schema.edgeLabel('ate').properties('mealDate').add()
schema.edgeLabel('ate').connection('person', 'meal').add()
schema.edgeLabel('knows').multiple().create()
schema.edgeLabel('knows').properties('since').add()
schema.edgeLabel('knows').connection('person', 'person').add()
schema.edgeLabel('includes').multiple().create()
schema.edgeLabel('includes').properties('numServ').add()
schema.edgeLabel('includes').connection('meal', 'meal_item').add()
schema.edgeLabel('includedIn').multiple().create()
schema.edgeLabel('includedIn').properties('amount').add()
schema.edgeLabel('includedIn').connection('recipe', 'meal').add()
schema.edgeLabel('includedIn').connection('meal', 'book').add()
schema.edgeLabel('includedIn').connection('recipe', 'book').add()
schema.edgeLabel('includedIn').connection('ingredient', 'recipe').add()
schema.edgeLabel('created').multiple().create()
schema.edgeLabel('created').properties('createDate').add()
schema.edgeLabel('created').connection('person', 'recipe').add()
schema.edgeLabel('reviewed').multiple().create()
schema.edgeLabel('reviewed').properties('time', 'year', 'stars', 'comment').add()
schema.edgeLabel('reviewed').connection('person', 'recipe').add()
schema.edgeLabel('authored').multiple().create()
schema.edgeLabel('authored').connection('person', 'book').add()
schema.edgeLabel('contains').multiple().ttl(60800).create()
schema.edgeLabel('contains').properties('expireDate').add()
schema.edgeLabel('contains').connection('fridgeSensor', 'ingredient').add()
schema.edgeLabel('isStockedWith').multiple().ttl(60800).create()
schema.edgeLabel('isStockedWith').properties('expireDate').add()
schema.edgeLabel('isStockedWith').connection('store', 'ingredient').add()
schema.edgeLabel('isLocatedAt').multiple().create()
schema.edgeLabel('isLocatedAt').connection('home', 'location').add()
schema.edgeLabel('isLocatedAt').connection('store', 'location').add()
```

```
schema.edgeLabel('isLocatedAt').connection('fridgeSensor', 'home').add()
```

The schema for edge labels defines the label *type*, and defines the two vertex labels that are connected by the edge label with `connection()`. The reviewed edge label defines edges between adjacent vertices with the outgoing vertex label `person` and the incoming vertex label `recipe`. By default, edges have [multiple cardinality \(page 454\)](#), but can be defined with single cardinality. Multiple cardinality allows more than one edge with differing property values but the same edge label to be assigned.

## QuickStart Indexing

Index graph schema.

[Indexing \(page 457\)](#) is a complex and highly important subject. Here, several types of indexes are created. Briefly, secondary and materialized indexes are two types of indexes that use the DSE database built-in indexing. Search indexes use DSE Search which is Solr-based. Only one search index per vertex label is allowed, but multiple properties can be included. Property indexes allow meta-properties to be indexed. Edge indexes allow properties on edges to be indexed. Note that indexes are added with `add()` to previously created vertex labels.

### 1. Create the index schema:

```
// *****
// VERTEX INDEX
// *****
// SYNTAX:
// index('index_name').
// [secondary() | materialized() | search()].
// by('propertykey_name').
// [asText() | asString()].
// add()
// *****

schema.vertexLabel('person').index('byName').materialized().by('name').add()
schema.vertexLabel('meal_item').index('byName').materialized().by('name').add()
schema.vertexLabel('ingredient').index('byName').materialized().by('name').add()
//
schema.vertexLabel('recipe').index('byCuisine').materialized().by('cuisine').add()
//
schema.vertexLabel('book').index('byName').materialized().by('name').add()

schema.vertexLabel('meal').index('byType').secondary().by('type').add()

schema.vertexLabel('recipe').index('search').search().
 by('instructions').by('name').by('cuisine').add()
schema.vertexLabel('book').index('search').search().
 by('name').by('publishYear').add()
schema.vertexLabel('location').index('search').search().
 by('geoPoint').withError(0.000009,0.0).add()
schema.vertexLabel('store').index('search').search().by('name').add()
schema.vertexLabel('home').index('search').search().by('name').add()
schema.vertexLabel('fridgeSensor').index('search').search()
```

```

by('cityId').by('sensorId').by('name').add()

// *****
// EDGE INDEX
// *****
// SYNTAX:
// index('index_name').
// [outE('edgeLabel') | inE('edgeLabel')].
// by('propertykey_name').
// add()
// *****

schema.vertexLabel('recipe').index('byStars').inE('reviewed').
 by('stars').ifNotExists().add()
schema.vertexLabel('person').index('ratedByStars').outE('reviewed').
 by('stars').ifNotExists().add()
schema.vertexLabel('person').index('ratedByDate').outE('reviewed').
 by('year').ifNotExists().add()
schema.vertexLabel('person').index('ratedByComments').outE('reviewed').
 by('comment').ifNotExists().add()
schema.vertexLabel('recipe').index('byPersonOrRecipe').bothE('created').
 by('createDate').ifNotExists().add()

// *****
// PROPERTY INDEX using meta-property 'livedIn'
// *****
// SYNTAX:
// index('index_name').
// property('propertykey_name').
// by('meta-propertykey_name').
// add()
// *****

schema.vertexLabel('person').index('byStartYear').
 property('country').by('startYear').add()
schema.vertexLabel('person').index('byEndYear').
 property('country').by('endYear').add()

```

## QuickStart Inspecting schema

Inspect graph schema.

The `schema.describe()` query displays schema you can use to recreate the schema entered. If you enter data without creating schema, you can use this command verify the data types set for each property.

### 1. Examine the schema:

```
schema.describe()
```

In Studio, a portion of the output:

## Using DataStax Enterprise advanced functionality

The screenshot shows the DataStax Enterprise Gremlin console interface. The query entered is `schema.describe()`. The results are displayed in a table with two columns: `index` and `value`. The value column contains a large block of Groovy code representing the schema definition. At the bottom of the interface, it says "Displaying 1 - 1 of 1" and "1 element returned. Duration: 0.013 s."

index	value
0	schema.propertyKey("livedIn").Text().single().create() schema.propertyKey("instructions").Text().single().create() schema.propertyKey("country").Text().multiple().properties("livedIn").create() schema.propertyKey("amount").Text().single().create() schema.propertyKey("gender").Text().single().create() schema.propertyKey("year").int().single().create() schema.propertyKey("calories").int().single().create() schema.propertyKey("stars").int().single().create() schema.propertyKey("ISBN").Text().single().create() schema.propertyKey("name").Text().single().create() schema.propertyKey("comment").Text().single().create() schema.propertyKey("category").Text().single().create() schema.propertyKey("timestamp").Timestamp().single().create() schema.edgeLabel("authored").multiple().create() schema.edgeLabel("rated").multiple().properties("stars").create() schema.edgeLabel("includedIn").multiple().create() schema.edgeLabel("created").multiple().create() schema.edgeLabel("includes").multiple().create() schema.vertexLabel("meal").properties("name").create() schema.vertexLabel("meal").index("byMeal").materialized().by("name").add() schema.vertexLabel("ingredient").properties("name").create() schema.vertexLabel("ingredient").index("byIngredient").materialized().by("name").add() schema.vertexLabel("author").properties("name", "country").create() schema.vertexLabel("book").create() schema.vertexLabel("recipe").properties("name").create() schema.vertexLabel("recipe").index("byRecipe").materialized().by("name").add() schema.vertexLabel("reviewer").properties("name").create() schema.vertexLabel("reviewer").index("byReviewer").materialized().by("name").add() schema.vertexLabel("reviewer").index("ratedByStars").outE("rated").by("stars").add() schema.edgeLabel("rated").connection("reviewer", "recipe").add()

The `schema.describe()` query displays schema you can use to recreate the schema entered. If you enter data without creating schema, you can use this command verify the data types set for each property. While entering data without schema creation is handy while developing and learning, it is strongly recommended against for actual applications. As a reminder, Production mode disallows schema creation once data is loaded.

2. Some groovy steps are useful in the Gremlin query to find specific schema descriptions. For instance, to find only the schema for vertex labels and their indexes, use the following command:

```
schema.describe().split('\n').grep(~/.vertexLabel.*/)
```

In Studio:

The screenshot shows the DataStax Enterprise Studio Gremlin console interface. The query entered is `schema.describe().split('\n').grep(~/.vertexLabel.*/)`. The results are displayed in a table with two columns: `index` and `value`. The value column contains a list of vertex labels and their properties and indexes. At the bottom of the interface, it says "Displaying 1 - 27 of 27" and has navigation buttons for the first, last, and middle pages.

index	value
0	schema.vertexLabel("recipe").partitionKey("recipId").properties("name", "cuisine", "instructions...")
1	schema.vertexLabel("recipe").index("search").search().by("instructions").by("name").by("cuisine")...
2	schema.vertexLabel("recipe").index("byStars").inE("reviewed").by("stars").add()
3	schema.vertexLabel("recipe").index("byPersonOnRecipe").bothE("created").by("createDate").add()
4	schema.vertexLabel("store").partitionKey("storeId").properties("name", "address").create()
5	schema.vertexLabel("store").index("search").search().by("name").add()
6	schema.vertexLabel("meal_item").partitionKey("itemId").properties("name", "servAmt", "macro", "ca...
7	schema.vertexLabel("meal_item").index("byName").materialized().by("name").add()
8	schema.vertexLabel("fridgeSensor").partitionKey("stateId", "cityId").clusteringKey("sensorId").pr...
9	schema.vertexLabel("fridgeSensor").index("search").search().by("cityId").by("sensorId").by("name")...
10	schema.vertexLabel("home").partitionKey("homId").properties("name", "address").create()
11	schema.vertexLabel("home").index("search").search().by("name").add()
12	schema.vertexLabel("ingredient").partitionKey("ingredId").properties("name").create()
13	schema.vertexLabel("ingredient").index("byName").materialized().by("name").add()
14	schema.vertexLabel("person").partitionKey("personId").properties("name", "nickname", "gender", "...")
15	schema.vertexLabel("person").index("byName").materialized().by("name").add()
16	schema.vertexLabel("person").index("ratedByStars").outE("reviewed").by("stars").add()
17	schema.vertexLabel("person").index("ratedByDate").outE("reviewed").by("year").add()
18	schema.vertexLabel("person").index("ratedByComments").outE("reviewed").by("comment").add()
19	schema.vertexLabel("person").index("byStartYear").property("country").by("startYear").add()
20	schema.vertexLabel("person").index("byEndYear").property("country").by("endYear").add()
21	schema.vertexLabel("book").partitionKey("bookId").properties("name", "publishYear", "ISBN", "book...")
22	schema.vertexLabel("book").index("search").search().by("name").by("publishYear").add()
23	schema.vertexLabel("location").partitionKey("locId").properties("name", "geoPoint").create()
24	schema.vertexLabel("location").index("search").search().by("geoPoint").withError(9.0E-6, 0).add()
25	schema.vertexLabel("meal").partitionKey("type", "mealId").create()
26	schema.vertexLabel("meal").index("byType").secondary().by("type").add()

In Gremlin console:

```

==>schema.vertexLabel("recipe").partitionKey("recipeId").properties("name",
 "cuisine", "instructions", "notes").create()
==>schema.vertexLabel("recipe").index("search").search().by("instructions").by("name")
==>schema.vertexLabel("recipe").index("byStars").inE("reviewed").by("stars").add()
==>schema.vertexLabel("recipe").index("byPersonOrRecipe").bothE("created").by("created")
==>schema.vertexLabel("store").partitionKey("storeId").properties("name",
 "address").create()
==>schema.vertexLabel("store").index("search").search().by("name").add()
==>schema.vertexLabel("meal_item").partitionKey("itemId").properties("name",
 "servAmt", "macro", "calories").create()
==>schema.vertexLabel("meal_item").index("byName").materialized().by("name").add()
==>schema.vertexLabel("fridgeSensor").partitionKey("stateId",
 "cityId").clusteringKey("sensorId").properties("name").create()
==>schema.vertexLabel("fridgeSensor").index("search").search().by("cityId").by("sensorId")
==>schema.vertexLabel("home").partitionKey("homeId").properties("name",
 "address").create()
==>schema.vertexLabel("home").index("search").search().by("name").add()
==>schema.vertexLabel("ingredient").partitionKey("ingredId").properties("name").create()
==>schema.vertexLabel("ingredient").index("byName").materialized().by("name").add()
==>schema.vertexLabel("person").partitionKey("personId").properties("name",
 "nickname", "gender", "calGoal", "macroGoal", "country").create()
==>schema.vertexLabel("person").index("byName").materialized().by("name").add()
==>schema.vertexLabel("person").index("ratedByStars").outE("reviewed").by("stars").add()
==>schema.vertexLabel("person").index("ratedByDate").outE("reviewed").by("year").add()
==>schema.vertexLabel("person").index("ratedByComments").outE("reviewed").by("comment")
==>schema.vertexLabel("person").index("byStartYear").property("country").by("startYear")
==>schema.vertexLabel("person").index("byEndYear").property("country").by("endYear")
==>schema.vertexLabel("book").partitionKey("bookId").properties("name",
 "publishYear", "ISBN", "bookDiscount").create()
==>schema.vertexLabel("book").index("search").search().by("name").by("publishYear").add()
==>schema.vertexLabel("location").partitionKey("locId").properties("name",
 "geoPoint").create()
==>schema.vertexLabel("location").index("search").search().by("geoPoint").withError(9
 0.0).add()
==>schema.vertexLabel("meal").partitionKey("type",
 "mealId").create()
==>schema.vertexLabel("meal").index("byType").secondary().by("type").add()

```

Additional steps can split the output per newline and grep for a string as shown for `index`. The Gremlin variant used here is based on [Apache Groovy](#), so any Groovy commands can be used to manipulate graph traversals. Apache Groovy is a language that smoothly integrates with Java to provide scripting capabilities.

## QuickStart Modifying schema

Modify graph schema.

Schema can be modified after creation, using schema `add()` to add additional properties, vertex labels, edge labels, or indexes, as shown in the schema creation above. The `drop()` step can also be used to remove any element; see [propertyKey \(page 642\)](#), [vertexLabel \(page 644\)](#), and [edgeLabel \(page 637\)](#). The data type of a property, however, cannot be changed, without removing and recreating the property. While entering data without

schema creation is useful when developing and learning, it is strongly recommended against for actual applications. As a reminder, Production mode disallows schema creation once data is loaded.

1. Create a property to drop in the next step:

```
schema.propertyKey('nonsenseToDelete').Int().single().create()
schema.describe().split('\n').grep(~/.*nonsenseToDelete.*/)
```

In Studio:

Gremlin

```
schema.propertyKey('nonsenseToDelete').Int().single().create()
schema.describe().split('\n').grep(~/.*nonsenseToDelete.*/)
```



```
schema.propertyKey("nonsenseToDelete").Int().single().create()
```

2. Drop the property:

```
schema.propertyKey('nonsenseToDelete').drop()
schema.describe().split('\n').grep(~/.*nonsenseToDelete.*/)
```

In Studio:

Gremlin

```
schema.propertyKey('nonsenseToDelete').drop()
schema.describe().split('\n').grep(~/.*nonsenseToDelete.*/)
```

Success - No Data Returned

Success. 0 elements returned. Duration: 1.254 s.

## QuickStart Add data

Adding data to a graph.

Now that schema is created, add more vertices and edges using the following script. To explore more connections in the recipe data model, more vertices and edges are input into the graph. A script, *generateRecipe.groovy*, is entered and then executed by the remote Gremlin server. Note the first command, `g.v().drop().iterate()`; this command can be used to drop all vertex and edge data from the graph before reading in new data. In Studio, be sure to select the **Graph view** after running the script.

### Adding more data

1. Run *generateRecipe.groovy* in either Studio or the Gremlin console:

If running in Gremlin console, use the following command to load:

```
:load /tmp/generateRecipe.groovy
```

replacing "/tmp" with the directory where you write the script. In Studio, run the script within a cell.

```
// Generates all Recipe Toy Graph vertices and edges except Reviews

// Add all vertices and edges for Recipe
g.V().drop().iterate()

// author vertices
juliaChild = graph.addVertex(label, 'person', 'personId', 1,
 'name', 'Julia Child', 'gender', 'F')
simoneBeck = graph.addVertex(label, 'person', 'personId', 2, 'name',
 'Simone Beck', 'gender', 'F')
louisetteBertholie = graph.addVertex(label, 'person', 'personId', 3,
 'name', 'Louisette Bertholie', 'gender', 'F')
patriciaSimon = graph.addVertex(label, 'person', 'personId', 4,
 'name', 'Patricia Simon', 'gender', 'F')
aliceWaters = graph.addVertex(label, 'person', 'personId', 5,
 'name', 'Alice Waters', 'gender', 'F')
patriciaCurtan = graph.addVertex(label, 'person', 'personId', 6,
 'name', 'Patricia Curtan', 'gender', 'F')
kelsieKerr = graph.addVertex(label, 'person', 'personId', 7, 'name',
 'Kelsie Kerr', 'gender', 'F')
fritzStreiff = graph.addVertex(label, 'person', 'personId', 8,
 'name', 'Fritz Streiff', 'gender', 'M')
emerilLagasse = graph.addVertex(label, 'person', 'personId', 9,
 'name', 'Emeril Lagasse', 'gender', 'M')
jamesBeard = graph.addVertex(label, 'person', 'personId', 10,
 'name', 'James Beard', 'gender', 'M')

// book vertices
artOfFrenchCookingVolOne = graph.addVertex(label, 'book', 'bookId',
 1001, 'name', 'The Art of French Cooking, Vol. 1', 'publishYear',
 1961)
simcasCuisine = graph.addVertex(label, 'book', 'bookId', 1002,
 'name', "Simca's Cuisine: 100 Classic French Recipes for Every
 Occasion", 'publishYear', 1972, 'ISBN', '0-394-40152-2')
frenchChefCookbook = graph.addVertex(label, 'book', 'bookId', 1003,
 'name', 'The French Chef Cookbook', 'publishYear', 1968, 'ISBN',
 '0-394-40135-2')
artOfSimpleFood = graph.addVertex(label, 'book', 'bookId', 1004,
 'name', 'The Art of Simple Food: Notes, Lessons, and Recipes
 from a Delicious Revolution', 'publishYear', 2007, 'ISBN',
 '0-307-33679-4')

// recipe vertices
beefBourguignon = graph.addVertex(label, 'recipe', 'recipeId', 2001,
 'name', 'Beef Bourguignon', 'instructions', 'Braise the beef. Saute
```

```

the onions and carrots. Add wine and cook in a dutch oven at 425
degrees for 1 hour.', 'notes', 'Takes a long time to make.')
ratatouille = graph.addVertex(label, 'recipe', 'recipeId', 2002,
 'name', 'Rataouille', 'instructions', 'Peel and cut the eggplant.
 Make sure you cut eggplant into lengthwise slices that are about 1-
 inch wmyIde, 3-inches long, and 3/8-inch thick', 'notes', "I've made
 this 13 times.")
saladeNicoise = graph.addVertex(label, 'recipe', 'recipeId', 2003,
 'name', 'Salade Nicoise', 'instructions', 'Take a salad bowl or
 platter and line it with lettuce leaves, shortly before serving.
 Drizzle some olive oil on the leaves and dust them with salt.',
 'notes', '')
wildMushroomStroganoff = graph.addVertex(label, 'recipe',
 'recipeId', 2004, 'name', 'Wild Mushroom Stroganoff',
 'instructions', 'Cook the egg noodles according to the package
 directions and keep warm. Heat 1 1/2 tablespoons of the oliveoil in
 a large saute pan over medium-high heat.', 'notes', 'Good for Jan
 and Bill.')
spicyMeatloaf = graph.addVertex(label, 'recipe', 'recipeId', 2005,
 'name', 'Spicy Meatloaf', 'instructions', 'Preheat the oven to 375
 degrees F. Cook bacon in a large skillet over medium heat until
 very crisp and fat has rendered, 8-10 minutes.', 'notes', '')
oystersRockefeller = graph.addVertex(label, 'recipe', 'recipeId',
 2006, 'name', 'Oysters Rockefeller', 'instructions', 'Saute
 the shallots, celery, herbs, and seasonings in 3 tablespoons of
 the butter for 3 minutes. Add the watercress and let it wilt.',
 'notes', '')
carrotSoup = graph.addVertex(label, 'recipe', 'recipeId', 2007,
 'name', 'Carrot Soup', 'instructions', 'In a heavy-bottomed pot,
 melt the butter. When it starts to foam, add the onions and thyme
 and cook over medium-low heat until tender, about 10 minutes.',
 'notes', 'Quick and easy.')
roastPorkLoin = graph.addVertex(label, 'recipe', 'recipeId',
 2008, 'name', 'Roast Pork Loin', 'instructions', 'The day before,
 separate the meat from the ribs, stopping about 1 inch before the
 end of the bones. Season the pork liberally inside and out with
 salt and pepper and refrigerate overnight.', 'notes', 'Love this
 one!')

// ingredients vertices
beef = graph.addVertex(label, 'ingredient', 'ingredId', 3001,
 'name', 'beef')
onion = graph.addVertex(label, 'ingredient', 'ingredId', 3002,
 'name', 'onion')
mashedGarlic = graph.addVertex(label, 'ingredient', 'ingredId',
 3003, 'name', 'mashed garlic')
butter = graph.addVertex(label, 'ingredient', 'ingredId', 3004,
 'name', 'butter')
tomatoPaste = graph.addVertex(label, 'ingredient', 'ingredId', 3005,
 'name', 'tomato paste')
eggplant = graph.addVertex(label, 'ingredient', 'ingredId', 3006,
 'name', 'eggplant')
zucchini = graph.addVertex(label, 'ingredient', 'ingredId', 3007,
 'name', 'zucchini')

```

```

oliveOil = graph.addVertex(label, 'ingredient', 'ingredId', 3008,
 'name', 'olive oil')
yellowOnion = graph.addVertex(label, 'ingredient', 'ingredId', 3009,
 'name', 'yellow onion')
greenBean = graph.addVertex(label, 'ingredient', 'ingredId', 3010,
 'name', 'green beans')
tuna = graph.addVertex(label, 'ingredient', 'ingredId', 3011,
 'name', 'tuna')
tomato = graph.addVertex(label, 'ingredient', 'ingredId', 3012,
 'name', 'tomato')
hardBoiledEgg = graph.addVertex(label, 'ingredient', 'ingredId',
 3013, 'name', 'hard-boiled egg')
eggNoodles = graph.addVertex(label, 'ingredient', 'ingredId', 3014,
 'name', 'egg noodles')
mushroom = graph.addVertex(label, 'ingredient', 'ingredId', 3015,
 'name', 'mushrooms')
bacon = graph.addVertex(label, 'ingredient', 'ingredId', 3016,
 'name', 'bacon')
celery = graph.addVertex(label, 'ingredient', 'ingredId', 3017,
 'name', 'celery')
greenBellPepper = graph.addVertex(label, 'ingredient', 'ingredId',
 3018, 'name', 'green bell pepper')
groundBeef = graph.addVertex(label, 'ingredient', 'ingredId', 3019,
 'name', 'ground beef')
porkSausage = graph.addVertex(label, 'ingredient', 'ingredId', 3020,
 'name', 'pork sausage')
shallot = graph.addVertex(label, 'ingredient', 'ingredId', 3021,
 'name', 'shallots')
chervil = graph.addVertex(label, 'ingredient', 'ingredId', 3022,
 'name', 'chervil')
fennel = graph.addVertex(label, 'ingredient', 'ingredId', 3023,
 'name', 'fennel')
parsley = graph.addVertex(label, 'ingredient', 'ingredId', 3024,
 'name', 'parsley')
oyster = graph.addVertex(label, 'ingredient', 'ingredId', 3025,
 'name', 'oyster')
pernod = graph.addVertex(label, 'ingredient', 'ingredId', 3026,
 'name', 'Pernod')
thyme = graph.addVertex(label, 'ingredient', 'ingredId', 3027,
 'name', 'thyme')
carrot = graph.addVertex(label, 'ingredient', 'ingredId', 3028,
 'name', 'carrots')
chickenBroth = graph.addVertex(label, 'ingredient', 'ingredId',
 3029, 'name', 'chicken broth')
porkLoin = graph.addVertex(label, 'ingredient', 'ingredId', 3030,
 'name', 'pork loin')
redWine = graph.addVertex(label, 'ingredient', 'ingredId', 3031,
 'name', 'red wine')

// meal vertices
meal1 = graph.addVertex(label, 'meal', 'mealId', 4001, 'type',
 'lunch')
meal2 = graph.addVertex(label, 'meal', 'mealId', 4002, 'type',
 'lunch')

```

```

meal3 = graph.addVertex(label, 'meal', 'mealId', 4003, 'type',
 'lunch')
meal4 = graph.addVertex(label, 'meal', 'mealId', 4004, 'type',
 'lunch')
meal5 = graph.addVertex(label, 'meal', 'mealId', 4005, 'type',
 'breakfast')
meal6 = graph.addVertex(label, 'meal', 'mealId', 4006, 'type',
 'snack')
meal7 = graph.addVertex(label, 'meal', 'mealId', 4007, 'type',
 'dinner')
meal8 = graph.addVertex(label, 'meal', 'mealId', 4008, 'type',
 'dinner')

// author-book edges
juliaChild.addEdge('authored', artOfFrenchCookingVolOne)
simoneBeck.addEdge('authored', artOfFrenchCookingVolOne)
louisetteBertholie.addEdge('authored', artOfFrenchCookingVolOne)
simoneBeck.addEdge('authored', simcasCuisine)
patriciaSimon.addEdge('authored', simcasCuisine)
juliaChild.addEdge('authored', frenchChefCookbook)
aliceWaters.addEdge('authored', artOfSimpleFood)
patriciaCurtan.addEdge('authored', artOfSimpleFood)
kelsieKerr.addEdge('authored', artOfSimpleFood)
fritzStreiff.addEdge('authored', artOfSimpleFood)

// author - recipe edges
juliaChild.addEdge('created', beefBourguignon, 'createDate',
 1961-01-01)
juliaChild.addEdge('created', ratatouille, 'createDate', 1965-02-02)
juliaChild.addEdge('created', saladeNicoise, 'createDate',
 1962-03-03)
emerilLagasse.addEdge('created', wildMushroomStroganoff,
 'createDate', 2003-04-04)
emerilLagasse.addEdge('created', spicyMeatloaf, 'createDate',
 2000-05-05)
aliceWaters.addEdge('created', carrotSoup, 'createDate', 1995-06-06)
aliceWaters.addEdge('created', roastPorkLoin, 'createDate',
 1996-07-07)
jamesBeard.addEdge('created', oystersRockefeller, 'createDate',
 1970-01-01)

// recipe - ingredient edges
beefBourguignon.addEdge('includedIn', beef, 'amount', '2 lbs')
beefBourguignon.addEdge('includedIn', onion, 'amount', '1 sliced')
beefBourguignon.addEdge('includedIn', mashedGarlic, 'amount', '2
 cloves')
beefBourguignon.addEdge('includedIn', butter, 'amount', '3.5 Tbsp')
beefBourguignon.addEdge('includedIn', tomatoPaste, 'amount', '1
 Tbsp')
ratatouille.addEdge('includedIn', eggplant, 'amount', '1 lb')
ratatouille.addEdge('includedIn', zucchini, 'amount', '1 lb')
ratatouille.addEdge('includedIn', mashedGarlic, 'amount', '2
 cloves')
ratatouille.addEdge('includedIn', oliveOil, 'amount', '4-6 Tbsp')

```

```

ratatouille.addEdge('includedIn', yellowOnion, 'amount', '1 1/2 cups
or 1/2 lb thinly sliced')
saladeNicoise.addEdge('includedIn', oliveOil, 'amount', '2-3 Tbsp')
saladeNicoise.addEdge('includedIn', greenBean, 'amount', '1 1/2 lbs
blanched, trimmed')
saladeNicoise.addEdge('includedIn', tuna, 'amount', '8-10 ozs oil-
packed, drained and flaked')
saladeNicoise.addEdge('includedIn', tomato, 'amount', '3 or 4 red,
peeled, quartered, cored, and seasoned')
saladeNicoise.addEdge('includedIn', hardBoiledEgg, 'amount', '8
halved lengthwise')
wildMushroomStroganoff.addEdge('includedIn', eggNoodles, 'amount',
'16 ozs wmyIde')
wildMushroomStroganoff.addEdge('includedIn', mushroom, 'amount', '2
lbs wild or exotic, cleaned, stemmed, and sliced')
wildMushroomStroganoff.addEdge('includedIn', yellowOnion, 'amount',
'1 cup thinly sliced')
spicyMeatloaf.addEdge('includedIn', bacon, 'amount', '3 ozs diced')
spicyMeatloaf.addEdge('includedIn', onion, 'amount', '2 cups finely
chopped')
spicyMeatloaf.addEdge('includedIn', celery, 'amount', '2 cups finely
chopped')
spicyMeatloaf.addEdge('includedIn', greenBellPepper, 'amount', '1/4
cup finely chopped')
spicyMeatloaf.addEdge('includedIn', porkSausage, 'amount', '3/4 lbs
hot')
spicyMeatloaf.addEdge('includedIn', groundBeef, 'amount', '1 1/2 lbs
chuck')
oystersRockefeller.addEdge('includedIn', shallot, 'amount', '1/4 cup
chopped')
oystersRockefeller.addEdge('includedIn', celery, 'amount', '1/4 cup
chopped')
oystersRockefeller.addEdge('includedIn', chervil, 'amount', '1 tsp')
oystersRockefeller.addEdge('includedIn', fennel, 'amount', '1/3 cup
chopped')
oystersRockefeller.addEdge('includedIn', parsley, 'amount', '1/3 cup
chopped')
oystersRockefeller.addEdge('includedIn', oyster, 'amount', '2 dozen
on the half shell')
oystersRockefeller.addEdge('includedIn', pernod, 'amount', '1/3
cup')
carrotSoup.addEdge('includedIn', butter, 'amount', '4 Tbsp')
carrotSoup.addEdge('includedIn', onion, 'amount', '2 medium sliced')
carrotSoup.addEdge('includedIn', thyme, 'amount', '1 sprig')
carrotSoup.addEdge('includedIn', carrot, 'amount', '2 1/2 lbs,
peeled and sliced')
carrotSoup.addEdge('includedIn', chickenBroth, 'amount', '6 cups')
roastPorkLoin.addEdge('includedIn', porkLoin, 'amount', '1 bone-in,
4-rib')
roastPorkLoin.addEdge('includedIn', redWine, 'amount', '1/2 cup')
roastPorkLoin.addEdge('includedIn', chickenBroth, 'amount', '1 cup')

// book - recipe edges
beefBourguignon.addEdge('includedIn', artOfFrenchCookingVolOne)

```

```

saladeNicoise.addEdge('includedIn', artOfFrenchCookingVolOne)
carrotSoup.addEdge('includedIn', artOfSimpleFood)

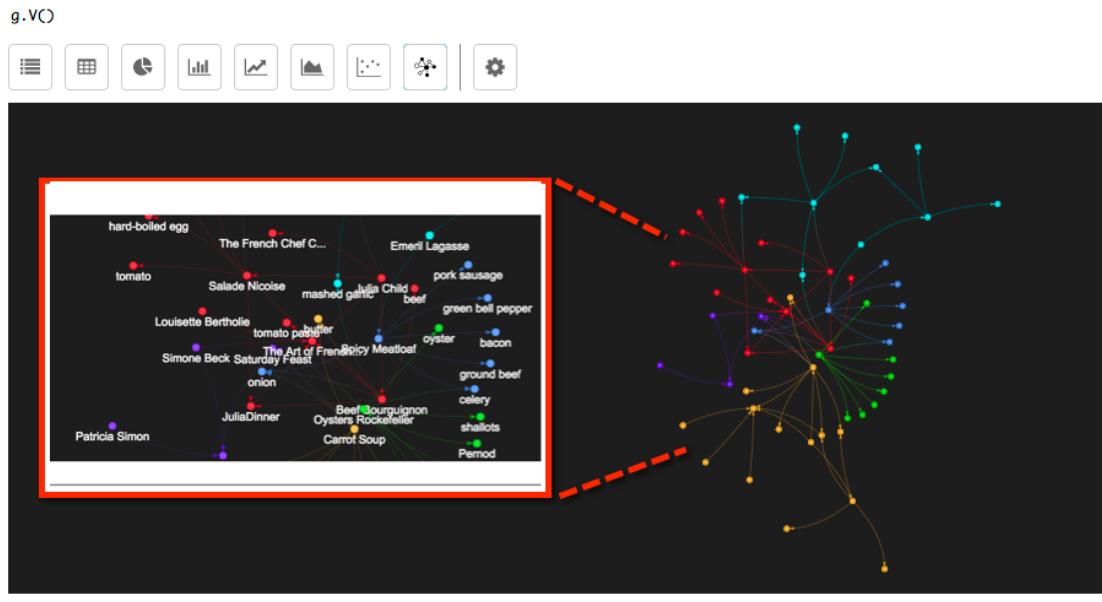
// meal - recipe edges
beefBourguignon.addEdge('includedIn', meal1)
saladeNicoise.addEdge('includedIn', meal1)
carrotSoup.addEdge('includedIn', meal4)
roastPorkLoin.addEdge('includedIn', meal4)

// meal - book edges
meal7.addEdge('includedIn', artOfFrenchCookingVolOne)
meal8.addEdge('includedIn', artOfSimpleFood)
meal5.addEdge('includedIn', frenchChefCookbook)
g.V()

```

In Studio:

Figure 22: Data for the Recipe Toy Graph



The `g.V()` command at the end of the script displays all the vertices created.

In Gremlin console:

```

// A series of returns for vertices and edges will mark the
// successful completion of the script
// Sample vertex
==>v[~label=meal, type="dinner", mealId=4008]
// Sample edge
==>e[~label=includedIn, ~out_vertex={~label=meal, type="dinner",
mealId=4008},
~in_vertex={~label=book, bookId=1004},
~local_id=5dec6ef7-0562-11e8-a4a1-4b3271ac7767}]

```

```
[{~label=meal, type="dinner", mealId=4008}-includedIn-
>{~label=book, bookId=1004}]
```

2. If a vertex count is run as either a transactional query or analytical query, there is now a higher count of 61 vertices. Run the vertex count again:

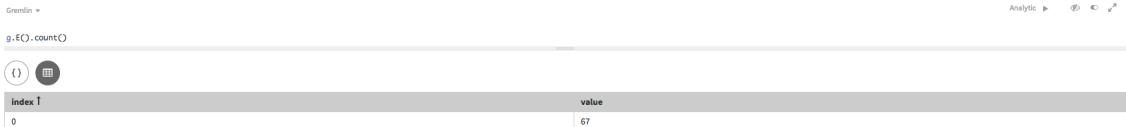
```
g.V().count()
```



The [DSE Graph Loader \(page 471\)](#) is the recommended method for scripting data loading. Using `graph.addVertex` or `g.addV()` are only practical for small toy graphs like the recipe example.

3. Similarly, the edge count can be run, to discover the higher edge count of 67:

```
g.E().count()
```



## QuickStart Exploring traversals

Explore graph data with query traversals.

Exploring the graph with graph traversals can lead to interesting conclusions. Here we'll explore a number of traversals, to show off the power of Gremlin in creating simple queries.

1. All queries can be profiled to see what the query path is and how the query performs.

```
g.V().has('person', 'name', 'Julia Child').profile()
```

In Studio:



Clicking on the bars in the graph in Studio will show more detail about underlying processes in the database.

In Gremlin console:

```
==>Traversal Metrics
Step
Count Traversers Time (ms) % Dur
=====
DsegGraphStep(vertex,[],(label = person & name ...
 1 1 10.097 65.69
query-optimizer
 1.848
 _condition=((label = person & name = Julia Child) & (true))
query-setup
 0.065
 _isFitted=true
 _isSorted=false
 _isScan=false
index-query
 1.645
 _indexType=Materialized
 _usesCache=false
 _statement=SELECT "personId" FROM
"DSE_GRAPH_QUICKSTART"."person_p_byName" WHERE "name" = ?
LIMIT ?; wit
 h params (java.lang.String) Julia Child,
(java.lang.Integer) 50000
 _options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
c=true}
DsegPropertyLoadStep
 1 1 5.274 34.31
 >TOTAL
 - - 15.372 -

```

In all the following queries, to investigate what happens, and why some queries are more efficient than others, try adding `.profile()` to any query will show you information similar to the information above.

2. With several *person* vertices in the graph, a specific *name* must be given to find a particular vertex. This traversal gets the stored vertex information for the vertex that *has the name of Julia Child*. Note that the constraint that the vertex is an *author* is also included in the `has()` clause. Graph queries will have lower latency if the query is more specific, and the `has()` step is a more tool for narrowing the search.

```
g.V().has('person', 'name', 'Julia Child')
```

Running the query in Studio will display the vertex id, label and all property values.

In Gremlin console, this query will only display the vertex id, and the `valueMap()` step must be appended to get the property values.

3. In this next traversal, `has()` filters vertex properties by `name = Julia Child` as seen above. The traversal step `outE()` discovers the outgoing edges from that vertex with the `authored` label.

```
g.V().has('name', 'Julia Child').outE('authored')
```

In Studio, either the listing of the **Raw JSON view** edge information:

The screenshot shows the Gremlin tab in DataStax Studio with the following command:

```
g.V().has('person', 'name', 'Julia Child').outE('authored')
```

The output is a JSON array of edges:

```
[
 {
 "id": "-{label=authored, ~out_vertex=~label=person, personId=1}, ~local_id=5deac140-0562-11e8-a4a1-4b3271ac7767, ~in_vertex=~label=book, bookId=1001}",
 "label": "authored",
 "type": "edge",
 "inV": "book",
 "outVLabel": "person",
 "inVLabel": "book",
 "outV": "-{label=book, bookId=1001}",
 "outF": "-{label=person, personId=1}"
 },
 {
 "id": "-{label=authored, ~out_vertex=~label=person, personId=1}, ~local_id=5deac145-0562-11e8-a4a1-4b3271ac7767, ~in_vertex=~label=book, bookId=1003}",
 "label": "authored",
 "type": "edge",
 "inV": "book",
 "outVLabel": "person",
 "inVLabel": "book",
 "outV": "-{label=book, bookId=1003}",
 "outF": "-{label=person, personId=1}"
 }
]
```

or the **Graph view** graph visualization where scrolling over a vertex provides additional information.

The screenshot shows the Graph view tab in DataStax Studio with the same Gremlin command. It displays a graph with two nodes: a green circle labeled "Chef Julia Child" and a blue circle labeled "book:The French Chef Cookbook". A blue arrow points from the green node to the blue node, representing the "authored" edge. To the left of the graph, there is a detailed tooltip for the blue node containing its properties:

**book:The French Chef Cookbook**

- id** : {~label=book, bookId=1003}
- label** : book
- bookId** : 1003
- ISBN** : 0-394-40135-2
- name** : The French Chef Cookbook
- publishYear** : 1968

In Gremlin console:

```
==>e[{~label=authored, ~out_vertex=~label=person, personId=1},
 ~in_vertex=~label=book, bookId=1001,
 ~local_id=5deac140-0562-11e8-a4a1-4b3271ac7767}]
[{~label=person, personId=1}-authored->{~label=book, bookId=1001}]
```

```
==>e[{~label=authored, ~out_vertex={~label=person, personId=1},
 ~in_vertex={~label=book, bookId=1003},
 ~local_id=5deac145-0562-11e8-a4a1-4b3271ac7767}]
[{~label=person, personId=1}-authored->{~label=book, bookId=1003}]
```

4. Spark SQL can also be used to discover information for a set of vertices or edges that match particular conditions. Here, all the edges with a `createdate` greater than May 1, 1975 are returned. Note the lack of camel case column names in Spark SQL.

```
SELECT * FROM DSE_GRAPH_QUICKSTART_edges WHERE createdate >
'1975-05-01';
```

In Studio:

Spark SQL - Database: dse_graph															
SELECT * FROM DSE_GRAPH_QUICKSTART_edges WHERE createdate > '1975-05-01';															
index	T	id	src	dst	-label	meadate	amount	expiredate	since	stars	time	year	comment	numserv	createdate
0		5deb8494-0562-11e8-a4a1-4b3271ac7767	person:AAAAABAAAAAU=	recipe:AAAAAAAB9g=	created										1975-06-07
1		5deb8495-0562-11e8-a4a1-4b3271ac7767	person:AAAAABAAAAU=	recipe:AAAAAAAB9g=	created										1975-06-06
2		5debaba0-0562-11e8-a4a1-4b3271ac7767	person:AAAAABAAAQo=	recipe:AAAAAAAB9y=	created										1975-05-23
3		5deb5d80-0562-11e8-a4a1-4b3271ac7767	person:AAAABAAAAAE=	recipe:AAAAABAAAB9E=	created										1975-05-14
4		5deb8490-0562-11e8-a4a1-4b3271ac7767	person:AAAABAAAAAE=	recipe:AAAAAAAB9j=	created										1975-05-16
5		5deb8491-0562-11e8-a4a1-4b3271ac7767	person:AAAABAAAAAE=	recipe:AAAAAAAB9M=	created										1975-05-11
6		5deb8492-0562-11e8-a4a1-4b3271ac7767	person:AAAABAAAAAk=	recipe:AAAAAAAB9Q=	created										1975-06-19
7		5deb8493-0562-11e8-a4a1-4b3271ac7767	person:AAAABAAAAk=	recipe:AAAAAAAB9U=	created										1975-06-14

Displaying 1 - 8 of 8 results for the last statement

< 1 >

The data presented in Spark SQL is different than the data stored in the database tables for graph. In Spark SQL tables, the source and destination vertices are listed for an edge, along with the edge label and properties.

5. If instead, you want to query for the books that all people have written, the query must be modified. The previous example retrieved edges, but not the adjacent book vertices. Add a traversal step `inv()` to find all the vertices that connect to the outgoing edges, then print the book titles of those vertices. Notice how the chained traversal steps go from the vertices along outgoing edges to the adjacent vertices with `V().outE().inv()`. The outgoing edges are given a particular filter value, `authored`.

```
g.V().outE('authored').inv().values('name')
```

**In**

GREMLIN >

```
g.V().outE('authored').inV().values('name')
```

index	↑	value
0		The Art of French Cooking, Vol. 1
1		Simca's Cuisine: 100 Classic French Recipes for Every Occasion
2		Simca's Cuisine: 100 Classic French Recipes for Every Occasion
3		The Art of Simple Food: Notes, Lessons, and Recipes from a Delicious Revolution
4		The Art of Simple Food: Notes, Lessons, and Recipes from a Delicious Revolution
5		The Art of French Cooking, Vol. 1
6		The French Chef Cookbook
7		The Art of Simple Food: Notes, Lessons, and Recipes from a Delicious Revolution
8		The Art of French Cooking, Vol. 1
9		The Art of Simple Food: Notes, Lessons, and Recipes from a Delicious Revolution

Studio:

and a similar listing in Gremlin console.

- Notice that the book titles are duplicated in the resulting list, because a listing is returned for each author. If a book has three authors, three listings are returned. The traversal step `dedup()` can eliminate the duplication.

```
g.V().outE('authored').inV().values('name').dedup()
```

**In**

GREMLIN >

```
g.V().outE('authored').inV().values('name').dedup()
```

index	↑	value
0		The Art of French Cooking, Vol. 1
1		Simca's Cuisine: 100 Classic French Recipes for Every Occasion
2		The Art of Simple Food: Notes, Lessons, and Recipes from a Delicious Revolution
3		The French Chef Cookbook

Studio:

and a similar listing in Gremlin console.

- Refine the traversal by reinserting the `has()` step for a particular author. Find all the books authored by *Julia Child*.

```
g.V().has('name', 'Julia
Child').outE('authored').inV().values('name')
```

**In**

GREMLIN >

```
g.V().has('name', 'Julia Child').outE('authored').inV().values('name')
```

index	↑	value
0		The Art of French Cooking, Vol. 1
1		The French Chef Cookbook

Studio:

and a similar listing in Gremlin console.

- The previous example and this example accomplish the same result. However, the number of traversal steps and the type of traversal steps can affect performance. The traversal step `outE()` should be only used if the edges are explicitly required. In this

example, the edges are traversed to get information about connected vertices, but the edge information is not important to the query.

```
g.V().has('name', 'Julia Child').out('authored').values('name')
```

In

The screenshot shows the Gremlin Studio interface with a table output. The table has two columns: 'index' and 'value'. The 'value' column contains two entries: 'The Art of French Cooking, Vol. 1' and 'The French Chef Cookbook'.

index	↑ value
0	The Art of French Cooking, Vol. 1
1	The French Chef Cookbook

Studio:

and a similar listing in Gremlin console.

The traversal step `out()` retrieves the connected book vertices based on the edge label `authored` without retrieving the edge information. In a larger graph traversal, this subtle difference in the traversal can become a latency issue.

9. Additional traversal steps continue to fine-tune the results. Adding another chained `has` traversal step finds only books authored by Julia Child published after 1967. This example also displays the use of the `gt`, or *greater than* function.

```
g.V().has('name', 'Julia Child').out('authored').has('publishYear', gt(1967)).values('name')
```

In Studio:

The screenshot shows the Gremlin Studio interface with a single result displayed: 'The French Chef Cookbook'.

and a similar listing in Gremlin console.

10. When developing or testing, oftentimes checking the number of vertices with each vertex label can confirm that data was read. To find the number of vertices by vertex label, use the traversal step `label()` followed by the traversal step `groupCount()`. The step `groupCount()` is useful for aggregating results from a previous step. Although this query can be run in real-time, it is an excellent example of a query that should be run in analytic (OLAP) mode. In Studio, under the run arrow, select **Execute using analytic engine (Spark)** before running.

```
g.V().label().groupCount()
```

> g.V().label().groupCount()

index	values	keys
0	3	meal
1	31	ingredient
2	10	author
3	4	book
4	8	recipe

## 11. An alternative method for getting the group count with Spark SQL uses:

```
SELECT `~label` AS label, COUNT(*) AS label_count FROM
DSE_GRAPH_QUICKSTART_vertices GROUP BY label;
```

## QuickStart Writing and reading data

Writing and reading graph data.

Writing data from DSE Graph to a file is most easily accomplished with the `graph.io()` command. The [DSE Graph Loader \(page 471\)](#) is the most appropriate tool for reading in data from files or other sources.

1. Write your data to an output file to save or exchange information. A Gryo file is a binary format file that can reload data to DSE Graph. In this next command, graph I/O writes the entire graph to a file. Other file formats can be written by substituting `gryo()` with `graphml()` or `graphson()`.

```
graph.io(gryo()).writeGraph("/tmp/recipe.gryo")
```

**Note:** `graph.io()` is disabled in sandbox mode.

In Studio:

```
GREMLIN <
| graph.io(gryo()).writeGraph("/tmp/recipe.gryo")
null
1 element returned. Duration: 0.192 s.
```

In Gremlin console:

```
==>null
```

2. To load a Gryo file, use the `graphloader`, after creating a mapping script:

```
graphloader mappingGRYO.groovy -graph recipe -address localhost
```

Details about loading Gryo data are found in [Loading Gryo Data \(page 504\)](#), in [Using DSE Graph Loader \(page 471\)](#).

## QuickStart Listing graphs

How to list graphs.

1. To discover all graphs that exist, use a system command:

```
system.graphs()
```

2. To display all the tables within Spark SQL:

```
SHOW TABLES FROM DSE_GRAPH_QUICKSTART;
```

## Increase your knowledge

Further increase your knowledge of DSE Graph.

Further adventures in traversing can be found in [Creating queries using traversals](#). If you want to explore various loading options, check out [DSE Graph Loader](#) or [Using DSE Graph](#).

DataStax also hosts a [DSE Graph self-paced course](#) on DataStax Academy; register for a free account to access the course.

## DSE Graph, OLTP, and OLAP

OLTP and OLAP are different processing methods that DSE Graph uses to search graph databases.

Online analytical processing (OLAP) is characterized by a large number of short, online transactions for very fast query processing. OLTP is typically used for data entry and retrieval with transaction-oriented applications. Online analytical processing (OLAP) is typically used to perform multidimensional analysis of data, doing complex calculations on aggregated historical data.

OLTP applications require sub-second response times, whereas OLAP applications take much longer to finish queries. Graph databases are a random access data system. In these databases, OLAP traversals do a linear scan of all vertices in the graph. Conversely, OLTP traversals are localized to a particular subgraph of the global graph. OLTP traversals leverage indexes to "jump" in to a particular vertex in the graph before starting a scan on the subgraph.

### OLTP queries

OLTP queries are best for questions that require access to a limited subset of the entire graph. OLTP queries use filters to limit the number of vertices that will be walked to find answers. DSE Graph co-locates vertices with their edges and adjacent neighbors. When a subgraph is specified in a traversal using indexes, the number of requests to disk are

reduced to locate and write the requested subgraph to memory. Once in memory, the traversal performs a link walk from vertex to vertex along the edges.

## OLAP queries

OLAP queries are best for questions that must access a significant portion of the data stored in a graph. Using the previous method to evaluate OLAP queries will not be efficient, so a different process is used. When OLAP queries are processed, the entire graph is interpreted as a sequence of star graphs, each composed of a single vertex, along with its properties, incident edges, and the edges' properties. The star graphs are linearly processed, jumping from one star graph to the next until all star graphs are processed and an aggregation of the discovered data is completed.

## Principles for writing graph traversals

Understanding these underlying principles can lead to writing better graph traversals to query the graph data. A simple example illustrates the differences. Using the food graph, the query is “How many recipes has Julia Child created?”

Consider the following graph traversal:

```
g.V().in().has('name', 'Julia Child').count()
==>6
```

This traversal completes the following processing:

1. Looks at all vertices.
2. Walks the incoming edges.
3. Finds the adjacent vertices that have the property key of `name` and property value of Julia Child.
4. Counts the number of vertices.

This graph traversal is a classic OLAP traversal, which must touch all vertices and does not use indexing. The count returned includes all vertices with edges to Julia Child, and not just the recipes, so as shown later, the count is incorrect and too high.

Consider the number of elements that must be traversed to complete this query. DSE Graph has profiling that aids in analyzing the traversal:

```
gremlin> g.V().in().has('name', 'Julia Child').count().profile()
==>Traversal Metrics
Step Count
Traversers Time (ms) % Dur
=====
DseGraphStep(vertex,[])
 61 28.932 18.71
query-optimizer
 0.563
 _condition=((label = FridgeSensor | label = author | label = book |
 label = ingredient | label = meal |
 label = recipe | label = reviewer) & (true))
query-setup
 0.048
 _isFitted=true
```

```

 _isSorted=false
 _isScan=true
index-query
 0.979
 _usesCache=false
 _statement=SELECT "city_id", "sensor_id" FROM
"DSEQuickStart"."FridgeSensor_p" WHERE "~~vertex_exists" =
 ? LIMIT ? ALLOW FILTERING; with params
(java.lang.Boolean) true, (java.lang.Integer) 50000
 _options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
 al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
 c=true}
index-query
 0.862
 _usesCache=false
 _statement=SELECT "community_id", "member_id" FROM
"DSEQuickStart"."author_p" WHERE "~~vertex_exists" =
 ? LIMIT ? ALLOW FILTERING; with params
(java.lang.Boolean) true, (java.lang.Integer) 50000
 _options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
 al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
 c=true}
index-query
 0.679
 _usesCache=false
 _statement=SELECT "community_id", "member_id" FROM
"DSEQuickStart"."book_p" WHERE "~~vertex_exists" = ?
 LIMIT ? ALLOW FILTERING; with params (java.lang.Boolean)
true, (java.lang.Integer) 50000
 _options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
 al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
 c=true}
index-query
 1.344
 _usesCache=false
 _statement=SELECT "community_id", "member_id" FROM
"DSEQuickStart"."ingredient_p" WHERE "~~vertex_exists"
 = ? LIMIT ? ALLOW FILTERING; with params
(java.lang.Boolean) true, (java.lang.Integer) 5000
 0
 _options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
 al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
 c=true}
index-query
 1.053
 _usesCache=false

```

```

 _statement=SELECT "community_id", "member_id" FROM
"DSEQuickStart"."meal_p" WHERE "~~vertex_exists" = ?
 LIMIT ? ALLOW FILTERING; with params (java.lang.Boolean)
true, (java.lang.Integer) 50000
 _options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
 al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
 c=true}
index-query
 4.173
 _usesCache=false
 _statement=SELECT "community_id", "member_id" FROM
"DSEQuickStart"."recipe_p" WHERE "~~vertex_exists" =
? LIMIT ? ALLOW FILTERING; with params
(java.lang.Boolean) true, (java.lang.Integer) 50000
 _options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
 al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
 c=true}
index-query
 1.291
 _usesCache=false
 _statement=SELECT "community_id", "member_id" FROM
"DSEQuickStart"."reviewer_p" WHERE "~~vertex_exists"
= ? LIMIT ? ALLOW FILTERING; with params
(java.lang.Boolean) true, (java.lang.Integer) 50000
 _options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
 al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
 c=true}
DsegVertexStep(IN,vertex) 78
 78 95.721 61.90
query-optimizer
 0.305
 _condition=((true) & direction = IN)
vertex-query
 4.136
 _usesCache=false
 _statement=SELECT * FROM "DSEQuickStart"."author_e" WHERE
"community_id" = ? AND "member_id" = ? LIMIT ?
 ALLOW FILTERING; with params (java.lang.Integer)
588941056, (java.lang.Long) 0, (java.lang.I
 nteger) 50000
 _options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
 al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
 c=true}
 _isPartitioned=false
 _usesIndex=false

```

```

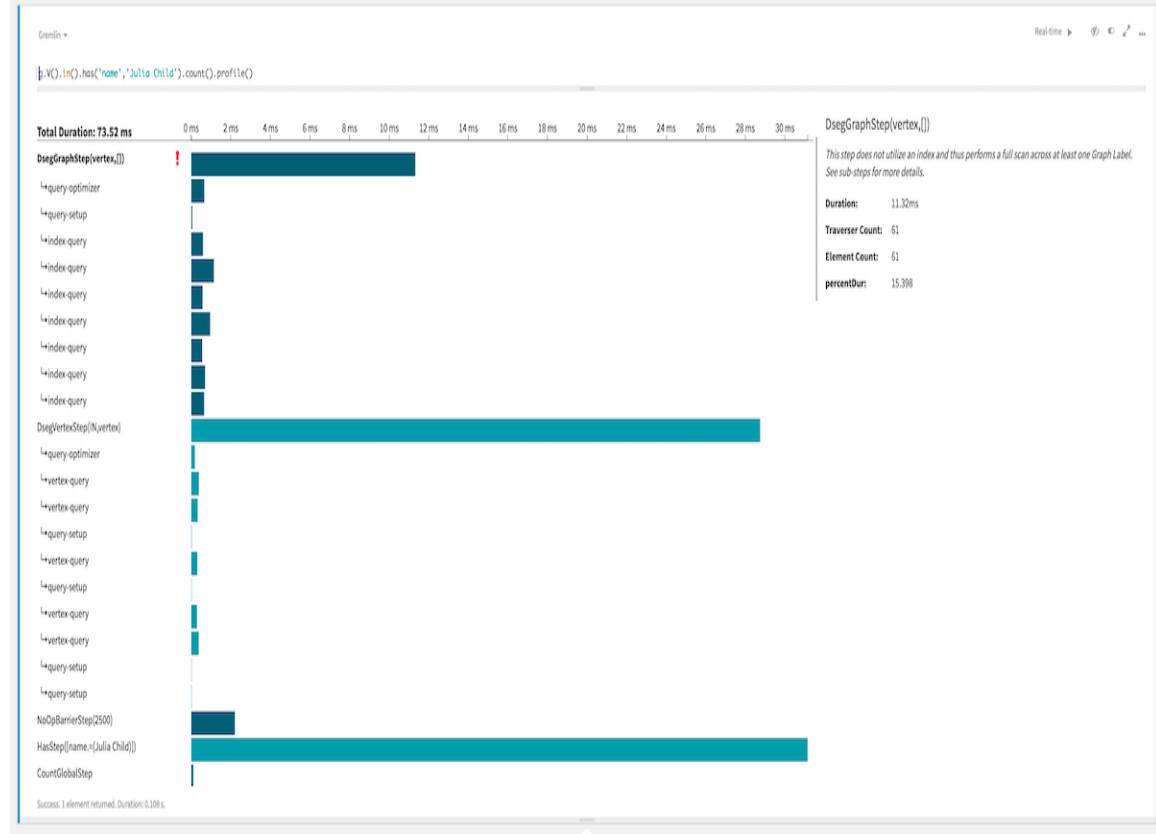
vertex-query
 0.558
 _usesCache=false
 _statement=SELECT * FROM "DSEQuickStart"."author_e" WHERE
"community_id" = ? AND "member_id" = ? LIMIT ?
 ALLOW FILTERING; with params (java.lang.Integer)
1432048000, (java.lang.Long) 1, (java.lang.
 Integer) 50000
 _options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
 al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
 c=true}
 _isPartitioned=false
 _usesIndex=false
vertex-query
 1.146
 _usesCache=false
 _statement=SELECT * FROM "DSEQuickStart"."author_e" WHERE
"community_id" = ? AND "member_id" = ? LIMIT ?
 ALLOW FILTERING; with params (java.lang.Integer)
153541376, (java.lang.Long) 1, (java.lang.I
 nteger) 50000
 _options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
 al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
 c=true}
 _isPartitioned=false
 _usesIndex=false
query-setup
 0.941
 _isFitted=false
 _isSorted=true
 _isScan=false
query-setup
 0.015
 _isFitted=false
 _isSorted=true
 _isScan=false
vertex-query
 1.966
 _usesCache=false
 _statement=SELECT * FROM "DSEQuickStart"."author_e" WHERE
"community_id" = ? AND "member_id" = ? LIMIT ?
 ALLOW FILTERING; with params (java.lang.Integer)
138026496, (java.lang.Long) 0, (java.lang.I
 nteger) 50000
 _options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
 al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
 c=true}
 _isPartitioned=false

```

\_usesIndex=false			
query-setup	0.015		
\_isFitted=false			
\_isSorted=true			
\_isScan=false			
query-setup	0.013		
\_isFitted=false			
\_isSorted=true			
\_isScan=false			
query-setup	0.016		
\_isFitted=false			
\_isSorted=true			
\_isScan=false			
NoOpBarrierStep(2500)			78
25	2.877	1.86	
HasStep([name.=(Julia Child)])			5
1	25.242	16.32	
CountGlobalStep			1
1	1.859	1.20	
		>TOTAL	-
-	154.632	-	

**Note:** The time each step takes depends on caching and other factors. For the purposes of this discussion, ignore the times reported. The `profile()` method now includes CQL commands that are executed due to Gremlin commands.

Figure 32: Studio profile output for Traversal 1



Looking at the first step, all vertices in the graph are traversed. This graph is very small, so the number of vertices is negligible compared to production graphs. In the next step, the traversal must find all incoming edges to the vertices. Again, for a small graph, the number of edges is negligible, but in production graphs, edges can number in the millions to billions. Now, the adjacent vertices are filtered for the property key information specified, narrowing the number of vertices to 6. The last two steps accomplish the count and profiling metrics.

## Specifying an edge label

Now consider a modification to the original traversal that specifies the edge label for the incoming edges:

```
g.V().in('created').has('name','Julia Child').count()
==>3
```

This modified traversal still looks at all vertices, but in walking the incoming edges, it is limited to those that are labeled as `created`. The following profile shows an improved picture:

```
gremlin> g.V().in('created').has('name','Julia Child').count().profile()
==>Traversal Metrics
Step
Traversers Time (ms) % Dur
=====
```

```

DsegGraphStep(vertex,[])
 61 22.251 16.91
query-optimizer
 1.760
 _condition=((label = FridgeSensor | label = author | label = book |
label = ingredient | label = meal |
 label = recipe | label = reviewer) & (true))
query-setup
 0.071
 _isFitted=true
 _isSorted=false
 _isScan=true
index-query
 1.139
 _usesCache=false
 _statement=SELECT "city_id", "sensor_id" FROM
"DSEQuickStart"."FridgeSensor_p" WHERE "~~vertex_exists" =
 ? LIMIT ? ALLOW FILTERING; with params
(java.lang.Boolean) true, (java.lang.Integer) 50000
 _options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
 al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
 c=true}
index-query
 2.012
 _usesCache=false
 _statement=SELECT "community_id", "member_id" FROM
"DSEQuickStart"."author_p" WHERE "~~vertex_exists" =
 ? LIMIT ? ALLOW FILTERING; with params
(java.lang.Boolean) true, (java.lang.Integer) 50000
 _options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
 al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
 c=true}
index-query
 0.549
 _usesCache=false
 _statement=SELECT "community_id", "member_id" FROM
"DSEQuickStart"."book_p" WHERE "~~vertex_exists" = ?
 LIMIT ? ALLOW FILTERING; with params (java.lang.Boolean)
true, (java.lang.Integer) 50000
 _options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
 al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
 c=true}
index-query
 0.849
 _usesCache=false
 _statement=SELECT "community_id", "member_id" FROM
"DSEQuickStart"."ingredient_p" WHERE "~~vertex_exists"

```

```

 " = ? LIMIT ? ALLOW FILTERING; with params
(java.lang.Boolean) true, (java.lang.Integer) 5000
 0
 _options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
 al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
 c=true}
index-query
 0.887
 _usesCache=false
 _statement=SELECT "community_id", "member_id" FROM
"DSEQuickStart"."meal_p" WHERE "~~vertex_exists" = ?
 ? LIMIT ? ALLOW FILTERING; with params (java.lang.Boolean)
true, (java.lang.Integer) 50000
 _options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
 al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
 c=true}
index-query
 0.889
 _usesCache=false
 _statement=SELECT "community_id", "member_id" FROM
"DSEQuickStart"."recipe_p" WHERE "~~vertex_exists" =
 ? LIMIT ? ALLOW FILTERING; with params
(java.lang.Boolean) true, (java.lang.Integer) 50000
 _options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
 al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
 c=true}
index-query
 0.499
 _usesCache=false
 _statement=SELECT "community_id", "member_id" FROM
"DSEQuickStart"."reviewer_p" WHERE "~~vertex_exists"
 = ? LIMIT ? ALLOW FILTERING; with params
(java.lang.Boolean) true, (java.lang.Integer) 50000
 _options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
 al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
 c=true}
DsegVertexStep(IN,[created],vertex)
 8 103.458 78.62
query-optimizer
 0.618
 _condition=((label = created) & (true)) & direction = IN)
vertex-query
 0.261
 _usesCache=false
 _statement=SELECT * FROM "DSEQuickStart"."author_e" WHERE
"community_id" = ? AND "member_id" = ? AND "~~

```

8

```

 edge_label_id" = ? LIMIT ? ALLOW FILTERING; with params
(java.lang.Integer) 1432048000, (java
 .lang.Long) 1, (java.lang.Integer) 65577,
(java.lang.Integer) 50000
 \options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
 al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
 c=true}
 \isPartitioned=false
 \usesIndex=false
vertex-query
 0.200
 \usesCache=false
 \statement=SELECT * FROM "DSEQuickStart"."author_e" WHERE
"community_id" = ? AND "member_id" = ? AND "~~
 edge_label_id" = ? LIMIT ? ALLOW FILTERING; with params
(java.lang.Integer) 153541376, (java.
 lang.Long) 1, (java.lang.Integer) 65577,
(java.lang.Integer) 50000
 \options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
 al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
 c=true}
 \isPartitioned=false
 \usesIndex=false
query-setup
 0.017
 \isFitted=true
 \isSorted=true
 \isScan=false
vertex-query
 6.140
 \usesCache=false
 \statement=SELECT * FROM "DSEQuickStart"."author_e" WHERE
"community_id" = ? AND "member_id" = ? AND "~~
 edge_label_id" = ? LIMIT ? ALLOW FILTERING; with params
(java.lang.Integer) 588941056, (java.
 lang.Long) 0, (java.lang.Integer) 65577,
(java.lang.Integer) 50000
 \options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
 al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
 c=true}
 \isPartitioned=false
 \usesIndex=false
query-setup
 0.017
 \isFitted=true
 \isSorted=true
 \isScan=false

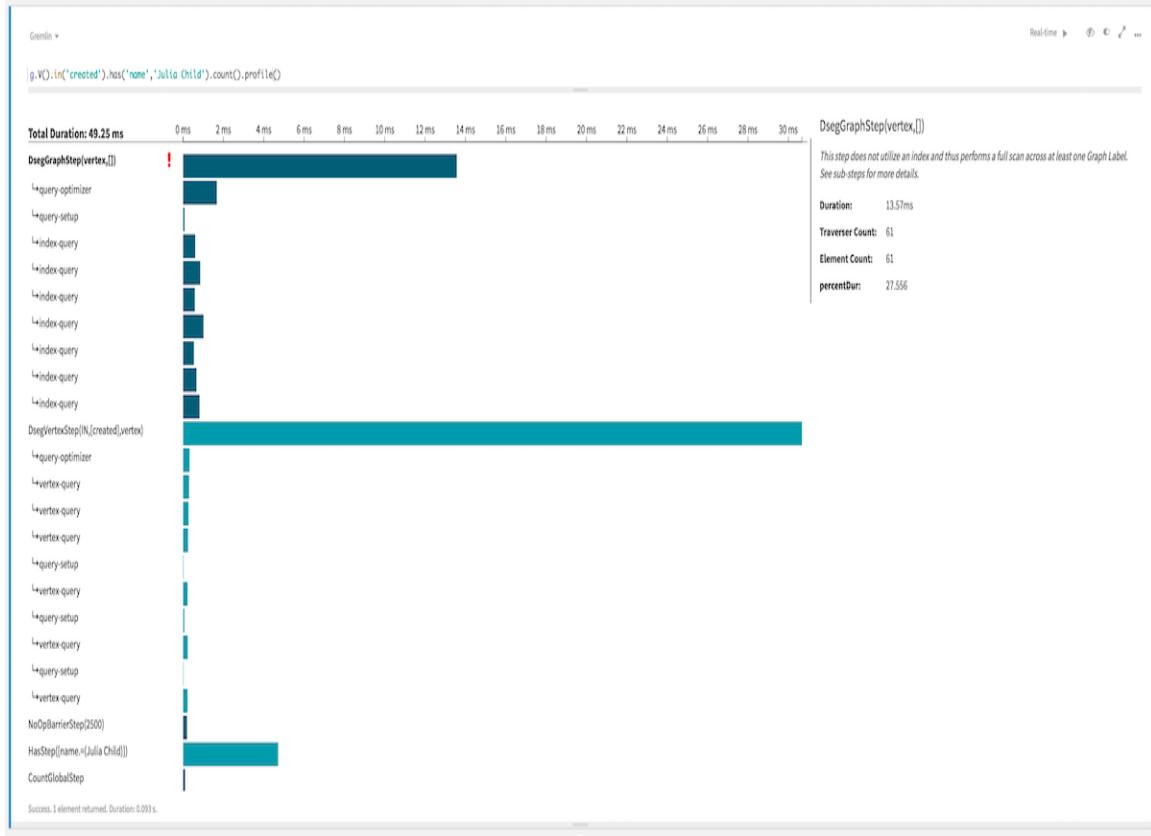
```

```

vertex-query
 0.201
 _usesCache=false
 _statement=SELECT * FROM "DSEQuickStart"."author_e" WHERE
"community_id" = ? AND "member_id" = ? AND "~~
 edge_label_id" = ? LIMIT ? ALLOW FILTERING; with params
(java.lang.Integer) 771301632, (java.
 lang.Long) 0, (java.lang.Integer) 65577,
(java.lang.Integer) 50000
 _options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
 al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
 c=true}
 _isPartitioned=false
 _usesIndex=false
query-setup
 0.012
 _isFitted=true
 _isSorted=true
 _isScan=false
vertex-query
 0.173
 _usesCache=false
 _statement=SELECT * FROM "DSEQuickStart"."author_e" WHERE
"community_id" = ? AND "member_id" = ? AND "~~
 edge_label_id" = ? LIMIT ? ALLOW FILTERING; with params
(java.lang.Integer) 994194304, (java.
 lang.Long) 0, (java.lang.Integer) 65577,
(java.lang.Integer) 50000
 _options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
 al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
 c=true}
 _isPartitioned=false
 _usesIndex=false
query-setup
 0.012
 _isFitted=true
 _isSorted=true
 _isScan=false
NoOpBarrierStep(2500) 8
 4 0.910 0.69
HasStep([name.=("Julia Child")]) 3
 1 4.903 3.73
CountGlobalStep 1
 1 0.075 0.06
 >TOTAL
 - 131.599 -

```

Figure 33: Studio profile output for Traversal 2



As with the original traversal, the first step still finds all the vertices. In the next step, however, the number of edges walked is significantly decreased. However, in a production graph, finding all the vertices in the entire graph will take a long time. The third step now reflects the true answer for how many recipes Julia Child has created; in the first traversal, other incoming edges for Julia Child's books were included in the count.

This graph traversal is still an OLAP traversal that touch all vertices and does not use indexes.

## Specifying the vertex label

What effect does specifying the vertex label have on improving the traversal?

```
g.V().hasLabel('recipe').in().has('name', 'Julia Child').count()
==>3
```

This modified traversal now is limited to the `recipe` vertices, but walks all incoming edges. The profile shows a somewhat better picture:

Step	Traversers	Time (ms)	% Dur	Count
Traversal Metrics				

```
=====
DsegGraphStep([~label.=(recipe)])
 8 2.598 9.25
query-optimizer
 0.241
 _condition=((label = recipe) & (true))
query-setup
 0.187
 _isFitted=true
 _isSorted=false
 _isScan=true
index-query
 1.225
 _usesCache=false
 _statement=SELECT "community_id", "member_id" FROM
"DSEQuickStart"."recipe_p" WHERE "~~vertex_exists" =
 ? LIMIT ? ALLOW FILTERING; with params
(java.lang.Boolean) true, (java.lang.Integer) 50000
 _options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
 al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
 c=true}
DsegVertexStep(IN,vertex)
 15 9.668 34.41
query-optimizer
 0.150
 _condition=((true) & direction = IN)
query-setup
 0.047
 _isFitted=false
 _isSorted=true
 _isScan=false
vertex-query
 0.896
 _usesCache=false
 _statement=SELECT * FROM "DSEQuickStart"."recipe_e" WHERE
"community_id" = ? AND "member_id" = ? LIMIT ?
 ALLOW FILTERING; with params (java.lang.Integer)
1315507840, (java.lang.Long) 1, (java.lang.
 Integer) 50000
 _options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
 al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
 c=true}
 _isPartitioned=false
 _usesIndex=false
vertex-query
 1.415
 _usesCache=false
 _statement=SELECT * FROM "DSEQuickStart"."recipe_e" WHERE
"community_id" = ? AND "member_id" = ? LIMIT ?

```

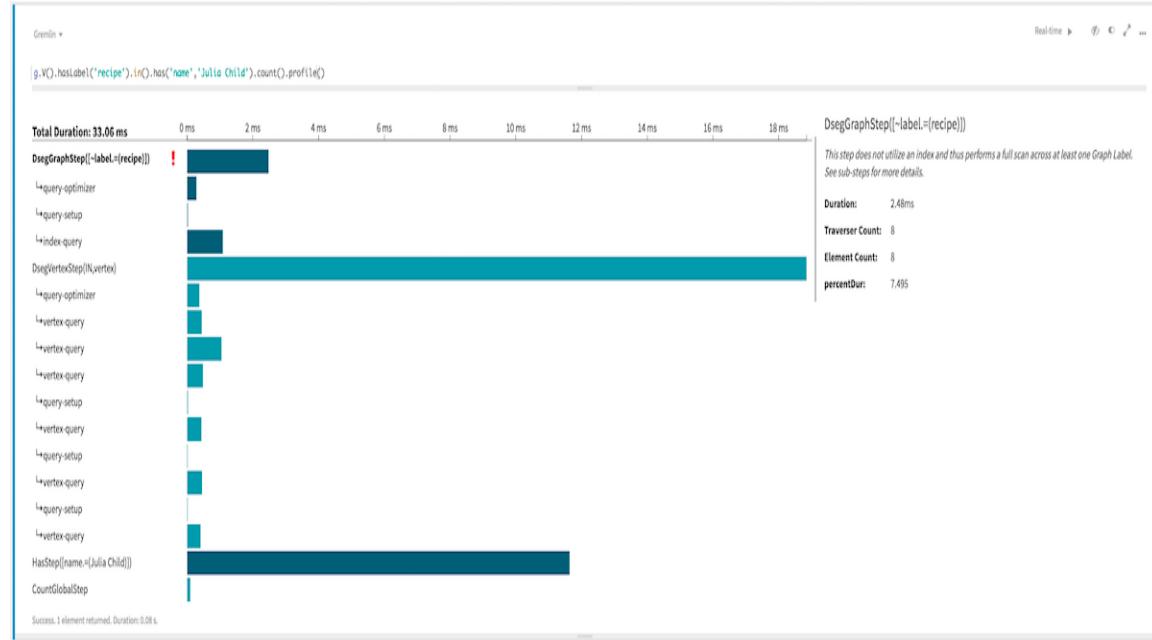
```

 ALLOW FILTERING; with params (java.lang.Integer)
96517120, (java.lang.Long) 1, (java.lang.In
teger) 50000
 _options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
c=true}
 _isPartitioned=false
 _usesIndex=false
vertex-query
 2.846
 _usesCache=false
 _statement=SELECT * FROM "DSEQuickStart"."recipe_e" WHERE
"community_id" = ? AND "member_id" = ? LIMIT ?
 ALLOW FILTERING; with params (java.lang.Integer)
1598713728, (java.lang.Long) 1, (java.lang.
Integer) 50000
 _options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
c=true}
 _isPartitioned=false
 _usesIndex=false
query-setup
 0.038
 _isFitted=false
 _isSorted=true
 _isScan=false
vertex-query
 0.364
 _usesCache=false
 _statement=SELECT * FROM "DSEQuickStart"."recipe_e" WHERE
"community_id" = ? AND "member_id" = ? LIMIT ?
 ALLOW FILTERING; with params (java.lang.Integer)
1146421632, (java.lang.Long) 1, (java.lang.
Integer) 50000
 _options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
c=true}
 _isPartitioned=false
 _usesIndex=false
query-setup
 0.014
 _isFitted=false
 _isSorted=true
 _isScan=false
vertex-query
 0.431
 _usesCache=false

```

<code>\_statement=SELECT * FROM "DSEQuickStart"."recipe_e" WHERE "community_id" = ? AND "member_id" = ? LIMIT ? ALLOW FILTERING; with params (java.lang.Integer) 384373760, (java.lang.Long) 2, (java.lang.I nteger) 50000</code>	
<code>\_options=Options{consistency=Optional[ONE], serialConsistency=Optional.empty, fallbackConsistency=Option al.empty, pagingState=null, pageSize=-1, user=Optional.empty, waitForSchemaAgreement=true, asyn c=true}</code>	
<code>\_isPartitioned=false</code>	
<code>\_usesIndex=false</code>	
<code>query-setup</code>	0.014
<code>\_isFitted=false</code>	
<code>\_isSorted=true</code>	
<code>\_isScan=false</code>	
<code>HasStep([name.=(Julia Child)])</code>	3
3	15.765
<code>CountGlobalStep</code>	1
1	0.068
	0.24
	>TOTAL
-	28.100
	-

Figure 34: Studio profile output for Traversal 3



A limited number of vertices are found in the first step. A number of edges are walked. However, in a production graph, finding even a limited number of vertices will take some time without indexing, and the number of edges walked could be quite large.

This graph traversal is still an OLAP traversal that does not use indexes. Although this traversal narrows the query by limiting the vertex label initially, an index is not used to find the starting point for the traversal.

## Using an edge label plus a vertex label

Indexes are identified by vertex label and property key. The following graph traversal twists the direction of the query:

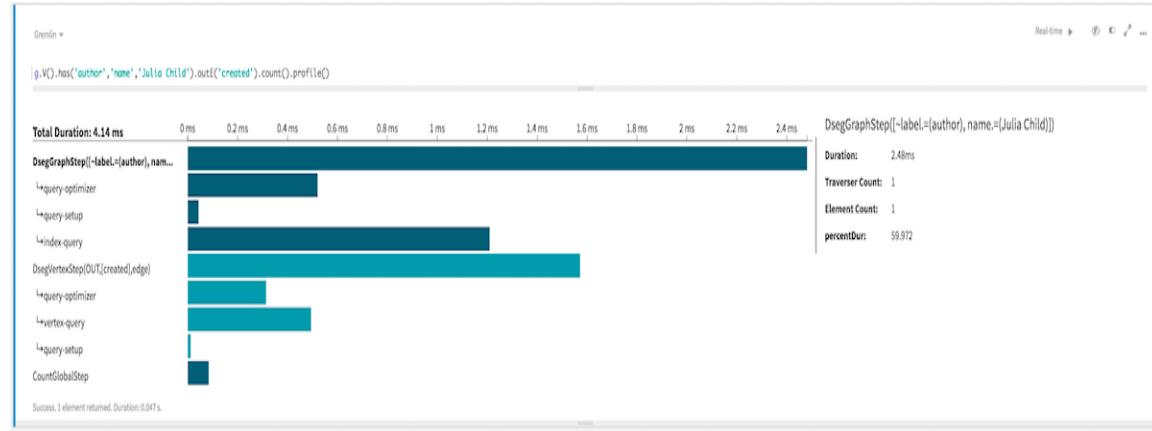
```
g.V().has('author', 'name', 'Julia Child').outE('created').count()
==>3
```

This traversal starts at a single vertex by specifying both vertex label `author` and a specific property key and value `Julia Child`, and walks only the outgoing edges that have an edge label `created`.

```
gremlin> g.V().has('author', 'name', 'Julia
Child').outE('created').count().profile()
==>Traversal Metrics
Step Count
Traversers Time (ms) % Dur
=====
DsegGraphStep([~label.=(author), name.=(Julia C... 1
 1 29.049 84.45
query-optimizer
 7.673
 _condition=((label = author) & (true)) & name = Julia Child
query-setup
 0.033
 _isFitted=true
 _isSorted=false
 _isScan=false
index-query
 17.694
 _indexType=Secondary
 _usesCache=false
 _statement=SELECT "community_id", "member_id" FROM
"DSEQuickStart"."author_p" WHERE "name" = ? LIMIT ?;
 with params (java.lang.String) Julia Child,
(java.lang.Integer) 50000
 _options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
 al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
 c=true}
DsegVertexStep(OUT,[created],edge)
 3 5.265 15.31
query-optimizer
 0.200
 _condition=((label = created) & (true)) & direction = OUT
vertex-query
 0.586
 _usesCache=false
```

<code>\_statement=SELECT * FROM "DSEQuickStart"."author_e" WHERE "community_id" = ? AND "member_id" = ? AND "~~ edge_label_id" = ? LIMIT ? ALLOW FILTERING; with params (java.lang.Integer) 1535517312, (java .lang.Long) 0, (java.lang.Integer) 65576, (java.lang.Integer) 50000 \_options=Options{consistency=Optional[ONE], serialConsistency=Optional.empty, fallbackConsistency=Option al.empty, pagingState=null, pageSize=-1, user=Optional.empty, waitForSchemaAgreement=true, asyn c=true} \_isPartitioned=false \_usesIndex=false query-setup</code>	0.057
<code>\_isFitted=true \_isSorted=true \_isScan=false</code>	
CountGlobalStep	1
1	0.081
0.24	
	>TOTAL
-	34.397
	-

Figure 35: Studio profile output for Traversal 4



A single vertex starts the traversal. An edge label filters the edges.

This graph traversal is an OLTP traversal. An index on the vertex label `author` and property key `name` can be used to start the traversal directly at an indexed vertex. This example results in a single vertex, but queries that use indexing to limit the starting point to even several vertices will be more efficient than a linear scan that must check all vertices in the graph. Thus, a subgraph, or portion of the graph is traversed.

The key to creating OLTP graph traversals is considering how the graph will be traversed. Use of indexing is critical to the success of fast transactional processing. The profiling tool included with DSE Graph is valuable to analyzing how the traversal performs.

For information on running OLAP queries using Spark, see [DSE Graph and Graph Analytics \(page 607\)](#).

## Graph anti-patterns

Examine common mistakes made with DSE Graph.

Some common mistakes are made with DSE Graph. Examining best practices can ease the learning curve and improve graph application performance.

### Not using indexing

Indexing is a key feature in decreasing the latency of queries in a distributed database. DSE Graph relies on indexing to speed up OLTP read latency for complex graph traversals. What is key to understand is that global indexing in DSE Graph involves both a vertex label and a property key. The vertex label narrows the search in the underlying DSE datastore to a partition, which in turn narrows the search to one or a small number of DSE database nodes in the cluster. Indexing a property key that is used for more than one vertex label and not supplying the vertex label in the query amounts to an almost full scan of the cluster. Thus, using this query:

```
g.V().has('name', 'James Beard')...
```

requires the traversal to check all vertices that use the property key `name`. Changing this query to:

```
g.V().has('author', 'name', 'James Beard')...
```

allows the query to consult an index that can be built for all names in author records, and retrieve just one vertex to start the traversal. The index would be added during [schema creation \(page 436\)](#):

```
schema.vertexLabel('author').index('byName').secondary().by('name').add()
```

In fact, this one change in the traversal will change the query from an OLAP query into an OLTP query.

### Property key creation

Property key creation can affect the performance of DSE Graph. Using unique property key names can seem beneficial at first, but reusing property keys for different vertex labels can improve the storage of property keys for the graphs. For example, consider the following:

```
schema.propertyKey('recipeCreationDate').Timestamp().create()
schema.propertyKey('mealCreationDate').Timestamp().create()
schema.propertyKey('reviewCreationDate').Timestamp().create()
```

While these property key names make code readable and ease tracking in graph traversals, each additional property key stored requires resources. Use one property key instead, such as:

```
schema.propertyKey('timestamp').Timestamp().create()
```

to decrease overhead. Since property keys are mostly used in graph traversals along with vertex labels, `timestamp` will be uniquely identified by the combination of vertex label and property key.

## Vertex label creation

Vertex label creation can affect the performance of DSE Graph. Using many unique vertex labels can seem useful, but like property keys, the fewest vertex labels created can improve the storage requirements. For example, consider the following:

```
schema.vertexLabel('recipeAuthor').create()
schema.vertexLabel('bookAuthor').create()
schema.vertexLabel('mealAuthor').create()
schema.vertexLabel('reviewAuthor').create()
```

While these vertex labels again have the advantage of readability, unless a vertex label will be uniquely queried, it is best to roll the functionality into a single vertex label. For instance, in the above code, it is likely that recipes, meals, and books will have the same authors, whereas reviews are likely to have a different set of writers and types of queries. Use two vertex labels instead of four:

```
schema.vertexLabel('author').create()
schema.vertexLabel('reviewer').create()
```

In fact, this case may even be better suited to using only one vertex label `person`, if the overlap in authors and reviewers is great enough. In some cases, a property key that identifies whether a person is an author or a reviewer is a viable option.

```
schema.propertyKey('type').Text().create()
schema.vertexLabel('person').create()
graph.addVertex(label, 'person', 'type', 'author', 'name', 'Jamie
Oliver')
```

## Mixing schema creation or configuration setting with traversal queries

Consider the following statements. The first statement configures a graph setting for read consistency. The second statement executes a count on a field `name` with a value `read vertex` for all vertices.

```
schema.config().option('graph.tx_groups.default.read_consistency').set('ALL');
g.V().has('name', 'read vertex').count()
```

In Gremlin Server, both statements are run in one transaction. Any changes made during this transaction are applied when it successfully commits both actions. The change in read consistency is not actually applied until the end of a transaction and thereby only affects the next transaction. The statements are not processed sequentially as individual requests.

To avoid such errors in processing, avoid mixing schema creation or configuration setting with traversal queries in applications. Best practice is to create schema and set configurations before querying the graph database with graph traversals.

## InterruptedException indicates OLTP query running too long

In general, seeing logs with this exception are indicative that an OLTP query is running too long. The typical cause is that indexes have not been created for elements used in graph traversal queries. [Create the indexes \(page 460\)](#) and retry the queries.

## g.V().count() and g.E().count() can cause long delays

Running a count on a large graph can cause serious issues. The command basically must iterate through all the vertices, taking hours if the graph is large. Any table scan (iterating all vertices) is simply not an OLTP process. Doing the same process on edges is essentially the same, a full table scan, as well. Using Spark commands are currently the recommended method to get these counts.

## Setting replication factor too low for *graph\_name\_system*

Each graph created in turn creates three DSE database keyspaces, *graph\_name*, *graph\_name\_system* and *graph\_name\_pvt*. The *graph\_name\_system* stores the graph schema, and loss of this data renders the entire graph inoperable. Be sure to [set the replication factor appropriately \(page 649\)](#) based on cluster configuration.

## Using string concatenation in application instead of parameterized queries

String concatenation in graph applications will critically impair performance. Each unique query string creates an object that is cached on a node, using up node resources. Use parameterized queries ([DSE Java Driver](#), [DSE Python Driver](#), [DSE Ruby Driver](#), [DSE Node.js Driver](#), [DSE C# Driver](#), [DSE C/C++ Driver](#)) to prevent problems due to resource allocation.

## DSE Graph data modeling

Introduce graph data modeling.

### Graph data modeling introduction

Brief introduction to the parts of a graph data model.

Data modeling for graph databases is generally a simple process. Imagine information written on a whiteboard as vertices and lines connecting them, and you are 90% done with a graph database data model.

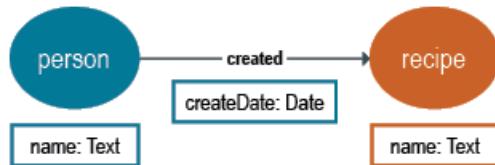
Figure 36: Julia Child creates beef bourguignon



Julia Child was a famous chef who created many recipes. One of the recipes she created for an American audience in 1961 was beef bourguignon. In the diagram above, a person, Julia Child, is linked to a recipe, beef bourguignon. Person and recipe are two types of **vertex**, and the line adjoining the vertices, or **edge**, identifies the relationship as "created". Vertices and edges have associated **properties**, such as a person's name, a recipe name,

and the date associated with the edge. Properties are a basic element that are used in a query about the graph, and consist of a **property key** and **property value**. In graph databases, a vertex is **incident** to an edge, and an edge is **incident** to a vertex. A vertex is **adjacent** to another vertex if they share an edge. A generalized view of this data model is shown below:

Figure 37: Generalized data model for author and recipe



Each vertex is assigned a **vertex label** to identify a specific type of vertex. The vertex labels shown here are `person` and `recipe`. Each edge must also have an **edge label** specifying its type. The edge label shown is `created`. The **properties** shown are `name` and `createDate`.

DSE Graph limits the number of vertex labels to 200 per graph.

For more complex graphs, [multiple edges \(page 454\)](#) can connect vertices, and multiple properties can be assigned to vertices and edges. Both properties and edges can have [multiple cardinality \(page 454\)](#). Vertex properties can have **meta-properties**, a [property on a property \(page 448\)](#).

An important concept to be aware of is the nature of vertices and edges as addressable elements. [Indexes \(page 457\)](#) play a critical role in querying graphs, and vertex labels must be a part of every index. Only vertices are globally addressable, whereas edges are only locally addressable. In practice, what this situation means is that edges can only be indexed locally for a particular vertex label. Edges are about the relationship of vertices, and are classified as second-class citizens; vertices are entities and are [first-class citizens](#) for which all graph operations are available. To illustrate the nature of the second-class citizenry of edges, meta-properties of edges cannot be indexed and used to narrow queries, making those edges better modeled as vertices if the data stored in the meta-properties must be used to narrow down a query.

For the remaining 10% of your effort, optimization of whether an aspect of your whiteboard graph should be a vertex or an edge is the most pressing factor. If an aspect used as an edge begins to be used more than a few times, it should become a vertex instead. For instance, we could add a vertex property to the author to add their country of origin. However, since many authors will come from the same country, such as the China or France, creating a location vertex type can be more advantageous to later querying operations.

## Graph data modeling example

Details of a larger data model creation.

Let's consider the example of recipes further to create a more complex data model. This example will go through some of the thinking behind creating a graph database data model.

- Obviously, we will need vertices that are connected by edges. What is a possible additional type of vertex besides person and recipe?

Not surprisingly, we can add an *ingredient* vertex label. This vertex will have some properties. Can you think of the possibilities for vertex properties?

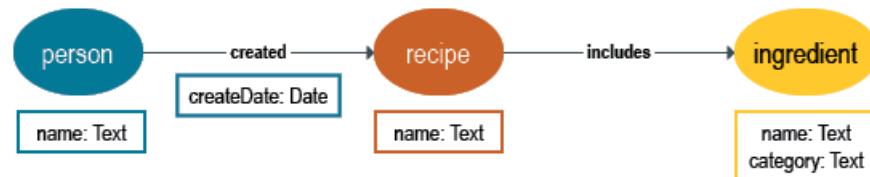
The most likely property for an ingredient vertex is the *name* of the ingredient. While we could use `ingredientName` to identify the name of the ingredient, keeping the schema small has advantages in DSE Graph. We'll reuse `name` for every vertex label in our example.



- There are other possibilities that might be important for the ingredient vertex properties. Think about it and write down some more possibilities. We will add them later. Let's move on to considering the edges that will connect *persons*, *recipes*, and *ingredients*.

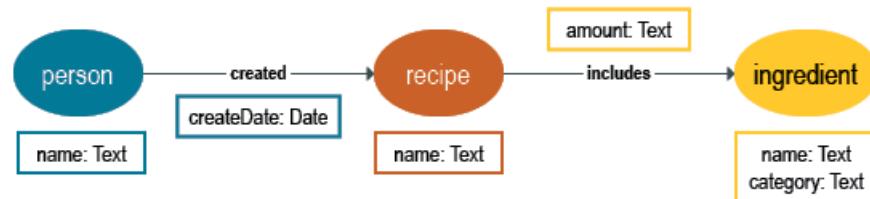
What are the edge labels we can use to identify different types of edges? Previously, you've seen that *persons* and *recipe* are connected by an edge *created*.

An ingredient must be included in a recipe, so an edge *includes* can connect the two vertices.



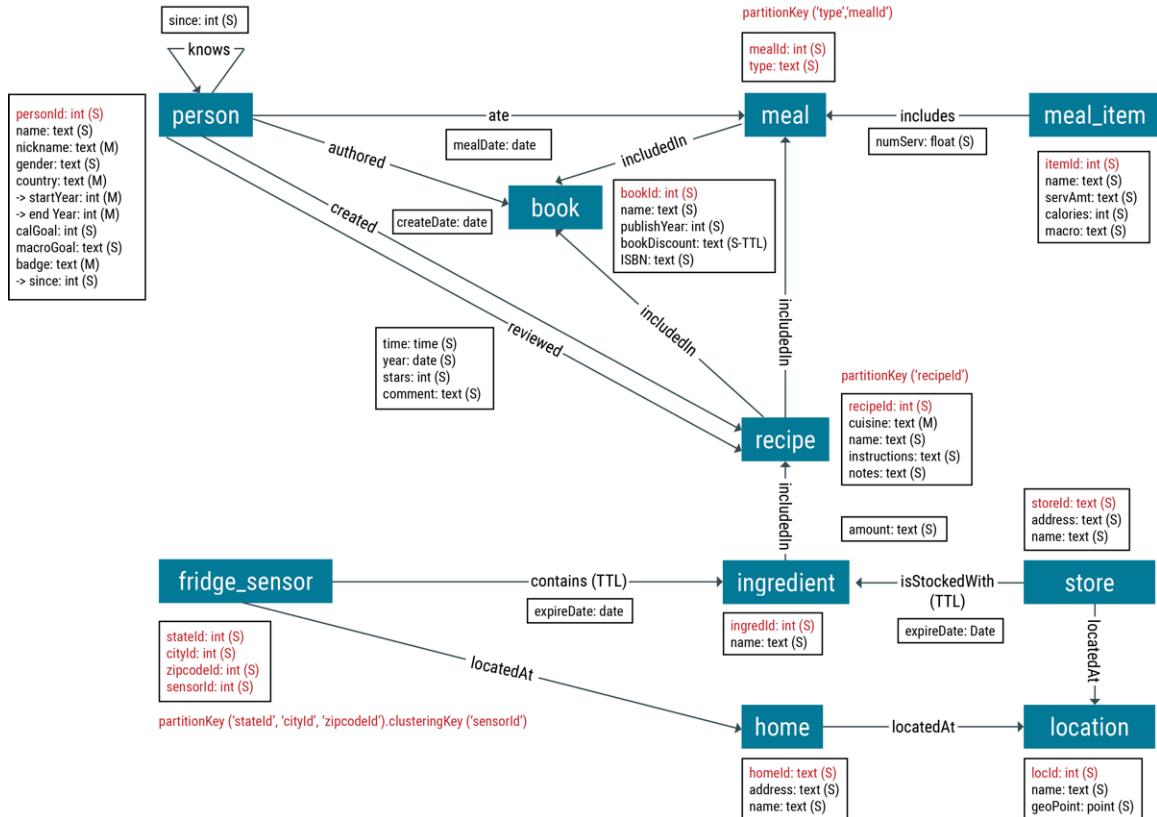
- Edges also have attached properties that can be used later in narrowing queries.

What edge property is appropriate for *includes*?

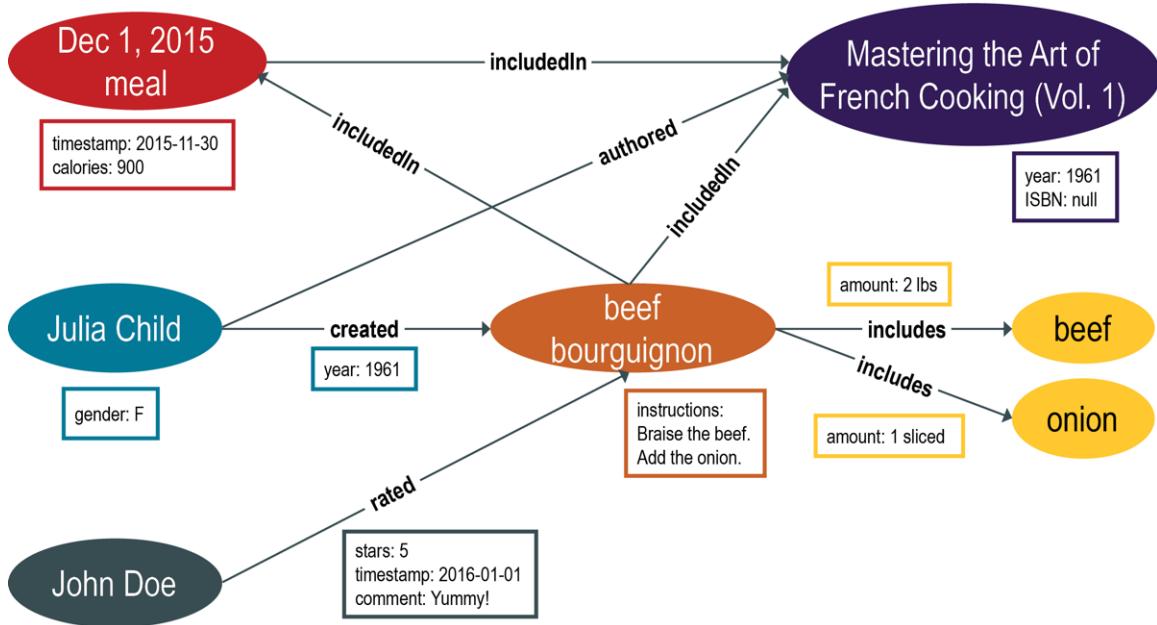


The *amount* of ingredient included in a recipe is important! One cup of salt instead of one teaspoon of salt will make a big difference in the results.

- Today, people publish their recipes online and in cookbooks. Restaurants create fixed price meals from recipes. Consumers review the recipes they try. The results are an intertwined graph of data.



- The additional vertices and edges that can be added to this graph are numerous. For instance, the gender of the recipe authors and reviewers (both *persons*) can be included. Nutritional information for the ingredients can be derived from the calories for a recipe. The number of servings that a recipe makes is useful to cooks. The resulting web of data can grow quickly.



Add a hundred authors, a thousand recipes, ten thousand reviews, and the enormity of the graph becomes obvious. However, as you will see in later sections, DSE Graph can transform complex searches and pattern matching into simple and powerful solutions.

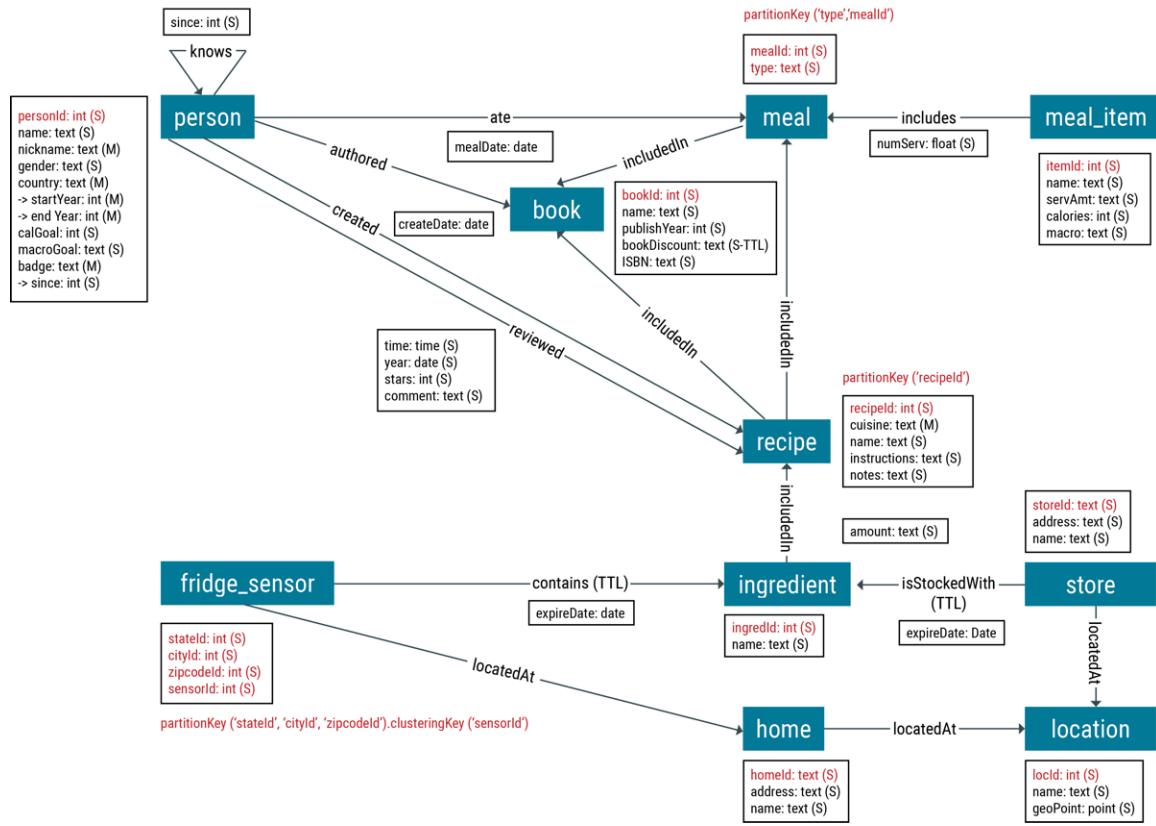
#### What's next:

The data model is the first step in creating a graph. Using the data model, a [schema can be created \(page 436\)](#) that defines how DSE Graph will store the data.

### Further data modeling concepts

Further data modeling concepts.

Graph data models can be expanded to encompass complex relationships. The whole graph can be digested better if subgraphs are considered. The recipe data model can be modified to include new layers of data.



Consider an ingredient. Many additional properties can be added to an ingredient:

#### category

vegetable, fruit, pasta, meat

#### nutritional value

% of vitamins, protein, carbohydrate, fat

#### calories

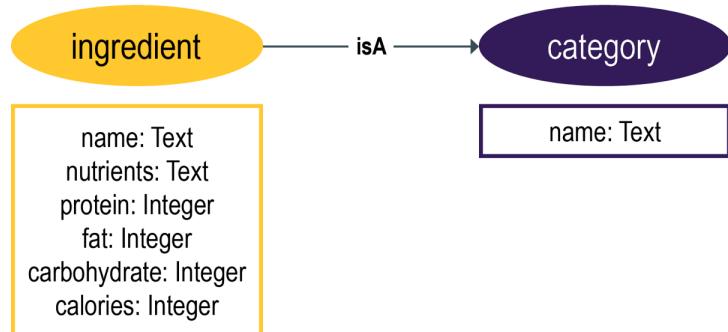
number of kcals

ingredient

name: Text
category: Text
nutrients: Text
protein: Integer
fat: Integer
carbohydrate: Integer
calories: Integer

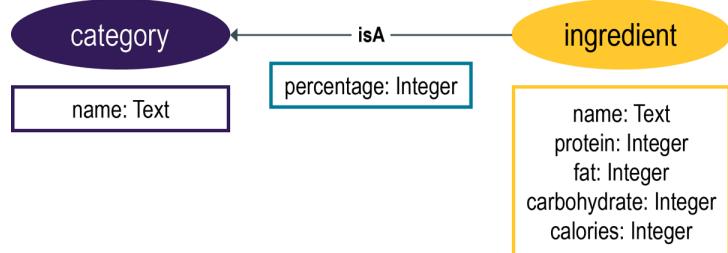
While it may seem simple to choose the property values for an ingredient, there can be more to consider. For instance, consider category. Depending on the number of categories used to describe the ingredients, it can be more advantageous to create a vertex

label or a property for `category`. Vertices can be the starting point for a graph traversal, but vertex properties cannot. In order to ask the question "what ingredients are dairy products?", a starting point at the `dairy` vertex requires one edge hop per ingredient to find all the ingredients categorized as dairy.

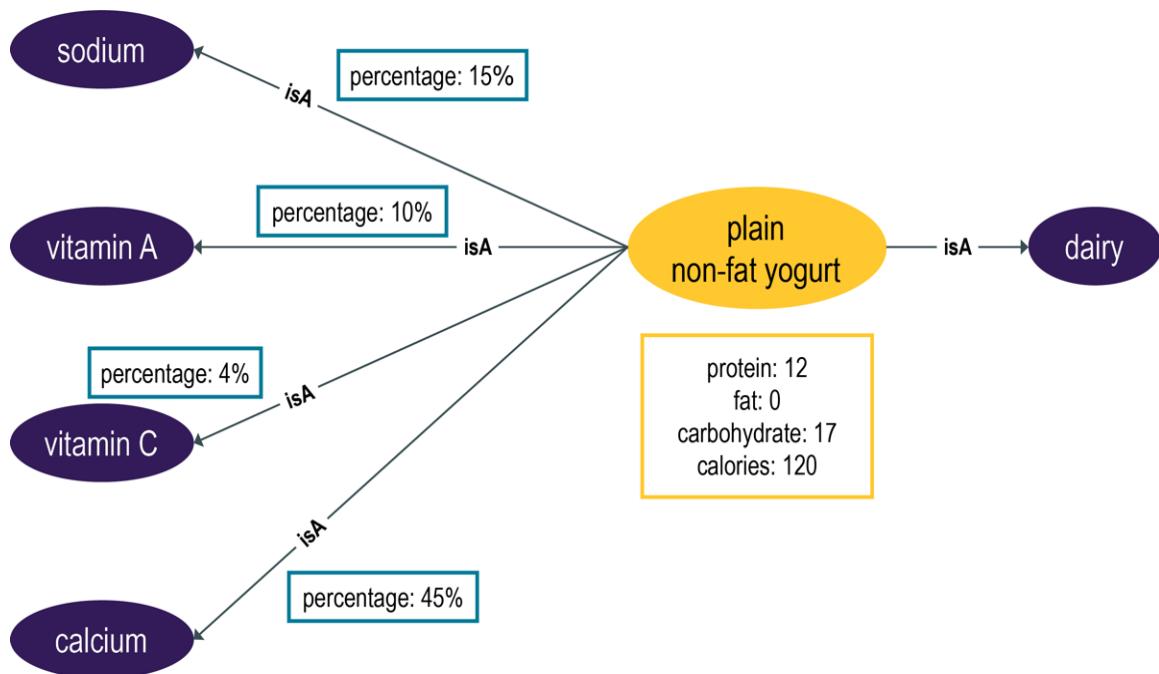


However, if too many ingredients are dairy, a super node, or node that is a hotspot with too many edges attached, can slow down queries that are searching for dairy ingredients. Using property indexing, an ingredient category can be better modeled as a property rather than a vertex label.

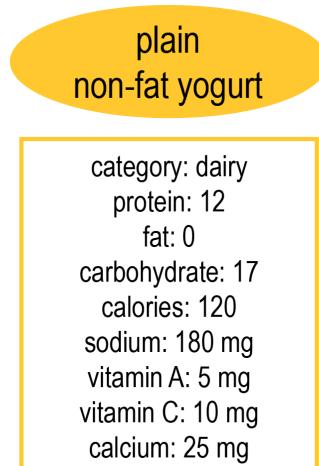
Nutrients are a set number of items, such as vitamin C, vitamin D, calcium, and sodium. Creating a vertex label for `nutrient` and weighting the edges between `ingredient` and `nutrient` with the percentage adds another dimension to the graph.



Look at the relationships that result for just one ingredient:



and imagine the graph resulting for even one hundred ingredients, let alone thousands of ingredients. Examine whether it is better to create a nutrient vertex label or nutrient vertex properties.



Imagine the possibilities for applications built using the ingredient properties. Look in the refrigerator and discover that you have mushrooms and beef, and query the graph database to find a recipe to cook, such as Beef Stroganoff. With the coming possibility of

tagged food in your refrigerator, you could even have your fridge tell you what's for dinner tonight, given the items stored.

## Using DSE Graph

Create graph schema, load external data files, and do advanced graph traversals.

### Getting started with graph databases

Getting started with graph databases.

Graph databases are useful for discovering simple and complex relationships between objects. These things can be people, software, locations, automobiles, or anything else you can think of. Relationships are fundamental to how objects interact with one another and their environment. Graph databases are the perfect representation of the relationships between objects.

Graph databases consist of three elements:

#### vertex

A vertex is an object, such as a person, location, automobile, recipe, or anything else you can think of as nouns.

#### edge

An edge defines the relationship between two vertices. A person can create software, or an author can write a book. Think verbs when defining edges.

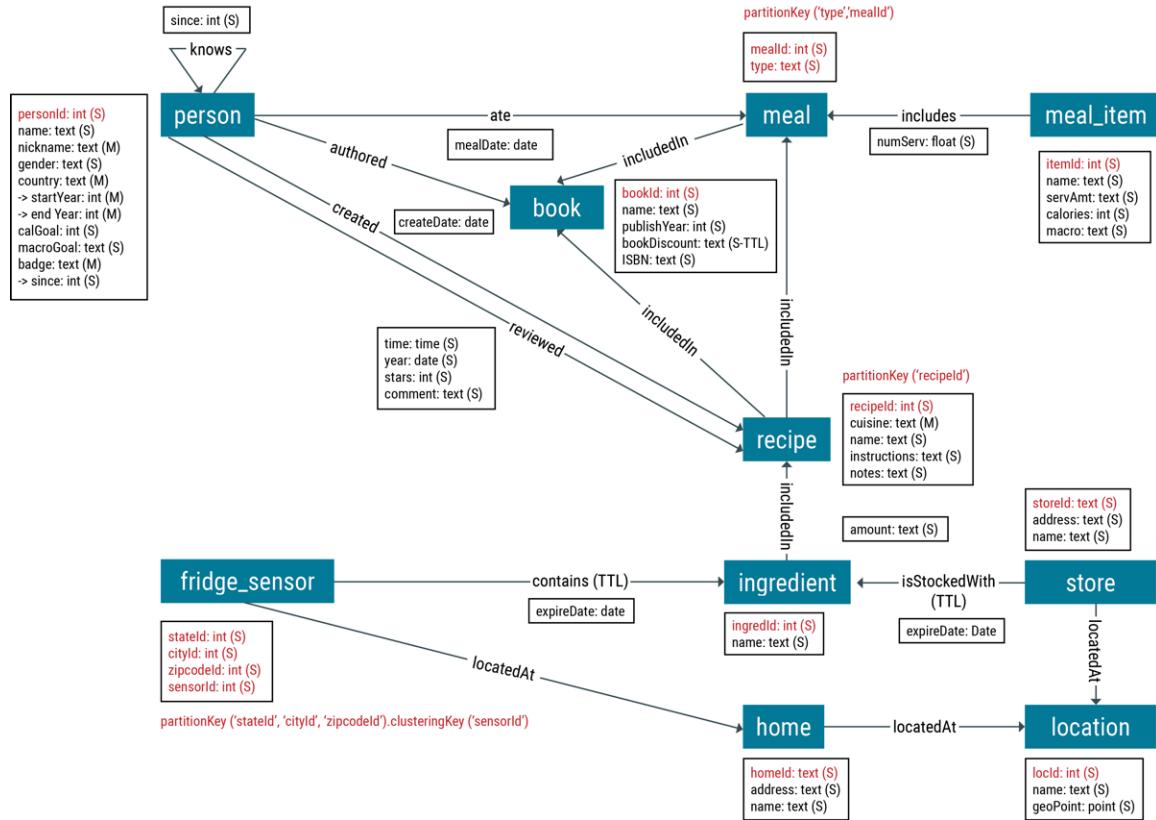
#### property

A key-value pair that describes some attribute of either a vertex or an edge. Property key is used to describe the key in the key-value pair. All properties are global in DSE Graph, meaning that a property can be used for any vertices. For example, "name" can be used for all vertices in a graph.

Vertices, edges and properties can have properties; for this reason, DSE Graph is classified as a **property graph**. The properties for elements are an important element of storing and querying information in a property graph.

Property graphs are typically quite large, although the nature of querying the graph varies depending on whether the graph has large numbers of vertices, edges, or both vertices and edges. To get started with graph database concepts, a *toy* graph is used for simplicity. The example used here explores the world of food.

Figure 49: Recipe Toy Graph



Elements are labeled to distinguish the type of vertices and edges in a graph database. A vertex that will hold information about a person is labeled *person*. An edge in the graph is labeled *authored*. Labels specify the types of vertices and edges that make up the graph. Specifying appropriate labels is an important step in [graph data modeling \(page 423\)](#).

Vertices and edges generally have properties. For instance, an *person* vertex can have a *name*. Gender and current job are examples of additional properties for a *author* vertex. Edges also have properties. A *created* edge can have a *createDate* property that identifies when the adjoining *recipe* vertex was created.

Properties can also have properties. Consider the locations that an author may have lived in while authoring books. While knowing the writing location may be interesting by itself, generally an inquirer is interested in the dates that a person lived in a particular location. Would it be interesting to know if Julia Child lived in France or the United States while writing her first cookbook? It could be relevant if the cookbook is on French cuisine.

There are a variety of methods for ingesting data into DSE Graph.

### DSE Graph Loader

Data can be loaded using the [DSE Graph Loader \(page 471\)](#). CSV, JSON, text parsed with regular expressions, and data selected from a JDBC compliant database can be loaded using a command line tool.

### DataStax Studio

Graph API and Traversal API can be used.

## Gremlin commands

Data can be added using Gremlin commands. This is a useful method for toy (small graphs) used for development and test. An API exists for adding data using Gremlin commands as well, so Gremlin is common in scripts. The [Quick Start \(page 368\)](#) shows some of the common Gremlin commands for creating a graph and running traversals.

## Gryo

Data can be loaded using [Gryo \(page 504\)](#), a binary format, if the data was previously stored in Titan or TinkerGraph. Gryo files can be transferred directly using the schema from the original database.

## GraphSON

Data can be entered with [GraphSON \(page 506\)](#), a JSON format that is useful for transferring human-readable data. GraphSON files can lose data type information in transfer unless lossless data is generated.

## GraphML

Data can be entered using [GraphML \(page 507\)](#), an XML format that is useful for transferring graph data. However, data type information is lost with GraphML data transfer.

After loading data, *graph traversals* are executed to retrieve filtered information. In relational databases, *queries* are retrieved that combine and filter information. In graph databases, the vertex properties, edge connections, and edge properties all play a role in picking a starting point in the graph and traversing the connections to provide a particular answer to a query. Several `TraversalSources`, that supply a traversal strategy and traversal engine to use in executing traversals, can be generated for any `Graph`. Queries in graph databases can consist of several traversals if a complex question is asked, or trivially include no traversals, if a mathematical calculation like  $1 + 1$  is submitted.

## Managing graphs

Creating and dropping graphs.

## Creating a graph

Creating a graph.

Depending on the DSE Graph schema mode, DataStax Studio will have differing behavior. In [Production mode](#), DataStax Studio will not auto-create a graph, and the [graph must be created in the Gremlin console \(page 433\)](#). In [Development mode](#), DataStax Studio creates a graph and aliases the graph to a graph traversal automatically for each connection that is created.

DataStax Studio creates a graph automatically for each connection that is created. In Gremlin console, a graph must be manually created. In addition to creating the graph, a graph traversal must be aliased to the graph in order to run queries.

## Studio

1. [Start DSE Graph \(page 867\)](#).
2. [Install and start Studio](#). Also create a Studio notebook, if needed.
3. In DataStax Studio, [create a connection \(page 877\)](#). Choose a graph name; any graph previously unused will work.

- In DataStax Studio, [create a notebook \(page 899\)](#). Select the connection created in the last step.

A blank notebook will open with a single cell. DSE Graph runs a Gremlin Server `tinkerpop.server` on each DSE node. DataStax Studio automatically connects to the Gremlin Server, and if it doesn't exist, creates a graph using the connection information. The graph is stored as one graph instance per DSE database keyspace with a replication factor of 1 and a strategy of `SimpleStrategy`. Once a graph exists, a graph traversal `g` is configured that will allow graph traversals to be executed. Graph traversals are used to query the graph data and return results. A graph traversal is bound to a specific traversal source which is the standard OLTP traversal engine.

### Gremlin console

- [Start the Gremlin console \(page 624\)](#).
- Create a simple graph with default settings to hold the data.

```
gremlin> system.graph('food').create()
=>null
```

- Create a graph with non-default [replication \(page 649\)](#), [systemReplication \(page 650\)](#), and [configuration settings \(page 647\)](#):

```
system.graph('food2').
 replication("{'class' : 'NetworkTopologyStrategy', 'dc1' : 3 }").
 systemReplication("{'class' : 'NetworkTopologyStrategy', 'dc1' :
 3 }").
 option("graph.schema_mode").set("Production").
 option("graph.allow_scan").set("false").
 option("graph.default_property_key_cardinality").set("multiple").

option("graph.tx_groups.*.write_consistency").set("QUORUM").create()
```

**Caution:** For graphs created in multi-datacenter clusters, the [DSE database settings](#) must use `NetworkTopologyStrategy` and a replication factor greater than one. If the graph is created with a replication setting of `SimpleStrategy` and a replication factor of 1, the graph data will be stored across the multiple datacenters rather than localizing the data in the graph datacenter.

The default replication strategy for a multi-node or multi-datacenter graph is `NetworkTopologyStrategy`, whereas for a single node, the replication strategy will default to `SimpleStrategy`. The number of nodes will determine the default replication factor:

number of nodes per datacenter	<code>graph_name</code> replication factor	<code>graph_name_system</code> replication factor
1-3	number of nodes per datacenter	number of nodes per datacenter

number of nodes per datacenter	<i>graph_name</i> replication factor	<i>graph_name_system</i> replication factor
greater than 3	3	5

8. On the remote Gremlin Server, set the alias for the graph traversal `g` to the graph traversal specified in `food`. To run traversals, the graph traversal must be aliased to a graph.

```
gremlin> :remote config alias g food.g
==>g=food.g
```

## Examining graphs

How to examine graphs.

1. A list of all graphs can be retrieved with the following command:

```
system.graphs()
```

In Studio and Gremlin console, a list is retrieved, although the presentation is different. Here is a Gremlin console result:

```
==> food
==> test
```

## Dropping graphs

Dropping (deleting) graphs.

Graphs can be dropped (deleted). All schema and data for the graph will be lost, so be sure that you intend to remove a graph before using the steps below.

- A system command is required to drop a graph. If using the Gremlin console, the graph traversal alias can be cleared, to be certain no action effects the currently selected graph in the Gremlin console:

```
gremlin> :remote config alias reset
==>Aliases cleared
```

- Optional: If unsure of the graph name, [examine what graphs exist \(page 435\)](#).
- Drop the desired graph by running the `drop()` command:

```
system.graph('food').drop()
```

Use the `IfExists()` step to check if a graph exists before dropping.

```
==>null
```

## Managing graph schema

Creating, modifying, and dropping graph database schema.

### Creating graph schema

Creating graph database schema.

Creating a [data model \(page 424\)](#) for a graph database is the critical first step towards creating a schema. Once the data model is designed and a graph is created, defining the schema for the vertices and edges and their properties is the next step in creating a graph database. Gremlin-Groovy is the language used to create scripts; Gremlin-Groovy is packaged with the [Apache TinkerPop](#) engine, and can be used with either DataStax Studio or the Gremlin console (`dse gremlin-console`) installed with DataStax Enterprise.

Graph schema can be created with `create()` or added to existing schema with `add()`.

#### Prerequisites:

[Create a graph \(page 433\).](#)

1. Optional. If you are reusing a graph that you previously created, [drop the graph schema and data \(page 435\)](#).
2. Optional. If running large scripts in Gremlin console, set the `timeout` value to `max` to prevent client-side timeouts. Use this setting to ensure that script processing will complete. This step cannot be completed in Studio.

```
gremlin> :remote config timeout max
```

3. Optional. If running large scripts, set the `evaluation_timeout` value to `max` to prevent server-side timeouts. Use this setting to ensure that script processing will complete.

```
graph.schema().config().option("graph.traversal_sources.g.evaluation_timeout").set("PT1
```

4. Load the example schema listed in the Example below:

- a. In Studio, copy and paste the entire code block into a single cell and execute the cell.
- b. In Gremlin console, copy and paste the example schema to a schema file. Two choices for loading are shown:

Use the `:load` command by specifying the location of the schema file:

```
gremlin> :load /tmp/RecipeSchema.groovy
```

Use a `cat` command in the machine shell to load the file, specifying the location of the schema file, and pass to the Gremlin console:

```
$ cat /tmp/RecipeSchema.groovy | dse gremlin-console
```

**NOTE:** Each command submitted is within a single session, so from cell to cell (Studio) or line to line (Gremlin console), the Gremlin server is not aware of any

variables set on the previous line. If any of the lines in the Recipe Schema are entered separately, an error will occur on the edge creation commands.

5. The following steps show the details of the full script broken down into sections.
6. Define the properties for the vertices and the edges. The data type of the property is specified in addition to a key name. All properties created in this example are Text, Integers, or Timestamps. Other [data types \(page 740\)](#) are available. Properties will be used to retrieve selective subsets of the graph and to retrieve stored values. Properties are global in nature, and the pairing of a vertex label and a property will uniquely identify a property for use in traversals. Edge properties are expensive to update, as because the whole edge with all its properties are deleted and recreated to update edge properties. Use edge properties only in situations that warrant their use.

```
// Property Keys
// Check for previous creation of property key with ifNotExists()
schema.propertyKey('name').Text().ifNotExists().create()
schema.propertyKey('gender').Text().create()
schema.propertyKey('instructions').Text().create()
schema.propertyKey('category').Text().create()
schema.propertyKey('year').Int().create()
schema.propertyKey('timestamp').Timestamp().create()
schema.propertyKey('ISBN').Text().create()
schema.propertyKey('calories').Int().create()
schema.propertyKey('amount').Text().create()
schema.propertyKey('stars').Int().create()
schema.propertyKey('comment').Text().single().create() // single()
 is optional - default
// Example of multiple property
// schema.propertyKey('nickname').Text().multiple().create();
// Example meta-property added to property:
// schema.propertyKey('livedIn').Text().create()
//
schema.propertyKey('country').Text().multiple().properties('livedIn').create()
```

Property keys can be checked for prior existence with `ifNotExists()`. Property keys can be created with either single or multiple cardinality with `single()` or `multiple()`. The default is single cardinality which does not have to be specified, but it can be explicitly stated as in the example.

Meta-properties, or properties of properties, can be created using `propertyKey()` followed by `properties()`. The property key must exist prior to the creation of a meta-property. Meta-properties cannot be nested, i.e., a meta-property cannot have a meta-property. In this example, `country` is the property that has a meta-property `livedIn`. This property and meta-property are used to represent the countries that an author has lived in at various times in their life.

```
{
 "name": "Julia Child",
 "gender": "F",
 [{"country": "United States", "livedIn": "1929-1949" },
 {"country": "France", "livedIn": "1949-1952" }],
 "authored": [{
```

```

"book": {
 "label": "book",
 "bookTitle": "Art of French Cooking Volume One",
 "publishDate": 1968
},
"book": {
 "label": "book",
 "bookTitle": "The French Chef Cookbook",
 "publishDate": 1968,
 "ISBN": "0-394-40135-2"
}
],
"created": [
{
 "type": "recipe",
 "recipeTitle": "Beef Bourguignon",
 "instructions": "Braise the beef.",
 "createDate": 1967
},
{
 "type": "recipe",
 "recipeTitle": "Salade Nicoise",
 "instructions": "Break the lettuce into pieces.",
 "createDate": 1970
}
]
}

```

7. Define the vertex labels. The vertex labels identify the type of vertices that can be created.

```

// Vertex Labels
schema.vertexLabel('author').ifNotExists().create()
schema.vertexLabel('recipe').create()
// Example of creating vertex label with properties
//
schema.vertexLabel('recipe').properties('name', 'instructions').create()
schema.vertexLabel('ingredient').create()
schema.vertexLabel('book').create()
schema.vertexLabel('meal').create()
schema.vertexLabel('reviewer').create()
// Example of custom vertex id:
// schema.propertyKey('city_id').Int().create()
// schema.propertyKey('sensor_id').Uuid().create()
//
schema().vertexLabel('FridgeSensor').partitionKey('city_id').clusteringKey('sensor_id')

```

Vertex labels can be checked for prior existence using `ifNotExists()`. Vertex labels can be created along with properties. Vertex labels can be created with [user-defined vertex ids \(page 451\)](#), rather than the  [autogenerated vertex ids \(page 451\)](#).

**Note:** Auto-generated vertex ids are deprecated with DSE 6.0.

DSE Graph limits the number of vertex labels to 200 per graph.

8. Define the edge labels. The edge labels identify the type of edges that can be created.

```
// Edge Labels
schema.edgeLabel('authored').ifNotExists().create()
schema.edgeLabel('created').create()
schema.edgeLabel('includes').create()
schema.edgeLabel('includedIn').create()
schema.edgeLabel('rated').properties('rating').connection('reviewer','recipe').create()
```

Edge labels can be checked for prior existence using `ifNotExists()`. Edge labels can be created with adjacent vertex labels identified using [connection\(\) \(page 630\)](#). Edge labels can identify properties that an edge has using [properties\(\) \(page 642\)](#).

9. Define indexes that can speed up the query processing. All types of indexes are presented here. [Indexing \(page 457\)](#) has more information.

```
// Vertex Indexes
// Secondary
schema.vertexLabel('author').index('byName').secondary().by('name').add()
// Materialized
schema.vertexLabel('recipe').index('byRecipe').materialized().by('name').add()
schema.vertexLabel('meal').index('byMeal').materialized().by('name').add()
schema.vertexLabel('ingredient').index('byIngredient').materialized().by('name').add()
schema.vertexLabel('reviewer').index('byReviewer').materialized().by('name').add()
// Search
//
 schema.vertexLabel('recipe').index('search').search().by('instructions').asText().add()
//
 schema.vertexLabel('recipe').index('search').search().by('instructions').asString().add()
// If more than one property key is search indexed
//
 schema.vertexLabel('recipe').index('search').search().by('instructions').asText().by('
// Edge Index
schema.vertexLabel('reviewer').index('ratedByStars').outE('rated').by('stars').add()

// Example of property index using meta-property 'livedIn':
//
 schema.vertexLabel('author').index('byLocation').property('country').by('livedIn').add()
```

These indexes are included to make the schema for the food example more efficient for data loading.

**Note:** The difference between `create()` and `add()` is subtle but important. If an entity (vertex label or edge label) has been created and already exists, if an index or property keys are associated with the entity, then an `add()` command is used. For example, a vertex label and property keys can be created, and then the property keys can be added to the vertex label.

10. After creating the graph schema, examine the schema to verify. A portion of the output is shown.

Using DataStax Enterprise advanced functionality

```
schema.describe()

Real-time ▶ ⌂ ⌓ ⌚

GREMLIN ▾
> schema.describe()
+-----+
| index | value |
+-----+
| 0 | schema.propertyKey("livedin").Text().single().create() schema.propertyKey("instructions").Text().single().create() schema.propertyKey("country").Text().multiple().properties("livedin").create()
| schema.propertyKey("amount").Text().single().create() schema.propertyKey("gender").Text().single().create() schema.propertyKey("year").Int().single().create() schema.propertyKey("calories").Int().single().create()
| schema.propertyKey("stars").Int().single().create() schema.propertyKey("ISBN").Text().single().create() schema.propertyKey("name").Text().single().create() schema.propertyKey("comment").Text().single().create()
| schema.propertyKey("category").Text().single().create() schema.propertyKey("timestamp").Timestamp().single().create() schema.edgeLabel("authored").multiple().create()
| schema.edgeLabel("rated").multiple().properties("stars").create() schema.edgeLabel("includedIn").multiple().create() schema.edgeLabel("created").multiple().create()
| schema.edgeLabel("includes").multiple().create() schema.vertexLabel("meal").properties("name").create() schema.vertexLabel("meal").index("byMeal").materialized().by("name").add()
| schema.vertexLabel("ingredient").properties("name").create() schema.vertexLabel("ingredient").index("byName").materialized().by("name").add() schema.vertexLabel("author").properties("name",
| "country").create() schema.vertexLabel("author").index("byName").secondary().by("name").add() schema.vertexLabel("author").index("byLocation").property("country").by("livedin").add()
| schema.vertexLabel("book").create() schema.vertexLabel("recipe").properties("name").create() schema.vertexLabel("recipe").index("byRecipe").materialized().by("name").add()
| schema.vertexLabel("reviewer").properties("name").create() schema.vertexLabel("reviewer").index("byReviewer").materialized().by("name").add()
| schema.vertexLabel("reviewer").index("ratedByStars").outE("rated").by("stars").add() schema.edgeLabel("rated").connection("reviewer", "recipe").add() |
+-----+

```

```
// RECIPE SCHEMA

// To run in Studio, copy and paste all lines to a cell and run.

// To run in Gremlin console, use the next two lines:
// script = new File('/tmp/RecipeSchema.groovy').text; []
// :> @script

// Property Keys
// Check for previous creation of property key with ifNotExists()
schema.propertyKey('name').Text().ifNotExists().create()
schema.propertyKey('gender').Text().create()
schema.propertyKey('instructions').Text().create()
schema.propertyKey('category').Text().create()
schema.propertyKey('year').Int().create()
schema.propertyKey('timestamp').Timestamp().create()
schema.propertyKey('ISBN').Text().create()
schema.propertyKey('calories').Int().create()
schema.propertyKey('amount').Text().create()
schema.propertyKey('stars').Int().create()
schema.propertyKey('comment').Text().single().create() // single()
is optional - default
// Example of multiple property
// schema.propertyKey('nickname').Text().multiple().create();
// Example meta-property added to property:
// schema.propertyKey('livedIn').Text().create()
//
schema.propertyKey('country').Text().properties('livedIn').create()

// Vertex Labels
schema.vertexLabel('author').ifNotExists().create()
schema.vertexLabel('recipe').create()
// Example of creating vertex label with properties
//
schema.vertexLabel('recipe').properties('name','instructions').create()
```

```

schema.vertexLabel('ingredient').create()
schema.vertexLabel('book').create()
schema.vertexLabel('meal').create()
schema.vertexLabel('reviewer').create()
// Example of custom vertex id:
// schema.propertyKey('city_id').Int().create()
// schema.propertyKey('sensor_id').Uuid().create()
//
schema().vertexLabel('FridgeSensor').partitionKey('city_id').clusteringKey('sensor_id')

// Edge Labels
schema.edgeLabel('authored').ifNotExists().create()
schema.edgeLabel('created').create()
schema.edgeLabel('includes').create()
schema.edgeLabel('includedIn').create()
schema.edgeLabel('rated').properties('stars').connection('reviewer', 'recipe').create()

// Vertex Indexes
// Secondary
schema.vertexLabel('author').index('byName').secondary().by('name').add()
// Materialized
schema.vertexLabel('recipe').index('byRecipe').materialized().by('name').add()
schema.vertexLabel('meal').index('byMeal').materialized().by('name').add()
schema.vertexLabel('ingredient').index('byIngredient').materialized().by('name').add()
schema.vertexLabel('reviewer').index('byReviewer').materialized().by('name').add()
// Search
//
schema.vertexLabel('recipe').index('search').search().by('instructions').asText().add()
//
schema.vertexLabel('recipe').index('search').search().by('instructions').asString().add()
// If more than one property key is search indexed
//
schema.vertexLabel('recipe').index('search').search().by('instructions').asText().by('c')

// Edge Index
schema.vertexLabel('reviewer').index('ratedByStars').outE('rated').by('stars').add()

// Example of property index using meta-property 'livedIn':
//
schema().vertexLabel('author').index('byLocation').property('country').by('livedIn').add()

// Schema description
// Use to check that the schema is built as desired
schema.describe()

```

## Examining schema

How to examine schema.

Examining schema can be done to either verify the existence or non-existence of schema before creation.

1. A list of all schema can be retrieved with the following command:

```
schema.describe()
```

In Studio and Gremlin console, a list is retrieved, although the presentation is different. Here is a portion of a Studio result:

index ↑	value
0	<pre> schema.propertyKey("livedIn").Text().single().create() schema.propertyKey("instructions").Text().single().create() schema.propertyKey("country").Text().multiple().properties("livedIn").create() schema.propertyKey("amount").Text().single().create() schema.propertyKey("gender").Text().single().create() schema.propertyKey("year").Int().single().create() schema.propertyKey("calories").Int().single().create() schema.propertyKey("stars").Int().single().create() schema.propertyKey("ISBN").Text().single().create() schema.propertyKey("name").Text().single().create() schema.propertyKey("comment").Text().single().create() schema.propertyKey("category").Text().single().create() schema.propertyKey("timestamp").Timestamp().single().create() schema.edgeLabel("authored").multiple().create() schema.edgeLabel("rated").multiple().properties("stars").create() schema.edgeLabel("includedIn").multiple().create() schema.edgeLabel("created").multiple().create() schema.edgeLabel("includes").multiple().create() schema.vertexLabel("meal").properties("name").create() schema.vertexLabel("meal").index("byMeal").materialized().by("name").add() schema.vertexLabel("ingredient").properties("name").create() schema.vertexLabel("ingredient").index("byIngredient").materialized().by("name").add() schema.vertexLabel("author").properties("name", "country").create() schema.vertexLabel("author").index("byName").secondary().by("name").add() schema.vertexLabel("author").index("byLocation").property("country").by("livedIn").add() schema.vertexLabel("book").create() schema.vertexLabel("recipe").properties("name").create() schema.vertexLabel("recipe").index("byRecipe").materialized().by("name").add() schema.vertexLabel("reviewer").properties("name").create() schema.vertexLabel("reviewer").index("byReviewer").materialized().by("name").add() schema.vertexLabel("reviewer").index("ratedByStars").outE("rated").by("stars").add() schema.edgeLabel("rated").connection("reviewer", "recipe").add() </pre>

Displaying 1 - 1 of 1  
1 element returned. Duration: 0.013 s.

2. The schema for a particular element can be examined by appending `describe()` to a more specific schema:

```
schema.edgeLabel('includedIn').describe()
```

```

schema.edgeLabel("includedIn").multiple().properties("amount").create()
schema.edgeLabel("includedIn").connection("recipe",
"meal").connection("recipe", "book").connection("recipe",
"ingredient").connection("ingredient",
"recipe").connection("meal", "book").add()

```

3. Some groovy steps are useful in the Gremlin query to find specific schema descriptions. For instance, to inspect only the vertex labels and their indexes, use the `split()` and `grep()` steps:

```
schema.describe().split('\n').grep(~/.vertexLabel.*/)
```

In Studio:

The screenshot shows the Gremlin interface with the command `[schema.describe().split('\n').grep(-/,^vertexLabel.*).]` entered. The results are displayed in a table with two columns: `index` and `value`. The table contains 27 rows of schema definitions, mostly for vertex labels like `recipe`, `store`, `person`, and `meal`, with their respective properties and indexes. The interface includes standard navigation buttons and a status bar at the bottom.

index	value
0	schema.vertexLabel("recipe").partitionKey("recipId").properties("name", "cuisine", "instructions...")
1	schema.vertexLabel("recipe").index("search").by("instructions").by("name").by("cuisine")...
2	schema.vertexLabel("recipe").index("byStars").by("reviewed").by("stars").add()
3	schema.vertexLabel("recipe").index("byPersonOrRecipe").bothE("created").by("createDate").add()
4	schema.vertexLabel("store").partitionKey("storeId").properties("name", "address").create()
5	schema.vertexLabel("store").index("search").by("name").add()
6	schema.vertexLabel("meal_item").partitionKey("itemId").properties("name", "servAmt", "macro", "ca...")
7	schema.vertexLabel("meal_item").index("byName").materialized().by("name").add()
8	schema.vertexLabel("fridgeSensor").partitionKey("stateId", "cityId").clusteringKey("sensorId").pr...
9	schema.vertexLabel("fridgeSensor").index("search").by("cityId").by("sensorId").by("name")...
10	schema.vertexLabel("home").partitionKey("homId").properties("name", "address").create()
11	schema.vertexLabel("home").index("search").by("name").add()
12	schema.vertexLabel("ingredient").partitionKey("ingredId").properties("name").create()
13	schema.vertexLabel("ingredient").index("byName").materialized().by("name").add()
14	schema.vertexLabel("person").partitionKey("personId").properties("name", "nickName", "gender", "c...")
15	schema.vertexLabel("person").index("byName").materialized().by("name").add()
16	schema.vertexLabel("person").index("ratedByStars").outE("reviewed").by("stars").add()
17	schema.vertexLabel("person").index("ratedByDate").outE("reviewed").by("year").add()
18	schema.vertexLabel("person").index("ratedByComments").outE("reviewed").by("comment").add()
19	schema.vertexLabel("person").index("byStartYear").property("country").by("startYear").add()
20	schema.vertexLabel("person").index("byEndYear").property("country").by("endYear").add()
21	schema.vertexLabel("book").partitionKey("bookId").properties("name", "publishYear", "ISBN", "book...")
22	schema.vertexLabel("book").index("search").by("name").by("publishYear").add()
23	schema.vertexLabel("location").partitionKey("locId").properties("name", "geoPoint").create()
24	schema.vertexLabel("location").index("search").by("geoPoint").withError(0.0E-6, 0.0).add()
25	schema.vertexLabel("meal").partitionKey("type").create()
26	schema.vertexLabel("meal").index("byType").secondary().by("type").add()

## Creating property key schema

Creating or adding property key schema.

Property keys ([page 642](#)), as discussed in the data model ([page 423](#)), allow attribute values to be assigned to either vertices or edges. Property keys can also have property keys assigned to them as meta-properties using `properties()`. Property keys are defined before assigning them to vertex or edge labels, and indexes can be created based on property keys.

Properties will be used to retrieve selective subsets of the graph and to retrieve stored values. Properties are global in nature, and the pairing of a vertex label and a property will uniquely identify a property for use in traversals. Edge properties are expensive to update, because the whole edge, with all its properties, are deleted and recreated to update edge properties. Use edge properties only in situations that warrant their use.

The cardinality of a property key is single [cardinality](#) by default, but multiple cardinality can be assigned. Additionally, property keys can be created with a specific Time-To-Live (TTL) value. When creating property keys, prior existence of a key can be checked with `ifNotExists()`.

Property keys ([page 642](#)) must have a key name and [data type](#) ([page 740](#)) assigned, at a minimum. Several data types exist to accommodate data storage in flexible formats. Some warrant special attention, such as geospatial and Cartesian. Three geospatial data types, `point`, `linestring`, and `polygon`, store data that can be searched with geospatial shapes. For most geospatial queries that look for geospatial points, points or linestrings within circles or polygons, [DSE Search indexes](#) ([page 461](#)) must also be created; the same is true for Cartesian data types.

Property key schema are created with `create()`. Either DataStax Studio or the Gremlin console (`dse gremlin-console`) installed with DataStax Enterprise can be used to create or modify property keys.

**Prerequisites:**

[Create a graph \(page 433\)](#).

**Single and multiple cardinality property keys**

1. Define a single cardinality property key:

The name of the propertyKey is `personId`, it is defined as an `Integer` type with `Int()`, with single cardinality explicitly stated with `single()`:

```
schema.propertyKey('personId').Int().single().create()
```

Since single cardinality is the default, `single()` can be omitted:

```
schema.propertyKey('personId').Int().create()
```

To check if a property key already exists before creating it, use `ifNotExists()`:

```
schema.propertyKey('createDate').Date().single().ifNotExists().create()
```

2. Define a multiple cardinality property key:

The name of the propertyKey is `nickname`, it is defined as an `Text` type with `Text()`, with multiple cardinality:

```
schema.propertyKey('nickname').Text().multiple().create()
```

Since multiple cardinality is not the default, `multiple()` must be included.

**Meta-properties**

3. Define a single cardinality meta-property to a multiple cardinality property key. First create the property keys for the meta-property, then create the property key that has the meta-property.

The name of the meta-property propertyKey is `since`, it is defined as an `Integer` type with `Int()`, with single cardinality. The name of the propertyKey that uses this meta-property is `badge`, it is defined as an `Text` type with `Text()`, with multiple cardinality. The meta-property is defined with `properties`.

```
// MULTIPLE CARDINALITY PROPERTY KEY WITH SINGLE CARDINALITY META-PROPERTY
schema.propertyKey('since').Int().single().create() // meta-property
schema.propertyKey('badge').Text().multiple().properties('since').create()
```

4. Both properties and meta-properties can be defined with multiple cardinality. Define a multiple cardinality meta-property to a multiple cardinality property key. First create the property keys for the meta-property, then create the property key that has the meta-property.

The name of the meta-property `propertyKey` is `startYear`, it is defined as an `Integer` type with `Int()`, with multiple cardinality. The name of the `propertyKey` that uses this meta-property is `country`, it is defined as an `Text` type with `Text()`, with multiple cardinality. The meta-property is defined with `properties`.

```
// MULTIPLE CARDINALITY PROPERTY KEY WITH MULTIPLE CARDINALITY
META-PROPERTY
schema.propertyKey('startYear').Int().multiple().create() // meta-property
schema.propertyKey('endYear').Int().multiple().create() // meta-property
schema.propertyKey('country').Text().multiple().properties('startYear', 'endYear').create()
```

This example includes two meta-properties, `startYear` and `endYear`.

### Property keys with Time-To-Live (TTL)

5. Property keys can be defined with a Time-To-Live (TTL) value; once the specified time is reached, the value is deleted from the graph. Define a single cardinality property key with TTL:

The name of the `propertyKey` is `bookDiscount`, it is defined as an `Text` type with `Text()`, single cardinality (default), and a TTL of 604,800 seconds with `ttl()`:

```
// PROPERTY KEY WITH TTL
schema.propertyKey('bookDiscount').Text().ttl(604800).create()
```

### Geospatial property keys

6. Create schema for `point` property key:

```
schema.propertyKey('point').Point().withGeoBounds().create()
```

**Note:** For all geospatial elements, the `withGeoBounds()` method limits searches to a default valid range of latitude in degrees from -90 to +90 (South Pole to North Pole) and a valid range of longitude in degrees from -180 to +180 (east to west from the [Greenwich Meridian](#)). The point is specified using `Geo.point(longitude, latitude)` when adding points, using [WellKnownText \(WKT\)](#) format. Note that it specifies **longitude** first, then **latitude**.

7. Create schema for a `linestring` property key:

```
schema.propertyKey('line').Linestring().withGeoBounds().create()
```

The same [boundary limits \(page 445\)](#) imposed on points are imposed on linestrings.

8. Create schema for a `polygon` property key:

```
schema.propertyKey('polygon').Polygon().withGeoBounds().create()
```

The same [boundary limits \(page 445\)](#) imposed on points are imposed on polygons.

## Cartesian property keys

9. Create schema for *point* property key:

```
schema.propertyKey('point').Point().withBounds(-3, -3, 3,
3).create()
```

**Note:** For Cartesian spatial types, the `withBounds(x1, y1, x2, y2)` method limit searches to a default valid range of values in the x-y grid.

10. Create schema for a *linestring* property key:

```
schema.propertyKey('line').Linestring().withBounds(-3, -3, 3,
3).create()
```

The same [boundary limits \(page 446\)](#) imposed on points are imposed on linestrings.

11. Create schema for a *polygon* property key:

```
schema.propertyKey('polygon').Polygon().withBounds(-3, -3, 3,
3).create()
```

The same [boundary limits \(page 446\)](#) imposed on points are imposed on polygons.

12. Define the properties for the vertices and the edges. The data type of the property is specified in addition to a key name. All properties created in this example are Text, Integers, or Timestamps. Other [data types \(page 740\)](#) are available. Properties will be used to retrieve selective subsets of the graph and to retrieve stored values. Properties are global in nature, and the pairing of a vertex label and a property will uniquely identify a property for use in traversals. Edge properties are expensive to update, as because the whole edge with all its properties are deleted and recreated to update edge properties. Use edge properties only in situations that warrant their use.

```
// *****
// PROPERTY KEYS
// *****
// SYNTAX:
// propertyKey('name').
// type().
// [single() | multiple()].
// [ttl].
// [properties(metadata_property)].
// [ifNotExists()].
// [create() | add() | describe() | exists()]
// DEFAULT IS SINGLE CARDINALITY
// *****

// SINGLE CARDINALITY PROPERTY KEY
schema.propertyKey('personId').Int().single().create()
schema.propertyKey('mealId').Int().single().create()
schema.propertyKey('itemId').Int().single().create()
schema.propertyKey('recipeId').Int().single().create()
schema.propertyKey('bookId').Int().single().create()
```

```

schema.propertyKey('ingredId').Int().single().create()
schema.propertyKey('homeId').Int().single().create()
schema.propertyKey('storeId').Int().single().create()
schema.propertyKey('locId').Text().single().create()
schema.propertyKey('stateId').Int().single().create()
schema.propertyKey('cityId').Int().single().create()
schema.propertyKey('sensorId').Int().single().create()
schema.propertyKey('name').Text().single().create()
schema.propertyKey('gender').Text().single().create()
schema.propertyKey('calGoal').Int().single().create()
schema.propertyKey('macroGoal').Text().single().create()
schema.propertyKey('publishYear').Int().single().create()
schema.propertyKey('ISBN').Text().single().create()
// PROPERTY KEY WITH TTL
schema.propertyKey('bookDiscount').Text().ttl(604800).create()
schema.propertyKey('instructions').Text().single().create()
schema.propertyKey('notes').Text().single().create()
schema.propertyKey('type').Text().single().create()
schema.propertyKey('servAmt').Text().single().create()
schema.propertyKey('macro').Text().single().create()
schema.propertyKey('calories').Int().single().create()
schema.propertyKey('geoPoint').Point().withGeoBounds().create()
schema.propertyKey('address').Text().single().create()
schema.propertyKey('amount').Text().single().create()
// MULTIPLE CARDINALITY PROPERTY KEY
schema.propertyKey('nickname').Text().multiple().create()
schema.propertyKey('cuisine').Text().multiple().create()
// MULTIPLE CARDINALITY PROPERTY KEY WITH SINGLE CARDINALITY META-PROPERTY
schema.propertyKey('since').Int().single().create() // meta-property
schema.propertyKey('badge').Text().multiple().properties('since').create()
// MULTIPLE CARDINALITY PROPERTY KEY WITH MULTIPLE CARDINALITY META-PROPERTY
schema.propertyKey('startYear').Int().multiple().create() // meta-property
schema.propertyKey('endYear').Int().multiple().create() // meta-property
schema.propertyKey('country').Text().multiple().properties('startYear', 'endYear').create()

// EDGE PROPERTIES
schema.propertyKey('numServ').Int().single().create()
schema.propertyKey('mealDate').Date().single().create()
schema.propertyKey('useDate').Date().single().create()
schema.propertyKey('createDate').Date().single().create()
schema.propertyKey('expireDate').Date().single().create()
schema.propertyKey('stars').Int().single().create()
schema.propertyKey('time').Time().single().create()
schema.propertyKey('year').Date().single().create()
schema.propertyKey('comment').Text().single().create()

// Property Keys
// Check for previous creation of property key with ifNotExists()

```

```

schema.propertyKey('name').Text().ifNotExists().create()
schema.propertyKey('gender').Text().create()
schema.propertyKey('instructions').Text().create()
schema.propertyKey('category').Text().create()
schema.propertyKey('year').Int().create()
schema.propertyKey('timestamp').Timestamp().create()
schema.propertyKey('ISBN').Text().create()
schema.propertyKey('calories').Int().create()
schema.propertyKey('amount').Text().create()
schema.propertyKey('stars').Int().create()
schema.propertyKey('comment').Text().single().create() // single()
 is optional - default
// Example of multiple property
// schema.propertyKey('nickname').Text().multiple().create();
// Example meta-property added to property:
// schema.propertyKey('livedIn').Text().create()
//
schema.propertyKey('country').Text().multiple().properties('livedIn').create()

```

Property keys can be checked for prior existence with `ifNotExists()`. Property keys can be created with either single or multiple cardinality with `single()` or `multiple()`. The default is single cardinality which does not have to be specified, but it can be explicitly stated as in the example.

Meta-properties, or properties of properties, can be created using `propertyKey()` followed by `properties()`. The property key must exist prior to the creation of a meta-property. Meta-properties cannot be nested, i.e., a meta-property cannot have a meta-property. In this example, `country` is the property that has a meta-property `livedIn`. This property and meta-property are used to represent the countries that an author has lived in at various times in their life.

```

{
 "name": "Julia Child",
 "gender": "F",
 [{"country": "United States", "livedIn": "1929-1949" },
 {"country": "France", "livedIn": "1949-1952" }],
 "authored": [{
 "book": {
 "label": "book",
 "bookTitle": "Art of French Cooking Volume One",
 "publishDate": 1968
 },
 "book": {
 "label": "book",
 "bookTitle": "The French Chef Cookbook",
 "publishDate": 1968,
 "ISBN": "0-394-40135-2"
 }
 }],
 "created": [{
 "type": "recipe",
 "recipeTitle": "Beef Bourguignon",

```

```

 "instructions" : "Braise the beef.",
 "createDate":1967
 },
 {
 "type" : "recipe",
 "recipeTitle" : "Salade Nicoise",
 "instructions" : "Break the lettuce into pieces.",
 "createDate": 1970
 }
]
}

```

```

// RECIPE SCHEMA

// To run in Studio, copy and paste all lines to a cell and run.

// To run in Gremlin console, use the next two lines:
// script = new File('/tmp/RecipeSchema.groovy').text; []
// :> @script

// Property Keys
// Check for previous creation of property key with ifNotExists()
schema.propertyKey('name').Text().ifNotExists().create()
schema.propertyKey('gender').Text().create()
schema.propertyKey('instructions').Text().create()
schema.propertyKey('category').Text().create()
schema.propertyKey('year').Int().create()
schema.propertyKey('timestamp').Timestamp().create()
schema.propertyKey('ISBN').Text().create()
schema.propertyKey('calories').Int().create()
schema.propertyKey('amount').Text().create()
schema.propertyKey('stars').Int().create()
schema.propertyKey('comment').Text().single().create() // single()
is optional - default
// Example of multiple property
// schema.propertyKey('nickname').Text().multiple().create();
// Example meta-property added to property:
// schema.propertyKey('livedIn').Text().create()
//
schema.propertyKey('country').Text().properties('livedIn').create()

// Vertex Labels
schema.vertexLabel('author').ifNotExists().create()
schema.vertexLabel('recipe').create()
// Example of creating vertex label with properties
//
schema.vertexLabel('recipe').properties('name','instructions').create()
schema.vertexLabel('ingredient').create()
schema.vertexLabel('book').create()
schema.vertexLabel('meal').create()
schema.vertexLabel('reviewer').create()

```

```

// Example of custom vertex id:
// schema.propertyKey('city_id').Int().create()
// schema.propertyKey('sensor_id').Uuid().create()
//
schema().vertexLabel('FridgeSensor').partitionKey('city_id').clusteringKey('sensor_id')

// Edge Labels
schema.edgeLabel('authored').ifNotExists().create()
schema.edgeLabel('created').create()
schema.edgeLabel('includes').create()
schema.edgeLabel('includedIn').create()
schema.edgeLabel('rated').properties('stars').connection('reviewer', 'recipe').create()

// Vertex Indexes
// Secondary
schema.vertexLabel('author').index('byName').secondary().by('name').add()
// Materialized
schema.vertexLabel('recipe').index('byRecipe').materialized().by('name').add()
schema.vertexLabel('meal').index('byMeal').materialized().by('name').add()
schema.vertexLabel('ingredient').index('byIngredient').materialized().by('name').add()
schema.vertexLabel('reviewer').index('byReviewer').materialized().by('name').add()
// Search
//
schema.vertexLabel('recipe').index('search').search().by('instructions').asText().add()
//
schema.vertexLabel('recipe').index('search').search().by('instructions').asString().add()
// If more than one property key is search indexed
//
schema.vertexLabel('recipe').index('search').search().by('instructions').asText().by('name').add()

// Edge Index
schema.vertexLabel('reviewer').index('ratedByStars').outE('rated').by('stars').add()

// Example of property index using meta-property 'livedIn':
//
schema().vertexLabel('author').index('byLocation').property('country').by('livedIn').add()

// Schema description
// Use to check that the schema is built as desired
schema.describe()

```

## Creating vertex label schema

Creating vertex label database schema.

[Vertex labels \(page 644\)](#), as discussed in the [data model \(page 423\)](#), define the vertex id and associated property keys for each type of vertex created. [Property keys must be created \(page 443\)](#) prior to using them in vertex label creation. Vertex label schema can be created with `create()` or property keys can be added to existing schema with `add()`. Vertex labels can be created with a specific Time-To-Live (TTL) value, or prior existence of a vertex label can be checked using `ifNotExists()`.

A key component of a vertex label is the *vertex id* which identifies the data locality with which vertices with a particular vertex label will be stored. User-defined vertex ids (UDV ids) are analogous to a primary key in RDBMS, and map directly to the underlying data representation of the graph in DataStax's distribution of Apache Cassandra™. UDV ids identify the unique property keys that define the `partitionKey` and `clusteringKey` of a vertex label. The values associated with the UDV ids define the node in a DSE cluster that vertices will be partitioned (`partitionKey`), and the order in which the data is stored in the associated tables (`clusteringKey`).

A UDV id can be defined using three different arrangements for the vertex id:

**Single-key: composed of a single property in a `partitionKey`**

Maps every instance of a vertex label to a distinct DSE partition and distributes the data around the DSE cluster based on DSE distribution methodologies.

**Multiple-key: composed of more than one property in a `partitionKey`**

Maps a particular vertex label and associated properties to a distinct DSE partition, but contains more than one property key to identify the uniqueness.

**Composite key: composed of a `partitionKey` and a `clusteringKey`**

Includes both a `partitionKey` which maps a particular vertex label to a distinct DSE partition and one or more `clusteringKeys` to group data within a partition.

**Caution:** Keep in mind that UDV ids must be globally unique within the graph.

Auto-generated vertex ids also exist, but are discouraged. If a `partitionKey` or `clusteringKey` are not specified, an auto-generated vertex id will be created that assigns values to two internal properties, `community_id` as a `partitionKey` and `member_id` as a `clusteringKey`. Because a unique id is created for every vertex, duplicate elements can be created with the same property values accidentally, leading to confusion.

**Note:** Auto-generated vertex ids are deprecated with DSE 6.0.

Caching can improve query performance and is configurable. DSE Graph has two types of cache: adjacency list cache and index/property cache. Either edges or properties can be cached using the `cache()` option with [vertexLabel\(\) \(page 644\)](#). Caching can be configured for all edges, all properties, or a filtered set of edges. Vertices are not cached directly, but caching properties and edges that define the relationship between vertices essentially accomplishes the same operation. The best use of caching is for static values.

Property caching is enabled if indexes exist and are used in the course of queries. Full graph scan queries will not be cached. If an index does not exist, then caching does not occur. Adjacency list caching is enabled if caching is configured for edges.

The caches are local to a node and data is loaded into cache when it is read with a query. Both caches are set to a default size of 128 MB in the `dse.yaml` file. The settings are `adjacency_cache_size_in_mb` and `index_cache_size_in_mb`. Both caches utilize off-heap memory implemented as Least Recently Used (LRU) cache.

Graph cache is local to each node in the cluster, so the cached data can be different between nodes. Thus, a query can use cache on one node, but not on another. The caches are updated only when the data is not found. Graph caching does not have any means of eviction. No flushing occurs, and the cache is not updated if an element is deleted or modified. The cache will only evict data based on the time-to-live (TTL)

value set when the cache is configured for an element. Set a low TTL value for elements ([property keys \(page 642\)](#), [vertex labels \(page 644\)](#), [edge labels \(page 637\)](#)) that change often to avoid stale data.

Caching is intended to help make queries more efficient if the same information is required in a later query. For instance, caching the `calories` property for `meal_item` vertices will improve the retrieval of a query asking for *all meal items with a calorie count less than 850 calories*. However, it is useful only for rarely changed graph data, and queries run in the same sort order. Caching `calories` for the query above will not reduce query latency if the query asks for *all recipes with a calorie count greater than 850 calories*.

DSE Graph limits the number of vertex labels to 200 per graph.

### Prerequisites:

[Create property key schema \(page 443\)](#).

### User-defined vertex ids

1. Create a vertex label with a single-key vertex id of `sensorId`. The property key `sensorId` must exist prior to use in creating the vertex label and cannot be a multiple cardinality property.

```
schema.vertexLabel('FridgeSensor').partitionKey('sensorId').create()
```

This vertex id will store data based on the unique `sensorId` value for each `FridgeSensor`, distributing the data throughout the entire DSE cluster.

2. Create a vertex label with a composite key vertex id of `cityId` and clustering key `sensorId`.

```
schema.vertexLabel('FridgeSensor').partitionKey('cityId').clusteringKey('sensorId').cre
```

The vertex id in this example will store all data for `FridgeSensors` with a particular `cityId` on the same partition, but order the data based on the `sensorId`. If the city has a large number of sensors, the table storing these vertices could grow quite big.

3. Create a vertex label with a multiple-key vertex id using both `cityId` and `sensorId` as part of the partitioning key.

```
schema.vertexLabel('FridgeSensor').partitionKey('cityId', 'sensorId').create()
```

This vertex id will hash both property keys before distributing the data in the cluster, so that each is uniquely stored based on more information.

### Auto-generated vertex ids

4. If no `partitionKey` or `clusteringKey` are specified, an auto-generated vertex id will be generated when data is created:

```
schema.vertexLabel('stupid').create()
```

The vertex id consists of the label plus the two attributes `community_id` and a `member_id`:

```
{~label=stupid, community_id=1270013568, member_id=0}
```

## Associating property keys with vertex labels

- Properties can be defined in either the `create()` or `add()` statement:

```
schema.vertexLabel('book').partitionKey('bookId').create()
schema.vertexLabel('book').properties('name', 'publishYear', 'ISBN', 'bookDiscount').add()
```

or

```
schema.vertexLabel('book').partitionKey('bookId').properties('publishYear',
 'ISBN', 'name', 'bookDiscount').create()
```

## Caching

- Cache all properties for `person` vertices up to an hour (3600 seconds):

```
schema.vertexLabel('person').cache().properties().ttl(3600).add()
```

Enabling property cache causes index queries to use an index cache for the specified vertex label.

- Cache both incoming and outgoing `created` edges for `person` vertices up to a minute (60 seconds):

```
schema.vertexLabel('person').cache().bothE('created').ttl(60).add()
```

The vertex labels used for the DSE QuickStart example used throughout the documentation:

```
// *****
// VERTEX LABELS
// *****
// SYNTAX:
// schema.vertexLabel('vertexLabel').
// [partitionKey(propertyKey,
// [partitionKey(propertyKey)]].
// [clusteringKey(propertyKey)].
// [ttl].
// [properties(property, property)].
// [index].
// [cache()].
// [ifNotExists()].
// [create() | add() | describe() | exists()]
// *****

// SINGLE-KEY VERTEX ID
schema.vertexLabel('person').partitionKey('personId').create()
schema.vertexLabel('person').properties('name', 'nickname', 'gender', 'calGoal', 'macroGoal')
schema.vertexLabel('book').partitionKey('bookId').create()
schema.vertexLabel('book').properties('name', 'publishYear', 'ISBN', 'bookDiscount').add()
schema.vertexLabel('meal_item').partitionKey('itemId').create()
```

```

schema.vertexLabel('meal_item').properties('name', 'servAmt',
 'macro', 'calories').add()
schema.vertexLabel('ingredient').partitionKey('ingredId').create()
schema.vertexLabel('ingredient').properties('name').add()
schema.vertexLabel('home').partitionKey('homeId').create()
schema.vertexLabel('home').properties('name', 'address').add()
schema.vertexLabel('store').partitionKey('storeId').create()
schema.vertexLabel('store').properties('name', 'address').add()
schema.vertexLabel('location').partitionKey('locId').create()
schema.vertexLabel('location').properties('name',
 'geoPoint').add()
schema.vertexLabel('recipe').partitionKey('recipeId').create()
schema.vertexLabel('recipe').properties('name', 'cuisine',
 'instructions', 'notes').add()
// MULTIPLE-KEY VERTEX ID
schema.vertexLabel('meal').partitionKey('type', 'mealId').create()
// COMPOSITE KEY VERTEX ID
schema.vertexLabel('fridgeSensor').partitionKey('stateId',
 'cityId').clusteringKey('sensorId').create()
schema.vertexLabel('fridgeSensor').properties('name').add()

```

## Creating edge label schema

Creating edge label database schema.

[Edge labels \(page 637\)](#), as discussed in the [data model \(page 423\)](#), define the name of the edge label, incoming and outgoing vertex labels connected by an edge, and associated property keys for the edge. [Property keys \(page 443\)](#) and [vertex labels \(page 450\)](#) must be created prior to using them in edge label creation. Edge label schema can be created with `create()`, and property keys or connections can be added to existing schema with `add()`. The cardinality of an edge label is multiple [cardinality](#) by default, but single cardinality can be assigned. Edge labels can be created with a specific Time-To-Live (TTL) value, or prior existence of a edge label can be checked using `ifNotExists()`.

A key component of an edge label is the `connection()` which identifies the two types of vertices that are connected with an edge using the edge label. The connection has directionality specified by the order, going from the outgoing vertex label listed first to the incoming vertex label listed second.

### Prerequisites:

[Create property key schema \(page 443\)](#).

### Single and multiple cardinality edge labels

1. Define a single cardinality edge label:

The name of the edgeLabel is `ate`, it is defined with multiple cardinality explicitly stated with `multiple()`:

```

schema.edgeLabel('ate').multiple().create()

```

Since multiple cardinality is the default, `multiple()` can be omitted:

```
schema.edgeLabel('ate').create()
```

To check if an edge label already exists before creating it, use `ifNotExists()`:

```
schema.edgeLabel('knows').multiple().create().ifNotExists().create()
```

## 2. Define a single cardinality edge label:

The name of the `edgeLabel` is *has*, it is defined with single cardinality:

```
schema.edgeLabel('has').single().create()
```

Since single cardinality is not the default, `single()` must be included.

### Associating property keys with edge labels

#### 3. Properties can be defined in either the `create()` or `add()` statement:

```
schema.edgeLabel('ate').multiple().create()
schema.edgeLabel('ate').properties('mealDate').add()
```

or

```
schema.edgeLabel('ate').multiple().properties('mealDate').create()
```

**Important:** If cardinality (`single` or `multiple`) or `properties()` are specified, those options must precede the `connection()` option in the schema statement.

### Connections

#### 4. Connections can be defined in either the `create()` or `add()` statement:

```
schema.edgeLabel('ate').multiple().create()
schema.edgeLabel('ate').connection('person', 'meal').add()
```

or

```
schema.edgeLabel('ate').multiple().connection('person',
'meal').create()
```

#### 5. Multiple connections between vertex labels can be created for the same edge label:

```
schema.edgeLabel('includedIn').connection('recipe', 'meal').add()
schema.edgeLabel('includedIn').connection('meal', 'book').add()
schema.edgeLabel('includedIn').connection('recipe', 'book').add()
```

These schema statements will allow edges to be inserted that use the edge label *includedIn* between recipes and meals, meals and books, and recipe and books.

### Order restrictions

#### 6. If cardinality (`single` or `multiple`) or `properties()` are specified, those options must precede the `connection()` option in the schema statement.

Multiple steps:

```
schema.edgeLabel('includedIn').multiple().create()
schema.edgeLabel('includedIn').properties('amount').add()
schema.edgeLabel('includedIn').connection('recipe', 'meal').add()
schema.edgeLabel('includedIn').connection('meal', 'book').add()
schema.edgeLabel('includedIn').connection('recipe', 'book').add()
schema.edgeLabel('includedIn').connection('ingredient', 'recipe').add()
```

Single step:

```
schema.edgeLabel('includedIn').multiple().properties('amount').connection('recipe', 'meal')
```

### Edge labels with Time-To-Live (TTL)

7. Edge labels can be defined with a Time-To-Live (TTL) value; once the specified time is reached, the value is deleted from the graph. Define an edge label with TTL:

The name of the edge label is *createDate* is defined with a TTL of 60 seconds with *ttl()*:

```
schema.edgeLabel('createDate').ttl(60).create()
```

The edge labels used for the DSE QuickStart example used throughout the documentation:

```
// *****
// EDGE LABELS
// *****
// SYNTAX:
//schema.edgeLabel('edgeLabel').
// [single() | multiple()].
// [connection(outVertex, inVertex)].
// [ttl].
// [properties(property[, property])].
// [ifNotExists()].
// [create() | add() | describe() | exists()]
// DEFAULT IS MULTIPLE CARDINALITY
// *****

schema.edgeLabel('ate').multiple().create()
schema.edgeLabel('ate').properties('mealDate').add()
schema.edgeLabel('ate').connection('person', 'meal').add()
schema.edgeLabel('knows').multiple().create()
schema.edgeLabel('knows').properties('since').add()
schema.edgeLabel('knows').connection('person', 'person').add()
schema.edgeLabel('includes').multiple().create()
schema.edgeLabel('includes').properties('numServ').add()
schema.edgeLabel('includes').connection('meal', 'meal_item').add()
schema.edgeLabel('includedIn').multiple().create()
schema.edgeLabel('includedIn').properties('amount').add()
schema.edgeLabel('includedIn').connection('recipe', 'meal').add()
schema.edgeLabel('includedIn').connection('meal', 'book').add()
schema.edgeLabel('includedIn').connection('recipe', 'book').add()
```

```

schema.edgeLabel('includedIn').connection('ingredient','recipe').add()
schema.edgeLabel('created').multiple().create()
schema.edgeLabel('created').properties('createDate').add()
schema.edgeLabel('created').connection('person', 'recipe').add()
schema.edgeLabel('reviewed').multiple().create()
schema.edgeLabel('reviewed').properties('time', 'year', 'stars', 'comment').add()
schema.edgeLabel('reviewed').connection('person', 'recipe').add()
schema.edgeLabel('authored').multiple().create()
schema.edgeLabel('authored').connection('person', 'book').add()
schema.edgeLabel('contains').multiple().ttl(60800).create()
schema.edgeLabel('contains').properties('expireDate').add()
schema.edgeLabel('contains').connection('fridgeSensor', 'ingredient').add()
schema.edgeLabel('isStockedWith').multiple().ttl(60800).create()
schema.edgeLabel('isStockedWith').properties('expireDate').add()
schema.edgeLabel('isStockedWith').connection('store', 'ingredient').add()
schema.edgeLabel('isLocatedAt').multiple().create()
schema.edgeLabel('isLocatedAt').connection('home', 'location').add()
schema.edgeLabel('isLocatedAt').connection('store', 'location').add()
schema.edgeLabel('isLocatedAt').connection('fridgeSensor', 'home').add()

```

## Indexing

Explain indexes and how they affect DSE Graph performance.

Indexes play a significant role in making DSE Graph queries performant. Graph queries that must traverse the entire graph to find information will have poor performance, which explains why full-scan queries are disallowed in production environments. Two aspects of querying a graph can be improved with indexing: the initial vertex or vertices from which to start a traversal, and the narrowing of the edges and vertices to traverse from this starting point. DSE Graph implements two types of indexes, global indexes and vertex-centric indexes (VCIs) to address these different aspects of query processing. Global indexes are used to find the starting point for a query and involve finding a matching value for a value of a vertex property. Vertex-centric indexes are used to narrow down the scope of a query after a starting point is defined.

### Global indexing overview

Global indexes identify the starting point for a graph traversal query using a [vertex label \(page 450\)](#) and a [property \(page 443\)](#). It is important to understand that graph queries must start from a vertex, not an edge. Although a vertex-centered index using an edge property can be used to narrow a traversal, a traversal cannot start from an edge. In a distributed graph database like DSE Graph, the most efficient traversal would start with a vertex identified by its [vertex id \(page 451\)](#), such as this query that uses the vertex id for Julia Child:

```
g.V(['~label':'person', 'personId':1])
```

However, identifying a vertex by vertex id is rather restrictive. Using a vertex label and a property in a traversal allows DSE Graph to identify the DSE node where the vertex data resides without reading all data from all DSE nodes. Most graph queries will first use a global index to find a starting vertex with a friendlier property:

```
g.V().has('person', 'name', 'Julia Child')
```

Since the property `name` is not part of the vertex id, an index is required to match the search conditions with the correct vertex, and that index is a global index.

Global indexing in DSE Graph can be accomplished with one of three DSE indexing methods: a [materialized view \(MV\)](#), a [search index \(page 304\)](#), or a [secondary index](#).

Materialized views are tables generated from a base table to provide a query based on a different primary key than the base table. This type of index is best used for values of high cardinality of nearly unique values, or high selectivity. Selectivity is derived from cardinality, using the following formula:

```
selectivity = (cardinality / number of rows) * 100%
```

In general, low cardinality results in low selectivity, and high cardinality results in high selectivity. Searching materialized views yields similar response times to searching base tables, although writing the data incurs a small time penalty. When data is written or updated in the graph, the index information is updated in the MV table along with the graph tables. A consequence of using a MV table is higher write latencies, but results in lower read latencies for graph traversals.

Search indexes are used when textual, numeric or geospatial indexing are required and rely on [DSE Search \(page 304\)](#). Since graph data is stored in DSE database tables, one search core is available per vertex label. For each vertex label that will be indexed with search, all properties must be added to a single search index named `search`. Because search is implemented with DSE Search, all data types can be indexed. For two indexing options, full text and string, the property key must be defined, as different indexing results. Full text indexing performs tokenization and secondary processing such as case normalization. Full text indexing is useful for queries where partial match of text is required, and lends itself to regular expressing (regEx) searching. String indexing is useful for queries where an exact string is sought and no tokenization is required, similar to [Solr facetting](#). This type of index is best for low selectivity, but lends itself to fuzzy matching for both tokenized and non-tokenized indexing.

Secondary indexing in DSE Graph follows the same rule of thumb as [DSE secondary indexing](#). This type of index is meant for lower cardinality values, or alternatively, for low selectivity values. The number of values for indexing should number in the tens to hundreds at most; for instance, searching by country is a good candidate for secondary indexing. In addition, only equality conditions can be used to match values, and no ordering or range queries on values can be used. If more complex value matching is required, search indexes are the superior choice.

To summarize global indexes:

- Composite index keys are not currently supported in DSE Graph.

## **Vertex-centric indexing (VCI) overview**

Vertex-centric indexes (VCI) are used to narrow the traversal based on an additional property. Global indexes can be applied across all vertices with a specified vertex label, as opposed to VCIs which apply to a filtered set of vertices. Vertex-centric indexes are

especially important in reducing the complexity of a traversal from  $O(n)$  to  $O(1)$  or  $O(\log n)$ , using Big O notation. Two types of VCI exist, edge indexes and property indexes. Edges indexes are useful for traversing edges based on associated properties, to avoid linear scans of all incident edges of a vertex, since traversing all incident [edges \(page 454\)](#) can quickly compound the cost of a traversal if many incident edges exist. For instance, an edge index is useful in picking just certain edges once a global index has initiated the traversal at a particular vertex (in this case, Julia Child):

```
g.V().has('person', 'name', 'Julia
Child').outE('created').has('createDate', gt(1960-01-01))
```

Property indexes are created to index [meta-properties \(page 444\)](#). Property indexes can support both equality and inequality predicates, and are useful in cases where a range of values must be returned by a query. This example will find all the countries that Fritz Streiff lived in and order them by the year he started living in the country:

```
g.V().has('person', 'name', 'Fritz
Streiff').properties('country').has('startYear', order().by(decr))
```

Vertex-centric indexing in DSE Graph is accomplished with [materialized views \(MVs\)](#) for both edge and property indexes, and have the same properties as described above for global indexes.

## Indexing best practices

More than one index can be created on the same property, such as creating both a materialized index and a search index on the property `amount`. The DSE Graph query optimizer will automatically use the appropriate index when processing a query; designation of an index type to use is not a feature. The order of preference that DSE Graph uses is MV index > secondary index > DSE Search index to ensure best performance. Different index types may be created on different properties as appropriate, based on the selectivity. In general, secondary indexes in DSE Graph are limited in usefulness, for the same reasons that constrict their general use in DSE. Materialized view indexing should be the first choice, unless textual search is required and a search index is selected.

If a search index is created, be aware that building the index can take time, and that until the index is available, [queries that depend on the index can fail](#). Applications that create schema, immediately followed by data insertion that require search indexes will likely experience errors. Also, queries that use search indexes should be run on DSE Search-enabled nodes in the cluster. Search indexes also require extra resources. Each index allocates a minimum of 256MB of memory by default, and each index will require two physical cores. For a typical 32GB node, 16 search indexes would be a reasonable number to create.

Queries that use textual predicates (`regex`, `tokenRegex`, `prefix`, `tokenPrefix`, `token`, and `eq/eq`) can be accomplished without DSE search indexes. However, such queries will not make use of secondary or materialized indexes and will instead use full graph scans to return results. By default, Production mode does not allow full graph scans, so such queries will fail. If such matching search methods are required, search indexes are strongly suggested.

**Caution:** `tokenRegex` will display case insensitivity in queries, whether a search index is used or not.

Textual search indexes are by default indexed in both tokenized (`TextField`) and non-tokenized (`StrField`) forms. This means that all textual predicates (`token`, `tokenPrefix`, `tokenRegex`, `eq`, `neq`, `regex`, `prefix`) will be usable with all textual vertex properties indexed. Practically, search indexes should be created using the `asString()` method only in cases where there is absolutely no use for tokenization and text analysis, such as for inventory categories (silverware, shoes, clothing). The `asText()` method is used if searching tokenized text, such as long multi-sentence descriptions. The query optimizer will choose whether to use analyzed or non-analyzed indexing based on the textual predicate used.

It is possible to modify [search index schema \(page 315\)](#) to change search characteristics. Although DSE Graph will not overwrite these out-of-band changes, it is recommended that you do not add or remove fields in this manner - only DSE Graph commands should be used. The general use of this feature is mainly to change the behavior of a search, such as adding case sensitivity to a type of search.

## Adding index schema

Adding index database schema.

All index schema is based on previously created properties and vertex labels and added to existing schema with `add()`.

### Prerequisites:

[Create property key schema \(page 443\)](#) and [vertex label schema \(page 450\)](#).

### Materialized index

1. Create a materialized view index for a global index:

```
schema.vertexLabel('person').index('byName').materialized().by('name').add()
```

Identify the vertex label and property key for the index, in the `vertexLabel()` and `by()` steps, respectively. In the `index()` step, name the index. The `materialized()` step identifies the index as a materialized view index.

2. Create a materialized view for an edge index. Edges indexes are vertex-centric to a particular vertex label. For instance, the example below indexes anything that a reviewer rates.

```
schema.vertexLabel('person').index('ratedByStars').outE('reviewed').
 by('stars').ifNotExists().add()
```

Identify the vertex label and property keys for the index, in the `vertexLabel()` and `by()` steps, respectively. In the `index()` step, name the index. The `outE()` step is used to define the direction of the edge.

**Note:** For edge and property indexes, the `materialized()` step is not included, because all edge indexes are created as materialized views.

3. Create an edge index that indexes both incoming and outgoing edges:

```
schema.vertexLabel('person').index('ratedByStars').bothE('reviewed').
 by('stars').ifNotExists().add()
```

Identify the vertex label and property keys for the index, in the `vertexLabel()` and `by()` steps, respectively. In the `index()` step, name the index. The `bothE()` step is used to define the direction of the edge.

#### 4. Create a materialized view for a property index:

```
schema.vertexLabel('person').index('byStartYear').
 property('country').by('startYear').add()
```

Identify the vertex label for the index in the `vertexLabel()` step. In the `index()` step, name the index. The `property()` and `by()` steps identify the property key and its meta-property, respectively.

**Note:** For edge and property indexes, the `materialized()` step is not included, because all edge indexes are created as materialized views.

### Search index

#### 5. Most commonly, a search index is created with multiple columns, chained together as multiple `by()` steps in one statement:

```
schema.vertexLabel('recipe').index('search').search().by('name').by('instructions').add()
```

Identify the vertex label and property keys for the index, in the `vertexLabel()` and `by()` steps, respectively. In the `index()` step, name the index `search`; only this naming convention can be used. The `search()` step identifies the index as a search index. Since no option is specified, the property will be indexed both as Text and String.

**Tip:** Only one search index can be created per vertex label.

Textual search indexes are by default indexed in both tokenized (`TextField`) and non-tokenized (`StrField`) forms. This means that all textual predicates (`token`, `tokenPrefix`, `tokenRegex`, `eq`, `neq`, `regex`, `prefix`) will be usable with all textual vertex properties indexed. Practically, search indexes should be created using the `asString()` method only in cases where there is absolutely no use for tokenization and text analysis, such as for inventory categories (silverware, shoes, clothing). The `asText()` method is used if searching tokenized text, such as long multi-sentence descriptions. The query optimizer will choose whether to use analyzed or non-analyzed indexing based on the textual predicate used.

#### 6. Create a search index with only one property key indexed as Text():

```
schema.vertexLabel('recipe').index('search').search().by('instructions').asText().add()
```

#### 7. A search index can specify the string indexing option along with the text indexing option:

```
schema.vertexLabel('recipe').index('search').search().by('instructions').asString().by()
```

This example specifically states how the search index will be constructed for each property.

## 8. Search indexes can also include non-text data types:

```
schema.vertexLabel('recipe').index('search').search().by('year').by('name').asString()
```

Data types other than text are inferred from the schema and DSE Search uses a comparable Solr data type. In this example, `year` is indexed as an integer.

**Caution:** The `Decimal` data type will index as a `SolrDecimalStrField`. Use `Int`, `Long`, `Float`, or `Double` to ensure that the Solr data types are used for sorting and range querying.

## 9. Create a search index for geospatial data:

```
schema.propertyKey("coordinates").Point().single().create()
schema.propertyKey("name").Text().single().create()

schema.vertexLabel("place").properties("coordinates",
 "name").create()
schema.vertexLabel("place").index("search").search().by("name").asText().by("coordinate")
```

In this example, the property `coordinates` is a point defining a longitude and latitude. The search index includes `coordinates` without a qualifying `asText()` or `asString()` method. See [Geospatial Schema \(page 445\)](#) for additional information.

## 10. While DSE Graph natively supports geospatial searches, performing them without a Search index does not scale as the number of vertices in the graph increases. Doing such queries without a search index results in very inefficient query performance because full scans are required. DSE Search indexes can index points and linestrings, but not polygons.

```
//SEARCH INDEX ONLY WORKS FOR POINT AND LINESTRING
schema.vertexLabel('location').index('search').search().by('point').add()
schema.vertexLabel('lineLocation').index('search').search().by('line').add()
```

Without a search index, spatial queries always return exact results. DSE Search indexes, however, can trade off performance for accuracy.

**Note:** A point of confusion can occur if the same geospatial query is run with or without a DSE Search index. Without a search index, geospatial queries always return exact results. DSE Search indexes, however, trade off write performance and index size for query accuracy with [two tunable parameters](#), `maxDistErr` (default: 0.000009) and `distErrPct` (default: 0.025). Inconsistent results in these two cases are due to the distance calculation algorithm variation of the default values of these parameters. DSE Graph can pass values for these two parameters when creating the search index. Change `maxDistErr` in `withError(maxDistErr, distErrPct)` to 0.0 to force both index-backed and non-index-backed queries to yield the same value:

```
schema.vertexLabel('location').index('search').search().by('point').withError(0.00000)
```

## 11. Create a search index for timestamp data:

```
schema.propertyKey('review_ts').Timestamp().create()
schema.propertyKey('name').Text().create()
schema.vertexLabel('rating').properties('name',
 'review_ts').create()
schema.vertexLabel('rating').index('search').search().by('name','review_ts').add()
```

## Secondary index

### 12. Create a global index as a secondary index:

```
schema.vertexLabel('recipe').index('byRecipe').secondary().by('name').add()
```

Identify the vertex label and property key for the index, in the `vertexLabel()` and `by()` steps, respectively. In the `index()` step, name the index. The `secondary()` step identifies the index as a secondary index.

The edge labels used for the DSE QuickStart example used throughout the documentation:

```
// *****
// VERTEX INDEX
// *****
// SYNTAX:
// index('index_name').
// [secondary() | materialized() | search()].
// by('propertykey_name').
// [asText() | asString()].
// add()
// *****

schema.vertexLabel('person').index('byName').materialized().by('name').add()
schema.vertexLabel('meal_item').index('byName').materialized().by('name').add()
schema.vertexLabel('ingredient').index('byName').materialized().by('name').add()
//
schema.vertexLabel('recipe').index('byCuisine').materialized().by('cuisine').add()
//
schema.vertexLabel('book').index('byName').materialized().by('name').add()

schema.vertexLabel('meal').index('byType').secondary().by('type').add()

schema.vertexLabel('recipe').index('search').search().
 by('instructions').by('name').by('cuisine').add()
schema.vertexLabel('book').index('search').search().
 by('name').by('publishYear').add()
schema.vertexLabel('location').index('search').search().
 by('geoPoint').withError(0.000009,0.0).add()
schema.vertexLabel('store').index('search').search().by('name').add()
```

```

schema.vertexLabel('home').index('search').search().by('name').add()
schema.vertexLabel('fridgeSensor').index('search').search().
 by('cityId').by('sensorId').by('name').add()

// *****
// EDGE INDEX
// *****
// SYNTAX:
// index('index_name').
// [outE('edgeLabel') | inE('edgeLabel')].
// by('propertykey_name').
// add()
// *****

schema.vertexLabel('recipe').index('byStars').inE('reviewed').
 by('stars').ifNotExists().add()
schema.vertexLabel('person').index('ratedByStars').outE('reviewed').
 by('stars').ifNotExists().add()
schema.vertexLabel('person').index('ratedByDate').outE('reviewed').
 by('year').ifNotExists().add()
schema.vertexLabel('person').index('ratedByComments').outE('reviewed').
 by('comment').ifNotExists().add()
schema.vertexLabel('recipe').index('byPersonOrRecipe').bothE('created').
 by('createDate').ifNotExists().add()

// *****
// PROPERTY INDEX using meta-property 'livedIn'
// *****
// SYNTAX:
// index('index_name').
// property('propertykey_name').
// by('meta-propertykey_name').
// add()
// *****

schema.vertexLabel('person').index('byStartYear').
 property('country').by('startYear').add()
schema.vertexLabel('person').index('byEndYear').
 property('country').by('endYear').add()

```

## Dropping graph schema

Dropping (deleting) graph schema.

Schema can be dropped (deleted) using the `drop()` step; see [propertyKey \(page 642\)](#), [vertexLabel \(page 644\)](#), and [edgeLabel \(page 637\)](#) for more information.

### Drop schema

- To drop the schema and all data without dropping the graph, use a `drop()` step. Running `describe()` after will verify that the schema is dropped. After the schema is dropped, new schema and data can be loaded to the graph.

```
schema.drop()
```

```
=>null
```

- To drop schema for a particular property key, specify the property key and then use the `drop()` step:

```
schema.propertyKey('aBadProperty').drop()
```

**Warning:** To drop a property key for a vertex label that has a materialized view index, additional steps are required to prevent data loss or cluster errors.

[Drop any materialized indexes \(page 465\)](#) for the vertex label, drop the property key, and then rebuild the schema for the materialized view indexes. The MV index data will be recreated from the base table, with the exception of the data for the dropped property key.

- To drop schema for a particular vertex label, specify the vertex label and then use the `drop()` step:

```
schema.vertexLabel('aBadVertexLabel').drop()
```

## Drop index

- To drop an index from the schema, such as the `byMeal` index, identify the index by name. Use `describe()` to examine all indexes for the desired vertex label first.

```
gremlin> schema.vertexLabel('meal').describe()
```

```
==>schema.vertexLabel('meal').properties("name").create()
schema.vertexLabel('meal').index('byMeal').materialized().by('name').add()
```

- Using the vertex label and index name, remove the index. Running `describe()` again will verify that the index is removed.

```
gremlin> schema.vertexLabel('meal').index('byMeal').drop()
```

```
=>null
```

- Remove only one property from a search index:

```
schema.vertexLabel('author').index('search').search().properties('nick_name').drop()
schema.vertexLabel('author').describe()
```

## Managing graph data

Inserting, updating, examining, and dropping graph data.

## Data Formats

DSE Graph handles many different formats. Each format has advantages, and the choice of which to use is a matter of preference.

DSE Graph can ingest comma-delimited (CSV), JSON, or text data using the DSE Graph Loader. The DSE Graph Loader is a tool that can transform data upon ingestion if required, but complex transformations will make the data ingestion slow.

Gryo is a compact binary format, the fastest and most space-saving format for populating DSE Graph with data. This method can be used if the data has been generated from DSE Graph or another graph database.

GraphSON is a JSON style file format that passes both data and schema in human-readable format. It is easy to work with, as the structure is evident, but it results in large files. GraphML is an XML-based file format that similarly passes both data and schema. It is widely supported by graph-related tools and libraries making it a solid interchange format for DSE Graph. GraphSON and GraphML are common formats and useful for importing data into DSE Graph from another graph database.

## CSV

A CSV file is a common file format that can be input into DSE Graph. A sample of CSV graph data:

```
personId|name|nickname|gender|CALORIES|macroGoal
1|Julia Child|null|F||
2|Simone Beck|null|F||
```

## JSON

A JSON file is a common file format that can be input into DSE Graph. A sample of JSON graph data:

```
{ "personId":1, "country": { "value":"USA", "startYear":1930,
"endYear":1949 } }
```

## Gryo

Writing data out of the graph into a Gryo file can be accomplished with a simple `graph.io()` command:

```
graph.io(gryo()).writeGraph("/tmp/recipe.gryo")
```

## GraphSON

Writing data out of the graph into a GraphSON file can be accomplished with a simple `graph.io()` command:

```
graph.io(graphson()).writeGraph("/tmp/recipe.json")
```

A sample of the output of a GraphSON file:

```
{"id":{ "@type":"g:Map", "@value":["~label", "recipe", "recipeId",
{ "@type":"g:Int32", "@value":2003 }] }, "label": "recipe", "inE":{ "created":
[{ "id":{ "@type":"g:Map", "@value":["~label", "created", ""] } }] }
```

```

~out_vertex", {"@type": "g:Map", "@value": ["~label", "person", "personId",
{ "@type": "g:Int32", "@value": 1 }]}, "~in_vertex",
{ "@type": "g:Map", "@value": ["~label", "recipe", "recipeId", { "@type": "g:Int32", "@value": 2003 }]}, "~local_id",
{ "@type": "g:UUID", "@value": "f38f9dd1-2978-11e8-8043-6bfe97ac83b9" }]}, "outV":
{ "@type": "g:Map", "@value": ["~label", "person", "personId",
{ "@type": "g:Int32", "@value": 1 }], "properties": { "createDate": {} } }]}, "outE": { "includedIn":
[{ "id": { "@type": "g:Map", "@value": ["~label", "includedIn", "~out_vertex",
{ "@type": "g:Map", "@value": ["~label", "recipe", "rec
ipeId", { "@type": "g:Int32", "@value": 2003 }]}, "~in_vertex",
{ "@type": "g:Map", "@value": ["~label", "book", "bookId",
{ "@type": "g:Int32", "@value": 1001 }]}, "~local_id",
{ "@type": "g:UUID", "@value": "f3
90fd61-2978-11e8-8043-6bfe97ac83b9" }]}, "inV": { "@type": "g:Map", "@value": [
["~label", "book", "bookId", { "@type": "g:Int32", "@value": 1001 }] } }, { "id": {
{ "@type": "g:Map", "@value": ["~label", "includedI
n", "~out_vertex", { "@type": "g:Map", "@value": [
["~label", "recipe", "recipeId",
{ "@type": "g:Int32", "@value": 2003 }]}, "~in_vertex",
{ "@type": "g:Map", "@value": ["~label", "ingredient", "ingredId", { "@t
ype": "g:Int32", "@value": 3008 }]}, "~local_id",
{ "@type": "g:UUID", "@value": "f3903a15-2978-11e8-8043-6bfe97ac83b9" }]}, "inV": {
{ "@type": "g:Map", "@value": ["~label", "ingredient", "ingredId", { "@type
": "g:Int32", "@value": 3008 }] }, "properties": { "amount": "2-3 Tbsp" } },
{ "id": { "@type": "g:Map", "@value": ["~label", "includedIn", "~out_vertex",
{ "@type": "g:Map", "@value": ["~label", "recipe", "recipe
Id", { "@type": "g:Int32", "@value": 2003 }]}, "~in_vertex",
{ "@type": "g:Map", "@value": ["~label", "ingredient", "ingredId",
{ "@type": "g:Int32", "@value": 3010 }]}, "~local_id",
{ "@type": "g:UUID", "@value": "f3903a16-2978-11e8-8043-6bfe97ac83b9" }]}, "inV": {
{ "@type": "g:Map", "@value": ["~label", "ingredient", "ingredId",
{ "@type": "g:Int32", "@value": 3010 }] }, "properties": { "amount": "1 1/2 lbs
blanch
ed, trimmed" } },

```

## GraphML

Writing data out of the graph into a GraphML file can be accomplished with a simple `graph.io()` command:

```
graph.io(graphml()).writeGraph("/tmp/recipe.gml")
```

A sample of the output of a GraphSON file:

```

<?xml version='1.0' encoding='UTF-8'?><graphml xmlns="http://
graphml.graphdrawing.org/xmlns" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:schemaLocation="http://graphml.grap
hdrawing.org/xmlns http://graphml.graphdrawing.org/xmlns/1.1/
graphml.xsd"><key id="instructions" for="node" attr.name="instructions"
attr.type="string"/><key id="notes" for="node" attr.n

```

```
ame="notes" attr.type="string"/><key id="mealId" for="node"
attr.name="mealId" attr.type="int"/><key id="gender" for="node"
attr.name="gender" attr.type="string"/><key id="ingredId" for=
"node" attr.name="ingredId" attr.type="int"/><key id="type" for="node"
attr.name="type" attr.type="string"/><key id="recipeId" for="node"
attr.name="recipeId" attr.type="int"/><key id="b
ookId" for="node" attr.name="bookId" attr.type="int"/><key
id="labelV" for="node" attr.name="labelV" attr.type="string"/><key
id="publishYear" for="node" attr.name="publishYear" attr.typ
e="int"/><key id="ISBN" for="node" attr.name="ISBN" attr.type="string"/
><key id="name" for="node" attr.name="name" attr.type="string"/><key
id="personId" for="node" attr.name="personId"
attr.type="int"/><key id="labelE" for="edge" attr.name="labelE"
attr.type="string"/><key id="amount" for="edge" attr.name="amount"
attr.type="string"/><key id="createDate" for="edge" att
r.name="createDate" attr.type="string"/><graph id="G"
edgedefault="directed"><node id="{'label=recipe, recipeId=2003}'><data
key="labelV">recipe</data><data key="instructions">Take a sal
ad bowl or platter and line it with lettuce leaves, shortly before
serving. Drizzle some olive oil on the leaves and dust them with
salt.</data><data key="notes"></data><data key="name">
Salade Nicoise</data><data key="recipeId">2003</data></node><node
id="{'label=recipe, recipeId=2005}'><data key="labelV">recipe</
data><data key="instructions">Preheat the oven to 375 deg
rees F. Cook bacon in a large skillet over medium heat until very
crisp and fat has rendered, 8-10 minutes.</data><data key="notes"> </
data><data key="name">Spicy Meatloaf</data><data ke
y="recipeId">2005</data></node><node id="{'label=recipe,
recipeId=2006}'><data key="labelV">recipe</data><data
key="instructions">Saute the shallots, celery, herbs, and seasonings
in 3 t
ablespoons of the butter for 3 minutes. Add the watercress and let
it wilt.</data><data key="notes"> </data><data key="name">Oysters
Rockefeller</data><data key="recipeId">2006</data></n
ode><node id="{'label=recipe, recipeId=2007}'><data
key="labelV">recipe</data><data key="instructions">In a heavy-bottomed
pot, melt the butter. When it starts to foam, add the onions an
d thyme and cook over medium-low heat until tender, about 10 minutes.</
data><data key="notes">Quick and easy.</data><data key="name">Carrot
Soup</data><data key="recipeId">2007</data></n
ode><node id="{'label=recipe, recipeId=2001}'><data
key="labelV">recipe</data><data key="instructions">Braise the beef.
Saute the onions and carrots. Add wine and cook in a dutch oven at
425 degrees for 1 hour.</data><data key="notes">Takes a long
time to make.</data><data key="name">Beef Bourguignon</data><data
key="recipeId">2001</data></node><node id="{'label=recipe,
recipeId=2002}'><data key="labelV">recipe</data><data
key="instructions">Peel and cut the eggplant. Make sure you cut
eggplant into lengthwise slices that are about 1-inch wide, 3-inch
```

```
es long, and 3/8-inch thick</data><data key="notes">I've made this 13
times.</data>
```

## Inserting data with traversal API

Inserting data with traversal API.

The [Traversal API \(page 651\)](#) can be used to insert data into DSE Graph.

1. Add a vertex:

```
g.addV('person').property('personId', 1).property('name', 'Julia
Child').property('gender', 'F')
```

The `addV()` step must identify the vertex label, and can be followed by key and value pairs in `property()` steps for any properties that are added.

2. To add an edge using only the graph API, the two vertices that are connected by the edge can be assigned variables that can be used in the `add()` statement:

```
juliaChild = g.V().has('person', 'personId', 1).next()
artOfFrenchCookingVolOne = g.V().has('book', 'bookId', 1001).next()
g.V(juliaChild).addE('authored').to(artOfFrenchCookingVolOne)
```

or alternatively, the query can be constructed without variables:

```
g.V().has('person', 'name', 'Julia Child')
 .addE('authored')
 .to(g.V().has('book', 'name', 'The Art of French Cooking, Vol.
1'))
```

The outgoing vertex, `juliaChild`, is connected to the incoming vertex, `artOfFrenchCookingVolOne` with a `to()` step, to create and authored edge with `addE()`,

If the edge has properties, the key and value pairs are appended to the `addE()` statement, similar to the `addV()` statement with `property()`:

```
g.V().has('person', 'name', 'Julia Child')
 .addE('created')
 .to(g.V().has('recipe', 'name', 'Beef Bourguignon'))
 .property('createDate', '1956-01-10')
```

3. A property can also be added to a previously created vertex, like `jamieOliver`:

```
jamieOliver = g.V().has('person', 'name', 'Jamie Oliver').next()
jamieOliver.property('gender', 'M').property('nickname', 'jimmy')
```

A `property()` value can also be modified by simply doing a similar traversal with a different value.

4. A property can also be added to a previously created edge by constructing a query that finds the edge and changes or adds the property with `property()`:

```
g.V().has('person', 'name', 'Julia Child')
```

```
.outE('created')
.has('createDate', '1956-01-10')
.property('createDate', '1956-09-09')
```

## Inserting data with graph API

Inserting data with graph API.

The [Graph API \(page 627\)](#) can be used to insert data into DSE Graph.

1. Add a vertex:

```
graph.addVertex(label, 'person', 'personId', 1, 'name', 'Julia
Child', 'gender', 'F')
```

The literal `label` followed by the vertex label are the first two items in the statement, followed by key and value pairs for any properties that are added.

2. To add an edge using only the graph API, the two vertices that are connected by the edge must be assigned variables that can be used in the `addEdge` statement:

```
juliaChild = graph.addVertex(label, 'person', 'personId', 1,
'name', 'Julia Child', 'gender', 'F')
artOfFrenchCookingVolOne = graph.addVertex(label, 'book', 'bookId',
1001, 'name', 'The Art of French Cooking, Vol. 1', 'publishYear',
1961)
juliaChild.addEdge('authored', artOfFrenchCookingVolOne)
```

The outgoing vertex, `juliaChild`, is connected to the incoming vertex, `artOfFrenchCookingVolOne`, to create and `authored` edge.

If the edge has properties, the key and value pairs are appended in the `addEdge` statement, similar to the `addVertex` statement:

```
beefBourguignon.addEdge('includedIn', beef, 'amount', '2 lbs')
```

3. A property can also be added to a previously created vertex, like `jamieOliver`:

```
jamieOliver.property('gender', 'M', 'nickname', 'jammy')
```

4. To add geospatial data:

```
graph.addVertex(label, 'location', 'name', 'Paris', 'point',
Geo.point(2.352222, 48.856614))
graph.addVertex(label, 'lineLocation', 'name', 'ParisLondon',
'line', "LINESTRING(2.352222 48.856614, -0.127758 51.507351)")
graph.addVertex(label, 'polyLocation', 'name',
'ParisLondonDublin', 'polygon', Geo.polygon(2.352222, 48.856614,
-0.127758, 51.507351, -6.26031, 53.349805))
```

A vertex label is created for `location` that has a `point` property. A vertex label is created for `lineLocation` that has a `LineString` property. A vertex label is created for `polyLocation` that has a `Polygon` property.

**Note:** For all geospatial elements, the `withGeoBounds()` method limits searches to a default valid range of latitude in degrees from -90 to +90 (South Pole to North Pole) and a valid range of longitude in degrees from -180 to +180 (east to west from the [Greenwich Meridian](#)). The point is specified using `Geo.point(longitude, latitude)` when adding points, using [WellKnownText \(WKT\)](#) format. Note that it specifies **longitude** first, then **latitude**.

## Using the DSE Graph Loader

Load schema and data using the DSE Graph Loader.

How to load schema and data using the DSE Graph Loader.

### DSE Graph Loader overview

DSE Graph Loader is a customizable, highly tunable command line utility for loading graph datasets into DSE Graph from various input sources.

DSE Graph Loader is a customizable, highly tunable command line utility for loading graph datasets into DSE Graph from various input sources. It is not included as part of DataStax Enterprise installations and must be [installed separately \(page 472\)](#).

DSE Graph Loader is built to load datasets containing hundreds of millions ( $10^8$ ) of vertices and billions ( $10^9$ ) of edges. DSE Graph Loader is efficient, using parallel loading and persistent cache to store vertices, provided [a sufficient machine \(page 472\)](#) is used to run the program.

Data can be loaded from CSV files, JSON files, delimited text (CSV with a header line to identify the fields), text parsed by regular expressions, and binary Gryo files. Distributed filesystem support exists to read input files from Hadoop Distributed File Systems (HDFS) and AWS S3 sources. In addition, DSE Graph Loader supports reading input data directly from a JDBC compatible database or a Neo4J database. Input files can be uncompressed or compressed files. All data can be [transformed \(page 527\)](#) upon reading to manipulate the data that is loaded into a graph.

Data from an input source file can be mapped to define vertices or edges, along with properties for both. The mapping script configures loading parameters, defines the input parameters, and identifies the mapping from each input record to graph element. Both vertex and edge properties can be included in the data that is loaded.

DSE Graph Loader processes input data with three stages:

#### Preparation

Reads entire input data to check for graph schema conformity. Suggests graph schema updates, or if enabled, changes graph schema. Supplies statistics about how much data will be added to graph when loaded. The `dryrun` configuration option ([page 472](#)) can be used to stop the loading process at this stage.

#### Vertex loading

Adds or retrieves all of the vertices in the input data and caches them locally to speed up subsequent edge loading.

Vertex validation is enabled unless the data is identified as new data with `load_new`. If data is new, validation is not executed, and performance improvement will be seen.

### Edge and property loading

Adds all edges and properties from the input data to the graph.

Edge validation is enabled unless the data is identified as new data with `load_new`. If data is new, validation is not executed, and performance improvement will be seen. Another method of handling mixed new and existing data is the use of [isNew\(\) \(page 548\)](#) and [exists\(\) \(page 546\)](#).

If duplicate edges are required, `isNew()` must be used to designate those edges as additive to the edges that already exist.

**Note:** Multiple cardinality input data must have graph schema created prior to data loading.

A critical feature to keep in mind when using DSE Graph Loader is the `upsert` nature of the underlying DSE database. If a vertex already exists, DSE Graph Loader updates the stored data with the new property values depending on the configuration choices made. [Configuration \(page 472\)](#) can be used to identify if the data loaded is new or will overwrite data that currently exists. Edges will be duplicated if the same edge is loaded multiple times and the edge label is set to the default of multiple cardinality.

It is **strongly** recommended that [graph schema is created \(page 436\)](#) before loading data using DSE Graph Loader. Without schema, the correct data types for the data are not enforced. [Creating indexes \(page 460\)](#) will greatly speed up the loading process, and are necessary to achieve acceptable performance for loading.

## Installing DSE Graph Loader

Install DSE Graph Loader using a binary tarball on any Linux-based platform.

DSE Graph Loader is not included as part of DataStax Enterprise installations. Follow the instructions for installing [DSE Graph Loader](#) in the DataStax Installation Guide.

## Configuring DSE Graph Loader

Configuring DSE Graph Loader files.

Before loading data using any of the methods detailed in the next topics, decide which configuration items to include in the mapping script.

The configuration settings can be applied in the command line using a `-` command, like `-read_threads`, or the settings can be included in the mapping script. All configuration settings are shown in the [DSE Graph Loader reference \(page 535\)](#) including [security options \(page 542\)](#).

- The `dryrun` setting will run the DSE Graph Loader with a mapping script, and output the results, but will not execute the loading process. It is useful for spotting potential errors in the mapping script or graphloader command.

```
config dryrun: true
```

This command may be more useful to use as a command line option, since it is not common to leave in after checking a mapping script:

```
graphloader map.groovy -graph food -address localhost -dryrun true
```

**Notice:** This configuration option discovers schema and suggests missing schema without executing any changes. In DSE 6.0, this option is deprecated and may possibly be removed in a future release.

- The `preparation` setting is a validity checking mechanism. If `preparation` is true, then a sample of the data is analyzed for whether or not the schema is valid. This setting is used in conjunction with `create_schema`. If `create_schema` and `preparation` are both true, then the data is analyzed, compared to the schema, and new schema is created if found missing.

```
/* CONFIGURATION */
/* Configures the data loader to analyze the schema */
config preparation: true
```

See the [table below \(page 473\)](#) for all permutations.

**Notice:** This configuration option validates and creates schema if used in conjunction with `create_schema`. The default will be set to `false`, and this option is deprecated with DSE 6.0. In a future release, it may be removed.

- This example sets `create_schema` to true, so that schema is created from the data. Setting `create_schema` to true is a good method of inputting new data, to get feedback on what schema may be required for the data. It is not recommended for Production data loading.

```
/* CONFIGURATION */
/* Configures the data loader to create the schema */
config create_schema: true
```

**Notice:** It is strongly recommended that [schema is created \(page 436\)](#) prior to data loading, so that the correct data types are enforced and indexes created. Setting `create_schema` to true is recommended only for testing. In DSE 6.0, this configuration option is deprecated and will be removed in a future release.

`preparation` and `create_schema` must be considered together

- The `load_new` setting is used if vertex records do not yet exist in the graph at the beginning of the loading process, such as for a new graph. Configuring `load_new` can significantly speed up the loading process. However, it is important that the user guarantee that the vertex records are indeed new, or duplicate vertices can be created in the graph. Edges that are created in the same script will use the newly created vertices for the outgoing vertex `outV` and incoming vertex `inV`.

```
config load_new: true
```

**Warning:** Duplicate vertices will be created if `load_new` is set to `false` and the data being loaded contain any vertex that already exists in the graph.

- Setting the number of threads used for loading vertices or edges uses `load_vertex_threads` and `load_edge_threads`, respectively; the default is 0, which will set `load_vertex_threads` to the number of cores divided by 2, and `load_edge_threads` to the number of nodes in the datacenter multiplied by six .

```
config load_vertex_threads: 3 load_edge_threads: 0
```

- Multiple configuration settings can be listed together.

```
config load_new: true, dryrun: true, schema_output: '/tmp/
loader_output.txt'
```

**What's next:** [Load data. \(page 474\)](#)

## Loading data

How to load data using DSE Graph Loader.

DSE Graph Loader can load data from many different input data formats. Pick the option that most resembles your data source:

Type	Description	Instructions
CSV	Strict format, with the first line of the file identifying the property keys used in the graph.	<a href="#">Loading CSV data (page 475)</a>
Text	Delimited text data of any format.	<a href="#">Loading TEXT data (page 490)</a>
Text with regular expressions	Delimited text data parsed using regular expressions (regex).	<a href="#">Loading TEXT data using regular expressions (regex) (page 494)</a>
JSON	Data stored in JSON (JavaScript Object Notation) format.	<a href="#">Loading JSON data (page 483)</a>
JDBC-compatible database	Data stored in a JDBC-compatible database	<a href="#">Loading data from a JDBC compatible database. (page 496)</a>
HDFS file	Data file stored in a Hadoop Distributed File System (HDFS) of any format.	<a href="#">Loading data from Hadoop (HDFS) (page 499)</a>
AWS S3 file	Data file stored in AWS S3 storage of any format.	<a href="#">Loading data from AWS S3 (page 501)</a>

Type	Description	Instructions
Gryo	Data stored in a binary Gryo format.	<a href="#">Loading Gryo data (page 504)</a>
GraphSON	Data stored in GraphSON format.	<a href="#">Loading GraphSON data (page 506)</a>
GraphML	Data stored in GraphML format.	<a href="#">Loading GraphML data (page 507)</a>

**Note:** Fields that contain `NULL`, `null`, or empty fields in text and CSV files will be pruned by DSE Graph Loader. A transform must be used if a different behavior is desired.

**Warning:** When loading [user-defined vertex ids \(page 451\)](#), the vertex cache that DSE Graph Loaders uses will be bypassed to facilitate faster write throughput. The client must ensure vertices are unique because no logic will validate the existence of a vertex with custom ids. To ensure the fastest performance, the DSE Graph configuration option [external\\_vertex\\_verify \(page 632\)](#) should be set to false.

The DSE Graph Loader also supports loading several files of the same format from a single directory. Example mapping scripts are shown for [CSV \(page 478\)](#) and [JSON \(page 485\)](#), but will work for all formats.

## Loading CSV data

How to use the DSE Graph Loader to load CSV data.

A common file format for loading graph data is CSV (comma-delimited data). An input CSV file generally identifies the property keys in the first line of the file with a header line. However, the mapping script can also identify the property keys to be read with `header()` in the data input line. If more flexibility is desired, such as manipulation of the vertex labels using [labelField \(page 512\)](#), use [Loading TEXT data \(page 490\)](#).

## Mapping several different CSV files

Mapping several different CSV files with DSE Graph Loader.

DSE Graph Loader can load several different CSV files that exist in a directory using the following steps. Sample input data:

```
SAMPLE INPUT
// For the author.csv file:
name|gender
Julia Child|F
// For the book.csv file:
name|year|ISBN
Simca's Cuisine: 100 Classic French Recipes for Every Occasion|1972|
0-394-40152-2
// For the authorBook.csv file:
bname|aname
```

Simca's Cuisine: 100 Classic French Recipes for Every Occasion |  
Simone Beck

Because the property key `name` is used for both vertex labels `author` and `book`, in the `authorBook` file, variables `aname` and `bname` are used for author name and book name, respectively. These variables are used in the mapping logic used to create the edges between `author` and `book` vertices.

1. If desired, [add configuration \(page 472\)](#) to the mapping script.
2. Specify the data input files. The variable `inputfiledir` specifies the directory for the input files. Each of the identified files will be used for loading.

```
// DATA INPUT
// Define the data input source (a file which can be specified
// via command line arguments)
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/CSV/'
authorInput = File.csv(inputfiledir +
 'author.csv').delimiter('|')
bookInput = File.csv(inputfiledir + 'book.csv').delimiter(' ')
authorBookInput = File.csv(inputfiledir +
 'authorBook.csv').delimiter('|')
```

It is important to note that CSV files can have a header line that shows the field names. For example, the `authorInput` will have the following as the first line in the file:

name|gender

If a `header()` is used in the mapping script and a header line is used in the data file, then both must match. Either a header line in the data file or a `header()` is required.

3. In each line, the file is specified as a `csv` file, the file name is specified, and a delimiter is set. A map, `authorInput`, is created that will be used to process the data. The map can be manipulated before loading using [transforms \(page 527\)](#).

```
authorInput = File.csv(inputfiledir +
 'author.csv').delimiter('|')
```

**Tip:** If you need to trim excess whitespace from data, use `trimWhitespace(true)` in the `File.csv()` statement.

4. [Create the main body of the mapping script. \(page 509\)](#) This part of the mapping script is the same regardless of the file format.
5. To run DSE Graph Loader for CSV loading as a dry run, use the following command:

```
graphloader authorBookMappingCSV.groovy -graph testCSV -address
localhost -dryrun true
```

For testing purposes, the graph specified does not have to exist prior to running `graphloader`. However, for production applications, the graph and schema must be created prior to using `graphloader`.

The fullscript is shown:

```

/* SAMPLE INPUT
author: Julia Child|F
book: Simca's Cuisine: 100 Classic French Recipes for Every
 Occasion|1972|0-394-40152-2
authorBook: Simca's Cuisine: 100 Classic French Recipes for
 Every Occasion|Simone Beck
 */

// CONFIGURATION
// Configures the data loader to create the schema
config create_schema: true, load_new: true,
 load_vertex_threads: 3

// DATA INPUT
// Define the data input source (a file which can be specified
// via command line arguments)
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/CSV/'
authorInput = File.csv(inputfiledir +
 "author.csv").delimiter('|')
bookInput = File.csv(inputfiledir + "book.csv").delimiter('|')
authorBookInput = File.csv(inputfiledir +
 "authorBook.csv").delimiter('|')

//Specifies what data source to load using which mapper (as
// defined inline)

load(authorInput).asVertices {
 label "author"
 key "name"
}

load(bookInput).asVertices {
 label "book"
 key "name"
}

load(authorBookInput).asEdges {
 label "authored"
 outV "aname", {
 label "author"
 key "name"
 }
 inV "bname", {
 label "book"
 }
}

```

```
 key "name"
 }
}
```

## Mapping several files with same format from a directory

Mapping several same format CSV files with DSE Graph Loader.

DSE Graph Loader can load several CSV files with same format that exist in a directory using the following steps. Sample input data:

```
SAMPLE INPUT
// For the author.csv file:
name|gender
Julia Child|F
Simone Beck|F

// For the knows.csv file:
aname|bname
Julia Child|James Beard
```

A number of files with the same format exist in a directory. If the files differ, the graphloader will issue an error and stop:

```
java.lang.IllegalArgumentException: /tmp/dirSource/data has more
than 1 input type.
```

1. If desired, [add configuration \(page 472\)](#) to the mapping script.
2. Specify the data input directory. The variable `inputfiledir` specifies the directory for the input files. Each of the identified files will be used for loading.

```
// DATA INPUT
// Define the data input source (a file which can be specified
// via command line arguments)
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/dirSource/data'
personInput =
 File.directory(inputfiledir).delimiter('|').header('name', 'gender')

//Specifies what data source to load using which mapper (as
defined inline)

load(personInput).asVertices {
 label "author"
 key "name"
}
```

The important element is `File.directory()`; this defines the directory where the files are stored.

It is important to note that CSV files must have a header line that shows the field names. For example, the `authorInput` will have the following as the first line in the file:

```
name|gender
```

3. Note that two directories could be used to load vertices and edges:

```
// DATA INPUT
// Define the data input source (a file which can be specified
// via command line arguments)
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/dirSource/data'
vertexfiledir = inputfiledir+'/vertices'
edgefiledir = inputfiledir+'/edges'
personInput =
 File.directory(vertexfiledir).delimiter('|').header('name','gender')
personEdgeInput =
 File.directory(edgefiledir).delimiter('|').header('aname','bname')

//Specifies what data source to load using which mapper (as
defined inline)

load(personInput).asVertices {
 label "author"
 key "name"
}

load(personEdgeInput).asEdges {
 label "knows"
 outV "aname", {
 label "author"
 key "name"
 }
 inV "bname", {
 label "book"
 key "name"
 }
}
```

4. To run DSE Graph Loader for CSV loading from a directory, use the following command:

```
graphloader dirSourceMapping.groovy -graph testdirSource -
address localhost
```

For testing purposes, the graph specified does not have to exist prior to running `graphloader`. However, for production applications, the graph and schema should be created prior to using `graphloader`.

## Mapping files from a directory using a file pattern

Mapping several same format CSV files with DSE Graph Loader.

DSE Graph Loader can load several files from a directory using file pattern matching  
Sample input files:

```
ls data

badOne.csv person1.csv person2.csv
```

A number of files with the same format exist in a directory. If the files differ, DSE Graph Loader will only load the files that match the pattern in the map script.

Several file patterns are defined for use:

### Mapping using \*

- If desired, [add configuration \(page 472\)](#) to the mapping script.
- Sample input file:

```
/* SAMPLE CSV INPUT:
id|name|gender
001|Julia Child|F
*/
```

- Specify the data input directory. The variable `inputfiledir` specifies the directory for the input files. Each of the identified files will be used for loading.

```
// DATA INPUT
// Define the data input source (a file which can be specified
// via command line arguments)
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/filePattern'
inputfileCSV = inputfiledir+'/data'
personInput =
 File.directory(inputfileCSV).fileMatches("person*.csv").delimiter('|').header('id','na

//Specifies what data source to load using which mapper (as
//defined inline)

load(personInput).asVertices {
 label "person"
 key "name"
}

/* RESULT:
 person1.csv and person2.csv will be loaded, but not
 badOne.csv
```

```
* /
```

The important element is `fileMatches("person*.csv")`; this defines the pattern that will be matched for loaded files. The file `badOne.csv` will not be loaded, because the pattern does not match. Note that a file `personExtra.csv` would also be loaded, as it would match the pattern.

This same pattern matching can be used for JSON input files, by substituting `person*.json` for `person*.csv` and using JSON input file parameters.

- To run DSE Graph Loader for CSV loading from a directory, use the following command:

```
graphloader filePatternCSV.groovy -graph testPattCSV -address
localhost
```

For testing purposes, the graph specified does not have to exist prior to running `graphloader`. However, for production applications, the graph and schema should be created prior to using `graphloader`.

### Mapping using [ ]

- If desired, [add configuration \(page 472\)](#) to the mapping script.
- Sample input file:

```
/* SAMPLE CSV INPUT:
id|name|gender
001|Julia Child|F
*/
```

- Specify the data input directory. The variable `inputfiledir` specifies the directory for the input files. Each of the identified files will be used for loading.

```
// DATA INPUT
// Define the data input source (a file which can be specified
// via command line arguments)
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/filePattern'
inputfileCSV = inputfiledir+'/data'
personInput =
 File.directory(inputfileCSV).fileMatches("person[1-9].csv").delimiter(' | ').header('id')

//Specifies what data source to load using which mapper (as
//defined inline)

load(personInput).asVertices {
 label "person"
 key "name"
}

/* RESULT:
 person1.csv and person2.csv will be loaded, but not
 badOne.csv
```

```
* /
```

The important element is `fileMatches("person[1-9].csv")`; this defines the pattern that will be matched for loaded files. All files `person1.csv` through `person9.csv` will be loaded, but `person15.csv` doesn't match the pattern and will not be loaded, as well as `badOne.csv`. Note that `fileMatches("person?.csv")` would achieve the same result.

This same pattern matching can be used for JSON input files, by substituting `person[1-9].json` for `person[1-9].csv` and using JSON input file parameters.

- Run DSE Graph Loader for this example use the following command:

```
graphloader filePatternRANGE.groovy -graph testPattRANGE -
address localhost
```

For testing purposes, the graph specified does not have to exist prior to running `graphloader`. However, for production applications, the graph and schema should be created prior to using `graphloader`.

### Mapping using {} with multiple patterns

- If desired, [add configuration \(page 472\)](#) to the mapping script.
- Sample input file:

```
/* SAMPLE CSV INPUT:
id|name|gender
001|Julia Child|F
*/
```

- Specify the data input directory. The variable `inputfiledir` specifies the directory for the input files. Each of the identified files will be used for loading.

```
// DATA INPUT
// Define the data input source (a file which can be specified
// via command line arguments)
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/filePattern/data'
personInput =
 File.directory(inputfiledir).fileMatches("{person*,badOne}.csv").delimiter('|').header(true)

//Specifies what data source to load using which mapper (as
//defined inline)

load(personInput).asVertices {
 label "person"
 key "name"
}

/* RESULT:
 person1.csv, person2.csv and badOne.csv will all be loaded
*/
```

The important element is `fileMatches( "{person*,badOne}.csv" )`; this defines the pattern that will be matched for loaded files. The files `person1.csv`, `person1.csv`, and `badOne.csv` will be loaded, because the pattern matches all three files. This same pattern matching can be used for JSON input files, by substituting `person*.json` for `person*.csv` and using JSON input file parameters.

- To run DSE Graph Loader for this example using the following command:

```
graphloader filePatternMULT.groovy -graph testPattMULT -address
localhost
```

For testing purposes, the graph specified does not have to exist prior to running `graphloader`. However, for production applications, the graph and schema should be created prior to using `graphloader`.

## Loading JSON data

How to use the DSE Graph Loader to load JSON data.

A common file format for loading graph data is JSON. An input JSON file holds all key and value information in a nested structure.

## Mapping several different JSON files

Mapping several different JSON files with DSE Graph Loader.

DSE Graph Loader can load several different CSV files that exist in a directory using the following steps. Sample input data:

```
SAMPLE INPUT
// For the author.json file:
{ "author_name": "Julia Child", "gender": "F" }
// For the book.json file:
{ "name": "The Art of French Cooking, Vol.
1", "year": "1961", "ISBN": "none" }
// For the authorBook.json file:
{ "name": "The Art of French Cooking, Vol. 1", "author": "Julia Child" }
```

Because the property key `name` is used for both vertex labels `author` and `book`, in the `authorBook` file, variables `aname` and `bname` are used for author name and book name, respectively. These variables are used in the mapping logic used to create the edges between `author` and `book` vertices.

1. If desired, [add configuration \(page 472\)](#) to the mapping script.
2. Specify the data input files. The variable `inputfiledir` specifies the directory for the input files. Each of the identified files will be used for loading.

```
// DATA INPUT
// Define the data input source (a file which can be specified
// via command line arguments)
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/JSON/'
```

```
authorInput = File.json(inputfiledir + 'author.json')
bookInput = File.json(inputfiledir + 'book.json')
authorBookInput = File.json(inputfiledir + 'authorBook.json')
```

3. In each line, the file is specified as a `json` file and the file name is specified. The JSON format for `File.json` is one JSON object per line. A map, `authorInput`, is created that will be used to process the data. The map can be manipulated before loading using [transforms \(page 527\)](#).

```
authorInput = File.json(inputfiledir + 'author.json')
```

4. [Create the main body of the mapping script. \(page 509\)](#) This part of the mapping script is the same regardless of the file format.
5. To run DSE Graph Loader for JSON loading as a dry run, use the following command:

```
graphloader authorBookMappingJSON.groovy -graph testJSON -
address localhost -dryrun true
```

For testing purposes, the graph specified does not have to exist prior to running `graphloader`. However, for production applications, the graph and schema should be created prior to using `graphloader`.

The fullscript is shown:

```
/* SAMPLE INPUT
author: { "name": "Julia Child", "gender": "F" }
book : { "name": "The Art of French Cooking, Vol.
1", "year": "1961", "ISBN": "none" }
authorBook: { "bname": "The Art of French Cooking, Vol.
1", "aname": "Julia Child" }
*/
// CONFIGURATION
// Configures the data loader to create the schema
config create_schema: true, load_new: true,
load_vertex_threads: 3
// DATA INPUT
// Define the data input source (a file which can be specified
// via command line arguments)
// inputfiledir is the directory for the input files that is
// given in the commandline
// as the "-filename" option

inputfiledir = '/tmp/JSON/'
authorInput = File.json(inputfiledir + 'author.json')
bookInput = File.json(inputfiledir + 'book.json')
authorBookInput = File.json(inputfiledir + 'authorBook.json')

//Specifies what data source to load using which mapper (as
defined inline)
```

```

load(authorInput).asVertices {
 label "author"
 key "name"
}

load(bookInput).asVertices {
 label "book"
 key "name"
}

load(authorBookInput).asEdges {
 label "authored"
 outV "aname", {
 label "author"
 key "name"
 }
 inV "bname", {
 label "book"
 key "name"
 }
}

```

## Mapping several files with same format from a directory

Mapping several same format JSON files with DSE Graph Loader.

DSE Graph Loader can load several JSON files with same format that exist in a directory using the following steps. Sample input data:

```

SAMPLE INPUT
// For the author.json file:
{ "author_name": "Julia Child", "gender": "F" }
// For the book.json file:
{ "name": "The Art of French Cooking, Vol.
 1", "year": "1961", "ISBN": "none" }
// For the authorBook.json file:
{ "name": "The Art of French Cooking, Vol. 1", "author": "Julia Child" }

```

A number of files with the same format exist in a directory. If the files differ, the graphloader will issue an error and stop:

```
java.lang.IllegalArgumentException: /tmp/dirSource/data has more
than 1 input type.
```

1. If desired, [add configuration \(page 472\)](#) to the mapping script.
2. Specify the data input directory. The variable `inputfiledir` specifies the directory for the input files. Each of the identified files will be used for loading.

```

// DATA INPUT
// Define the data input source (a file which can be specified
// via command line arguments)
// inputfiledir is the directory for the input files

```

```
inputfiledir = '/tmp/dirSource/data'
personInput = File.directory(inputfiledir)

//Specifies what data source to load using which mapper (as
defined inline)

load(personInput).asVertices {
 label "author"
 key "name"
}
```

The important element is `File.directory()`; this defines the directory where the files are stored.

3. Note that two directories could be used to load vertices and edges:

```
// DATA INPUT
// Define the data input source (a file which can be specified
// via command line arguments)
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/dirSource/data'
vertexfiledir = inputfiledir+'/vertices'
edgefiledir = inputfiledir+'/edges'
personInput = File.directory(vertexfiledir)
personEdgeInput = File.directory(edgefiledir)

//Specifies what data source to load using which mapper (as
defined inline)

load(personInput).asVertices {
 label "author"
 key "name"
}

load(personEdgeInput).asEdges {
 label "knows"
 outV "aname", {
 label "author"
 key "name"
 }
 inV "bname", {
 label "book"
 key "name"
 }
}
```

4. To run DSE Graph Loader for JSON loading from a directory, use the following command:

```
graphloader dirSourceJSONMapping.groovy -graph testdirSource -
address localhost
```

For testing purposes, the graph specified does not have to exist prior to running `graphloader`. However, for production applications, the graph and schema should be created prior to using `graphloader`.

## Mapping files from a directory using a file pattern

Mapping several same format CSV files with DSE Graph Loader.

DSE Graph Loader can load several files from a directory using file pattern matching  
Sample input files:

```
ls data
badOne.csv person1.csv person2.csv
```

A number of files with the same format exist in a directory. If the files differ, DSE Graph Loader will only load the files that match the pattern in the map script.

Several file patterns are defined for use:

### Mapping using \*

- If desired, [add configuration \(page 472\)](#) to the mapping script.
- Sample input file:

```
/* SAMPLE CSV INPUT:
id|name|gender
001|Julia Child|F
*/
```

- Specify the data input directory. The variable `inputfiledir` specifies the directory for the input files. Each of the identified files will be used for loading.

```
// DATA INPUT
// Define the data input source (a file which can be specified
// via command line arguments)
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/filePattern'
inputfileCSV = inputfiledir+'/data'
personInput =
 File.directory(inputfileCSV).fileMatches("person*.csv").delimiter('|').header('id','na

//Specifies what data source to load using which mapper (as
defined inline)

load(personInput).asVertices {
 label "person"
 key "name"
}

/* RESULT:
 person1.csv and person2.csv will be loaded, but not
 badOne.csv
```

```
* /
```

The important element is `fileMatches("person*.csv")`; this defines the pattern that will be matched for loaded files. The file `badOne.csv` will not be loaded, because the pattern does not match. Note that a file `personExtra.csv` would also be loaded, as it would match the pattern.

This same pattern matching can be used for JSON input files, by substituting `person*.json` for `person*.csv` and using JSON input file parameters.

- To run DSE Graph Loader for CSV loading from a directory, use the following command:

```
graphloader filePatternCSV.groovy -graph testPattCSV -address
localhost
```

For testing purposes, the graph specified does not have to exist prior to running `graphloader`. However, for production applications, the graph and schema should be created prior to using `graphloader`.

### Mapping using [ ]

- If desired, [add configuration \(page 472\)](#) to the mapping script.
- Sample input file:

```
/* SAMPLE CSV INPUT:
id|name|gender
001|Julia Child|F
*/
```

- Specify the data input directory. The variable `inputfiledir` specifies the directory for the input files. Each of the identified files will be used for loading.

```
// DATA INPUT
// Define the data input source (a file which can be specified
// via command line arguments)
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/filePattern'
inputfileCSV = inputfiledir+'/data'
personInput =
 File.directory(inputfileCSV).fileMatches("person[1-9].csv").delimiter(' | ').header('id')

//Specifies what data source to load using which mapper (as
//defined inline)

load(personInput).asVertices {
 label "person"
 key "name"
}

/* RESULT:
 person1.csv and person2.csv will be loaded, but not
 badOne.csv
```

```
* /
```

The important element is `fileMatches("person[1-9].csv")`; this defines the pattern that will be matched for loaded files. All files `person1.csv` through `person9.csv` will be loaded, but `person15.csv` doesn't match the pattern and will not be loaded, as well as `badOne.csv`. Note that `fileMatches("person?.csv")` would achieve the same result.

This same pattern matching can be used for JSON input files, by substituting `person[1-9].json` for `person[1-9].csv` and using JSON input file parameters.

- Run DSE Graph Loader for this example use the following command:

```
graphloader filePatternRANGE.groovy -graph testPattRANGE -
address localhost
```

For testing purposes, the graph specified does not have to exist prior to running `graphloader`. However, for production applications, the graph and schema should be created prior to using `graphloader`.

### Mapping using {} with multiple patterns

- If desired, [add configuration \(page 472\)](#) to the mapping script.
- Sample input file:

```
/* SAMPLE CSV INPUT:
id|name|gender
001|Julia Child|F
*/
```

- Specify the data input directory. The variable `inputfiledir` specifies the directory for the input files. Each of the identified files will be used for loading.

```
// DATA INPUT
// Define the data input source (a file which can be specified
// via command line arguments)
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/filePattern/data'
personInput =
 File.directory(inputfiledir).fileMatches("{person*,badOne}.csv").delimiter('|').header(true)

//Specifies what data source to load using which mapper (as
//defined inline)

load(personInput).asVertices {
 label "person"
 key "name"
}

/* RESULT:
 person1.csv, person2.csv and badOne.csv will all be loaded
*/
```

The important element is `fileMatches( "{person*,badOne}.csv" )`; this defines the pattern that will be matched for loaded files. The files `person1.csv`, `person1.csv`, and `badOne.csv` will be loaded, because the pattern matches all three files. This same pattern matching can be used for JSON input files, by substituting `person*.json` for `person*.csv` and using JSON input file parameters.

- To run DSE Graph Loader for this example using the following command:

```
graphloader filePatternMULT.groovy -graph testPattMULT -address
localhost
```

For testing purposes, the graph specified does not have to exist prior to running `graphloader`. However, for production applications, the graph and schema should be created prior to using `graphloader`.

## Loading TEXT data

How to use the DSE Graph Loader to load delimited text data.

The data mapping script for delimited text data is shown with explanation. The full script is found at the bottom of the page.

- If desired, [add configuration \(page 472\)](#) to the mapping script.
- A sample of the data for load looks like the following:

```
SAMPLE INPUT
// For the author.dat file:
Julia Child|F
// For the book.dat file:
Simca's Cuisine: 100 Classic French Recipes for Every Occasion|
1972|0-394-40152-2
// For the authorBook.dat file:
Simca's Cuisine: 100 Classic French Recipes for Every Occasion|
Simone Beck
```

- Specify the data input files. The variable `inputfiledir` specifies the directory name for the input files. Each of the identified files will be used for loading.

```
// DATA INPUT
// Define the data input source (a file which can be specified
// via command line arguments)
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/TEXT/'
authorInput = File.text(inputfiledir + "author.dat").
 delimiter("|").
 header('name', 'gender')
bookInput = File.text(inputfiledir + "book.dat").
 delimiter("|").
 header('name', 'year', 'ISBN')
authorBookInput = File.text(inputfiledir + "authorBook.dat").
 delimiter("|").
```

```
header('bname' , 'aname')
```

Because the property key `name` is used for both vertex labels `author` and `book`, in the `authorBook` file, variables `aname` and `bname` are used for author name and book name, respectively. These variables are used in the mapping logic used to create the edges between `author` and `book` vertices.

- In each line, the file is specified as a `text` file, the file name is specified, a delimiter is set, and a header can be specified to identify the fields that will be read. The header can alternatively be specified on the first line of the data file. A map, `authorInput`, is created that will be used to process the data. The map can be manipulated before loading using [transforms \(page 527\)](#).

```
authorInput = File.text(inputfiledir +
 "author.dat").delimiter("|").header('name', 'gender')
```

If a `header()` is used in the mapping script and a header line is used in the data file, then both must match. Either a header line in the data file or a `header()` is required.

- [Create the main body of the mapping script. \(page 509\)](#) This part of the mapping script is the same regardless of the file format.
- To run DSE Graph Loader for text loading as a dry run, use the following command:

```
graphloader authorBookMappingTEXT.groovy -graph testTEXT -address
localhost -dryrun true
```

For testing purposes, the graph specified does not have to exist prior to running `graphloader`. However, for production applications, the graph and schema should be created prior to using `graphloader`.

- The full loading script is shown.

```
/** SAMPLE INPUT
author: Julia Child|F
book : Simca's Cuisine: 100 Classic French Recipes for Every
Occasion|1972|0-394-40152-2
authorBook: Simca's Cuisine: 100 Classic French Recipes for Every
Occasion|Simone Beck
 */

// CONFIGURATION
// Configures the data loader to create the schema
config create_schema: true, load_new: true, load_vertex_threads:
3

// DATA INPUT
// Define the data input source (a file which can be specified
via command line arguments)
// inputfiledir is the directory for the input files that is
given in the commandline
// as the "-filename" option
```

```
inputfiledir = '/tmp/CSV/'
authorInput = File.text(inputfiledir + "author.dat").
 delimiter("|").
 header('name', 'gender')
bookInput = File.text(inputfiledir + "book.dat").
 delimiter("|").
 header('name', 'year', 'ISBN')
authorBookInput = File.text(inputfiledir + "authorBook.dat").
 delimiter("|").
 header('bname', 'aname')

//Specifies what data source to load using which mapper (as
defined inline)

load(authorInput).asVertices {
 label "author"
 key "name"
}

load(bookInput).asVertices {
 label "book"
 key "name"
}

load(authorBookInput).asEdges {
 label "authored"
 outV "aname", {
 label "author"
 key "name"
 }
 inv "bname", {
 label "book"
 key "name"
 }
}
```

## Mapping several files with same format from a directory

- A sample of the data for load looks like the following:

```
SAMPLE INPUT
// For the author.text file:
name|gender
Julia Child|F
Simone Beck|F

// For the knows.text file:
aname|bname
Julia Child|James Beard
```

A number of files with the same format exist in a directory. If the files differ, the graphloader will issue an error and stop:

```
java.lang.IllegalArgumentException: /tmp/dirSource/data has more
than 1 input type.
```

- Specify the data input directory. The variable `inputfiledir` specifies the directory for the input files. Each of the identified files will be used for loading.

```
// DATA INPUT
// Define the data input source (a file which can be specified
// via command line arguments)
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/dirSource/data'
personInput =
 File.directory(inputfiledir).delimiter(' | ').header('name', 'gender')

//Specifies what data source to load using which mapper (as
//defined inline)

load(personInput).asVertices {
 label "author"
 key "name"
}
```

The important element is `File.directory()`; this defines the directory where the files are stored.

- Note that two directories could be used to load vertices and edges:

```
// DATA INPUT
// Define the data input source (a file which can be specified
// via command line arguments)
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/dirSource/data'
vertexfiledir = inputfiledir + '/vertices'
edgefiledir = inputfiledir + '/edges'
personInput =
 File.directory(vertexfiledir).delimiter(' | ').header('name', 'gender')
personEdgeInput =
 File.directory(edgefiledir).delimiter(' | ').header('aname', 'bname')

//Specifies what data source to load using which mapper (as
//defined inline)

load(personInput).asVertices {
 label "author"
 key "name"
}

load(personEdgeInput).asEdges {
 label "knows"
 outV "aname", {
 label "author"
```

```

 key "name"
 }
 inV "bname", {
 label "book"
 key "name"
 }
}

```

- To run DSE Graph Loader for text file loading from a directory, use the following command:

```
graphloader dirSourceMapping.groovy -graph testdirSource -address localhost
```

For testing purposes, the graph specified does not have to exist prior to running `graphloader`. However, for production applications, the graph and schema should be created prior to using `graphloader`.

## Loading TEXT data using regular expressions (regex)

How to use the DSE Graph Loader to load text data using regex.

The data mapping script for text data parsed using regular expressions (regex) is shown with explanation. The full script is found at the bottom of the page.

- If desired, [add configuration \(page 472\)](#) to the mapping script.
- A sample of the data for load looks like the following:

```

SAMPLE INPUT
// This file uses tabs between fields
// For the authorREGEX.data file:
name:Julia Child gender:F
// For the bookREGEX.dat file:
name:Simca's Cuisine: 100 Classic French Recipes for Every
Occasion year:1972 ISBN:0-394-40152-2
// For the authorBookREGEX.dat file:
bname:Simca's Cuisine: 100 Classic French Recipes for Every
Occasion aname:Simone Beck

```

- Specify the data input files. The variable `inputfiledir` specifies the directory name for the input files. Each of the identified files will be used for loading.

```

// DATA INPUT
// Define the data input source
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/REGEX/'
authorInput = File.text(inputfiledir + "authorREGEX.dat").
 regex("name:(.*)\\tgender:(MF)").

 header('name', 'gender')
bookInput = File.text(inputfiledir + "bookREGEX.dat").
 regex("name:(.*)\\tyear:(\\d{4})\\tISBN:(\\d{1}\\d{3}[-]\\d{1}\\d{5}[-]\\d{1}\\d{2}0,1)").

 header('name', 'year', 'ISBN')

```

```
authorBookInput = File.text(inputfiledir +
 "authorBookREGEX.dat").
 regex("bname:(.*)\\tname:(.*)").
 header('bname', 'aname')
```

Because the property key `name` is used for both vertex labels `author` and `book`, in the `authorBook` file, variables `aname` and `bname` are used for author name and book name, respectively. These variables are used in the mapping logic used to create the edges between `author` and `book` vertices.

- In each line, the file is specified as a text file, the file name is specified, a delimiter is set, and a header must be specified to identify the fields that will be read. In addition, to parse each line of the text file using regex, the regex logic is included. A map, `authorInput`, is created that will be used to process the data. The map can be manipulated before loading using [transforms \(page 527\)](#).

```
authorInput = File.text(inputfiledir +
 "authorREGEX.dat").regex("name:(.*)\\tgender:
 ([MF])").header('name', 'gender')
```

- [Create the main body of the mapping script. \(page 509\)](#) This part of the mapping script is the same regardless of the file format.
- To run DSE Graph Loader for text loading as a dry run, use the following command:

```
graphloader authorBookMappingREGEX.groovy -graph testREGEX -
address localhost -dryrun true
```

For testing purposes, the graph specified does not have to exist prior to running `graphloader`. However, for production applications, the graph and schema should be created prior to using `graphloader`.

- The full loading script is shown:

```
/* SAMPLE INPUT - uses tabs
author:
name:Julia Child gender:F
book:
name:Simca's Cuisine: 100 Classic French Recipes for Every
Occasion year:1972 ISBN:0-394-40152-2
authorBook:
bname:Simca's Cuisine: 100 Classic French Recipes for Every
Occasion aname:Simone Beck
*/
// CONFIGURATION
// Configures the data loader to create the schema
config create_schema: true, load_new: true, load_vertex_threads:
3
// DATA INPUT
// Define the data input source (a file which can be specified
via command line arguments)
```

```

// inputfiledir is the directory for the input files that is
// given in the commandline
// as the "-filename" option
inputfiledir = '/tmp/REGEX/'
authorInput = File.text(inputfiledir + "authorREGEX.dat").
 regex("name:(.*)\\tgender:([MF])").
 header('name', 'gender')
bookInput = File.text(inputfiledir + "bookREGEX.dat").
 regex("name:(.*)\\tyear:([0-9]{4})\\tISBN:([0-9]{1}{1}[-]{1}{1}[0-9]
{3}[-]{1}[0-9]{5}[-]{1}[0-9]{0,1})").
 header('name', 'year', 'ISBN')
authorBookInput = File.text(inputfiledir +
 "authorBookREGEX.dat").
 regex("bname:(.*)\\taname:(.*)").
 header('bname', 'aname')

//Specifies what data source to load using which mapper (as
defined inline)

load(authorInput).asVertices {
 label "author"
 key "name"
}

load(bookInput).asVertices {
 label "book"
 key "name"
}

load(authorBookInput).asEdges {
 label "authored"
 outV "aname", {
 label "author"
 key "name"
 }
 inV "bname", {
 label "book"
 key "name"
 }
}

```

## Loading data from a JDBC compatible database.

How to use the DSE Graph Loader to load data from a JDBC compatible database.

The data mapping script for loading from a JDBC compatible database is shown with explanation. The full script is found at the bottom of the page.

**Note:** Using DSE Graph Loader to load directly from a JDBC compatible database is convenient, but very slow for a large database. Test a small dataset first, to see if the time required to move a larger dataset makes this method efficient.

- If desired, [add configuration \(page 472\)](#) to the mapping script.
- A sample of the data for load looks like the following:

```
SAMPLE INPUT
// For the author data:
name:Julia Child gender:F
// For the book data:
name:Simca's Cuisine: 100 Classic French Recipes for Every
Occasion year:1972 ISBN:0-394-40152-2
// For the authorBook data:
bname:Simca's Cuisine: 100 Classic French Recipes for Every
Occasion aname:Simone Beck
```

Because the property key `name` is used for both vertex labels `author` and `book`, in the `authorBook` file, variables `aname` and `bname` are used for author name and book name, respectively. These variables are used in the mapping logic used to create the edges between `author` and `book` vertices.

- Some databases will need a driver installed in the same directory as the `graphloader` script. For the following example using MySQL, the [driver can be downloaded](#). Unzip the file and copy the `mysql-connector-java-5.1.44-bin.jar` file to the correct directory. A similar download would be required for the other databases as well.
- Specify the data input database with JDBC information. The variable `inputDatabase` specifies the data input database. This example uses the `MySQLdatabase`, but any JDBC-compliant database (H2, MySQL, Postgres, Oracle) can be used. The `connection to a localhost` and a MySQL database `sample` are specified. In addition, `user` and `password` are defined. The `MySQL()` step denotes the data connection to a MySQL database. The connection can alternatively define a remote machine address.

```
// DATA INPUT
// Define the data input source (a database connection and SQL
// statements for data selection)
// inputDatabase is the database name
inputDatabase = 'localhost/sample'
db = Database.connection('jdbc:mysql:///' +
 inputDatabase).user('root').password('foo').MySQL()
// Define multiple data inputs from the database source via SQL
// queries
authorInput = db.query "select * from author";
bookInput = db.query "select * from book";
authorBookInput = db.query "select * from authorbook";
```

**Note:** To load data from H2, the connection line could be:

```
inputDatabase = '~/test'
db = Database.connection("jdbc:h2:" +
 inputDatabase).H2().user("sa")
```

For Postgres, `Postgre()` is used, and for Oracle, `Oracle()`.

- In each line, the database query is specified that will be used to retrieve the data. A map, `authorInput`, is created that will be used to process the data. The map can be manipulated before loading using [transforms \(page 527\)](#).

```
authorInput = db.query "SELECT * FROM AUTHOR";
```

**Important:** DSE Graph Loader will retrieve all column names from the database with lower-cased names. Create the graph schema with corresponding lower-cased names to avoid read errors.

- [Create the main body of the mapping script. \(page 509\)](#) This part of the mapping script is the same regardless of the file format.
- To run DSE Graph Loader for text loading as a dry run, use the following command:

```
graphloader authorBookMappingJDBC.groovy -graph testJDBC -address localhost -dryrun true
```

For testing purposes, the graph specified does not have to exist prior to running `graphloader`. However, for production applications, the graph and schema should be created prior to using `graphloader`.

- The full loading script is shown.

```
/* SAMPLE INPUT
author:
name:Julia Child gender:F
book:
name:Simca's Cuisine: 100 Classic French Recipes for Every
Occasion year:1972 ISBN:0-394-40152-2
authorBook:
bname:Simca's Cuisine: 100 Classic French Recipes for Every
Occasion aname:Simone Beck
*/
// CONFIGURATION
// Configures the data loader to create the schema
config create_schema: true, load_new: true, load_vertex_threads:
3
// DATA INPUT
// Define the data input source (a database connection and SQL
// statements for data selection)
inputDatabase = 'localhost/sample'
db = Database.connection('jdbc:mysql:///' +
inputDatabase).user('root').password('foo').MySQL()
// Define multiple data inputs from the database source via SQL
// queries
authorInput = db.query "select * from author";
bookInput = db.query "select * from book";
```

```

authorBookInput = db.query "select * from authorbook";

//Specifies what data source to load using which mapper (as
defined inline)

load(authorInput).asVertices {
 label "author"
 key "name"
}

load(bookInput).asVertices {
 label "book"
 key "name"
}

load(authorBookInput).asEdges {
 label "authored"
 outV "aname", {
 label "author"
 key "name"
 }
 inV "bname", {
 label "book"
 key "name"
 }
}
}

```

## Loading data from Hadoop (HDFS)

How to use the DSE Graph Loader to load data from Hadoop (HDFS).

The data mapping script for loading from HDFS is shown with explanation. The full script is found at the bottom of the page.

- If desired, [add configuration \(page 472\)](#) to the mapping script.
- A sample of the CSV data residing on HDFS:

```

// SAMPLE INPUT
// For the author.csv file:
// name|gender
// Julia Child|F
// For the book.csv file:
// name|year|ISBN
// Simca's Cuisine: 100 Classic French Recipes for Every
// Occasion|1972|0-394-40152-2
// For the authorBook.csv file:
// bname|aname
// Simca's Cuisine: 100 Classic French Recipes for Every
// Occasion|Simone Beck

```

Because the property key name is used for both vertex labels author and book, in the authorBook file, variables aname and bname are used for author name and

book name, respectively. These variables are used in the mapping logic used to create the edges between `author` and `book` vertices.

- Specify the data inputs using a HDFS reference `dfs_uri` and the filenames:

```
// DATA INPUT
// Define the data input sources /
// dfs_uri specifies the URI to the HDFS directory in which the
// files are stored

dfs_uri = 'hdfs://hadoopNode:9000/food/'
authorInput = File.csv(dfs_uri + 'author.csv.gz').
 gzip().
 delimiter('|')
bookInput = File.csv(dfs_uri + 'book.csv.gz').
 gzip().
 delimiter('|')
authorBookInput = File.csv(dfs_uri + 'authorBook.csv.gz').
 gzip().
 delimiter('|')
```

This example uses compressed files and the additional step `gzip()`.

- Create the main body of the mapping script. ([page 509](#)) This part of the mapping script is the same regardless of the file format.
- To run DSE Graph Loader for text loading as a dry run, use the following command:

```
graphloader authorBookMappingHDFS.groovy -graph testHDFS -address
localhost -dryrun true
```

For testing purposes, the graph specified does not have to exist prior to running `graphloader`. However, for production applications, the graph and schema should be created prior to using `graphloader`. The `-dryrun true` option runs the command without loading data.

- The full loading script is shown.

```
// SAMPLE INPUT
// For the author.csv file:
// name|gender
// Julia Child|F
// For the book.csv file:
// name|year|ISBN
// Simca's Cuisine: 100 Classic French Recipes for Every
// Occasion|1972|0-394-40152-2
// For the authorBook.csv file:
// bname|aname
// Simca's Cuisine: 100 Classic French Recipes for Every
// Occasion|Simone Beck

// CONFIGURATION
// Configures the data loader to create the schema
```

```

config create_schema: true, load_new: true, preparation: true

// DATA INPUT
// Define the data input sources
// dfs_uri specifies the URI to the HDFS directory in which the
// files are stored

dfs_uri = 'hdfs://hadoopNode:9000/food/'
authorInput = File.csv(dfs_uri + 'author.csv.gz').
 gzip().
 delimiter('|')
bookInput = File.csv(dfs_uri + 'book.csv.gz').
 gzip().
 delimiter('|')
authorBookInput = File.csv(dfs_uri + 'authorBook.csv.gz').
 gzip().
 delimiter('|')

// Specifies what data source to load using which mapper (as
// defined inline)

load(authorInput).asVertices {
 label "author"
 key "name"
}

load(bookInput).asVertices {
 label "book"
 key "name"
}

load(authorBookInput).asEdges {
 label "authored"
 outV "aname", {
 label "author"
 key "name"
 }
 inV "bname", {
 label "book"
 key "name"
 }
}

```

## Loading data from AWS S3

How to use the DSE Graph Loader to load data from AWS S3.

The data mapping script for loading from AWS S3 is shown with explanation. The full script is found at the bottom of the page.

- If desired, [add configuration \(page 472\)](#) to the mapping script.
- A sample of the CSV data residing on AWS S3:

```
// SAMPLE INPUT
// For the author.csv file:
// name|gender
// Julia Child|F
// For the book.csv file:
// name|year|ISBN
// Simca's Cuisine: 100 Classic French Recipes for Every
// Occasion|1972|0-394-40152-2
// For the authorBook.csv file:
// bname|aname
// Simca's Cuisine: 100 Classic French Recipes for Every
// Occasion|Simone Beck
```

Because the property key `name` is used for both vertex labels `author` and `book`, in the `authorBook` file, variables `aname` and `bname` are used for author name and book name, respectively. These variables are used in the mapping logic used to create the edges between `author` and `book` vertices.

- Specify the data inputs using a AWS S3 reference `dfs_uri` that defines `s3://[bucket]` and the filenames:

```
// DATA INPUT
// Define the data input sources /
// dfs_uri specifies the URI to the HDFS directory in which the
// files are stored

dfs_uri = 's3://food/'
authorInput = File.csv(dfs_uri +
 'author.csv.gz').gzip().delimiter('|')
bookInput = File.csv(dfs_uri +
 'book.csv.gz').gzip().delimiter('|')
authorBookInput = File.csv(dfs_uri +
 'authorBook.csv.gz').gzip().delimiter('|')
```

This example uses compressed files and the additional step `gzip()`.

- Create the main body of the mapping script.** ([page 509](#)) This part of the mapping script is the same regardless of the file format.
- To run DSE Graph Loader for text loading as a dry run, use the following command:

```
graphloader authorBookMappingS3.groovy -graph testS3 -address
localhost -dryrun true
```

For testing purposes, the graph specified does not have to exist prior to running `graphloader`. However, for production applications, the graph and schema should be created prior to using `graphloader`. The `-dryrun true` option runs the command without loading data.

- The full loading script is shown.

```
// SAMPLE INPUT
// For the author.csv file:
```

```

// name|gender
// Julia Child|F
// For the book.csv file:
// name|year|ISBN
// Simca's Cuisine: 100 Classic French Recipes for Every
// Occasion|1972|0-394-40152-2
// For the authorBook.csv file:
// bname|aname
// Simca's Cuisine: 100 Classic French Recipes for Every
// Occasion|Simone Beck

// CONFIGURATION
// Configures the data loader to create the schema

config create_schema: true, load_new: true, preparation: true

// DATA INPUT
// Define the data input sources
// dfs_uri specifies the URI to the HDFS directory in which the
// files are stored

dfs_uri = 's3://food/'
authorInput = File.csv(dfs_uri +
 'author.csv.gz').gzip().delimiter('|')
bookInput = File.csv(dfs_uri +
 'book.csv.gz').gzip().delimiter('|')
authorBookInput = File.csv(dfs_uri +
 'authorBook.csv.gz').gzip().delimiter('|')

// Specifies what data source to load using which mapper (as
// defined inline)

load(authorInput).asVertices {
 label "author"
 key "name"
}

load(bookInput).asVertices {
 label "book"
 key "name"
}

load(authorBookInput).asEdges {
 label "authored"
 outV "aname", {
 label "author"
 key "name"
 }
 inV "bname", {
 label "book"
 key "name"
 }
}

```

```
}
```

## Loading Gryo data

How to use the DSE Graph Loader to load Gryo data.

One file format for importing and exporting data to and from DSE Graph is Gryo, a binary file format. Gryo is a Gremlin variant of Kryo, a fast and efficient object graph serialization framework for Java.

The data mapping script for Gryo data is shown with explanation. The full script is found at the bottom of the page.

**Note:** DSE Graph Loader can load Gryo files generated with DSE Graph or with [TinkerGraph](#), the in-memory graph database included with Apache TinkerPop. The Gryo files generated with DSE Graph have a different format from TinkerGraph Gryo files, and the [mapping script is different \(page 524\)](#) for loading data from each source.

- If desired, [add configuration \(page 472\)](#) to the mapping script.
- Specify the data input file. The variable `inputfiledir` specifies the directory for the input file. The identified file will be used for loading.

```
// DATA INPUT
// Define the data input source
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/Gryo/'
recipeInput = Graph.file(inputfiledir + 'recipe.gryo').gryo()
```

If the Gryo input file is generated from DSE Graph, an additional step `dse()` will allow the input data to be streamed, facilitating large file transfers.

```
// DATA INPUT
// Define the data input source
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/Gryo/'
recipeInput = Graph.file(inputfiledir +
 'recipe.gryo').gryo().dse()
```

- The file is specified as a `gryo` file and an additional step `gryo()` identifies that the file should be processed as a Gryo file. A map, `recipeInput`, is created that will be used to process the data.

```
recipeInput = Graph.file(inputfiledir + 'recipe.gryo')
```

Note that `Graph.file` is used, in contrast to `File.csv` or `File.json`.

**Tip:** If you wish to access a `java.io.File` object, fully namespace the first call; otherwise, DSE Graph Loader overrides the `File` object:

```
currentDir = new java.io.File('.').getCanonicalPath() + '/'
```

```
source = Graph.file(currentDir + 'myfile.kryo').gryo()
```

- **Create the main body of the mapping script.** ([page 509](#)) This part of the mapping script is the same regardless of the file format, although Gryo files use a [slightly modified version](#) ([page 524](#)).
- To run DSE Graph Loader for Gryo loading as a dry run, use the following command:

```
graphloader recipeMappingGRYO.groovy -graph testGRYO -address
localhost -dryrun true
```

For testing purposes, the graph specified does not have to exist prior to running `graphloader`. However, for production applications, the graph and schema should be created prior to using `graphloader`.

- The full loading script is shown:

```
/* SAMPLE INPUT
Gryo file is a binary file
 */

// CONFIGURATION
// Configures the data loader to create the schema
config create_schema: true, load_new: true

// DATA INPUT
// Define the data input source
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/GRYO/'
recipeInput = Graph.file(inputfiledir + 'recipe.gryo').gryo()

load(recipeInput.vertices()).asVertices {
 labelField "~label"
 key "~id", "id"
}

load(recipeInput.edges()).asEdges {
 labelField "~label"
 outV "outV", {
 labelField "~label"
 key "~id", "id"
 }
 inV "inV", {
 labelField "~label"
 key "~id", "id"
 }
}
```

```
}
```

## Loading GraphSON data

How to use the DSE Graph Loader to load GraphSON data.

The data mapping script for GraphSON data is shown with explanation. The full script is found at the bottom of the page.

**Note:** DSE Graph Loader can load GraphSON files generated with [Apache TinkerGraph](#), the in-memory graph database included with Apache TinkerPop. GraphSON files generated with DSE Graph cannot be loaded using DSE Graph Loader.

- If desired, [add configuration \(page 472\)](#) to the mapping script.
- Specify the data input file. The variable `inputfiledir` specifies the directory for the input file. The identified file will be used for loading.

```
// DATA INPUT
// Define the data input source
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/GraphSON/'
recipeInput = Graph.file(inputfiledir + 'recipe.json').graphson()
```

If the GraphSON input file is generated from DSE Graph, an additional step `dse()` will allow the input data to be streamed, facilitating large file transfers.

```
// DATA INPUT
// Define the data input source
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/GraphSON/'
recipeInput = Graph.file(inputfiledir +
'recipe.json').graphson().dse()
```

- The file is specified as a `json` file and an additional step `graphson()` identifies that the file should be processed as a GraphSON file. A map, `recipeInput`, is created that will be used to process the data.

```
recipeInput = Graph.file(inputfiledir + 'recipe.json')
```

Note that `Graph.file` is used, in contrast to `File.csv` or `File.json`.

- [Create the main body of the mapping script. \(page 509\)](#) This part of the mapping script is the same regardless of the file format, although GraphSON files use a [slightly modified version \(page 527\)](#).
- To run DSE Graph Loader for GraphSON loading as a dry run, use the following command:

```
graphloader recipeMappingGraphSON.groovy -graph testGraphSON -
address localhost -dryrun true
```

For testing purposes, the graph specified does not have to exist prior to running `graphloader`. However, for production applications, the graph and schema should be created prior to using `graphloader`.

- The full loading script is shown:

```
/* SAMPLE INPUT
GraphSON file is a JSON-like file
 */

// CONFIGURATION
// Configures the data loader to create the schema
config create_schema: true, load_new: true

// DATA INPUT
// Define the data input source
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/GraphSON/'
recipeInput = Graph.file(inputfiledir + 'recipe.json').graphson()

//Specifies what data source to load using which mapper (as
defined inline)

load(recipeInput.vertices()).asVertices {
 labelField "~label"
 key "~id", "id"
}

load(recipeInput.edges()).asEdges {
 labelField "~label"
 outV "outV", {
 labelField "~label"
 key "~id", "id"
 }
 inV "inV", {
 labelField "~label"
 key "~id", "id"
 }
}
```

## Loading GraphML data

How to use the DSE Graph Loader to load GraphML data.

The data mapping script for GraphML data is shown with explanation. The full script is found at the bottom of the page.

**Note:** DSE Graph Loader can load GraphML files generated with [TinkerGraph](#), the in-memory graph database included with Apache TinkerPop. GraphML files generated with DSE Graph cannot be loaded using DSE Graph Loader.

- If desired, [add configuration \(page 472\)](#) to the mapping script.

- Specify the data input file. The variable `inputfiledir` specifies the directory for the input file. The identified file will be used for loading.

```
// DATA INPUT
// Define the data input source
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/GraphML/'
recipeInput = Graph.file(inputfiledir + 'recipe.xml').graphml()
```

- The file is specified as a `xml` file and an additional step `graphml()` identifies that the file should be processed as a GraphML file. A map, `recipeInput`, is created that will be used to process the data.

```
recipeInput = Graph.file(inputfiledir + 'recipe.xml')
```

Note that `Graph.file` is used, in contrast to `File.csv` or `File.json`.

- Create the main body of the mapping script. ([page 509](#)) This part of the mapping script is the same regardless of the file format, although GraphML files use a slightly modified version ([page 526](#)).
- To run DSE Graph Loader for GraphML loading as a dry run, use the following command:

```
graphloader recipeMappingGraphML.groovy -graph testGraphML -
address localhost -dryrun true
```

For testing purposes, the graph specified does not have to exist prior to running `graphloader`. However, for production applications, the graph and schema should be created prior to using `graphloader`.

- The full loading script is shown:

```
/* SAMPLE INPUT
GraphML file is an XML file
 */

// CONFIGURATION
// Configures the data loader to create the schema
config create_schema: true, load_new: true

// DATA INPUT
// Define the data input source
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/GraphML/'
recipeInput = Graph.file(inputfiledir + 'recipe.xml').graphml()

//Specifies what data source to load using which mapper (as
defined inline)

load(recipeInput.vertices()).asVertices {
 labelField "~label"
```

```

 key "~id", "id"
 }

load(recipeInput.edges()).asEdges {
 labelField "~label"
 outV "outV", {
 labelField "~label"
 key "~id", "id"
 }
 inV "inV", {
 labelField "~label"
 key "~id", "id"
 }
}

```

## Mapping script

Explain the main body of the mapping script.

Regardless of the file format selected, the main body of the mapping script is the same. After setting configuration and adding a data input source, the mapping commands are specified.

- Vertices are loaded from `authorInput`, with the vertex label `author` and the property `key name` which uniquely identifies a vertex listed for the `key`. Note that, in this example, if `gender` were chosen for the `key`, it would not be unique enough to load each record from the data file. Using the configuration setting `load_new: true` can significantly speed up the loading process, but a duplicate vertex will be created if the record already exists. All other property keys will be loaded, but do not have to be identified in the loading script. For `author` vertices, `gender` will also be loaded.

```

load(authorInput).asVertices {
 label "author"
 key "name"
}

```

**Note:** If more than 256 property key values are present in the input file, see [important information \(page 147\)](#) on the `max_query_params` value in the `dse.yaml` file.

One load statement must be created for each vertex loaded, even if the same file is reused for one or more vertices. When using the same input file for multiple vertices, sometimes a field exists in the input file that should be ignored for a particular vertex. See the instructions for [ignoring a field \(page 511\)](#). If an input file includes multiple types of lines, for instance, `authors` and `reviewers`, that should be read into different vertex labels, see the instructions for [labelField \(page 512\)](#).

- Loading the book vertices follows a similar pattern. Note that both vertex labels `author` and `book` use `name` as the unique key for identifying a vertex. This declares that the vertex record does not yet exist in the graph at the beginning of the loading process.

```
load(bookInput).asVertices {
 label "book"
 key "name"
}
```

- After vertices are loaded, edges are loaded. Similar to the vertex mapping, an edge label is specified. In addition, the outgoing vertex (`outV`) and incoming vertex (`inV`) for the edge must be identified. For each vertex in `outV` or `inV`, the vertex label is specified with `label`, and the unique key is specified with `key`.

```
load(authorBookInput).asEdges {
 label "authored"
 outV "aname", {
 label "author"
 key "name"
 }
 inV "bname", {
 label "book"
 key "name"
 }
}
```

Note the naming convention used for the `outV` and `inV` designations. Because both the outgoing vertex and the incoming vertex keys are listed as `name`, the designators `aname` and `bname` are used to distinguish between the author name and the book name as the field names in the input file.

- An alternative to the definitions shown above is to specify the mapping logic with variables, and then list the load statements separately.

```
authorMapper = {
 label "author"
 key "name"
}
bookMapper = {
 label "book"
 key "name"
}
authorBookMapper = {
 label "authored"
 outV "aname", {
 label "author"
 key "name"
 }
 inV "bname", {
 label "book"
 key "name"
 }
}

load(authorInput).asVertices(authorMapper)
load(bookInput).asVertices(bookMapper)
```

```
load(authorBookInput).asEdges(authorBookMapper)
```

## Ignoring a field in input file

Mapping data while ignoring a field with DSE Graph Loader.

If the input file includes a field that should be ignored for a particular vertex load, use `ignore`.

1. Create a map script that ignores the field `restaurant`:

```
// authorInput includes name, gender, and restaurant
// but restaurant is not loaded
/* Sample input:
name|gender|restaurant
Alice Waters|F|Chez Panisse
*/

load(authorInput).asVertices {
 label "author"
 key "name"
 ignore "restaurant"
}
```

2. An additional example shows the use of `ignore` where two different types of vertices are created, `book` and `author`, using the same input file.

```
/* Sample input:
name|gender|bname
Julia Child|F|The French Chef Cookbook
Simone Beck|F|The Art of French Cooking, Vol. 1
*/

//inputfiledir = '/tmp/TEXT/'
authorInput = File.text("author.dat").
 delimiter("|").
 header('name', 'gender', 'bname')

//Specifies what data source to load using which mapper (as
//defined inline)

load(authorInput).asVertices {
 label "book"
 key "bname"
 ignore "name"
 ignore "gender"
}

load(authorInput).asVertices {
 label "author"
 key "name"
 outV "book", "authored", {
 label "book"
 key "bname"
```

```
 }
```

## Using labelField to parse input into different vertex labels

Mapping data using labelField to parse input into different vertex labels with DSE Graph Loader.

Oftentimes, an input file includes a field that is used to identify the vertex label. In order to load the file and create different vertex labels on-the-fly, `labelField` is used to identify that particular field.

1. Create a map to input both *authors* and *reviewers* from the same file using `labelField`:

```
/* SAMPLE INPUT
The input personInput includes type of person, name, gender; type
can be either author or reviewer.
type::name::gender
author::Julia Child::F
reviewer::Jane Doe::F
*/
personInput = File.text('people.dat').
 delimiter("::").
 header('type','name','gender')

load(personInput).asVertices{
 labelField "type"
 key "name"
}
```

Running this map script using the sample data results in two different vertex labels, with one record for each.

```
g.V().hasLabel('author').valueMap()
{gender=[F], name=[Julia Child]}
g.V().hasLabel('reviewer').valueMap()
{gender=[F], name=[Jane Doe]}
```

## Using compressed files to load data

Mapping compressed data with DSE Graph Loader.

Compressed files can be loaded using DSE Graph Loader to load both vertices and edges. This example loads vertices and edges, as well as edge properties, using gzipped files.

1. Create a map script that specifies the input files as compressed \*.gz files:

```
/* SAMPLE INPUT
rev_name|recipe_name|timestamp|stars|comment
John Doe|Beef Bourguignon|2014-01-01|5|comment
```

```

*/

// CONFIGURATION
// Configures the data loader to create the schema
config create_schema: false, load_new: false

// DATA INPUT
// Define the data input source (a file which can be specified
// via command line arguments)
// inputfiledir is the directory for the input files that is
// given in the commandline
// as the "-filename" option
inputfiledir = '/tmp/CSV/'

// This next file is not required if the reviewers already exist
reviewerInput = File.csv(inputfiledir + "reviewers.csv.gz").
 gzip().
 delimiter('|')

// This next file is not required if the recipes already exist
recipeInput = File.csv(inputfiledir + "recipes.csv.gz").
 gzip().
 delimiter('|')

// This is the file that is used to create the edges with edge
// properties
reviewerRatingInput = File.csv(inputfiledir +
 "reviewerRatings.csv.gz").
 gzip().
 delimiter(|)

//Specifies what data source to load using which mapper (as
defined inline)

load(reviewerInput).asVertices {
 label "reviewer"
 key "name"
}

load(recipeInput).asVertices {
 label "recipe"
 key "name"
}

load(reviewerRatingInput).asEdges {
 label "rated"
 outV "rev_name", {
 label "reviewer"
 key "name"
 }
 inV "recipe_name", {
 label "recipe"
 key "name"
 }
 // properties are automatically added from the file, using the
header line as property keys
 // from previously created schema
}

```

```
}
```

The compressed files are designated as `.gz` files, followed by a `gzip()` step for processing. Edge properties are loaded from one of the input files based on the header identifying the property keys to use for the values listed in each line of the CSV file. The edge properties populate a `rated` edge between a `reviewer` vertex and a `recipe` vertex with the properties `timestamp`, `stars`, and `comment`.

## Mapping data with a composite custom id

Mapping data with a composite custom id with DSE Graph Loader.

Data with a [composite primary key \(page 451\)](#) requires some additional definition when specifying the key for loading, if the custom id uses multiple keys for definition (either `partitionKeys` and/or `clusteringKeys`).

1. Inserting data for vertices with a composite custom id requires the declaration of two or more keys:

```
/* SAMPLE INPUT
cityId|sensorId|fridgeItem
santaCruz|93c4ec9b-68ff-455e-8668-1056ebc3689f|asparagus
 */

load(fridgeItemInput).asVertices {
 label "fridgeSensor"
 // The vertexLabel schema for fridgeSensor includes two keys:
 // partition key: cityId and clustering key: sensorId
 key cityId: "cityId", sensorId: "sensorId"
}
```

**Tip:** The schema for the composite custom id must be created prior to using DSE Graph Loader, and cannot be inferred from the data. In addition, create a [search index \(page 460\)](#) that includes all properties in the composite key. The search index is required to use DSE Graph Loader for inserting composite custom id data.

Check the vertex id results with `id()` to retrieve the full primary key definition:

```
gremlin> g.V().hasLabel('fridgeSensor').id()
==>{~label=fridgeSensor,
 sensorId=93c4ec9b-68ff-455e-8668-1056ebc3689f,
 cityId=santaCruz}
==>{~label=fridgeSensor, sensorId=9c23b683-1de2-4c97-
a26a-277b3733732a, cityId=sacramento}
==>{~label=fridgeSensor, sensorId=eff4a8af-2b0d-4ba9-a063-
c170130e2d84, cityId=sacramento}
```

Each vertex stores `fridgeItem` as data:

```
gremlin> g.V().valueMap()
==>{fridgeItem=[asparagus]}
==>{fridgeItem=[ham]}
```

```
=>{fridgeItem=[eggs]}
```

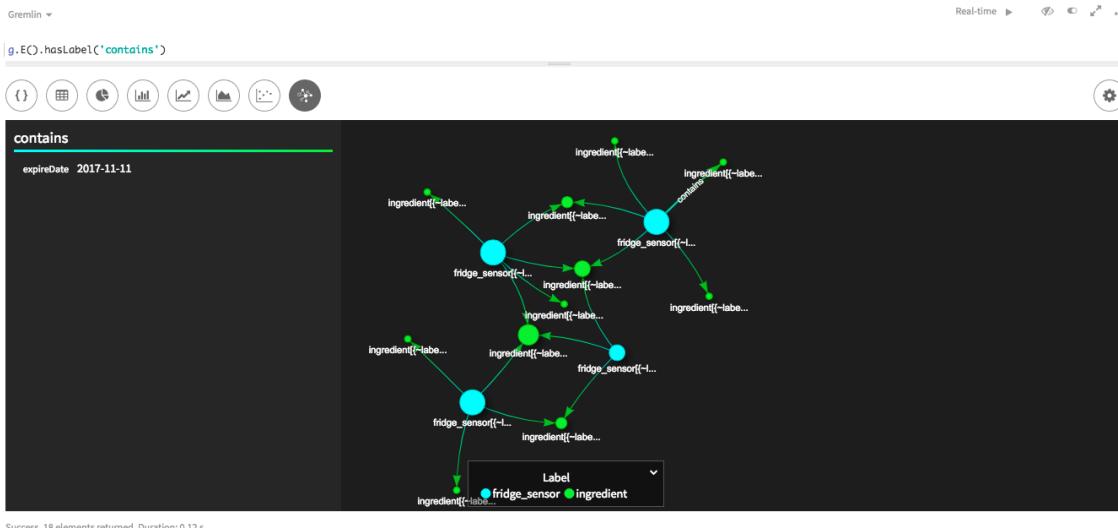
- To load edges based on a composite key, a transformation is required:

```
/* SAMPLE EDGE DATA
cityId|sensorId|name
santaCruz|93c4ec9b-68ff-455e-8668-1056ebc3689f|asparagus
*/
the_edges = File.csv(inputfiledir +
 "fridgeItemEdges.csv").delimiter(' | ')

the_edges = the_edges.transform {
 it['fridgeSensor'] = [
 'cityId' : it['cityId'],
 'sensorId' : it['sensorId']];
 it['ingredient'] = [
 'name' : it['name']];
 it
}
load(the_edges).asEdges {
 label "contains"
 outV "ingredient", {
 label "ingredient"
 key "name"
 }
 inV "fridgeSensor", {
 label "fridgeSensor"
 key cityId:"cityId", sensorId:"sensorId"
 }
}
```

The edge file transforms the partition key and clustering key into a map of `cityId` and `sensorId`. This map can then be used to designate the key for a `fridgeSensor` vertex when the edges are loaded.

The resulting map shows the edges created between ingredient and `fridgeSensor` vertices.



3. For DSE 5.1.3 and later, an alternative method of loading edge data from CSV files can be used:

```
/* SAMPLE EDGE DATA
cityId|sensorId|homeId
100|001|9001
*/

isLocatedAt_fridgeSensor = File.csv(/tmp/data/edges/ +
 "isLocatedAt_fridgeSensor.csv").delimiter(' | ')

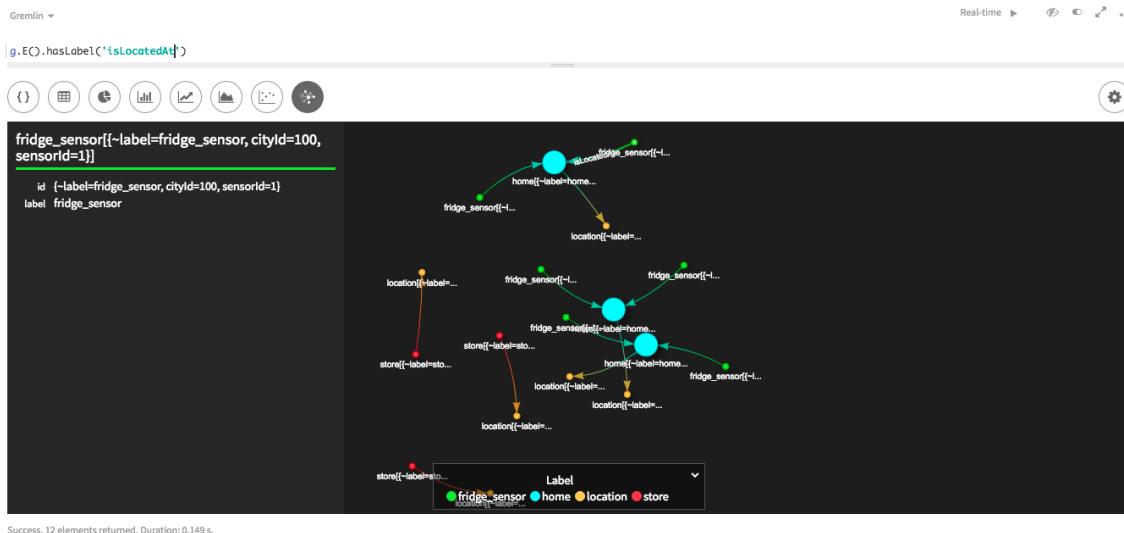
load(isLocatedAt_fridgeSensor).asEdges {
 label "isLocatedAt"
 outV {
 label "fridgeSensor"
 key cityId: "cityId", sensorId: "sensorId"
 exists()
 ignore "homeId"
 }
 inV {
 label "home"
 key "homeId"
 exists()
 ignore "cityId"
 ignore "sensorId"
 }
 ignore "cityId"
 ignore "sensorId"
 ignore "homeId"
}
```

In this example, no transform is required, but `ignore` statements are required in both the `inv` and `outV` declarations, as well as the edge properties section. Removing the `exists()` statement in the incoming and outgoing vertex

declarations can enable loading the vertices as well as the edges in this mapping script.

**Important:** There is a new subtle change in the `inV` and `outV` declarations. An input field name is no longer used, such as `inV "home"`, `{`, due to the requirement to support multiple-key custom ids.

The resulting map:



## Mapping multi-cardinality edges

Mapping multi-cardinality edges data with DSE Graph Loader.

Multiple cardinality edges are a common type of data that is inserted into graphs. Often, the input file has both vertex and edge information for loading.

1. Inserting vertices and multi-cardinality edges can be accomplished from one file with judicious use of `ignore` while loading vertices:

```
/* SAMPLE INPUT
authorCity:
author|city|dateStart|dateEnd
Julia Child|Paris|1961-01-01|1967-02-10
*/

// CONFIGURATION
// Configures the data loader to create the schema
config dryrun: false, preparation: true, create_schema: false,
load_new: true, schema_output: 'loader_output.txt'

// DATA INPUT
// Define the data input source (a file which can be specified
// via command line arguments)
// inputfiledir is the directory for the input files
inputfiledir = '/tmp/multiCard/'
```

```

authorCityInput = File.csv(inputfiledir +
 "authorCity.csv").delimiter('|')

//Specifies what data source to load using which mapper (as
defined inline)

// Ignore city, dateStart, and dateEnd when creating author
vertices
load(authorCityInput).asVertices {
 label "author"
 key "author"
 ignore "city"
 ignore "dateStart"
 ignore "dateEnd"
}

// Ignore author, dateStart, and dateEnd when creating city
vertices

load(authorCityInput).asVertices {
 label "city"
 key "city"
 ignore "author"
 ignore "dateStart"
 ignore "dateEnd"
}

// create edges from author -> city and include the edge
properties dateStart and dateEnd
load(authorCityInput).asEdges {
 label "livedIn"
 outV "author", {
 label "author"
 key "author"
 }
 inV "city", {
 label "city"
 key "city"
 }
}
}

```

## Mapping meta-properties

Mapping meta-property data with DSE Graph Loader.

If the input file includes meta-properties, or properties that have properties, use `vertexProperty`.

The schema for this data load should be created prior to running `graphloader`

```

// PROPERTY KEYS
schema.propertyKey('name').Text().single().create()
schema.propertyKey('gender').Text().single().create()
schema.propertyKey('badge').Text().single().create()

```

```

schema.propertyKey('since').Int().single().create()
// Create the meta-property since on the property badge
schema.propertyKey('badge').properties('since').add()
// VERTEX LABELS
schema.vertexLabel('reviewer').properties('name', 'gender', 'badge').create()
// INDEXES
schema.vertexLabel('reviewer').index('byname').materialized().by('name').add()

```

1. The mapping script uses `vertexProperty` to identify `badge` as a vertex property.  
Note the structure of the nested fields for `badge` in the JSON file.

```

* SAMPLE INPUT
reviewer: { "name": "Jon Doe", "gender": "M", "badge" : { "value":
 "Gold Badge", "since" : 2012 } }
/*
// CONFIGURATION
// Configures the data loader to create the schema
config dryrun: false, preparation: true, create_schema:
 true, load_new: true, load_vertex_threads: 3, schema_output:
 'loader_output.txt'

// DATA INPUT
// Define the data input source (a file which can be specified
// via command line arguments)
// inputfiledir is the directory for the input files
inputfiledir = '/tmp/'
reviewerInput = File.json(inputfiledir + "reviewer.json")

//Specifies what data source to load using which mapper (as
defined inline)

load(reviewerInput).asVertices{
 label "reviewer"
 key "name"
 vertexProperty "badge", {
 value "value"
 }
}

```

Running this mapping script using the sample data results in a `reviewer` vertex where the property `badge` has a meta-property `since`.

```

g.V().valueMap()
{badge=[Gold Badge], gender=[M], name=[Jane Doe]}
g.V().properties('badge').valueMap()
{since=2012}

```

## Mapping multiple meta-properties

Mapping multiple meta-property data with DSE Graph Loader.

If the input file includes multiple meta-properties, or properties that have multiple properties, use `vertexProperty`.

The schema for this data load should be created prior to running `graphloader`

```
// PROPERTY KEYS
schema.propertyKey('badge').Text().multiple().create()
schema.propertyKey('gender').Text().single().create()
schema.propertyKey('name').Text().single().create()
schema.propertyKey('since').Int().single().create()

// VERTEX LABELS
schema.vertexLabel('reviewer').properties('name', 'gender',
'badge').create()
schema.propertyKey('badge').properties('since').add()

// INDEXES
schema.vertexLabel('reviewer').index('byname').materialized().by('name').add()
```

1. The mapping script uses `vertexProperty` to identify `badge` as a vertex property.

Note the structure of the nested fields for `badge` in the JSON file.

```
/* SAMPLE INPUT
reviewer: { "name": "Jane Doe", "gender": "F",
 "badge" : [{ "value": "Gold Badge", "since" : 2012 },
 { "value": "Silver Badge", "since" : 2005 }] }
 */

// CONFIGURATION
// Configures the data loader to create the schema
config dryrun: false, preparation: true, create_schema:
false, load_new: true, load_vertex_threads: 3, schema_output:
'loader_output.txt'

// DATA INPUT
// Define the data input source (a file which can be specified
via command line arguments)
// inputfiledir is the directory for the input files
inputfiledir = '/tmp/'
reviewerInput = File.json(inputfiledir +
"reviewerMultiMeta.json")

//Specifies what data source to load using which mapper (as
defined inline)

load(reviewerInput).asVertices{
 label "reviewer"
 key "name"
 vertexProperty "badge", {
 value "value"
 }
}
```

Optionally, the data can be loaded from a CSV file if a transform is used before loading:

```

/* SAMPLE INPUT
name|gender|value|since
Jane Doe|F|Gold Badge|2011
Jane Doe|F|Silver Badge|2005
Jon Doe|M|Gold Badge|2012
 */

// CONFIGURATION
// Configures the data loader to create the schema
config dryrun: false, preparation: true, create_schema:
 false, load_new: true, load_vertex_threads: 3, schema_output:
 'loader_output.txt'

// DATA INPUT
// Define the data input source (a file which can be specified
// via command line arguments)
// inputfiledir is the directory for the input files
inputfiledir = '/tmp/'
reviewerInput = File.csv(inputfiledir +
 "reviewerMultiMeta.csv").delimiter('|')

//Specifies what data source to load using which mapper (as
defined inline)
reviewerInput = reviewerInput.transform {
 badge1 = [
 "value": it.remove("value"),
 "since": it.remove("since")]
 it["badge"] = [badge1]
 it
}

load(reviewerInput).asVertices{
 label "reviewer"
 key "name"
 vertexProperty "badge", {
 value "value"
 }
}

```

Running this mapping script using the sample data results in a `reviewer` vertex where the property `badge` has multiple values.

The screenshot shows the DataStax Enterprise Gremlin interface. At the top, there's a navigation bar with 'Gremlin' and other options like 'Real-time', 'refresh', and '...', followed by a search bar with 'g.V()' and a set of circular icons. Below the search bar is a table with the following data:

index ↑	id	label	badge	gender	name
0	reviewer:312263936:0	reviewer	Gold Badge, Silver Badge	M	Jane Doe
1	reviewer:887358848:0	reviewer	Gold Badge	M	Jon Doe

Choosing the pop-up link for `badge` reveals the meta-property values:

The screenshot shows the same Gremlin interface as above, but with a modal window open over the table. The modal has a title 'VERTEXPROPERTY DETAILS' with a close button 'X'. It contains a table with two rows:

badge	since
Gold Badge	2012
Silver Badge	2005

At the bottom right of the modal is a blue 'Close' button.

## Mapping geospatial and Cartesian data

Mapping geospatial and Cartesian data with DSE Graph Loader.

Geospatial and Cartesian data can be loaded with DSE Graph Loader. The DSE Graph Loader is not capable of creating schema for [geospatial \(page 445\)](#) and [Cartesian](#)

([page 446](#)) data, so schema must be created before loading and the `create_schema` configuration must be set to `false`.

An example of geospatial schema for the example:

```
//SCHEMA
schema.propertyKey('name').Text().create()
schema.propertyKey('point').Point().withGeoBounds().create()
schema.vertexLabel('location').properties('name','point').create()
schema.propertyKey('line').Linestring().withGeoBounds().create()
schema.vertexLabel('lineLocation').properties('name','line').create()
schema.propertyKey('polygon').Polygon().withGeoBounds().create()
schema.vertexLabel('polyLocation').properties('name','polygon').create()

schema.vertexLabel('location').index('byname').materialized().by('name').add()
schema.vertexLabel('lineLocation').index('byname').materialized().by('name').add()
schema.vertexLabel('polyLocation').index('byname').materialized().by('name').add()
schema.vertexLabel('location').index('search').search().by('point').add()
schema.vertexLabel('lineLocation').index('search').search().by('line').add()
schema.vertexLabel('polyLocation').index('search').search().by('polygon').add()
```

[Search indexes \(page 461\)](#) must be used for geospatial and Cartesian points, linestrings or polygons in graph queries. DSE Graph uses one index per query, and because geospatial data consists of latitude and longitude (two parameters), only search indexes can be used to optimize query performance.

1. If desired, [add configuration \(page 472\)](#) to the mapping script.
2. Specify the data input directory. The variable `inputfiledir` specifies the directory for the input files. Each of the identified files will be used for loading.

```
/* SAMPLE DATA
name|point
New York|POINT(74.0059 40.7128)
Paris|POINT(2.3522 48.8566)
*/

// DATA INPUT
// Define the data input source (a file which can be specified
// via command line arguments)
// inputfiledir is the directory for the input files

inputfiledir = '/tmp/geo_dgl/data/'
ptsInput = File.csv(inputfiledir + "vertices/
place.csv").delimiter('|')
linesInput = File.csv(inputfiledir + "vertices/
place_lines.csv").delimiter('|')
polysInput = File.csv(inputfiledir + "vertices/
place_polys.csv").delimiter('|')

//Specifies what data source to load using which mapper (as
defined inline)

load(ptsInput).asVertices {
 label "location"
```

```

 key "name"
 }

import com.datastax.driver.dse.geometry.Point
ptsInput = ptsInput.transform {
 it['point'] = Point.fromWellKnownText(it['point']);
 return it;
}

load(linesInput).asVertices {
 label "lineLocation"
 key "name"
}
import com.datastax.driver.dse.geometry.LineString
linesInput = linesInput.transform {
 it['line'] = LineString.fromWellKnownText(it['line']);
 return it;
}

load(polysInput).asVertices {
 label "polyLocation"
 key "name"
}

import com.datastax.driver.dse.geometry.Polygon
polysInput = polysInput.transform {
 it['polygon'] = Polygon.fromWellKnownText(it['polygon']);
 return it;
}

```

A transformation of the input data is required, converting the point from the WKT format into the format DSE Graph stores. For a point, the transformation imports a Point library and uses the `fromWellKnownText` method:

```

import com.datastax.driver.dse.geometry.Point
ptsInput = ptsInput.transform {
 it['point'] = Point.fromWellKnownText(it['point']);
 return it;
}

```

Linestrings and polygons use the same library and method, respectively.

3. To run DSE Graph Loader for CSV loading from a directory, use the following command:

```
graphloader geoMap.groovy -graph testGeo -address localhost
```

## Mapping Gryo data generated from DSE Graph

Inserting Gryo binary data requires a slightly modified map script. To load Gryo data, allow DSE Graph Loader to create schema and load new data. Loading will require a graph `schema_mode` set to Development.

1. Create a map script for DSE Graph generated Gryo input:

```
inputfiledir = '/tmp/Gryo/'
recipeInput =
 com.datastax.dsgraphloader.api.Graph.file(inputfiledir +
 'recipesDSEG.gryo').gryo().dse()
load(recipeInput.vertices()).asVertices {
 labelField '~label'
 key 'name'
}

load(recipeInput.edges()).asEdges {
 labelField '~label'
 outV 'outV', {
 labelField '~label'
 key 'name' : 'name', 'personId' : 'personId'
 }
 inV 'inV', {
 labelField '~label'
 key 'name' : 'name', 'bookId' : 'bookId'
 }
}
```

The Gryo data format will include `~label` and `name` field values that must be used to create the vertices. For instance, a record that is an author will have a `~label` of `person` and property `name`. For the edges, notice that a user-defined vertex ID consisting of both `name` and `bookId` is used to identify the vertex to use as the incoming vertex for the edge.

## Mapping Gryo data generated with TinkerGraph

Inserting Gryo binary data requires a slightly modified map script. To load Gryo data, allow DSE Graph Loader to create schema and load new data. Loading will require a graph `schema_mode` set to Development.

1. Create a map script for TinkerGraph generated Gryo input:

```
//Specifies what data source to load using which mapper (as
defined inline)

load(recipeInput.vertices()).asVertices {
 labelField "~label"
 key "~id", "id"
}

load(recipeInput.edges()).asEdges {
 labelField "~label"
 outV "outV", {
 labelField "~label"
```

```

 key "~id", "id"
 }
 inV "inV", {
 labelField "~label"
 key "~id", "id"
 }
}

```

The Gryo data format will include `~label` and `name` field values that must be used to create the vertices and edges. For instance, a record that is an author will have a `~label` of `author` and property `name`. The `vertexKeyMap` creates a map of each vertex label to a unique property. This map is used to create unique keys used while loading vertices from the binary file.

## Mapping GraphML binary data

Mapping GraphML data with DSE Graph Loader.

Inserting GraphML binary data requires a slightly modified map script. To load GraphML data, allow DSE Graph Loader to create schema and load new data. Loading will require a graph `schema_mode` set to Development.

1. Create a map script for GraphML data:

```

//Specifies what data source to load using which mapper (as
defined inline)

load(recipeInput.vertices()).asVertices {
 labelField "~label"
 key "~id", "id"
}

load(recipeInput.edges()).asEdges {
 labelField "~label"
 outV "outV", {
 labelField "~label"
 key "~id", "id"
 }
 inV "inV", {
 labelField "~label"
 key "~id", "id"
 }
}

```

The GraphML data format will include `~label` and `~id` field values that must be used to create the label and key for each record loaded. For instance, a record that is an author will have a `~label` of `author`. The `~id` will similarly be set in the record, a difference from other data. The difference can be seen by looking at a record and noting the presence of the `id` field, based on the second item in each `key` setting in the mapping script:

```

g.V().hasLabel('author').valueMap()
{gender=[F], name=[Julia Child], id=[0]}

```

```
{gender=[F], name=[Simone Beck], id=[3]}
```

## Mapping GraphSON binary data

Mapping GraphSON data with DSE Graph Loader.

Inserting GraphSON data requires a slightly modified map script. To load GraphSON data, allow DSE Graph Loader to create schema and load new data. Loading will require a graph [schema\\_modeset](#) to Development.

1. Create a map script for GraphML data:

```
//Specifies what data source to load using which mapper (as
defined inline)

load(recipeInput.vertices()).asVertices {
 labelField "~label"
 key "~id", "id"
}

load(recipeInput.edges()).asEdges {
 labelField "~label"
 outV "outV", {
 labelField "~label"
 key "~id", "id"
 }
 inV "inV", {
 labelField "~label"
 key "~id", "id"
 }
}
```

The GraphSON data format will include `~label` and `~id` field values that must be used to create the label and key for each record loaded. For instance, a record that is an author will have a `~label` of `author`. The `~id` will similarly be set in the record, a difference from other data. The difference can be seen by looking at a record and noting the presence of the `id` field, based on the second item in each `key` setting in the mapping script:

```
g.V().hasLabel('author').valueMap()
{gender=[F], name=[Julia Child], id=[0]}
{gender=[F], name=[Simone Beck], id=[3]}
```

## Using transforms (filter, flatMap, and map) with DSE Graph Loader

How to use transforms (filter, flatMap, and map) with DSE Graph Loader

All data inputs support arbitrary user transformations to manipulate or truncate the input data according to a user provided function. The available transforms for DSE Graph Loader are:

- [filter \(page 528\)](#)
- [flatMap \(page 531\)](#)

- [map \(page 533\)](#)

**Notice:** As of DSE Graph Loader 6.0, transformation functions may be deprecated; be aware that changes may occur.

The data record for each data input is a document structure or nested map defined from an input file. A transformation acts upon the nested map and returns a nested map. Any provided transformation function must be thread-safe or the behavior of the data loader becomes undefined.

The transforms used are Groovy closures, or open anonymous blocks of code that can take arguments, return values and be assigned for a variable. These closures often make use of a Groovy implicit parameter, `it`. When a closure does not explicitly define a parameter list, `it` is always a defined parameter that can be used. In the following examples, `it` is used to get each record in an input file and apply the transformation.

The placement of the transform in the mapping script is arbitrary; as long as the input file is defined before the transform is defined, a transform may be placed anywhere in the mapping script.

Here's a simple [introduction to Groovy](#) for those unfamiliar with it.

## filter

How to use filter with DSE Graph Loader

The `filter` function can apply criteria to the input file, selecting only the objects that meet the criteria and loading them. The criteria can match any data type used in a field.

### Filter based on inequality operation on integer

The defined input file in this example is `chefs`. The filter is applied to the input file using the syntax `<input_file_name>.filter { ... }`. Given an integer field for `age`, all chefs 41 years old and younger can be filtered, and loaded into the graph with vertex label `chefYoung`:

```
/** SAMPLE INPUT
name|gender|status|age
Jamie Oliver|M|alive|41
**/

inputfiledir = '/tmp/filter_map_flatmap/'
chefs = File.csv(inputfiledir + "filterData.csv").delimiter(' | ')

// filter
def chefsYoung = chefs.filter { it["age"].toInteger() <= 41 }

//Specifies what data source to load using which mapper (as defined
//inline)

load(chefsYoung).asVertices {
 label "chefYoung"
 key "name"
}
```

The value for `age` is converted to an Integer for the function operation, and compared to the value of 41.

Only the records that match the criteria will create vertices, as reflected in the resulting values:

```
g.V().hasLabel('chefYoung').valueMap()
==>{gender=[M], name=[Jamie Oliver], age=[41], status=[alive]}
==>{gender=[F], name=[Amanda Cohen], age=[35], status=[alive]}
==>{gender=[M], name=[Patrick Connolly], age=[31], status=[alive]}
```

## Filter based on equality match operation on string

Another example of two filters finds all the chefs who are alive and who are deceased:

```
/** SAMPLE INPUT
name|gender|status|age
Jamie Oliver|M|alive|41
**/

inputfiledir = '/tmp/filter_map_flatmap/'
chefs = File.csv(inputfiledir + "filterData.csv").delimiter(' | ')
def chefsAlive = chefs.filter { it["status"] == "alive" }
def chefsDeceased = chefs.filter { it["status"] == "deceased" }

load(chefsAlive).asVertices {
 label "chefAlive"
 key "name"
}

load(chefsDeceased).asVertices {
 label "chefDeceased"
 key "name"
}
```

The filter checks the value of the string `status` and creates two new inputs, `chefsAlive` and `chefsDeceased` to use for loading the vertices, with the respective vertex labels `chefAlive` and `chefDeceased`.

The resulting vertices are:

```
// List all the living chefs
g.V().hasLabel('chefAlive').valueMap()
==>{gender=[F], name=[Alice Waters], age=[73], status=[alive]}
==>{gender=[F], name=[Patricia Curtan], age=[66], status=[alive]}
==>{gender=[F], name=[Kelsie Kerr], age=[57], status=[alive]}
==>{gender=[M], name=[Fritz Streiff], age=[500], status=[alive]}
==>{gender=[M], name=[Emeril Lagasse], age=[57], status=[alive]}
==>{gender=[M], name=[Jamie Oliver], age=[41], status=[alive]}
==>{gender=[F], name=[Amanda Cohen], age=[35], status=[alive]}
==>{gender=[M], name=[Patrick Connolly], age=[31], status=[alive]}

// List all the deceased chefs
g.V().hasLabel('chefDeceased').valueMap()
```

```
==>{gender=[F], name=[Julia Child], age=[500], status=[deceased]}
==>{gender=[F], name=[Simone Beck], age=[500], status=[deceased]}
==>{gender=[F], name=[Louisette Bertholie], age=[500],
status=[deceased]}
==>{gender=[F], name=[Patricia Simon], age=[500], status=[deceased]}
==>{gender=[M], name=[James Beard], age=[500], status=[deceased]}
```

## Full filter data set

The full sample data set used in this example:

name	gender	status	age
Julia Child	F	deceased	500
Simone Beck	F	deceased	500
Louisette Bertholie	F	deceased	500
Patricia Simon	F	deceased	500
Alice Waters	F	alive	73
Patricia Curtan	F	alive	66
Kelsie Kerr	F	alive	57
Fritz Streiff	M	alive	500
Emeril Lagasse	M	alive	57
James Beard	M	deceased	500
Jamie Oliver	M	alive	41
Amanda Cohen	F	alive	35
Patrick Connolly	M	alive	31

Note the use of 500 as a placeholder for the age of deceased chefs.

## Full filter mapping script

The full map script with all three filters:

```
/** SAMPLE INPUT
name|gender|status|age
Jamie Oliver|M|alive|41
**/

// SCHEMA
schema.propertyKey('name').Text().ifNotExists().create()
schema.propertyKey('gender').Text().ifNotExists().create()
schema.propertyKey('status').Text().ifNotExists().create()
schema.propertyKey('age').Int().ifNotExists().create()

schema.vertexLabel('chefAlive').properties('name','gender','status','age').create()
schema.vertexLabel('chefAlive').index('byname').materialized().by('name').add()
schema.vertexLabel('chefDeceased').properties('name','gender','status','age').create()
schema.vertexLabel('chefDeceased').index('byname').materialized().by('name').add()
schema.vertexLabel('chefYoung').properties('name','gender','status','age').create()
schema.vertexLabel('chefYoung').index('byname').materialized().by('name').add()

// CONFIGURATION
// Configures the data loader to create the schema
config create_schema: false, load_new: true

// DATA INPUT
```

```

// Define the data input source (a file which can be specified via
// command line arguments)
// inputfiledir is the directory for the input files that is given in
// the commandline
// as the "-filename" option

inputfiledir = '/tmp/filter_map_flatmap/'
chefs = File.csv(inputfiledir + "filterData.csv").delimiter('|')
def chefsYoung = chefs.filter { it["age"].toInteger() <= 41 }
def chefsAlive = chefs.filter { it["status"] == "alive" }
def chefsDeceased = chefs.filter { it["status"] == "deceased" }

//Specifies what data source to load using which mapper (as defined
//inline)

load(chefsYoung).asVertices {
 label "chefYoung"
 key "name"
}

load(chefsAlive).asVertices {
 label "chefAlive"
 key "name"
}

load(chefsDeceased).asVertices {
 label "chefDeceased"
 key "name"
}

```

## flatMap

### How to use flatMap with DSE Graph Loader

The `flatMap` function (also called `expand`) can break a single field in the input file into separate objects before loading them. In general, this function is used to convert more compacted data into an expanded form.

### FlatMap based on multiple cuisine values for a recipe

The input file for this example is `recipes`. The `flatMap` is applied to the input file using the syntax `<input_file_name>.flatMap { ... }`. Given a field for `cuisine` that identifies all the possible `cuisine` choices for a recipe, a record for each vertex can be created using the recipe name and the cuisine type as a separate vertex when loading the vertices into the graph:

```

/** SAMPLE INPUT
name|cuisine
Beef Bourguignon|English::French
**/

inputfiledir = '/tmp/filter_map_flatmap/'
recipes = File.csv(inputfiledir + "flatmapData.csv").delimiter('|')

```

```

def recipesCuisine = recipes.flatMap {
 def name = it["name"];
 it["cuisine"].
 split(":").
 collect {
 it = ['name': name, 'cuisine': it]
 }
}
//Specifies what data source to load using which mapper (as defined
//inline)

load(recipesCuisine).asVertices {
 label "recipe"
 key name: "name", cuisine: "cuisine"
}

```

The `flatMap` function gets each record, retrieves the recipe name, splits the cuisine field, and then collects each name/cuisine pair to use as the composite key for identifying each separate vertex. The Groovy `split` method splits a string (*cuisine*) using the supplied delimiter (:) and returns an array of strings (each cuisine). The Groovy `collect` method iterates over a collection and transforms each element of the collection.

The result of the loading reflects all the possible vertices based on cuisine:

```

g.V().valueMap()
==>{name=[Beef Bourguignon], cuisine=[English]}
==>{name=[Beef Bourguignon], cuisine=[French]}
==>{name=[Nicoise Salade], cuisine=[French]}
==>{name=[Wild Mushroom Stroganoff], cuisine=[American]}
==>{name=[Wild Mushroom Stroganoff], cuisine=[English]}

```

## Full flatMap data set

The full sample data set used in this example:

name		cuisine
Beef Bourguignon		English::French
Nicoise Salade		French
Wild Mushroom Stroganoff		American::English

## Full flatMap mapping script

The full map script with `flatMap`:

```

/** SAMPLE INPUT
name|cuisine
Beef Bourguignon|English::French
**/

// SCHEMA
schema.propertyKey('name').Text().ifNotExists().create()
schema.propertyKey('cuisine').Text().ifNotExists().create()

```

```

schema.vertexLabel('recipe').properties('name','cuisine').create()
schema.vertexLabel('recipe').index('byname').materialized().by('name').add()

// CONFIGURATION
// Configures the data loader to create the schema
config create_schema: false, load_new: true

// DATA INPUT
// Define the data input source (a file which can be specified via
// command line arguments)
// inputfiledir is the directory for the input files that is given in
// the commandline
// as the "-filename" option

inputfiledir = '/tmp/filter_map_flatmap/'
recipes = File.csv(inputfiledir + "flatmapData.csv").delimiter('|')

def recipesCuisine = recipes.flatMap {
 def name = it["name"];
 it["cuisine"].
 split("::").
 collect {
 it = ['name': name, 'cuisine': it]
 }
}
//Specifies what data source to load using which mapper (as defined
//inline)

load(recipesCuisine).asVertices {
 label "recipe"
 key name: "name", cuisine: "cuisine"
}

```

## map

How to use map with DSE Graph Loader

The `map()` (also called `transform()`) applies a function to a field's values before loading the data.

### map converts gender field from to lower case from any case

The input file for this example is `authorInput`. The map is applied to the input file using the syntax `<input_file_name>.map { ... }`. Given a field `gender`, the `Groovy toLowerCase()` method is performed on each `genderValue` in the nested map `authorInput`:

```

inputfiledir = '/tmp/TEXT/'
authorInput = File.text(inputfiledir + "author.dat").
 delimiter("|").
 header('name', 'gender')

```

```
authorInput = authorInput.map { it['gender'] =
 it['gender'].toLowerCase(); it }
```

This `map()` transformation ensures that the `gender` values in the graph are only lowercase.

The result of the loading reflects the change to the case of `gender`:

```
g.V().valueMap()
==>{gender=[f], name=[Julia Child], age=[500]}
==>{gender=[f], name=[Simone Beck], age=[500]}
==>{gender=[f], name=[Louisette Bertholie], age=[500]}
==>{gender=[f], name=[Patricia Simon], age=[500]}
==>{gender=[f], name=[Alice Waters], age=[73]}
==>{gender=[f], name=[Patricia Curtan], age=[66]}
==>{gender=[f], name=[Kelsie Kerr], age=[57]}
==>{gender=[m], name=[Fritz Streiff], age=[500]}
==>{gender=[m], name=[Emeril Lagasse], age=[57]}
==>{gender=[m], name=[James Beard], age=[500]}
==>{gender=[m], name=[Jamie Oliver], age=[41]}
==>{gender=[f], name=[Amanda Cohen], age=[35]}
==>{gender=[m], name=[Patrick Connolly], age=[31]}
```

## Full map data set

The full sample data set used in this example:

name	gender	age
Julia Child	F	500
Simone Beck	F	500
Louisette Bertholie	F	500
Patricia Simon	F	500
Alice Waters	F	73
Patricia Curtan	F	66
Kelsie Kerr	F	57
Fritz Streiff	M	500
Emeril Lagasse	M	57
James Beard	M	500
Jamie Oliver	M	41
Amanda Cohen	F	35
Patrick Connolly	M	31

## Full map mapping script

The full map script with `map`:

```
/** SAMPLE INPUT
name|gender|age
Jamie Oliver|M|41
**/

// SCHEMA
schema.propertyKey('name').Text().ifNotExists().create()
schema.propertyKey('gender').Text().ifNotExists().create()
schema.propertyKey('age').Int().ifNotExists().create()
```

```

schema.vertexLabel('chef').properties('name','gender','age').create()
schema.vertexLabel('chef').index('byname').materialized().by('name').add()

// CONFIGURATION
// Configures the data loader to create the schema
config create_schema: false, load_new: true

// DATA INPUT
// Define the data input source (a file which can be specified via
// command line arguments)
// inputfiledir is the directory for the input files that is given in
// the commandline
// as the "-filename" option

inputfiledir = '/tmp/filter_map_flatmap/'
chefs = File.csv(inputfiledir + "mapData.csv").delimiter('|')
chefInput = chefs.map { it['gender'] = it['gender'].toLowerCase();
it }

//Specifies what data source to load using which mapper (as defined
//inline)

load(chefInput).asVertices {
 label "chef"
 key "name"
}

```

## DSE Graph Loader reference

DSE Graph Loader reference.

### Synopsis

```
graphloader loadingScript [[-option value]...]
```

**Table 10: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>Italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.

Syntax conventions	Description
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ <i>key:value</i> }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
< <i>datatype1,datatype2</i> >	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
<i>cql_statement</i> ;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' < <i>schema</i> > ... </ <i>schema</i> > '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.
@ <i>xml_entity</i> =' <i>xml_entity_type</i> '	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

Options can be invoked in the command line or included in the loading script. Required options are marked.

Option	Data type	Default	Description
-abort_on_num_failures	Integer	100	Number of failures after which loading is aborted.
-abort_on_prep_errors	Boolean	true	Normally if errors occur in the preparation, or during the vertex insertion phase we abort, setting this to false will force the loader to continue up to the maximum number of allowed failures.
-address	String		The IP address (and port) of the DSE Graph instance to connect to. REQUIRED

Option	Data type	Default	Description
-allow_remote_hosts_in_quorum	Boolean	false	Allows hosts in a different datacenter to participate in a local consistency level, so that a node from a remote datacenter can be used to reach a consistency level of QUORUM, for instance, for a query. Choices are: true, false.
-batch-size	Integer	100	Size of loading batches.
-compress	String	none	The compression of the file. Choices are none, gzip, and xzip.
-consistency_level	CL	ONE	Choices are: ANY, ONE, TWO, THREE, QUORUM, ALL, LOCAL_QUORUM, EACH_QUORUM, SERIAL, LOCAL_SERIAL, LOCAL_ONE.
-create_graph	Boolean	true	Check if the target graph exists, and if it doesn't, creates it if true. Note that this option can fail on the default consistency level of QUORUM if a datacenter is unreachable.

Option	Data type	Default	Description
-create_schema	Boolean	true	<p>Whether to update or create the schema for missing schema elements.</p> <p><b>Notice:</b> It is strongly recommended that <a href="#">schema is created (page 436)</a> prior to data loading, so that the correct data types are enforced and indexes created.</p> <p>Setting <code>create_schema</code> to true is recommended only for testing. In DSE 6.0, this configuration option is deprecated and will be removed in a future release.</p>
-driver_retry_attempts	Integer	3	Number of retry attempts. If greater than zero, requests will be resubmitted after some recoverable failures.
-driver_retry_delay	milliseconds	1000	Number of milliseconds between driver retries.

Option	Data type	Default	Description
-dryrun	Boolean	false	<p>Whether to only conduct a trial run to verify data integrity and schema consistency. Does not create a graph if it doesn't exist.</p> <p><b>Notice:</b> This configuration option discovers schema and suggests missing schema without executing any changes. In DSE 6.0, this option is deprecated and may possibly be removed in a future release.</p>
-filename	String		The file to load the vertex data from. REQUIRED if not defined in the mapping script.
-graph	String		The name of the graph to load into. REQUIRED
-label	String		The label of the vertex to be populated with data. If left blank, the name of the input file is used as the vertex label name.
-load_failure_log	String	load_failures.txt	Name and location of the file where failed records will be stored.

Option	Data type	Default	Description
-load_new	Boolean	false	Whether the vertices loaded are new and do not yet exist in the graph.
-load_edge_threads	Integer	0	Number of threads to use for loading edge and property data into the graph (0 will force the value to be the number of nodes in the DC * 6).
-load_vertex_threads	Integer	0	Number of threads to use for loading vertices into the graph (0 will force the value to the number of cores/2).
-preparation	Boolean	true	Whether to do a preparation run to analyze the data and update the schema, if necessary.  <b>Notice:</b> This configuration option validates and creates schema if used in conjunction with <code>create_schema</code> . The default will be set to <code>false</code> , and this option is deprecated with DSE 6.0. In a future release, it may be removed.

Option	Data type	Default	Description
-preparation_limit	Intger	0	The number of records that the preparation phase will use to attempt to determine if the schema should be updated. Zero indicates no limit.
-queue-size	Integer	10000	Data retrieval queue size.
-read_threads	Integer	1	Number of threads to use for reading data from data input.
-remote_hosts_in_dc	Integer	2	Number of remote nodes that can participate in the consistency level for a query.
-reporting_interval	Integer	1	Number of seconds between each progress report written to the log.
-schema_output	String	proposed_schema.txt	The name of the file to save the proposed schema in when executing a dry-run. Leave blank to disable.
-skip_blank_values	Boolean	true	When false, loader will insert a blank ("") for all unspecified (empty/blank) property values in a CSV file.
-timeout	Integer	120000	Number of milliseconds until a connection times out.
-v   --version	N/A	N/A	Print the version of DSE Graph Loader.

Option	Data type	Default	Description
-vertex_complete	Boolean	false	The loader assumes that all vertexes referenced by properties and edges in this load are also included as vertexes of this load. No new vertices will be created from edge data or property data files.
-username	String		Username for DSE authentication.
-password	String		Password for DSE authentication.
-ssl	Boolean	false	Enable SSL.
-kerberos	Boolean	false	Enable kerberos.
-sasl	String		An optional sasl protocol name used in conjunction with kerberos.

Security options:

Option	Data type	Default	Description
-kerberos	Boolean	false	Enable kerberos.
-password	String		Password for DSE authentication.
-sasl	String		An optional sasl protocol name used in conjunction with kerberos.
-ssl	Boolean	false	Enable SSL.
-username	String		Username for DSE authentication.

## Description

DSE Graph Loader is an utility for loading up to 100 million vertices and 1 billion edges. The utility runs on a sufficiently powerful computer that can cache all vertices in memory and includes enough cores to parallelize the loading process. For larger loads, the utility must be run on a different machine.

DSE Graph Loader is invoked on the command line with a loading script as argument and a variable number of configuration option-value pairs. The loading script specifies what input data is being loaded and how that data maps onto the graph. The loading script can also configure the option-value pairs.

The three stages of load processing are:

### Preparation

Reads entire input data. This stage either ensures that the data conforms to the graph schema, or the stage updates the graph schema according to the provided data (if enabled). At the end of this stage, statistical estimates are provided on how much data will be added to the graph but no data is loaded.  
Set

```
-dryrun true
```

to abort the loading process after the preparation stage and before any changes are made. Inspect the output and verify that it matches your expectations. For large datasets, doing a dry run is important for spotting errors.

### Vertex Loading

The second stage adds or retrieves all of the vertices in the input data and caches them locally to speed up the subsequent edge loading.

### Edge and Property Loading

Adds all edges and properties from the input data to the graph.

A loading, or mapping, script is required to specify the particular mapping used to load the data from the input file to the graph. DSE Graph Loader supports four file-based data input types: CSV, JSON, delimited text, and text parsed by regular expressions. All file-based input formats support compression of the input data files.

Logging during the loading process can provide useful information if troubleshooting is required. The three stages of load processing are detailed in the log.

## Examples

To get the listing of possible options, use `-help`.

```
graphloader -help
```

This example will use the loading script `mymapscript.groovy` to read data from a file `/tmp/recipe/all.dat` into the graph `test` that is running on the localhost. Dry run is specified to test the loading without inserting the data.

```
graphloader mymapscript.groovy -filename /tmp/recipe/all.dat -graph
test -address localhost -dryrun true
```

This example will use the loading script `csv2Vertex.groovy` to read data from a file `MyUsers.csv` into the graph `csvTest` that is running on the localhost. The `-label` option specifies that the vertex label will be `User`, rather than the filename `MyUsers`.

```
graphloader ./scripts/csv2Vertex.groovy -filename MyUsers.csv -graph csvTest -label User -address 127.0.0.1
```

The configuration settings can also be specified in the loading script. A fragment of a loading script is shown here that sets `create_schema` to true and `load_vertex_threads` to 3.

```
// CONFIGURATION
// Configures the data loader to create the schema and set
// load_vertex_threads to 3
config load_new: true, load_vertex_threads: 3
```

By default, the `graphloader` logs debug information to the file `loader.log` in the directory from which `graphloader` is run. The location of the log can be specified with `-load_failure_log`:

```
graphloader mymapscript.groovy -graph test -address localhost -
load_failure_log /tmp/dgl.log
```

If `log4j` modifications are desired to log information differently, a configuration file can be created, and used in conjunction with the `-load_failure_log`. Here is a sample configuration file:

```
Set root logger level to the designated level and its appenders to
F1 and stdout
log4j.rootLogger=INFO, WARN, A1, stdout
#/dev/stdout
Log INFO messages to A1. A1 is set to be a ConsoleAppender.
log4j.appender.A1=org.apache.log4j.ConsoleAppender
log4j.appender.A1.Target=System.out
log4j.appender.A1.Threshold=INFO
A1 uses PatternLayout.
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss}
%5p %c{1}:%L - %m%n
Direct INFO log messages to stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.Threshold=INFO
stdout uses PatternLayout.
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss}
%5p %c{1}:%L - %m%n
```

and a sample `graphloader` command:

```
java -Dlog4j.configuration=file:./lib/log4j.properties -jar
graphLoaderJar mymapscript.groovy -graph test -address localhost -
load_failure_log /dev/stdout
```

that will write the log information to `stdout`.

The preparation stage has additional options. To use the input data to discover the schema, use `-preparation true`. If preparation discovers missing elements in the schema, those elements can be added if `-create_schema true`. If desired, preparation can be performed, but schema creation must be manually created if `-create_schema false`. Setting `-create_schema true` without `-preparation true` will result in a stopped job. Without sampling the data to discover the schema that the data describes, `graphloader` cannot create schema because the manner of the schema is unknown. To summarize, if you wish to create schema manually, use `-preparation true -create_schema false`. If you wish `graphloader` to automatically create schema, use `-preparation true -create_schema true`.

To use authentication, configure `graphloader` with `-user` and `-password`:

```
graphloader mymapscript.groovy -graph test -address localhost -
username myName -password myPasswd
```

To configure `graphloader` with SSL encryption and using Kerberos:

```
java -Djavax.net.ssl.trustStore=<TRUSTSTORE_PATH>
-Djavax.net.ssl.trustStorePassword=<PASSWORD> -
Djavax.net.ssl.keyStore=<KEYSTORE_PATH> \
-Djavax.net.ssl.keyStorePassword=<PASSWORD> -jar dse-graph-
loader-5.0.3-uberjar.jar -kerberos true -sasl dsename -graph new -
address localhost mymapscript.groovy
```

If the truststore and keystore java options are set in `cassandra-env.sh`, the command is simplified:

```
java -jar dse-graph-loader.jar -kerberos true -sasl dsename -graph new
-address localhost mymapscript.groovy
```

## Runtime parameters

Some modifications are necessary if certain conditions must be set. For instance, the JAR file can be run directly to use Java modifiers, or the `graphloader` script may be modified to allow additional parameters to be set.

If a large data set is loaded, configure the heap space to cache all vertices. This command runs Java and calls the jar file for DSE Graph Loader. For example:

```
java -Xmx10g -jar dse-graph-loader.jar
```

Vertex caching uses a temporary directory to store data during loading. If the temporary directory is not large enough, loading is blocked. To change the location of the temporary directory, use a runtime variable `LOADER_TMP_DIR`:

```
LOADER_TMP_DIR=/home/user ./graphloader -graph new -address localhost
mymapscript.groovy
```

## Successful loading

When graphloader has successfully loaded the data specified, notification of the results are logged to `/var/lib/cassandra/system.log`:

```
2017-02-09 23:27:22 INFO Reporter:97 - Current total additions:
1155735 vertices 1982536 edges 6583940 properties 0 anonymous
```

## Tuning graphloader JVM options

How to tune graphloader JVM options.

The DSE Graph Loader is written in Java and has some configurable JVM tuning in the `graphloader` script.

The default maximum heap size is 10G, generally a good heap size for appropriately sized machine used with `graphloader`. Two environment variables, `MAX_HEAP_SIZE` and `HEAP_NEWSIZE` were added in DSE 5.0.5 and later. `graphloader` now calculates the values of these two environment variables in the [same manner as the DSE database](#). If a particular value is desired for either variable, the value can be set directly in the `graphloader` script.

## graphloader API

Reference guide to graphloader mapping options.

`graphloader` mapping options are used to designate the manner in which a data file will be parsed for loading.

### exists()

How to identify that vertices or edges already exist when loading data from a data file.

#### Synopsis

```
exists()
```

#### Description

When loading edges, often the specified vertices for incoming or outgoing endpoints already exist in the database. The `exists()` method will identify that the vertices do not need creation when the edges are created.

The `exists()` method can also be used to specify that edges already exist.

#### Examples

Identify that the vertices for the outgoing vertices identified in the field `aname` in `outV` already exist in the database and do not need to be created:

```
load(authorBookInput).asEdges {
 label "authored"
 outV "aname", {
```

```

 label "author"
 key "name"
 exists()
 }
 inV "bname", {
 label "book"
 key "name"
 }
}

```

## ignore

How to ignore a field when loading data from a data file.

### Synopsis

```
ignore "fieldName"
```

### Description

Each record read from an input data file will insert every field included unless `ignore` is used.

### Examples

Ignore the field `gender` in the input data file:

```
ignore "gender"
```

## inE

How to designate the incoming edge to use when loading data from a data file.

### Synopsis

```

inE "edgeLabel" {
 labelField "fieldName"
 vertex "vertexLabel" {
 label "labelName"
 key "fieldName"
 }
}

```

### Description

Sets the information for an incoming edge to the given edge label and vertex. The edge label must already exist. `labelField` is optional.

### Examples

Set the incoming edge in a mapping script to `FridgeSensor`.

```

inE "authored", {
 vertex "author", {
 label "author"
 key "name"
 }
}

```

```
 }
}
```

The vertex with its [label \(page 550\)](#) and [key \(page 550\)](#) must be set along with `inE`.

## isNew()

How to identify that vertices or edges will be created when loading data from a data file.

### Synopsis

```
isNew()
```

### Description

The `isNew()` method will identify that vertices or edges need creation during the loading process. This method is used instead of the graphloader parameter `load_new` when only a portion of the loading needs identification. `load_new` requires either the entire creation of all vertices and edges during loading to be true or false.

### Examples

Identify that the edges between existing author vertices and existing book vertices will be created as new edges during the loading into the database:

```
load(authorBookInput).asEdges {
 isNew()
 label "authored"
 outV "aname", { label "author"
 key "name"
 exists()
 }
 inV "bname", {
 label "book"
 key "name"
 exists()
 }
}
```

## inV

How to designate the incoming vertex to use when loading data from a data file.

### Synopsis

```
DSE5.1.2 and earlier:
inV "field_name", {
 label "field_name"
 [key "key_name" | key key1_name: "key1_name", key2_name:
 "key2_name"]
}
```

```
DSE5.1.3 and later:
inV {
 label "field_name"
```

```
[key "key_name" | key key1_name: "key1_name", key2_name:
"key2_name"]
ignore "field_name"
}
```

## Description

In DSE versions 5.1.2 and earlier, sets the field name in the input file that will define the incoming vertex of an edge. Both `inV` and `outV` ([page 552](#)) must be defined in an edge mapping statement. In DSE 5.1.3 and later, the `field_name` is deleted from between the `inV` keyword and the `{`.

## Examples

DSE 5.1.2 and earlier: Sets the field name for the incoming vertex in a mapping script to `fridgeSensor`.

```
//Sample line read:
// cityId|sensorId|name
// santaCruz|93c4ec9b-68ff-455e-8668-1056ebc3689f|asparagus
// or JSON
// {"sensor": {"cityId": "santaCruz", "sensorId":
"93c4ec9b-68ff-455e-8668-1056ebc3689f"}, "name": "asparagus"}

// The incoming vertex has a vertex label of fridgeSensor, the
// particular vertex is defined as the one with
// the cityId of santaCruz and a sensorId of
// 93c4ec9b-68ff-455e-8668-1056ebc3689f
inV "fridgeSensor", {
 label "fridgeSensor"
 key cityId: "cityId", sensorId: "sensorId"
```

The field name in the input file that defines the outgoing vertex is `fridgeSensor`, the vertex has a vertex label of `fridgeSensor`, and the composite key value `cityId`, `sensorId` is supplied in the input file field set in this statement. The [label \(page 550\)](#) and [key \(page 550\)](#) must be set along with `inv`.

DSE5.1.3 and later:

```
//Sample line read:
// cityId|sensorId|homeId|name
// santaCruz|93c4ec9b-68ff-455e-8668-1056ebc3689f|asparagus
// or JSON
// {"sensor": {"cityId": "santaCruz", "sensorId":
"93c4ec9b-68ff-455e-8668-1056ebc3689f"}, "name": "asparagus"}

// The incoming vertex has a vertex label of fridgeSensor, the
// particular vertex is defined as the one with
// the cityId of santaCruz and a sensorId of
// 93c4ec9b-68ff-455e-8668-1056ebc3689f
inV {
 label "fridgeSensor"
 key cityId: "cityId", sensorId: "sensorId"
 exists()
```

```
 ignore "homeId"
 ignore "name"
}
```

## key

How to designate the incoming a simple unique key to use when loading data from a data file.

### Synopsis

```
key "fieldName"
```

### Description

Each record read from an input data file must be unique to avoid duplication. `key` defines a simple unique key for this element comprised of a single field and associated property key name.

### Examples

Set the key in a mapping script to `name`:

```
key "name"
```

If the data file includes unique ids, such as a GraphSON or Gryo file written from DataStax Enterprise, the `key` can be set to identify the id:

```
key "~id" "id"
```

where `~id` defines that the id is found in the data file, and `id` renames the field to `id` in the loaded file.

Set a key in a mapping script to a composite custom id:

```
key city_id: "city_id", sensor_id: "sensor_id"
```

This definition uses the following pattern:

```
key <csv_column_name1>: "vertex_property_key1", <csv_column_name2>:
"vertex_property_key2"
```

where `<csv_column_name>` is the column in the input file that specifies the value to be assigned to the `vertex_property_key` in the graph.

## label

How to designate the vertex label to use when loading data from a data file.

### Synopsis

```
label "labelName"
```

### Description

Sets the label of the vertex to the given name. The vertex label must already exist.

`label` can be used in both vertex and incident edge mapping ([inE \(page 547\)](#), [outE \(page 551\)](#)).

## Examples

Set the label in a mapping script to *recipe*.

```
label "recipe"
```

## labelField

How to designate the vertex label to use when loading data from a data file using the contents of a field in the data file.

## Synopsis

```
labelField "fieldName"
```

## Description

Sets the label of the vertex to the name associated with the given field in the input data file. The vertex label must already exist.

`labelField` can be used in both vertex and incident edge mapping ([inE \(page 547\)](#), [outE \(page 551\)](#)).

## Examples

Set the label in a mapping script to the field name *type*.

```
labelField "type"
```

The contents of the field *type* will designate the vertex label. For instance, if a record in the data file has the field *type* entered as *author*, then the record will be read into a vertex with the vertex label set to *author*. The next record might instead have a value of *recipe* for the *type* field, and the data will be read into a vertex with a vertex label set to *recipe*. Thus, mixed sets of data can be read from a single input data file.

## outE

How to designate the outgoing edge to use when loading data from a data file.

## Synopsis

```
outE "edgeLabel" {
 labelField "fieldName"
 vertex "vertexLabel" {
 label "labelName"
 key "fieldName"
 }
}
```

## Description

Sets the information for an outgoing edge to the given edge label and vertex. The edge label must already exist. `labelField` is optional.

## Examples

Set the outgoing edge in a mapping script to *ingredient*.

```
outE "authored", {
 vertex "book", {
 label "book"
 key "name"
 }
}
```

The vertex with its [label \(page 550\)](#) and [key \(page 550\)](#) must be set along with `outE`.

## outV

How to designate the outgoing vertex to use when loading data from a data file.

## Synopsis

```
DSE5.1.2 and earlier:
outV "field_name", {
 label "field_name"
 [key "key_name" | key key1_name: "key1_name", key2_name:
 "key2_name"]
 [exists()]
}
```

```
DSE5.1.3 and later:
outV {
 label "field_name"
 [key "key_name" | key key1_name: "key1_name", key2_name:
 "key2_name"]
 ignore "field_name"
 [exists()]
}
```

## Description

In DSE versions 5.1.2 and earlier, sets the field name in the input file that will define the outgoing vertex of an edge. Both `outV` and [inV \(page 548\)](#) must be defined in an edge mapping statement. In DSE 5.1.3 and later, the `field_name` is deleted from between the `outV` keyword and the `{`.

## Examples

DSE 5.1.2 and earlier: Set the field name for the outgoing vertex of an edge in a mapping script to *ingredient*.

```
//Sample line read:
```

```
// city_id|sensor_id|name
// santaCruz|93c4ec9b-68ff-455e-8668-1056ebc3689f|asparagus

// The outgoing vertex has a vertex label of ingredient, the
// particular vertex is defined as the one with
// the name of asparagus

outV "ingredient", {
 label "ingredient"
 key "name"
}
```

The field name in the input file that defines the outgoing vertex is *ingredient*, the vertex has a vertex label of *ingredient*, and the key value *name* is supplied in the input file field set in this statement. The [label \(page 550\)](#) and [key \(page 550\)](#) must be set along with `outV`.

DSE 5.1.3 and later:

```
//Sample line read:
// cityId|sensorId|homeId|name
// santaCruz|93c4ec9b-68ff-455e-8668-1056ebc3689f|asparagus
// or JSON
// {"sensor": {"cityId": "santaCruz", "sensorId":
"93c4ec9b-68ff-455e-8668-1056ebc3689f"}, "name": "asparagus"}

// The incoming vertex has a vertex label of fridgeSensor, the
// particular vertex is defined as the one with
// the cityId of santaCruz and a sensorId of
// 93c4ec9b-68ff-455e-8668-1056ebc3689f
outV {
 label "ingredient"
 key "name"
 exists()
 ignore "homeId"
 ignore "cityId"
 ignore "sensorId"
}
```

## property

How to designate a property name to use when loading data from a data file.

### Synopsis

```
property ["csv_column"] "propertyName"
```

### Description

Identifies a property of a vertex to the given field name that will be mapped onto a property from an input data file. If a CSV column is named differently than the field in the graph, the CSV column name may be optionally set. If no property is set, all properties are read. If [ignore \(page 547\)](#) is used, a property will be bypassed.

During the `graphloader` stage 1 schema discovery, if a property key sequentially occurs multiple times on the same vertex, then the property is considered to be a multi-property. This is a reasonable deduction; however, it might change the discovered schema in certain fringe cases. Typically, in JSON input data, nested lists are mapped to multi-cardinality properties and should be read in that manner. If multi-cardinal fields exist in the input file, schema must define those elements prior to loading. Another example is a JDBC column that is of data type `java.sql.Array`.

## Examples

Set `property` in a mapping script to `gender`. The values of the property will be read based on the values set in the `value` field of `badge`.

```
property "gender"
```

For this mapping, the data could be:

```
{ "name": "Jane Doe", "gender": "M" }
```

Set `property` to read a column `nick` in a CSV file to `nickname` in a mapping script.

```
property "nick" "nickname"
```

For this mapping, the data could be:

```
name, nick
Jane Doe|Janey
```

## vertex

How to designate a vertex to use when loading data from a data file.

## Synopsis

```
vertex "fieldName", {
 label "labelName"
 key "fieldName"
}
```

## Description

Each record read from an input data file will have vertex information. `vertex` defines the label and key for a vertex that will be read. If no `property` values are set, all properties in the record will be input.

## Examples

Set the parameters for a vertex in a mapping script:

```
vertex "recipe", {
 label "recipe"
 key "name"
}
```

For this vertex, each vertex created with the vertex label *recipe* will be created from a record in the input file in which the vertex label is defined as *recipe* and the key field as *name*.

## vertexProperty

How to designate the vertex property name to use when loading data from a data file.

### Synopsis

```
vertexProperty "propertyName"
```

### Description

Sets the vertex property of a vertex to the given name.

### Examples

Set `vertexProperty` in a mapping script to *badge*. The values of the vertex property will be read based on the values set in the *value* field of *badge*.

```
vertexProperty "badge" , {
 "value" "since"
}
```

For this mapping, the data could be:

```
{ "name": "Jane Doe", "gender": "M", "badge" : { "value": "Gold Badge",
"since" : 2012 } }
```

## Dropping graph data

How to drop (delete) data.

Data can be dropped as follows:

### Drop data

- To drop all data without dropping a graph and schema, drop all vertices.

```
g.V().drop().iterate()
```

- To drop specific data, such as all `person` vertices, identify the vertices along with a `drop` traversal step.

```
g.V().hasLabel('person').drop()
```

**Note:** If a very large number of vertices will be dropped with the command shown above, DSE Graph may complain. In that case, modify the `drop()` command in the following manner:

```
g.V().hasLabel('person').limit(100).drop()
```

and repeat until all vertices are dropped.

- To drop a specific value, such as person vertices, identify the vertices along with a drop traversal step.

```
g.V().hasLabel('person').properties('gender').hasValue('M').drop()
```

This query will drop the gender value for all person vertices that have a gender value of M.

```
gremlin> g.V().hasLabel('person').valueMap()
==>{gender=[F], name=[Julia Child]}
==>{gender=[F], name=[Patricia Curtan]}
==>{gender=[F], name=[Kelsie Kerr]}
==>{gender=[F], name=[Simone Beck]}
==>{gender=[F], name=[Alice Waters]}
==>{gender=[F], name=[Patricia Simon]}
==>{name=[James Beard]}
==>{name=[Fritz Streiff]}
==>{name=[Emeril Lagasse]}
```

- To drop a property key from an edge, such as reviewed edges, identify the edges, the property key stars along with a drop traversal step.

```
g.E().hasLabel('reviewed').properties('stars').drop()
```

This query will drop the property key stars for all edges that have a rated edge label.

```
g.E().hasLabel('reviewed').properties('stars').valueMap()
```

returns no values.

**Important:** When deleting schema elements, data for edge labels and meta-properties will not always be removed from the underlying database tables. If vertex labels are removed, edge data to vertices with that vertex label will still exist, but will be filtered out during any traversal. If those edge labels are added back to the graph, then the stale edge data will reappear. The same is true for meta-properties; if meta-properties are readded to property keys, then stale meta-property data will reappear. Dropping vertices with this command will also drop all edges associated with the vertices. Any vertex at the other end of an edge dropped will remain, but the edges and edge properties will be hidden from traversals.

**Warning:** For data created earlier than DSE 5.0.5, conditions exist that will drop all edges as well as the edge property during a property key drop. See [Dropping edge property drops edges](#)

## Discovering properties about graphs and traversals

Discover simple information about graphs and traversals.

After schema and data are inserted into a graph, it is important to verify that the information is correct. Checking simple information about inserted data is a good way to get started

with traversals. The `graph.schema()` calls can be used to check how the graph is storing data.

- Use the graph traversal instance `g` to check if data is loaded by checking the count of vertices. Note that the command is a remote command to Gremlin Server, as are all commands of discovery shown below.

```
g.V().count()
```

```
==>56
```

- Check the properties of a loaded vertex. Find all the information for the vertex with an name value of Julia Child.

```
g.V().has('name', 'Julia Child').valueMap()
```

```
==>{gender=[F], name=[Julia Child]}
```

- Check the properties of a loaded edge. Find all the information for the edges with a label of rated.

```
g.E().hasLabel('rated').values()
```

```
==>5
==>Pretty tasty!
==>2014-01-01T00:00:00Z
```

- Find the id information for vertices:

```
g.V().hasLabel('FridgeSensor').id()
```

```
==>{~label=FridgeSensor,
 sensor_id=93c4ec9b-68ff-455e-8668-1056ebc3689f,
 city_id=santaCruz}
==>{~label=FridgeSensor, sensor_id=9c23b683-1de2-4c97-
 a26a-277b3733732a, city_id=sacramento}
==>{~label=FridgeSensor, sensor_id=eff4a8af-2b0d-4ba9-a063-
 c170130e2d84, city_id=sacramento}
```

- Discover schema information using a `describe()` step. This traversal step provides a sorted list of the same information as the next alternative below.

```
schema.describe()
```

```
==>schema.propertyKey("member_id").Smallint().single().create()
schema.propertyKey("instructions").Text().single().create()
schema.propertyKey("amount").Text().single().create()
schema.propertyKey("gender").Text().single().create()
schema.propertyKey("year").Int().single().create()
schema.propertyKey("calories").Int().single().create()
schema.propertyKey("stars").Int().single().create()
```

```

schema.propertyKey("community_id").Int().single().create()
schema.propertyKey("ISBN").Text().single().create()
schema.propertyKey("name").Text().single().create()
schema.propertyKey("comment").Text().single().create()
schema.propertyKey("timestamp").Timestamp().single().create()
schema.edgeLabel("authored").multiple().create()
schema.edgeLabel("rated").multiple().properties("timestamp",
 "stars", "comment").create()
schema.edgeLabel("includedIn").multiple().create()
schema.edgeLabel("created").multiple().properties("year").create()
schema.edgeLabel("includes").multiple().properties("amount").create()
schema.vertexLabel("meal").properties("name", "timestamp",
 "calories").create()
schema.vertexLabel("ingredient").properties("name").create()
schema.vertexLabel("author").properties("name", "gender").create()
schema.vertexLabel("book").properties("name", "year",
 "ISBN").create()
schema.vertexLabel("recipe").properties("name",
 "instructions").create()
schema.vertexLabel("reviewer").properties("name").create()
schema.edgeLabel("authored").connection("author",
 "book").connection("book", "author").add()
schema.edgeLabel("rated").connection("recipe",
 "reviewer").connection("reviewer", "recipe").add()
schema.edgeLabel("includedIn").connection("meal",
 "recipe").connection("meal", "book").connection("book",
 "recipe").connection("book", "meal").connection("recipe",
 "book").connection("recipe", "meal").add()
schema.edgeLabel("created").connection("author",
 "recipe").connection("recipe", "author").add()
schema.edgeLabel("includes").connection("ingredient",
 "recipe").connection("recipe", "ingredient").add()
gremlin> schema.edgeLabel('includes').describe()
==>schema.edgeLabel("includes").multiple().properties("amount").create()
schema.edgeLabel("includes").connection("ingredient",
 "recipe").connection("recipe", "ingredient").add()
gremlin> schema.vertexLabel('author').describe()
==>schema.vertexLabel("author").properties("name",
 "gender").create()

```

- An alternative to discover schema information uses a `valueMap()` step on the traversal.

```

schema.traversal().V().valueMap()

==>{mode=[Development]}
==>{name=[author]}
==>{name=[recipe]}
==>{name=[ingredient]}
==>{name=[book]}
==>{name=[meal]}
==>{name=[reviewer]}
==>{name=[byName], type=[Secondary]}

```

```

==>{name=[includedIn], directionality=[Bidirectional],
cardinality=[Multiple]}
==>{name=[fridgeItem_single]}
==>{name=[rated], directionality=[Bidirectional],
cardinality=[Multiple]}
==>{name=[fridgeItem_multiple]}
==>{dataType=[Timestamp], name=[timestamp], cardinality=[Single]}
==>{dataType=[Text], name=[ISBN], cardinality=[Single]}
==>{dataType=[Text], name=[category], cardinality=[Single]}
==>{name=[byLocation]}
==>{dataType=[Int], name=[year], cardinality=[Single]}
==>{name=[ratedByStars], directionality=[OUT]}
==>{dataType=[Text], name=[gender], cardinality=[Single]}
==>{unique=[false], name=[byIngredient], type=[Materialized]}
==>{dataType=[Text], name=[instructions], cardinality=[Single]}
==>{unique=[false], name=[byReviewer], type=[Materialized]}
==>{unique=[false], name=[byRecipe], type=[Materialized]}
==>{unique=[false], name=[byMeal], type=[Materialized]}
==>{dataType=[Int], name=[stars], cardinality=[Single]}
==>{dataType=[Text], name=[comment], cardinality=[Single]}
==>{dataType=[Int], name=[calories], cardinality=[Single]}
==>{dataType=[Text], name=[blah], cardinality=[Single]}
==>{dataType=[Text], name=[amount], cardinality=[Single]}
==>{name=[created], directionality=[Bidirectional],
cardinality=[Multiple]}
==>{name=[includes], directionality=[Bidirectional],
cardinality=[Multiple]}
==>{dataType=[Bigint], name=[member_id], cardinality=[Single]}
==>{name=[authored], directionality=[Bidirectional],
cardinality=[Multiple]}
==>{dataType=[Text], name=[country], cardinality=[Multiple]}
==>{dataType=[Text], name=[item_mult], cardinality=[Multiple]}
==>{dataType=[Int], name=[community_id], cardinality=[Single]}
==>{dataType=[Text], name=[livedIn], cardinality=[Single]}
==>{dataType=[Text], name=[item_single], cardinality=[Single]}
==>{dataType=[Text], name=[name], cardinality=[Single]}

```

**Caution:** Using `valueMap()` without specifying properties can result in slow query latencies, if a large number of property keys exist for the queried vertex or edge. Specific properties can be specified, such as `valueMap('name')`.

- Changing `valueMap()` to `valueMap(true)` adds the id for each field.

```
graph.schema().traversal().V().valueMap(true)
```

```

==>{mode=[Development], id=0, label=schema}
==>{id=129, label=incident}
==>{id=133, label=incident}
==>{id=73, label=incident}
==>{id=137, label=incident}
==>{id=77, label=incident}
==>{id=141, label=incident}

```

```

==>{id=32784, dataType=[Text], name=[comment], label=propertyKey,
cardinality=[Single]}
==>{id=32782, name=[rated], directionality=[Bidirectional],
label=edgeLabel, cardinality=[Multiple]}
==>{id=32783, dataType=[Int], name=[stars], label=propertyKey,
cardinality=[Single]}
==>{id=85, label=incident}
==>{id=149, label=incident}
==>{id=32780, dataType=[Timestamp], name=[timestamp],
label=propertyKey, cardinality=[Single]}
==>{id=32781, dataType=[Int], name=[calories], label=propertyKey,
cardinality=[Single]}
==>{id=32769, dataType=[Smallint], name=[member_id],
label=propertyKey, cardinality=[Single]}
==>{id=94, label=incident}
==>{id=32767, dataType=[Int], name=[community_id],
label=propertyKey, cardinality=[Single]}
==>{id=98, label=incident}
==>{id=108, label=incident}
==>{id=32775, name=[authored], directionality=[Bidirectional],
label=edgeLabel, cardinality=[Multiple]}
==>{id=112, label=incident}
==>{id=32776, name=[created], directionality=[Bidirectional],
label=edgeLabel, cardinality=[Multiple]}
==>{id=1, name=[author], label=vertexLabel}
==>{id=32773, dataType=[Text], name=[ISBN], label=propertyKey,
cardinality=[Single]}
==>{id=2, name=[book], label=vertexLabel}
==>{id=32774, dataType=[Text], name=[instructions],
label=propertyKey, cardinality=[Single]}
==>{id=3, name=[recipe], label=vertexLabel}
==>{id=32771, dataType=[Text], name=[gender], label=propertyKey,
cardinality=[Single]}
==>{id=4, name=[ingredient], label=vertexLabel}
==>{id=116, label=incident}
==>{id=32772, dataType=[Int], name=[year], label=propertyKey,
cardinality=[Single]}
==>{id=5, name=[meal], label=vertexLabel}
==>{id=6, name=[reviewer], label=vertexLabel}
==>{id=32770, dataType=[Text], name=[name], label=propertyKey,
cardinality=[Single]}
==>{id=32779, name=[includedIn], directionality=[Bidirectional],
label=edgeLabel, cardinality=[Multiple]}
==>{id=125, label=incident}
==>{id=32777, name=[includes], directionality=[Bidirectional],
label=edgeLabel, cardinality=[Multiple]}
==>{id=32778, dataType=[Text], name=[amount], label=propertyKey,
cardinality=[Single]}

```

- Running `traversal()` will supply information about the number of schema element exist for vertices and edges, as well as the `TraversalSource` type.

```
schema.traversal()
```

```
==>graphtraversalsource[tinkergraph[vertices:58 edges:106],
 standard]
```

- A list of all vertex labels using utilities `split()` and `grep()`.

```
schema.describe().split('\n').grep(~/.*vertexLabel.*/)
```

```
gremlin> schema.describe().split('\n').grep(~/.*vertexLabel.*/)
==>schema.vertexLabel("meal").properties("name", "timestamp",
 "calories").create()
==>schema.vertexLabel("ingredient").properties("name").create()
==>schema.vertexLabel("ingredient").index("byIngredient").materialized().by("name").add()
==>schema.vertexLabel("test").partitionKey("tester").clusteringKey("foor").create()
==>schema.vertexLabel("FridgeSensor").create()
==>schema.vertexLabel("author").properties("name", "gender",
 "nationality").create()
==>schema.vertexLabel("author").index("byName").secondary().by("name").add()
==>schema.vertexLabel("author").index("byAuthor").materialized().by("name").add()
==>schema.vertexLabel("FridgeItem").properties("name",
 "expiration_date", "amount").create()
==>schema.vertexLabel("book").properties("name", "year",
 "ISBN").create()
==>schema.vertexLabel("recipe").properties("name",
 "instructions").create()
==>schema.vertexLabel("recipe").index("byRecipe").materialized().by("name").add()
==>schema.vertexLabel("reviewer").properties("name").create()
==>schema.vertexLabel("reviewer").index("byReviewer").materialized().by("name").add()
==>schema.vertexLabel("reviewer").index("ratedByStars").outE("rated").by("stars").add()
```

- Get the name of the current graph.

```
graph.name()
```

```
==>quickstart
```

## Creating queries using traversals

Create queries using graph traversals.

DSE Graph can create complex queries that traverse the relationships of the graph structure. If the complex queries require real-time results, DSE Graph is the best product for discovering answers. Start with the [Quick Start \(page 368\)](#) traversals that increase in complexity in a stepwise fashion. The examples shown here will continue with the [Recipe Toy Graph example \(page 432\)](#).

Additional complex Gremlin recipes can also be found at [Apache TinkerPop Recipes](#).

## Anatomy of a graph traversal

The anatomy of a graph traversal explores the results of each traversal step.

## Structure of a graph traversal

Simple traversals can be complex, but generally do not employ specialized techniques such as recursion or branching.

Break down the chain of a graph traversal into traversal steps:

```
g.V().hasLabel('recipe').count()
```

This graph traversal to find the number of recipes in the graph has four parts:

### The graph traversal g

g will return an error if run alone

### All vertices are gathered with v()

All the vertices will be returned. A sample of the result:

```
gremlin> g.V() ==>v[{~label=recipe, recipeId=2003}]
==>v[{~label=recipe, recipeId=2005}]
==>v[{~label=ingredient, ingredId=3017}]
==>v[{~label=ingredient, ingredId=3028}]
==>v[{~label=person, personId=1}]
==>v[{~label=person, personId=3}]
==>v[{~label=book, bookId=1004}]
==>v[{~label=book, bookId=1001}]
==>v[{~label=meal, type="lunch", mealId=4003}]
```

### Filter out the vertices labeled as a recipe with hasLabel('recipe')

Only the vertices that are recipes will be returned:

```
gremlin> g.V().hasLabel('recipe')
==>v[{~label=recipe, recipeId=2003}]
==>v[{~label=recipe, recipeId=2005}]
==>v[{~label=recipe, recipeId=2006}]
==>v[{~label=recipe, recipeId=2007}]
==>v[{~label=recipe, recipeId=2001}]
==>v[{~label=recipe, recipeId=2002}]
==>v[{~label=recipe, recipeId=2004}]
==>v[{~label=recipe, recipeId=2008}]
```

### Count the number of vertices with count()

The number of vertices returned from the last traversal step is totalled:

```
gremlin> g.V().hasLabel('recipe').count()
==>8
```

## Graph traversal with edges

Before trying the traversals displayed below, run the following script either in Studio (copy and paste) or Gremlin console (:load /tmp/generateReviews.groovy):

```
// reviewer vertices
johnDoe = graph.addVertex(label, 'person', 'personId', 11, 'name', 'John
Doe')
```

```

johnSmith = graph.addVertex(label, 'person', 'personId', 12, 'name',
 'John Smith')
janeDoe = graph.addVertex(label, 'person', 'personId', 13, 'name', 'Jane
 Doe')
sharonSmith = graph.addVertex(label, 'person', 'personId', 14,
 'name', 'Sharon Smith')
betsyJones = graph.addVertex(label, 'person', 'personId', 15,
 'name', 'Betsy Jones')

beefBourguignon = g.V().has('recipe', 'recipeId', 2001, 'name', 'Beef
 Bourguignon').tryNext().orElseGet {graph.addVertex(label, 'recipe',
 'recipeId', 2001, 'name', 'Beef Bourguignon')}
spicyMeatLoaf = g.V().has('recipe', 'recipeId', 2005, 'name', 'Spicy
 Meatloaf').tryNext().orElseGet {graph.addVertex(label, 'recipe',
 'recipeId', 2005, 'name', 'Spicy Meatloaf')}
carrotSoup = g.V().has('recipe', 'recipeId', 2007, 'name', 'Carrot
 Soup').tryNext().orElseGet {graph.addVertex(label, 'recipe',
 'recipedId', 2007, 'name', 'Carrot Soup')}

// reviewer - recipe edges
johnDoe.addEdge('reviewed', beefBourguignon, 'timestamp',
 Instant.parse('2014-01-01T00:00:00.00Z'), 'stars', 5, 'comment',
 'Pretty tasty!')
johnSmith.addEdge('reviewed', beefBourguignon, 'timestamp',
 Instant.parse('2014-01-23T00:00:00.00Z'), 'stars', 4)
janeDoe.addEdge('reviewed', beefBourguignon, 'timestamp',
 Instant.parse('2014-02-01T00:00:00.00Z'), 'stars', 5, 'comment',
 'Yummy!')
sharonSmith.addEdge('reviewed', beefBourguignon, 'timestamp',
 Instant.parse('2015-01-01T00:00:00.00Z'), 'stars', 3, 'comment', 'It
 was okay.')
johnDoe.addEdge('reviewed', spicyMeatLoaf, 'timestamp',
 Instant.parse('2015-12-31T00:00:00.00Z'), 'stars', 4, 'comment',
 'Really spicy - be careful!')
sharonSmith.addEdge('reviewed', spicyMeatLoaf,
 'timestamp', Instant.parse('2014-07-23T00:00:00.00Z'), 'stars', 3,
 'comment', 'Too spicy for me. Use less garlic.')
janeDoe.addEdge('reviewed', carrotSoup, 'timestamp',
 Instant.parse('2015-12-30T00:00:00.00Z'), 'stars', 5, 'comment',
 'Loved this soup! Yummy vegetarian!')

```

Any number of traversal steps can be chained into a traversal, filtering and transforming the graph data as required. In some cases, edges will be the result, and perhaps unexpected. Consider the following traversal:

```

g.V().hasLabel('recipe').has('name', 'Beef
 Bourguignon').inE().values('comment')

```

This graph traversal begins as the last traversal did with `g.V().hasLabel('recipe')`. It is then followed by:

#### A traversal step to pick only the vertices with the recipe title specified

The filter should capture one recipe if recipe titles are unique.

```
gremlin> g.V().hasLabel('recipe').has('name', 'Beef
 Bourguignon')
==>v[{\~label=recipe, recipeId=2001}]
```

### A traversal step that retrieves incoming edges

Notice from the two edges sampled from the complete result that edges with any label are filtered with this step. Using `inE('reviewed')` would be more precise if the target result are only ratings.

```
gremlin> g.V().hasLabel('recipe').has('name', 'Beef
 Bourguignon').inE()
==>e[{\~label=includedIn, ~out_vertex={\~label=ingredient,
 ingredId=3001},
 ~in_vertex={\~label=recipe, recipeId=2001},
 ~local_id=95672dd0-2e3d-11e8-8043-6bfe97ac83b9}]
[{\~label=ingredient, ingredId=3001}-includedIn->{\~label=recipe,
 recipeId=2001}]
==>e[{\~label=includedIn, ~out_vertex={\~label=ingredient,
 ingredId=3002},
 ~in_vertex={\~label=recipe, recipeId=2001},
 ~local_id=9567ca17-2e3d-11e8-8043-6bfe97ac83b9}]
[{\~label=ingredient, ingredId=3002}-includedIn->{\~label=recipe,
 recipeId=2001}]
==>e[{\~label=includedIn, ~out_vertex={\~label=ingredient,
 ingredId=3003},
 ~in_vertex={\~label=recipe, recipeId=2001},
 ~local_id=9567ca16-2e3d-11e8-8043-6bfe97ac83b9}]
[{\~label=ingredient, ingredId=3003}-includedIn->{\~label=recipe,
 recipeId=2001}]
==>e[{\~label=includedIn, ~out_vertex={\~label=ingredient,
 ingredId=3004},
 ~in_vertex={\~label=recipe, recipeId=2001},
 ~local_id=9567ca14-2e3d-11e8-8043-6bfe97ac83b9}]
[{\~label=ingredient, ingredId=3004}-includedIn->{\~label=recipe,
 recipeId=2001}]
==>e[{\~label=includedIn, ~out_vertex={\~label=ingredient,
 ingredId=3005},
 ~in_vertex={\~label=recipe, recipeId=2001},
 ~local_id=956754e0-2e3d-11e8-8043-6bfe97ac83b9}]
[{\~label=ingredient, ingredId=3005}-includedIn->{\~label=recipe,
 recipeId=2001}]
==>e[{\~label=created, ~out_vertex={\~label=person, personId=1},
 ~in_vertex={\~label=recipe, recipeId=2001},
 ~local_id=956495c4-2e3d-11e8-8043-6bfe97ac83b9}]
[{\~label=person, personId=1}-created->{\~label=recipe,
 recipeId=2001}]
==>e[{\~label=reviewed, ~out_vertex={\~label=person,
 personId=11},
 ~in_vertex={\~label=recipe, recipeId=2001},
 ~local_id=95775a70-2e3d-11e8-8043-6bfe97ac83b9}]
[{\~label=person, personId=11}-reviewed->{\~label=recipe,
 recipeId=2001}]
```

```

==>e[{~label=reviewed, ~out_vertex={~label=person,
 personId=12},
~in_vertex={~label=recipe, recipeId=2001},
~local_id=95773361-2e3d-11e8-8043-6bfe97ac83b9}]
[{~label=person, personId=12}-reviewed->{~label=recipe,
 recipeId=2001}]
==>e[{~label=reviewed, ~out_vertex={~label=person,
 personId=13},
~in_vertex={~label=recipe, recipeId=2001},
~local_id=95773360-2e3d-11e8-8043-6bfe97ac83b9}]
[{~label=person, personId=13}-reviewed->{~label=recipe,
 recipeId=2001}]
==>e[{~label=reviewed, ~out_vertex={~label=person,
 personId=14},
~in_vertex={~label=recipe, recipeId=2001},
~local_id=95778180-2e3d-11e8-8043-6bfe97ac83b9}]
[{~label=person, personId=14}-reviewed->{~label=recipe,
 recipeId=2001}]

```

### Parsing out the comment property from the rated edges

Here, the `inE()` is specified with the edge label `reviewed`. The property values are retrieved for the property key `comment`:

```

gremlin> g.V().hasLabel('recipe').has('name', 'Beef
Bourguignon').inE('reviewed').values('comment')
==>Yummy!
==>Pretty tasty!
==>It was okay.

```

Building graph traversals one step at a time can yield interesting results and insight into how to create traversals.

### The path of a graph traversal

A traversal step exists that will show the path taken by a graph traversal. First, find the results for a traversal that answers the question about what recipes that list beef and carrots as ingredients are included in the cookbooks, given the cookbook and recipe title?

```

gremlin>
g.V().hasLabel('ingredient').has('name',within('beef','carrots')).in().as('Recipe').
out().hasLabel('book').as('Book').
select('Book','Recipe').by('name').
by('name')
==>{Book=The Art of French Cooking, Vol. 1, Recipe=Beef Bourguignon}
==>{Book=The Art of Simple Food: Notes, Lessons, and Recipes from a
Delicious Revolution, Recipe=Carrot Soup}

```

One expects that the traversal path will be from `ingredient` to `recipe` to `book`. To check if this assumption is correct, add `path()` to the end of the traversal.

```

gremlin>
g.V().hasLabel('ingredient').has('name',within('beef','carrots')).in().as('Recipe').
out().hasLabel('book').as('Book').

```

```

select('Book','Recipe').
by('name').by('name').path()
==>[v[{\~label=ingredient, ingredId=3001}], v[{\~label=recipe,
recipeId=2001}], v[{\~label=book, bookId=1001}],
{Book=The Art of French Cooking, Vol. 1, Recipe=Beef
Bourguignon}]
==>[v[{\~label=ingredient, ingredId=3028}], v[{\~label=recipe,
recipeId=2007}], v[{\~label=book, bookId=1004}],
{Book=The Art of Simple Food: Notes, Lessons, and Recipes from a
Delicious Revolution, Recipe=Carrot Soup}]

```

For each case, notice that the traversal does follow the expected path.

## Traversal metrics

In addition to tracing the output of each graph traversal step, metrics can produce interesting insights as well. To add metrics to the last traversal shown, use some additional chained steps:

```

gremlin> g.V().has('recipe', 'name', 'Beef
Bourguignon').inE().values('comment').profile()
==>Traversal Metrics
Step
Count Traversers Time (ms) % Dur
=====
DsegGraphStep(vertex,[],(label = recipe & name ...
1 1 9.710 77.19
query-optimizer
 0.946
 _condition=((label = recipe & name = Beef Bourguignon) & (true))
query-setup
 0.051
 _isFitted=true
 _isSorted=false
 _isScan=false
index-query
 7.131
 _indexType=Search
 _usesCache=false
 _statement=SELECT "recipeId" FROM "newComp"."recipe_p" WHERE
"solr_query" = '{"q":"*:*", "fq":["name:Bee
f\\ Bourguignon"]}' LIMIT ?; with params
(java.lang.Integer) 50000
 _options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
c=true}
DsegVertexStep(IN,edge,direction = IN,Unordered)
10 10 2.275 18.09
query-optimizer
 0.421
 _condition=((true) & direction = IN)

```

```

query-setup 0.016
 _isFitted=true
 _isSorted=false
 _isScan=false
vertex-query 0.475
 _usesCache=false
 _statement=SELECT * FROM "newComp"."recipe_e" WHERE "recipeId" = ?
 LIMIT ? ALLOW FILTERING; with params
 (java.lang.Integer) 2001, (java.lang.Integer) 50000
 _options=Options{consistency=Optional[ONE],
 serialConsistency=Optional.empty, fallbackConsistency=Option
 al.empty, pagingState=null, pageSize=-1,
 user=Optional.empty, waitForSchemaAgreement=true, asyn
 c=true}
 _usesIndex=false
DsegPropertiesStep([comment],value,label = comm...
 3 3 0.593 4.71
 >TOTAL
 - - 12.579 -

```

The type of traversal step is listed, along with the number of traversers and the time to complete the traversal step. If a traversal step can be processed in parallel, multiple traversers will be employed to retrieve data. Some traversal steps are graph-global requiring retrieval from the entire graph; DsegGraphStep is a graph-global retrieval that finds vertices that match certain conditions. Other traversal steps are graph-local walks and can be processed in parallel; DsegVertexStep is a graph-local walk that walks through the graph along constrained paths. DSE Graph uses automatic query optimization to determine the traversal strategies to efficiently use any index structures that exist.

Looking at the metrics, the question of performance comes to mind. For instance, is there any way to optimize the traversal shown above? In fact, a simple modification results in a time savings:

```

gremlin> g.V().has('recipe', 'name', 'Beef
 Bourguignon').inE('reviewed').values('comment').profile()
==>Traversal Metrics
Step
Count Traversers Time (ms) % Dur
=====
DsegGraphStep(vertex,[],(label = recipe & name ...
 1 1 7.109 83.34
query-optimizer
 0.342
 _condition=((label = recipe & name = Beef Bourguignon) & (true))
query-setup
 0.013
 _isFitted=true
 _isSorted=false
 _isScan=false
index-query
 6.110

```

```

_indexType=Search
_usesCache=false
_statement=SELECT "recipeId" FROM
"DSE_GRAPH_QUICKSTART"."recipe_p" WHERE "solr_query" = '{"q": "*:*",
"fq": [{"name:Bee
 f"\ Bourguignon"]}' LIMIT ?; with params
(java.lang.Integer) 50000
_options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
c=true}
DsegVertexStep(IN,[reviewed],edge,(direction = ...
4 4 1.234 14.47
query-optimizer
 0.187
_condition=((direction = IN & label = reviewed) & (true))
query-setup
 0.009
_isFitted=true
_isSorted=false
_isScan=false
vertex-query
 0.474
_usesCache=false
_statement=SELECT * FROM "DSE_GRAPH_QUICKSTART"."recipe_e" WHERE
"recipeId" = ? AND "~~edge_label_id" = ? LIMIT ? ALL
 OW FILTERING; with params (java.lang.Integer) 2001,
(java.lang.Integer) 65633, (java.lang.Int
eger) 50000
_options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
c=true}
_usesIndex=false
DsegPropertiesStep([comment],value,label = comm...
3 3 0.186 2.19
 >TOTAL
- - 8.530 -

```

The change made is subtle. The traversal steps, `inE()` has been replaced by `inE('reviewed')`, to find only the edges that are reviews. Although each measurement can vary, generally the second traversal will outperform the first traversal.

DataStax Studio provides more easy to use metrics, in which scrolling over or selecting any phase of the traversal can be explored:



## Using user-defined vertex ids

How to use a user-defined vertex id in queries.

Vertices can be filtered using a `hasId()` traversal step and providing a particular user-defined vertex id:

```
g.V().hasId(['~label':'fridgeSensor', 'stateId':31, 'cityId':100,
'sensorId':1]).valueMap()
```

A vertex can be queried directly using a user-defined vertex id:

```
g.V(['~label':'fridgeSensor', 'stateId':31, 'cityId':100,
'sensorId':1]).valueMap()
```

## Using indexes

Using indexes for graph queries.

Global indexes can be used in graph traversal queries for the first traversal step reached after the `v()` step, and is used to trim down the number of vertices that are initially fetched. In general, the traversal step will involve a vertex label and can include a property key and a particular property value. In a traversal, the step following `g.v()` is generally the step in which an index will be consulted. If a mid-traversal `v()` step is called, then an additional indexed step can be consulted to narrow the list of vertices that will be traversed.

**Note:** Graph traversals will only use indexes if both the vertex label and property key are specified. If both are not specified, indexing will not be used and a full graph scan for the property key can result. If full graph scan is disabled, a query will fail, as shown in this example where a property is specified, but a vertex label is not specified:

```
g.V().has('name','Julia Child')
Could not find an index to answer query clause and
graph.allow_scan is disabled:
((label = FridgeSensor & name WITHIN [Julia Child]) | (label =
author & name WITHIN [Julia Child]) |
(label = book & name WITHIN [Julia Child]) | (label = ingredient &
name WITHIN [Julia Child]) |
(label = meal & name WITHIN [Julia Child]) | (label = recipe &
name WITHIN [Julia Child]) |
```

```
(label = reviewer & name WITHIN [Julia Child]))
```

Edge indexes and property indexes (vertex-centric indexes) can be used to narrow the query after a global index has found the starting vertex. They allow

### Global index

- The graph traversal shown uses an index to discover certain person vertices to start the query.

```
g.V().has(person, 'name', 'Emeril
Lagasse').out('created').values('name')
```

index ↑	value
0	Wild Mushroom Stroganoff
1	Spicy Meatloaf

Displaying 1 - 2 of 2

Success. 2 elements returned. Duration: 0.007 s.

This graph traversal uses an index, if the index exists, because the traversal step `has('person', 'name', 'Emeril Lagasse')` identifies the vertex label and the property key indexed. After finding the initial vertex to traverse from, the outgoing `created` edges are walked and the adjacent vertices are listed by `name`. This graph traversal shows the importance of using the vertex label in combination with the property key, as two different elements, persons and recipes, use the same property key `name`.

Checking for the use of indexing can be accomplished with the `profile()` method:

```
gremlin> g.V().has('person', 'name', 'Emeril
Lagasse').out('created').values('name').profile()
==>Traversal Metrics
Step
 Count Traversers Time (ms) % Dur
=====
DsegGraphStep([~label.=(person), name.=(Emeril ...
 1 1 2.196 51.37
 query-optimizer
 0.199
 query-setup
 0.004
 index-query
 0.946
DsegVertexStep(OUT,[created],vertex)
 2 2 0.935 21.88
 query-optimizer
 0.101
```

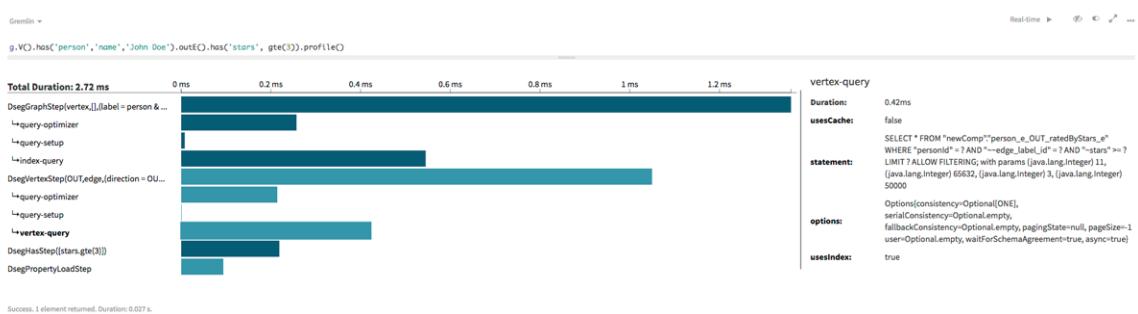
query-setup		0.000		
vertex-query		0.282		
DsegPropertiesStep( [name], value)	2	1.030	24.11	
query-optimizer		0.044		
query-setup		0.005		
vertex-query		0.347		
vertex-query		0.639		
query-setup		0.000		
NoOpBarrierStep(2500)	2	0.113	2.64	>TOTAL
-	-	4.276	-	

Note the *index-query* used in the first step `DsegGraphStep`. If an index was not used, *index-query* would be missing from the profile output.

## Edge index

- An edge index can narrow the query, such as this one that finds all the outgoing edges for reviews that *John Doe* wrote that have a rating of greater or equal to 3 stars:

```
g.V().has('person', 'name', 'John Doe').outE().has('stars', gte(3))
```



Using `profile()` on the query shows that a global index query was used in the initial step, and the output shown here shows that in the second step, the `ratedByStars` edge index was used to cut the latency of the query.

**Tip:** The `local()` (page 705) step can be used to affect how an edge index narrows a query.

## Property index

- A property index can narrow the query, such as this one that finds the countries that *Julia Child* lived in, starting in the year 1961 (in this case, only one country):

```

g.V().has('person', 'name','Julia Child').as('person').
 local(properties('country').has('startYear',
 1961)).value().as('country').
 select('person','country').
 by('name').by().profile()

gremlin> g.V().has('person', 'name','Julia Child').as('person').
.....1> local(properties('country').has('startYear',
 1961)).value().as('country').
.....2> select('person','country').
.....3> by('name').by().profile()
==>Traversal Metrics
Step
 Count Traversers Time (ms) % Dur
=====
DsegGraphStep(vertex,[],(label = person & name ...
 1 1 1.274 37.35
query-optimizer
 0.253
 _condition=((label = person & name = Julia Child) & (true))
query-setup
 0.008
 _isFitted=true
 _isSorted=false
 _isScan=false
index-query
 0.557
 _indexType=Materialized
 _usesCache=false
 _statement=SELECT "personId" FROM
"newComp"."person_p_byName" WHERE "name" = ? LIMIT ?; with
params (jav
 a.lang.String) Julia Child, (java.lang.Integer)
50000
 _options=Options{consistency=Optional[ONE],
serialConsistency=Optional.empty, fallbackConsistency=Option
 al.empty, pagingState=null, pageSize=-1,
user=Optional.empty, waitForSchemaAgreement=true, asyn
 c=true}
DsegHasStep@[person]
 1 1 0.060 1.76
LocalStep([DsegPropertiesStep([country],prop...
 1 1 1.300 38.12
DsegPropertiesStep([country],property,(label ...
 1 1 1.149
query-optimizer
 0.239
 _condition=((label = country & startYear = 1961) & (true))
query-setup
 0.001
 _isFitted=true
 _isSorted=false

```

```

_isScan=false
vertex-query
 0.564
 \usesCache=false
 \statement=SELECT * FROM
 "newComp"."person_p_OUT_byStartYear_p" WHERE "personId" = ? AND
 "~~property_ke
 y_id" = ? AND "startYear" = ? LIMIT ? ALLOW
 FILTERING; with params (java.lang.Integer) 1, (
 java.lang.Integer) 32801, (java.lang.Integer)
 1961, (java.lang.Integer) 50000
 \options=Options{consistency=Optional[ONE],
 serialConsistency=Optional.empty, fallbackConsistency=Optio
 nal.empty, pagingState=null, pageSize=-1,
 user=Optional.empty, waitForSchemaAgreement=true, as
 ync=true}
 \usesIndex=true
 DsegHasStep([startYear.eq(1961)])
 1 1 0.081
 PropertyValueStep
 1 1 0.026
SelectStep(last,[person, country],[value(name),...
 1 1 0.720 21.13
NoOpBarrierStep(2500)
 1 1 0.032 0.95
DsegPropertyLoadStep
 1 1 0.023 0.69
 >TOTAL
 - - 3.411 -

```

Using `profile()` on the query shows that a global index query was used in the initial step, and the output shown here shows that in the second SELECT step, the `byStartYear` property index was used to cut the latency of the query.

**Tip:** The [local\(\) \(page 705\)](#) step can also be handy for use with property indexes.

## Using search indexes

Using search indexes for graph traversals.

DSE Graph leverages DSE Search [indexes \(page 457\)](#) to efficiently filter vertices by properties, and reducing query latency. DSE Search uses a [modified Apache Solr \(page 305\)](#) to create the search indexes. Graph search indexes can be created using textual, numeric and geospatial data.

It is important to note that traversal queries with search predicates can be completed whether a search index exists or not. However, full graph scans will occur without a search index and performance will degrade severely as the graph grows, an unacceptable solution in a production environment. Create search indexes during schema creation before inserting data and querying the graph. Search indexes will only be created if DSE Search is started in conjunction with DSE Graph. If search indexes are used, the queries must be run on DSE Search nodes in the cluster.

In general, the traversal step will involve a vertex label and can include a property key and a particular property value. In a traversal, the step following `g.v()` is generally the step in which an index will be consulted. If a mid-traversal `v()` step is called, then an additional indexed step can be consulted to narrow the list of vertices that will be traversed.

Textual search indexes are by default indexed in both tokenized (TextField) and non-tokenized (StrField) forms. This means that all textual predicates (token, tokenPrefix, tokenRegex, eq, neq, regex, prefix) will be usable with all textual vertex properties indexed. Practically, search indexes should be created using the `asString()` method only in cases where there is absolutely no use for tokenization and text analysis, such as for inventory categories (silverware, shoes, clothing). The `asText()` method is used if searching tokenized text, such as long multi-sentence descriptions. The query optimizer will choose whether to use analyzed or non-analyzed indexing based on the textual predicate used.

Property key indexes defined with `asText()` or undefined (since this is the default) can use the following options for search:

- [token \(page 575\)](#)
- [tokenPrefix \(page 575\)](#)
- [tokenRegex \(page 576\)](#)

Property key indexes defined with `asString()` can use the following options for search:

- [eq/neq \(page 576\)](#)
- [prefix \(page 577\)](#)
- [regex \(page 578\)](#)

**Note:** The `eq()` search cannot be used with property key indexes created with `asText()` because they contain tokenized data and are therefore not suitable for exact text matches.

In addition, in DSE 5.1 and later, fuzzy search predicates have been added:

- [phrase \(page 578\)](#)
- [fuzzy \(page 579\)](#)
- [tokenFuzzy \(page 579\)](#)

Two of the predicates, `fuzzy` and `tokenFuzzy`, can be used with `TextFields` and `StrFields`, respectively, while `phrase` can be used only with `TextFields`.

### Creating a textual search index

- An example search index from [Creating indexes \(page 460\)](#) for vertex label `recipe` that will be used for all examples below:

```
schema.vertexLabel('recipe').index('search').search().
 by('instructions').asText().
 by('name').asString().add()
```

This search index uses DSE Search to index `instructions` as full text using tokenization, and `name` as a string. Note that, as of DSE 5.1, only those properties that specifically should be indexed as non-tokenized data must specify `asString()`.

If there are properties that specifically should be indexed only as tokenized data, specify `asText()`.

### Search using `token()` methods on full text

- In a traversal query, use a token search to find list the names of all recipes that have the word `Saute` in the instructions. The method `token()` is used with a supplied word.

```
g.V().has('recipe','instructions', token('Saute')).values('name')
```

GREMLIN ▾

```
g.V().has('recipe','instructions', Search.token('Saute')).values('name')
```

index ↑	value
0	Wild Mushroom Stroganoff
1	Beef Bourguignon
2	Oysters Rockefeller

Displaying 1 - 3 of 3

Why does this search find these three recipes? Because the instructions for each meet the search requirements:

```
gremlin> g.V().has('recipe','instructions', token('Saute')).values('instructions')
==>Braise the beef. Saute the onions and carrots. Add wine and cook in a dutch oven at 425 degrees for 1 hour.
==>Saute the shallots, celery, herbs, and seasonings in 3 tablespoons of the butter for 3 minutes. Add the watercress and let it wilt.
==>Cook the egg noodles according to the package directions and keep warm. Heat 1 1/2 tablespoons of the oliveoil in a large saute pan over medium-high heat.
```

### Search using `tokenPrefix()` methods on full text

- In a traversal query, use a token prefix search to list the names of all recipes that have a word that includes a prefix of `Sea` in the instructions. The method `tokenPrefix()` is used with a supplied prefix (a set of alphanumeric characters).

```
g.V().hasLabel('recipe').has('instructions',
 tokenPrefix('Sea')).values('name','instructions')
```

```
g.V().hasLabel('recipe').has('instructions', Search.tokenPrefix('Sea')).values('name','instructions')
```

The screenshot shows the DataStax Studio interface with a traversal query results table. The table has two columns: 'index ↑' and 'value'. The data rows are:

index ↑	value
0	Roast Pork Loin
1	The day before, separate the meat from the ribs, stopping about 1 inch before the end of the bones. Season the pork liberally inside and out with salt and pepper and refrigerate overnight.
2	Oysters Rockefeller
3	Saute the shallots, celery, herbs, and seasonings in 3 tablespoons of the butter for 3 minutes. Add the watercress and let it wilt.

Two recipes are returned, one with the word **Season** in the instructions, and one with the word **seasonings** in the instructions. Case is insensitive in `tokenPrefix()` indexing.

### Search using `tokenRegex()` methods on full text

- In a traversal query, use a token regular expression (regex) search to find all recipes that have a word that includes the regular expression specified. The regex, `.*sea.*in.*`, looks for the letters **sea** preceded by any number of other characters and followed by any number of other characters until the letters **in** are found and also followed by any number of other characters in the instructions and list the recipe names. The method `tokenRegex()` is used with a supplied regex.

```
g.V().hasLabel('recipe').has('instructions',
 tokenRegex('.*sea.*in.*')).values('name','instructions')
```

The screenshot shows the DataStax Studio interface with a traversal query results table. The table has two columns: 'index ↑' and 'value'. The data rows are:

index ↑	value
0	Oysters Rockefeller
1	Saute the shallots, celery, herbs, and seasonings in 3 tablespoons of the butter for 3 minutes. Add the watercress and let it wilt.

Note that in this query, only the Oysters Rockefeller recipe is returned because the word **Season** in the Roast Pork Loin recipe does not meet the requirements for the regular expression.

### Search using `eq()` on non-token methods on strings

- In a traversal query, use a non-token search to list all recipes that have `Carrot Soup` in the recipe name. Note that this search is case-sensitive, so using `carrot soup` would not find a vertex. The method `eq()` is used with a supplied name.

```
g.V().hasLabel('recipe').has('name', eq('Carrot
Soup')).values('name')
```

GREMLIN ▾

```
g.V().hasLabel('recipe').has('name', eq('Carrot Soup')).values('name')
```



Carrot Soup

The match is found for the full author name listed. Note that `neq()` can also be used to find all strings that do not match the specified string.

- In a traversal query, use a non-token search to list all recipes that have `Carrot` in the recipe name. The method `eq()` is used with a supplied name.

```
g.V().hasLabel('recipe').has('name', eq('Carrot')).valueMap()
```

GREMLIN ▾

```
g.V().hasLabel('recipe').has('name', eq('Carrot')).valueMap()
```

Success - No Results

No match is found, because only a partial name was specified. For `asString()` indexes, the string must match.

### Search using `prefix()` on non-token methods on strings

- In a traversal query, use a non-token search to find all authors that have a name beginning with the letter `R`. The method `prefix()` is used with a supplied string.

```
g.V().hasLabel('recipe').has('name', prefix('R')).values('name')
```

GREMLIN ▾

Real-time ▶ ⌛ ⚡ ...

```
g.V().hasLabel('recipe').has('name', Search.prefix('R')).values('name')
```



index ↑	value
0	Roast Pork Loin
1	Ratatouille

Matches are found for each author name that begins with `R`, provided the recipe name was designated with `asString()` in the search index.

### Search using `regex()` on non-token methods on strings

- In a traversal query, use a non-token search to find all recipes that have a name that includes a specified regular expression. The method `regex()` is used with a supplied regex.

```
g.V().hasLabel('recipe').has('name',
 regex('.*ee.*')).values('name')
```

GREMLIN ▾

```
g.V().hasLabel('recipe').has('name', Search.regex('.*ee.*')).values('name')
```

{}

T

Beef Bourguignon

Matches are found for each author name that include the regex `.*ee.*` to find all strings that include `ee` preceded and followed by any number of other characters, provided the recipe name was designated with `asString()` in the search index.

### Search using `phrase()`

- The `phrase()` predicate is used with properties designated as `TextFields`.

Find the exact phrase *Wild Mushroom Stroganoff* in a recipe name:

```
g.V().hasLabel('recipe').has('name', phrase('Wild Mushroom
Stroganoff',0))
```

The `0` designates that the result must be an exact phrase.

```
v[{~label=recipe, community_id=2123369856, member_id=0}]
```

The vertex for the correct recipe is returned.

- The `phrase()` predicate can be used for [proximity searches](#), to discover phrases that have terms that are within a certain distance of one another in the tokenized text.

```
g.V().hasLabel('recipe').has('name', phrase('Wild Stroganoff',1))
```

The value of `1` designates that the result must only have words in the recipe name that are one term away from one another. In this example, the variation is the addition of the word *Mushroom*.

```
v[{~label=recipe, community_id=2123369856, member_id=0}]
```

The vertex for the correct recipe is returned. A match for

```
g.V().hasLabel('recipe').has('name', phrase('Wild Mushroom',1))
```

 will

also return the correct vertex, but `g.V().hasLabel('recipe').has('name', phrase('Mushroom Wild',1))` will not.

### Search using fuzzy()

- The `fuzzy()` predicate uses [optimal string alignment distance calculations](#) to match properties designated as `StrFields`. Variations in the letters used in words, such as misspellings, are the focus of this predicate. The edit distance specified refers to the number of transpositions of letters, with a single transposition of letters constituting one edit.

Find the exact name of *James Beard* in an author name:

```
g.V().hasLabel('author').has('name', fuzzy('James Beard', 0)).values('name')
```

The `0` designates that the result must be an exact match.

James Beard

- Changing the last value in a `fuzzy()` predicate will find misspellings:

```
g.V().hasLabel('author').has('name', fuzzy('James Beard', 1)).values('name')
```

The `1` designates that the result matches with an edit distance of at most one.

James Beard, Jmaes Beard

If an author vertex exists with the misspelling *Jmaes Beard*, the query shown will find both vertices. The value of `1` finds this misspelling because of the single transposition of the letters *a* and *m*.

- Note that searching for a misspelling will find the records with the correct spelling, as well as the misspelled name

```
g.V().hasLabel('author').has('name', fuzzy('Jmase Beard', 2)).values('name')
```

The `2` designates that the result must match with at most two transpositions.

James Beard, Jmaes Beard

If an author vertex exists with the misspelling *Jmaes Beard*, the query shown will find both vertices. The value of `2` finds both the misspelling because of the single transposition of letters, *e* and *s* in *Jmaes Beard*, as well as the correct spelling with a second transposition of letters from *Jmase Beard* to *James Beard*.

**Caution:** Specifying an edit distance of `3` or greater matches too many terms for useful results. The resulting search index will be too large to efficiently filter queries.

### Search using tokenFuzzy()

- The `tokenFuzzy()` predicate is similar to `fuzzy()`, but searches for variation across individual tokens in analyzed textual data (`TextFields`).

Find the recipe name that includes the word *Wild* while searching for the word with a one-letter misspelling:

```
g.V().hasLabel('recipe').has('name',
 tokenFuzzy('Wlid',1)).values('name')
```

The `1` designates that one letter misspelling (one transposition) is acceptable.

```
Wild Beef Stroganoff
```

### Using two search indexes for a single traversal query

- Create a second search index like an example search index from [Creating indexes \(page 460\)](#) for vertex label `author`.

```
schema.vertexLabel('author').index('search').search()
 by('name').asString()
 by('nickname').ifNotExists().add()
```

This search index will use DSE Search to index `nickname` as full text using tokenization, and `name` as a string.

- This traversal query demonstrates a mid-traversal `v()` that allows a search index for `author` as well as a search index for `recipe` to be used to execute the query. The first index uses a `tokenRegex()` to find recipe instructions that start with the word **Braise**; this part of the query is labeled as `r` for use later in the query. Then the search index for `author` is searched for an author name that starts with the letter **J**, and traversed through an outgoing edge to a vertex where the search found in the first part of the query is found with `where(eq('r'))`.

```
g.V().has('recipe', 'instructions',
 tokenRegex('Braise.*')).as('r').
 V().has('author', 'name',
 prefix('J')).out().where(eq('r')).values('name')
```

GREMLIN ▾

```
g.V().has('recipe', 'instructions', Search.tokenRegex('Braise.*')).as('r').
V().has('author', 'name', Search.prefix('J')).out().where(eq('r')).values('name')
```



Beef Bourguignon

This query traversal finds the recipe `Beef Bourguignon` authored by `Julia Child`, and illustrates some of the complexity that can be successfully used with search indexes.

### Search using geospatial values

- Geospatial search is used to discover geospatial relationships. Search indexes are used to make such searches possible. First, a search index must be created.

```
schema.vertexLabel('FridgeSensor').index('search').search()
 by('location').ifNotExists().add()
```

- Some sample data will be helpful for understanding the search results. Two vertices are entered for fridge sensor:

```
graph.addVertex(label, 'FridgeSensor', 'name', 'jones1', 'city_id',
 100, 'sensor_id', '60bcae02-f6e5-11e5-9ce9-5e5517507c66',
 'location', Geo.point(-118.359770, 34.171221))
graph.addVertex(label, 'FridgeSensor', 'name', 'smith1', 'city_id',
 100, 'sensor_id', '61deada0-3bb2-4d6d-a606-a44d963f03b5',
 'location', Geo.point(-115.655068, 35.163427))
```

The sensors are named and given a city ID and sensor ID in addition to the location with data type `Point`.

- A query can find all sensors that meet the requirement of being inside the described polygon `Distance` that is designated as a circle with a center at (-110, 30) and a radius of 20 degrees with the method `Geo.inside()`.

```
Distance d = Geo.point(-110,30),20, Geo.Unit.DEGREES)
g.V().hasLabel('FridgeSensor').has('location',
 Geo.inside(d)).values('name')
```

GREMLIN ▾

```
Distance d = Geo.distance(-110,30,20)
g.V().hasLabel('FridgeSensor').has('location', Geo.inside(d)).values('name')
```

index ↑	value
0	jones1
1	smith1

Displaying 1 - 2 of 2

More information on geospatial queries can be found in [Geospatial traversals \(page 591\)](#).

## Search using numerical values

- Search indexes can also be used for non-textual values:

```
schema.propertyKey('name').Text().create()
schema.propertyKey('age').Int().create()
schema.vertexLabel('person').properties('name','age').create()
```

```
schema.vertexLabel('person').index('search').search().by('name').by('age').add()
```

This example includes a search index by the integer property `age`. Here is data to query:

```
graph.addVertex(label, 'person', 'name', 'Julia', 'age', 56)
graph.addVertex(label, 'person', 'name', 'Emeril', 'age', 48)
graph.addVertex(label, 'person', 'name', 'Simone', 'age', 50)
graph.addVertex(label, 'person', 'name', 'James', 'age', 52)
```

and the query itself:

```
g.V().has('person', 'age', gt(50)).values()
```

to find all persons over the age of 50.

```
==>Julia
==>56
==>James
==>52
```

- To sort the previous search, add additional methods:

```
g.V().hasLabel("person").has("age", gt(50)).order().by("age",
incr).values()
```

to get:

```
==>James
==>52
==>Julia
==>56
```

## Simple Traversals

Simple traversals can be complex, but not employ specialized techniques such as recursion or branching.

Returning to the Recipe Toy Graph, let's expand the graph to include reviewers and ratings. Load the following script to add the `reviewer` vertices and `recipe-reviewer` edges. You must have inserted the [DSE Graph QuickStart data \(page 390\)](#) previously, so that the recipe vertices exist before loading this script:

```
// Generates review vertices and edges for Recipe Toy Graph
// :load /tmp/generateReviews.groovy

// reviewer vertices
johnDoe = graph.addVertex(label, 'reviewer', 'name', 'John Doe')
johnSmith = graph.addVertex(label, 'reviewer', 'name', 'John Smith')
janeDoe = graph.addVertex(label, 'reviewer', 'name', 'Jane Doe')
sharonSmith = graph.addVertex(label, 'reviewer', 'name', 'Sharon Smith')
betsyJones = graph.addVertex(label, 'reviewer', 'name', 'Betsy Jones')
```

```

beefBourguignon = g.V().has('recipe', 'name', 'Beef
Bourguignon').tryNext().orElseGet {graph.addVertex(label, 'recipe',
'name', 'Beef Bourguignon')}
spicyMeatLoaf = g.V().has('recipe', 'name', 'Spicy
Meatloaf').tryNext().orElseGet {graph.addVertex(label, 'recipe',
'name', 'Spicy Meatloaf')}
carrotSoup = g.V().has('recipe', 'name', 'Carrot
Soup').tryNext().orElseGet {graph.addVertex(label, 'recipe', 'name',
'Carrot Soup')}

// reviewer - recipe edges
johnDoe.addEdge('rated', beefBourguignon, 'timestamp',
'2014-01-01T05:15:00.00Z', 'stars', 5, 'comment', 'Pretty tasty!')
johnSmith.addEdge('rated', beefBourguignon, 'timestamp',
'2014-01-23T00:00:00.00Z', 'stars', 4)
janeDoe.addEdge('rated', beefBourguignon, 'timestamp',
'2014-02-01T00:00:00.00Z', 'stars', 5, 'comment', 'Yummy!')
sharonSmith.addEdge('rated', beefBourguignon, 'timestamp',
'2015-01-01T00:00:00.00Z', 'stars', 3, 'comment', 'It was okay.')
johnDoe.addEdge('rated', spicyMeatLoaf, 'timestamp',
'2015-12-31T10:56:00.00Z', 'stars', 4, 'comment', 'Really spicy - be
careful!')
sharonSmith.addEdge('rated', spicyMeatLoaf, 'timestamp',
'2014-07-23T00:30:00.00Z', 'stars', 3, 'comment', 'Too spicy for me.
Use less garlic.')
janeDoe.addEdge('rated', carrotSoup, 'timestamp',
'2015-12-30T01:20:00.00Z', 'stars', 5, 'comment', 'Loved this soup!
Yummy vegetarian!')

```

Run the script by first identifying the script, and then remotely executing it.

```
gremlin> :load /tmp/generateReviews.groovy
```

The recipes that were previously entered are queried to assign the result to recipe variables. The variables are then used to create the reviewer-recipe edges. These queries make use of two Apache Tinkerpop methods, `tryNext()` and `orElseGet()`; see the [Apache Tinkerpop Java API](#) for more information.

## Exploring recipe ratings

Check if the vertices are created by counting the number of vertices with the `reviewer` label.

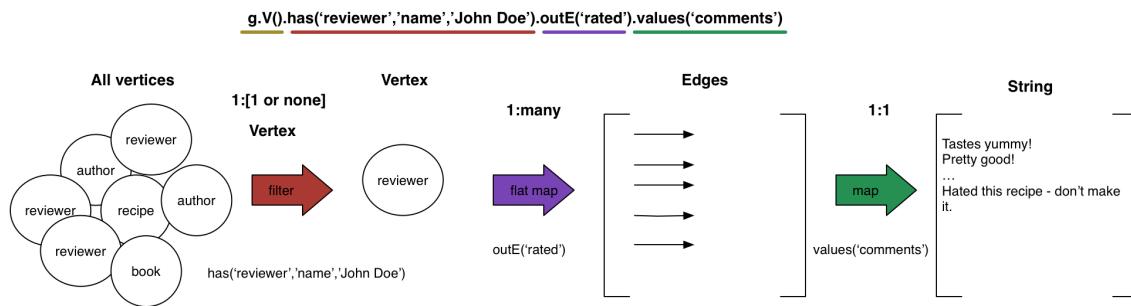
```
gremlin> g.V().hasLabel('reviewer').count()
==>5
```

List all the reviewer using values:

```
// Get the names of all the reviewers
gremlin> g.V().hasLabel('reviewer').values('name')
==>John Smith
==>Sharon Smith
==>Betsy Jones
==>Jane Doe
```

```
==>John Doe
```

Verifying that the reviewers are created is useful, but creating traversals that answer queries is more important. For instance, what does John Doe say about recipes?



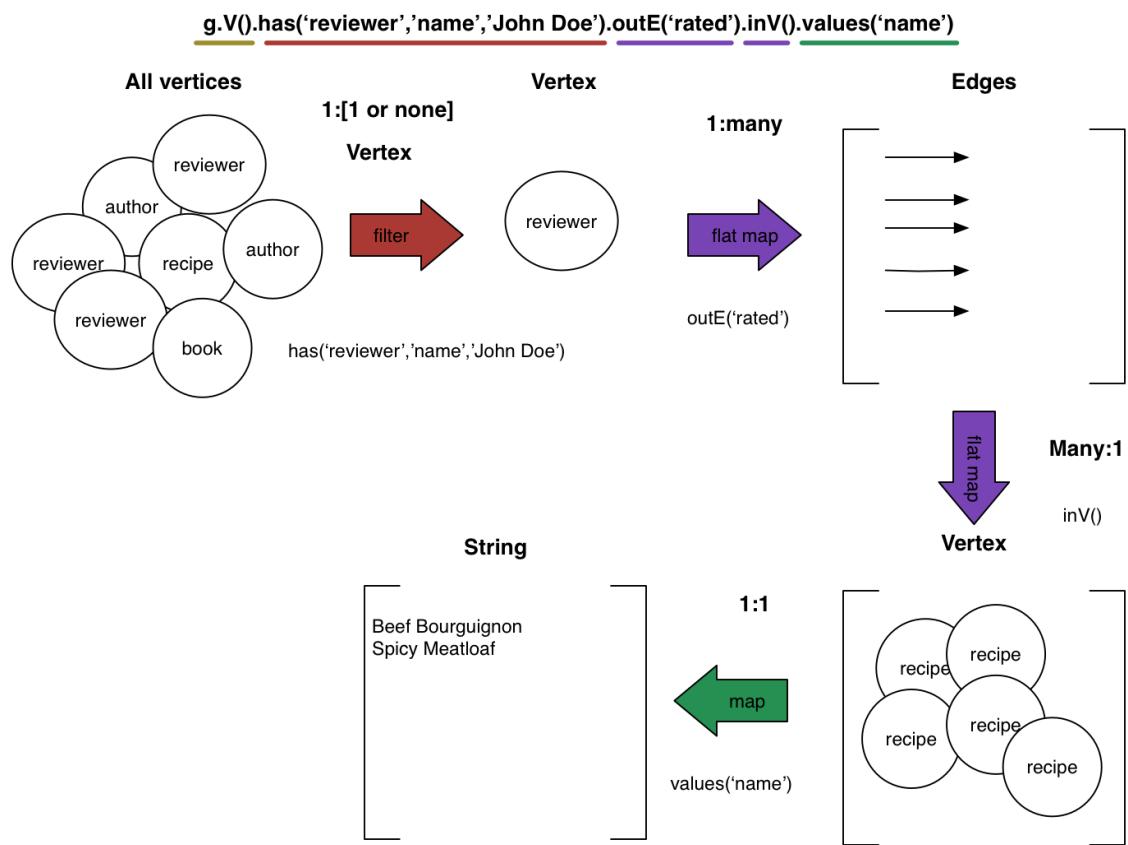
Use a query that identifies a vertex label as *reviewer* with a *name* value of *John Doe*.

```
g.V().has('reviewer', 'name', 'John Doe').outE('rated').values('comment')
```

The use of the outgoing edges command `outE('rated')` to find all the recipes that John Doe has rated allows the value of the property `comments` to be retrieved:

```
==>Pretty tasty!
==>Really spicy - be careful!
```

It might be nice to know which recipes John Doe reviewed, so another traversal can be used.



```

g.V().has('reviewer', 'name', 'John
Doe').outE('rated').inV().values('name')

```

resulting in:

```

==>Beef Bourguignon
==>Spicy Meatloaf

```

To find all the reviews that give a recipe more than 3 stars is a reasonable question to ask.  
Try a traversal using `gt(3)`, or *greater than 3* to filter the `stars` values:

```

gremlin> g.E().hasLabel('rated').has('stars', gt(3)).valueMap()
==>[stars:4, timestamp:2014-01-23T00:00:00Z]
==>[comment:Loved this soup! Yummy vegetarian!, stars:5,
timestamp:2015-12-30T00:00:00Z]
==>[comment:Yummy!, stars:5, timestamp:2014-02-01T00:00:00Z]
==>[comment:Pretty tasty!, stars:5, timestamp:2014-01-01T00:00:00Z]
==>[comment:Really spicy - be careful!, stars:4,
timestamp:2015-12-31T00:00:00Z]

```

The traversal shown finds each edge that is labeled `rated` and filters the edges found to output only those edges with a `star` rating of 4 or 5. But this traversal doesn't output

the answer to the original question. The traversal needs modification to get the incoming vertices with `inV()`, and to list those incoming vertices by name with `values('name')`:

```
gremlin> g.E().hasLabel('rated').has('stars',
 gt(3)).inV().values('name')
==>Beef Bourguignon
==>Spicy Meatloaf
==>Beef Bourguignon
==>Carrot Soup
==>Beef Bourguignon
```

The results indicate that `Beef Bourguignon` has been rated three times, although we don't have any reviewer information, just duplication of the recipe title in the list.

Returning to the previous query, let's look for more recent reviews. Adding an additional traversal step to filter by the `timestamp` can find the 4 and 5 star ratings using `gte(4)` or *greater than or equal to 4*, with a review date of Jan 1, 2015 or later.

```
gremlin> g.E().hasLabel('rated').has('stars',gte(4)).has('timestamp',
 gte(Instant.parse('2015-01-01T00:00:00.00Z'))).valueMap()
==>[comment:Loved this soup! Yummy vegetarian!,
 timestamp:2015-12-30T00:00:00Z, stars:5]
==>[comment:Really spicy - be careful!, timestamp:2015-12-31T00:00:00Z,
 stars:4]
```

Chaining traversal steps together can yield very exacting results. For instance, if we added the `inV().values('name')` to the last query, we'd now refine the results to find all 4-5 star reviews since the beginning of the year 2015.

Manipulating the ratings with statistical functions yields interesting answers. For instance, what is the *mean* value of all the recipe ratings?

```
gremlin> g.E().hasLabel('rated').values('stars').mean()
==>4.142857142857143
```

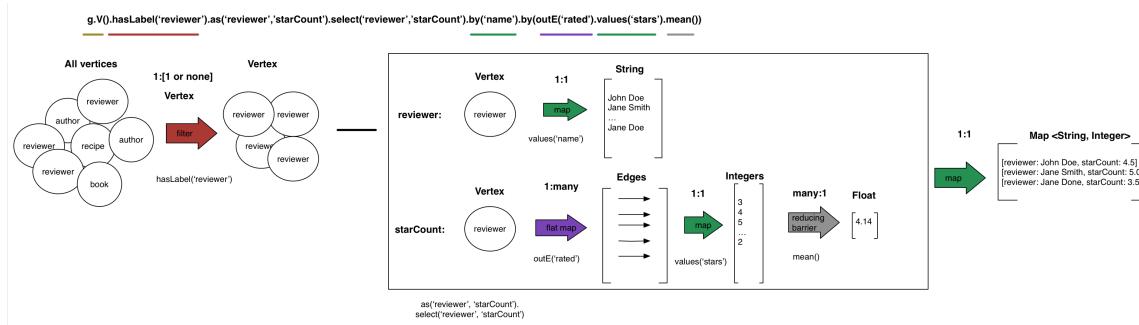
The results show that the reviewers like the recipes they reviewed, and establishes that reviewers in this sample did not write reviews for recipes that they did not like.

Perhaps a prolific reviewer would have a wider range of reviews. Find the maximum number of reviews that a single reviewer has written.

```
gremlin> g.V().hasLabel('reviewer').map(outE('rated').count()).max()
==>2
```

This traversal maps all the outgoing edges using `outE('rated')` of each reviewer and counts them, then determines which count has the highest value using `max()`.

Another measure that can be investigated is the mean rating of each reviewer. This traversal query uses a number of Apache TinkerPop traversal steps.



The `as()` step allows display labels to be created for the two items that will be lists, the reviewer's name and the mean stars value for each reviewer. These display labels, `reviewer` and `starCount` are then used in a `select()` step that gets each value, first the reviewer's name using `by('name')` and then the `starCount` using `by(outE('rated').values('stars').mean())`. The `select()` step checks each reviewer vertex and then traverses to discover the associated `starCount` value.

```

gremlin> g.V().hasLabel('reviewer').as('reviewer','starCount').
 select('reviewer','starCount').
 by('name').
 by(outE('rated').values('stars').mean())
==>[reviewer:Jane Doe, starCount:5.0]
==>[reviewer:Betsy Jones, starCount:NaN]
==>[reviewer:John Doe, starCount:4.5]
==>[reviewer:John Smith, starCount:4.0]
==>[reviewer:Sharon Smith, starCount:3.0]

```

Notice that Betsy Jones is listed as a reviewer, but has not reviewed any recipes. Her `starCount` lists `NaN` (not a number). It is clear from the results that Jane Doe really likes at least one recipe, while Sharon Smith does not.

Ordering the results by the `starCount`, or mean star rating, can allow the highest rater and the lowest rater to be discovered. Here, the traversal steps `order().by(select('starCount').decr())` use the output of the `select('starCount')` step to order the display in decremental order.

```

gremlin> g.V().hasLabel('reviewer').as('reviewer','starCount').
 select('reviewer','starCount').
 by('name').
 by(outE('rated').values('stars').mean()).
 order().by(select('starCount'), decr)
==>[reviewer:Betsy Jones, starCount:NaN]
==>[reviewer:Jane Doe, starCount:5.0]
==>[reviewer:John Doe, starCount:4.5]
==>[reviewer:John Smith, starCount:4.0]
==>[reviewer:Sharon Smith, starCount:3.0]

```

Betsy Jones and her lack of ratings still cause the listing to be incorrect. We could add a traversal step `limit(1)` to the traversal and get the highest rater, Jane Doe, if Betsy were not listed.

A tricky traversal step, `coalesce()`, is used to change NaN to a zero value.

```
gremlin> g.V().hasLabel('reviewer').as('reviewer', 'starCount').
 select('reviewer', 'starCount').
 by('name').
 by(coalesce(outE('rated').values('stars'), constant(0)).mean()).
 order().by(select('starCount'), decr)
==>[reviewer:Jane Doe, starCount:5.0]
==>[reviewer:John Doe, starCount:4.5]
==>[reviewer:John Smith, starCount:4.0]
==>[reviewer:Sharon Smith, starCount:3.0]
==>[reviewer:Betsy Jones, starCount:0.0]
```

Note that now Betsy Jones has a `starCount` of 0.0, the true value.

Find the star rating each reviewer has given to recipes:

```
g.V().hasLabel('reviewer').as('reviewer', 'rating').out().as('recipe').
select('reviewer', 'rating', 'recipe').
by('name').
by(outE('rated').values('stars')).
by(values('name'))
```

Note how the recipe name is traversed and named with the step modulator `as('recipe')` after the reviewer and rating are labeled from the reviewer vertices with `as('reviewer', 'rating')`. The first two items in the output listing are retrieved starting at the reviewer vertex while the third item is retrieved from the adjacent recipe vertex.

```
==>{reviewer=John Doe, rating=5, recipe=Beef Bourguignon}
==>{reviewer=John Doe, rating=5, recipe=Spicy Meatloaf}
==>{reviewer=John Smith, rating=4, recipe=Beef Bourguignon}
==>{reviewer=Jane Doe, rating=5, recipe=Beef Bourguignon}
==>{reviewer=Jane Doe, rating=5, recipe=Carrot Soup}
==>{reviewer=Sharon Smith, rating=3, recipe=Beef Bourguignon}
==>{reviewer=Sharon Smith, rating=3, recipe=Spicy Meatloaf}
```

In general, the most interesting statistic from the reviews answers the question about how many people rated a particular recipe, and what the mean rating is for that particular recipe. The graph traversal starts from a recipe vertex this time, and retrieves the recipe name, the number of reviews by counting the incoming edges with `inE('rated').count()`, and the mean value of the incoming edges with `inE('rated').values('stars').mean()`. The `coalesce()` step shown earlier could be used to change all NaN values for `meanRating` into zeroes.

```
g.V().hasLabel('recipe').as('recipe', 'numberOfReviews', 'meanRating').
select('recipe', 'numberOfReviews', 'meanRating').
by('name').
by(inE('rated').count()).
by(inE('rated').values('stars').mean())

==>{recipe=Beef Bourguignon, numberOfReviews=4, meanRating=4.25}
==>{recipe=Wild Mushroom Stroganoff, numberOfReviews=0, meanRating=NaN}
==>{recipe=Spicy Meatloaf, numberOfReviews=2, meanRating=3.5}
```

```

==>{recipe=Rataouille, numberOfReviews=0, meanRating=NaN}
==>{recipe=Salade Nicoise, numberOfReviews=0, meanRating=NaN}
==>{recipe=Roast Pork Loin, numberOfReviews=0, meanRating=NaN}
==>{recipe=Oysters Rockefeller, numberOfReviews=0, meanRating=NaN}
==>{recipe=Carrot Soup, numberOfReviews=1, meanRating=5.0}

```

## Searching recipes

A common query for recipes is finding recipes that contain a certain ingredient.

```

g.V().hasLabel('recipe').out().has('name', 'beef').in().hasLabel('recipe').values('name')

==>Beef Bourguignon

```

A modification allows a query that includes either one ingredient or another.

```

g.V().hasLabel('recipe').out().has('name', within('beef', 'carrots')).in().hasLabel('recipe')

==>Beef Bourguignon
==>Carrot Soup

```

Finding all the ingredients for a particular recipe is a common query.

```

g.V().match(
 __.as('a').hasLabel('ingredient'),
 __.as('a').in('includes').has('name', 'Beef Bourguignon')).

select('a').by('name')

```

This query uses a `match()` step to find a match for the ingredients used to make Beef Bourguignon. The traversal starts by filtering all vertices to find the ingredients, then traverses to the recipe vertices along the `includes` edges using `in('includes')`. This query also uses a Groovy double underscore variable as a private variable for the `match` method. The results are:

```

==>tomato paste
==>beef
==>onion
==>mashed garlic
==>butter

```

Although `inside()` is most commonly used for geospatial searches, the method can be used to find anything that falls within a particular range of values. An example is finding books that have a publishing date between 1960 and 1970:

```

g.V().has('book', 'year', inside(1960, 1970)).valueMap()

```

The results are:

```

==>{ISBN=[0-394-40135-2], year=[1968], name=[The French Chef Cookbook]}
==>{year=[1961], name=[The Art of French Cooking, Vol. 1]}

```

## Grouping output

Group output from a graph traversal using the `group()` traversal step. For example, display all the vertices by name, grouped by label:

```
g.V().group().by(label).by('name')
```

Note that the meals, ingredients, authors, books, recipes, and reviewers are all grouped in the results:

```
-->[meal:[JuliaDinner, Saturday Feast, EverydayDinner], ingredient:
[olive oil, chicken broth,
eggplant, pork sausage, green bell pepper, yellow onion, celery, hard-boiled egg, shallots,
zucchini, butter, green beans, mashed garlic, onion, mushrooms, bacon,
parsley, oyster,
tomato, thyme, pork loin, tuna, tomato paste, ground beef, red wine,
fennel, Pernod,
chervil, egg noodles, carrots, beef], author:[Louisette Bertholie,
Kelsie Kerr,
Alice Waters, Julia Child, Emeril Lagasse, Simone Beck, Patricia
Curtan, Patricia Simon,
James Beard, Fritz Streiff], book:[Simca's Cuisine: 100 Classic French
Recipes for Every
Occasion, The French Chef Cookbook, The Art of Simple Food: Notes,
Lessons, and Recipes
from a Delicious Revolution, The Art of French Cooking, Vol. 1],
recipe:[Wild Mushroom
Stroganoff, Roast Pork Loin, Spicy Meatloaf, Rataouille, Beef
Bourguignon, Oysters
Rockefeller, Salade Niçoise, Carrot Soup], reviewer:[Sharon Smith, John
Smith, Jane Doe,
Betsy Jones, John Doe]]
```

Another example groups all books by year and displays a listing of each year books were published followed by the book titles:

```
g.V().hasLabel('book').group().by('year').by('name')
```

and lists:

```
-->{1968=[The French Chef Cookbook, The French Chef Cookbook],
1972=[Simca's Cuisine: 100 Classic French Recipes for Every Occasion,
Simca's Cuisine: 100
Classic French Recipes for Every Occasion], 2007=[The Art of Simple
Food: Notes, Lessons,
and Recipes from a Delicious Revolution, The Art of Simple Food: Notes,
Lessons, and Recipes
from a Delicious Revolution], 1961=[The Art of French Cooking, Vol. 1,
The Art of French
Cooking, Vol. 1]}
```

## Grouping for processing using local()

Oftentimes, it is critical to do local processing for a particular step in the graph traversal. The next two examples use the `limit()` command to show how `local()` can change the processing from the whole stream entering the query to a portion of the query. First, find just two authors and the year that they have published books:

```
g.V().hasLabel('author').as('author').out().properties('year').as('year').
 select('author', 'year').
 by('name').
 by().
 limit(2)
```

This query results in returning the first two records in the database:

```
==>{author=Julia Child, year=vp[year->1961]}
==>{author=Julia Child, year=vp[year->1968]}
```

Using `local()`, change this query to find the first two books that each author in the graph has published:

```
g.V().hasLabel('author').as('author').
 local(out().properties('year').as('year').limit(2)).
 select('author', 'year').
 by('name').
 by()
```

Note that up to two books are displayed for each author:

```
==>{author=Julia Child, year=vp[year->1961]}
==>{author=Julia Child, year=vp[year->1968]}
==>{author=Simone Beck, year=vp[year->1961]}
==>{author=Simone Beck, year=vp[year->1972]}
==>{author=Louisette Bertholie, year=vp[year->1961]}
==>{author=Patricia Simon, year=vp[year->1972]}
==>{author=Alice Waters, year=vp[year->2007]}
==>{author=Patricia Curtan, year=vp[year->2007]}
==>{author=Kelsie Kerr, year=vp[year->2007]}
==>{author=Fritz Streiff, year=vp[year->2007]}
```

The traversal step `local()` has many applications for processing a subsection of a graph within a graph traversal to return results before moving on to further processing.

## Geospatial traversals

Creating geospatial traversal queries.

Geospatial queries are used to discover geospatial information. All geospatial data types (points, linestrings, and polygons) can be searched for specified values with simple queries. More interesting traversal queries discover points or linestrings within a radius from a specified point or within a specified geospatial polygon.

**Important:** All points must be specified in (longitude, latitude) following WKT format.

Distance calculations are crucial to proper results. DSE Search indexes can be created for geospatial data in DSE Graph, and DSE Search uses the [Haversine formula](#) to determine the great-circle distance between two points. DSE Search indexes cannot be created for polygons, but are essential to making geospatial point and linestring queries performant.

**Note:** A point of confusion can occur if the same geospatial query is run with or without a DSE Search index. Without a search index, geospatial queries always return exact results. DSE Search indexes, however, trade off write performance and index size for query accuracy with [two tunable parameters](#), `maxDistErr` (default: 0.000009) and `distErrPct` (default: 0.025). Inconsistent results in these two cases are due to the distance calculation algorithm variation of the default values of these parameters. DSE Graph can pass values for these two parameters when creating the search index. Change `maxDistErr` in `withError(maxDistErr, distErrPct)` to 0.0 to force both index-backed and non-index-backed queries to yield the same value:

```
schema.vertexLabel('location').index('search').search().by('point').withError(0.000009, 0.025)
```

## Schema and data

The examples here use the following schema:

```
// SCHEMA
// POINT
schema.propertyKey('name').Text().create()
schema.propertyKey('point').Point().withGeoBounds().create()
schema.vertexLabel('location').properties('name', 'point').create()
// LINESTRING
schema.propertyKey('line').Linestring().withGeoBounds().create()
schema.vertexLabel('lineLocation').properties('name', 'line').create()
// POLYGON
schema.propertyKey('polygon').Polygon().withGeoBounds().create()
schema.vertexLabel('polyLocation').properties('name', 'polygon').create()
// MATERIALIZED VIEW INDEXES
schema.vertexLabel('location').index('byname').materialized().by('name').add()
schema.vertexLabel('lineLocation').index('byname').materialized().by('name').add()
schema.vertexLabel('polyLocation').index('byname').materialized().by('name').add()
//SEARCH INDEX - ONLY WORKS FOR POINT AND LINESTRING
schema.vertexLabel('location').index('search').search().by('point').add()
schema.vertexLabel('lineLocation').index('search').search().by('line').add()
```

The example use the following data:

```
// Create a point
graph.addVertex(label, 'location', 'name', 'Paris', 'point', Geo.point(2.352222,
48.856614))
graph.addVertex(label, 'location', 'name', 'London', 'point', Geo.point(-0.127758, 51.507351))
graph.addVertex(label, 'location', 'name', 'Dublin', 'point', Geo.point(-6.26031,
53.349805))
```

```

graph.addVertex(label, 'location', 'name', 'Aachen', 'point',Geo.point(6.083887,
50.775346))
graph.addVertex(label, 'location', 'name', 'Tokyo', 'point',Geo.point(139.691706,
35.689487))

// Create a linestring
graph.addVertex(label, 'lineLocation', 'name', 'ParisLondon', 'line',
"LINESTRING(2.352222 48.856614, -0.127758 51.507351)")
graph.addVertex(label, 'lineLocation', 'name', 'LondonDublin', 'line',
"LINESTRING(-0.127758 51.507351, -6.26031 53.349805)")
graph.addVertex(label, 'lineLocation', 'name', 'ParisAachen', 'line',
"LINESTRING(2.352222 48.856614, 6.083887 50.775346)")
graph.addVertex(label, 'lineLocation', 'name', 'AachenTokyo', 'line',
"LINESTRING(6.083887 50.775346, 139.691706 35.689487)")

// Create a polygon
graph.addVertex(label, 'polyLocation', 'name', 'ParisLondonDublin',
'polygon',Geo.polygon(2.352222, 48.856614, -0.127758, 51.507351,
-6.26031, 53.349805))
graph.addVertex(label, 'polyLocation', 'name', 'LondonDublinAachen',
'polygon',Geo.polygon(-0.127758, 51.507351, -6.26031, 53.349805,
6.083887, 50.775346))
graph.addVertex(label, 'polyLocation', 'name', 'DublinAachenTokyo',
'polygon',Geo.polygon(-6.26031, 53.349805, 6.083887, 50.775346,
139.691706, 35.689487))

```

The example data has the following approximate distances:

```

// PARIS TO LONDON: 3.08 DEGREES; 344 KM; 214 MI; 344,000 M
// PARIS TO AACHE: 3.07 DEGREES; 343 KM; 213 MI; 343,000 M
// PARIS TO DUBLIN: 7.02 DEGREES; 781 KM; 485 MI; 781,000 M
// PARIS TO TOKYO: 86.3 DEGREES; 9713 KM; 6035 MI; 9,713,000 M

```

## Find stored geospatial data that matches specified information

Find the stored data that matches a point mapped to the specified (longitude, latitude):

```

g.V().
has('location','point', Geo.point(2.352222, 48.856614)).
valueMap()

```

results in:

```
{name=[Paris], point=[POINT (2.352222 48.856614)]}
```

Find the stored data that matches a line mapped to the specified points:

```

g.V().
has('lineLocation','line',Geo.lineString(2.352222, 48.856614,
-0.127758, 51.507351)).
valueMap()

```

results in:

```
{line=[LINESTRING (2.352222 48.856614, -0.127758 51.507351)],
name=[ParisLondon]}
```

Find the stored data that matches a polygon mapped to the specified points:

```
g.V().
has('polyLocation', 'polygon',Geo.polygon(2.352222, 48.856614,
-0.127758, 51.507351, -6.26031, 53.349805)).
valueMap()
```

results in:

```
{polygon=[POLYGON ((2.352222 48.856614, -0.127758 51.507351, -6.26031
53.349805, 2.352222 48.856614))], name=[ParisLondonDublin]}
```

## Find stored geospatial points or linestrings within a specified radius from a specified point

These queries, as well as the queries that use a specified geospatial polygon use a method `Geo.inside()` that specifies a point, a radius, and the units to be used.

Several units are available with use of the `Geo.inside()` method:

### DEGREES

Degrees of distance. One degree of latitude is approximately 111.2 kilometers, whereas one degree of longitude depends on the distance from the equator.

At the equator, one degree of longitude equals 111.2 kilometers, but at 45 degrees of latitude, one degree of longitude is 78.6 kilometers. While the physical distance over a single degree of longitude changes with latitude, we calculate only [great-circle distances](#) in degrees.

### KILOMETERS

Kilometers of distance.

### MILES

Miles of distance.

### METERS

Meters of distance.

Find all the cities (points) within a radius from a particular location (centerpoint):

```
g.V().
has('location', 'point', Geo.inside(Geo.point(2.352222, 48.856614),
4.2, Geo.Unit.DEGREES)).
values('name')
```

lists:

```
==>Paris
==>London
==>Aachen
```

Centering the query on Paris and searching within 4.2 degrees returns three cities: Paris, London, and Aachen from the dataset.

Find all the linestrings within a radius from a particular location (centerpoint):

```
g.V().
has('lineLocation', 'line', Geo.inside(Geo.point(2.352222, 48.856614),
 9713, Geo.Unit.KILOMETERS)).
values('name')
```

lists:

```
==>ParisLondon
==>LondonDublin
==>AachenTokyo
==>ParisAachen
```

Centering the query on Paris and searching within 9713 kilometers returns four stored linestrings: Paris to London, London to Dublin, Aachen to Tokyo, and Paris to Aachen. Note that London to Dublin was not a stored linestring.

## Find stored geospatial points or linestrings within a specified geospatial polygon

Polygons may be used in these queries with a search index on *point* if a JAR file from the [JTS library is added to DSE Search](#). Be sure to add the JAR file before attempting to insert the graph data.

Find all cities (points) within a specified geospatial polygon:

```
g.V().
has('location', 'point', Geo.inside(Geo.polygon(-6.26031, 53.349805,
 6.083887, 50.775346, 139.691706, 35.689487))).
values('name')
```

lists:

```
==>Dublin
==>Aachen
==>Tokyo
```

This results is not surprising, since the three points used to create the polygon represent the three cities discovered.

Find all cities (points) within a specified geospatial polygon generated with a [WKT tool](#):

```
g.V().
has('location', 'point', Geo.inside(Geo.polygon(-7.9541015625,
 55.148273231753834,-9.6240234375, 51.47539580264131,1.0986328125,
 50.86924482345238,0.5712890625, 53.29887631763788,-7.9541015625,
 55.148273231753834))).
values('name')
```

lists:

```
==>London
==>Dublin
```

The polygon used encompasses most of the Republic of Ireland as well as the southern half of the United Kingdom, and finds London and Dublin within the polygon.

find linestrings within a polygon

```
g.V().
has('lineLocation', 'line', Geo.inside(Geo.polygon(-6.26031, 53.349805,
6.083887, 50.775346, 139.691706, 35.689487))).
values('name')
```

lists:

```
==>AachenTokyo
```

Since two of the points in the specified polygon represent Aachen and Tokyo, it is reassuring that the linestring of Aachen to Tokyo is found.

## Schema and data

The examples here use the following schema:

```
//SCHEMA
// PROPERTY KEYS
// Check for previous creation of property key with ifNotExists()
schema.propertyKey('name').Text().ifNotExists().create()
schema.propertyKey('gender').Text().ifNotExists().create()
schema.propertyKey('location').Point().withGeoBounds().ifNotExists().create()
// VERTEX LABELS
schema.vertexLabel('author').properties('name', 'gender').ifNotExists().create()
schema.vertexLabel('place').properties('name', 'location').create()
// EDGE LABELS
schema.edgeLabel('livesIn').connection('author', 'place').ifNotExists().create()
// VERTEX INDEXES
// Secondary
schema.vertexLabel('author').index('byName').secondary().by('name').add()
// Search
schema.vertexLabel('author').index('search').search().by('name').asString().ifNotExists()
schema.vertexLabel('place').index('search').search().by('location').ifNotExists().add();
```

The examples use the following data:

```
//VERTICES
// AUTHOR VERTICES
juliaChild = graph.addVertex(label, 'author', 'name', 'Julia Child',
'gender', 'F')
simoneBeck = graph.addVertex(label, 'author', 'name', 'Simone Beck',
'gender', 'F')
louisetteBertholie = graph.addVertex(label, 'author', 'name',
'Louisette Bertholie', 'gender', 'F')
patriciaSimon = graph.addVertex(label, 'author', 'name', 'Patricia
Simon', 'gender', 'F')
aliceWaters = graph.addVertex(label, 'author', 'name', 'Alice Waters',
'gender', 'F')
patriciaCurtan = graph.addVertex(label, 'author', 'name', 'Patricia
Curtan', 'gender', 'F')
```

```

kelsieKerr = graph.addVertex(label, 'author', 'name', 'Kelsie Kerr',
 'gender', 'F')
fritzStreiff = graph.addVertex(label, 'author', 'name', 'Fritz
 Streiff', 'gender', 'M')
emerilLagasse = graph.addVertex(label, 'author', 'name', 'Emeril
 Lagasse', 'gender', 'M')
jamesBeard = graph.addVertex(label, 'author', 'name', 'James Beard',
 'gender', 'M')

// PLACE VERTICES
newYork = graph.addVertex(label, 'place', 'name', 'New York',
 'location', Geo.point(74.0059,40.7128));
paris = graph.addVertex(label, 'place', 'name', 'Paris', 'location',
 Geo.point(2.3522, 48.8566));
newOrleans = graph.addVertex(label, 'place', 'name', 'New Orleans',
 'location', Geo.point(90.0715, 29.9511));
losAngeles = graph.addVertex(label, 'place', 'name', 'Los Angeles',
 'location', Geo.point(118.2437, 34.0522));
london = graph.addVertex(label, 'place', 'name', 'London', 'location',
 Geo.point(-0.1278, 51.5074));
chicago = graph.addVertex(label, 'place', 'name', 'Chicago',
 'location', Geo.point(-87.6298, 41.8781136));
tokyo = graph.addVertex(label, 'place', 'name', 'Tokyo', 'location',
 Geo.point(139.6917, 35.6895));

// EDGES
juliaChild.addEdge('livesIn', newYork);
simoneBeck.addEdge('livesIn', paris);
louisetteBertholie.addEdge('livesIn', london);
patriciaSimon.addEdge('livesIn', newYork);
aliceWaters.addEdge('livesIn', losAngeles);
patriciaCurtan.addEdge('livesIn', chicago);
kelsieKerr.addEdge('livesIn', tokyo);
fritzStreiff.addEdge('livesIn', tokyo);
emerilLagasse.addEdge('livesIn', newOrleans);
jamesBeard.addEdge('livesIn', london);

```

Of course, this data can be loaded using the [DSE Graph Loader \(page 471\)](#) as well, from CSV or other formatted files.

## Find authors who live within a certain distance from a specified city in sorted order

First list the place names for all cities within the given radius (50 degrees) from New York (the approximate centerpoint listed:

```

g.V().
has('place', 'location',
 Geo.inside(Geo.point(74.0,40.5),50,Geo.Unit.DEGREES)).
values('name')

```

results in:

```
=>New York
```

```
==>Paris
==>New Orleans
==>Los Angeles
```

Now list the place names and authors who live in those cities for all cities within the given radius (50 degrees) from New York (the approximate centerpoint), sorted in alphabetical order:

```
// Order by name, not by distance from location point given
g.V().has('place', 'location',
 Geo.inside(Geo.point(74.0,40.5),50,Geo.Unit.DEGREES)).
order().by('name').
as('Location').
in().as('Author').
select('Location','Author').
by('name').
by('name')
```

finds:

```
==>{Location=Los Angeles, Author=Alice Waters}
==>{Location>New Orleans, Author=Emeril Lagasse}
==>{Location>New York, Author=Patricia Simon}
==>{Location>New York, Author=Julia Child}
==>{Location=Paris, Author=Simone Beck}
```

This query uses some additional methods such as `order()` and `select()` that are explained in [Simple Traversals \(page 582\)](#).

Now list the place names and authors who live in those cities for all cities within the given radius (50 degrees) from New York (the approximate centerpoint), sorted by the distance from the centerpoint:

```
// Order by distance from NYC
g.V().has('place', 'location',
 Geo.inside(Geo.point(74.0,40.5),50,Geo.Unit.DEGREES)).
order().by{it.value('location').getOgcGeometry().distance(Geo.point(74.0059,40.7128).getOg
as('Location').
in().as('Author').
select('Location', 'Author').
by('name').
by('name')

==>{Location>New York, Author=Patricia Simon}
==>{Location>New York, Author=Julia Child}
==>{Location>New Orleans, Author=Emeril Lagasse}
==>{Location>Los Angeles, Author=Alice Waters}
==>{Location=Paris, Author=Simone Beck}
```

This query introduces some additional methods that must be imported in order for the query to succeed: `getOgcGeometry()` and `distance()`. Importing the library is accomplished in the original script using:

```
import com.esri.core.geometry.ogc.OGCGeometry;
```

## Cartesian spatial traversals

Creating Cartesian spatial traversal queries.

Cartesian spatial queries are used to discover Cartesian (graphable) information. All Cartesian data types (points, linestrings, and polygons) can be searched for specified values with simple queries. More interesting traversal queries discover points or linestrings within a radius from a specified point or within a specified spatial polygon.

DSE Search indexes can be created to decrease the latency in response time, but are not required. Create schema to use a search index for `point` and `linestring` properties in the [Cartesian schema \(page 446\)](#).

### Schema and data

The examples here use the following schema:

```
// SCHEMA
// POINT
schema.propertyKey('name').Text().create()
schema.propertyKey('point').Point().withBounds(-3, -3, 3, 3).create()
schema.vertexLabel('location').properties('name', 'point').create()
// LINESTRING
schema.propertyKey('line').Linestring().withBounds(-3, -3, 3,
 3).create()
schema.vertexLabel('lineLocation').properties('name', 'line').create()
// POLYGON
schema.propertyKey('polygon').Polygon().withBounds(-3, -3, 3,
 3).create()
schema.vertexLabel('polyLocation').properties('name', 'polygon').create()
// MATERIALIZED VIEW INDEXES
schema.vertexLabel('location').index('byname').materialized().by('name').add()
schema.vertexLabel('lineLocation').index('byname').materialized().by('name').add()
schema.vertexLabel('polyLocation').index('byname').materialized().by('name').add()
//SEARCH INDEX - ONLY WORKS FOR POINT AND LINESTRING
schema.vertexLabel('location').index('search').search().by('point').add()
schema.vertexLabel('lineLocation').index('search').search().by('line').add()
```

The example use the following data:

```
/// Create a point
graph.addVertex(label, 'location', 'name', 'p0', 'point', Geo.point(0.5,0.5))
graph.addVertex(label, 'location', 'name', 'p1', 'point', Geo.point(1,1))
graph.addVertex(label, 'location', 'name', 'p2', 'point', Geo.point(-1,1))
graph.addVertex(label, 'location', 'name', 'p3', 'point', Geo.point(-2,-2))
graph.addVertex(label, 'location', 'name', 'p4', 'point', Geo.point(2,2))

// Create a linestring
graph.addVertex(label, 'lineLocation', 'name', 'l1', 'line',
 "LINESTRING(0 0, 1 1)")
graph.addVertex(label, 'lineLocation', 'name', 'l2', 'line',
 "LINESTRING(0 0, -1 1)")
```

```
graph.addVertex(label, 'lineLocation', 'name', 'l3', 'line',
 "LINESTRING(0 0, -2 -2)")
graph.addVertex(label, 'lineLocation', 'name', 'l4', 'line',
 "LINESTRING(0 0, 2 -2)")

// Create a polygon
graph.addVertex(label, 'polyLocation', 'name', 'g1',
 'polygon',Geo.polygon(0,0,1,1,0,1,0,0))
graph.addVertex(label, 'polyLocation', 'name', 'g2',
 'polygon',Geo.polygon(0,0,0,1,-1,1,0,0))
graph.addVertex(label, 'polyLocation', 'name', 'g3',
 'polygon',Geo.polygon(0,0,-2,0,-2,-2,0,0))
graph.addVertex(label, 'polyLocation', 'name', 'g4',
 'polygon',Geo.polygon(0,0,2,0,2,-2,0,0))
```

## Find stored Cartesian spatial data that matches specified information

Find the stored data that matches a point mapped to the specified (x, y):

```
g.V().
has('location','point', Geo.point(0.5, 0.5)).
valueMap()
```

results in:

```
{name=[p0], point=[POINT (0.5 0.5)]}
```

Find the stored data that matches a line mapped to the specified points:

```
g.V().
has('lineLocation','line',Geo.lineString(0, 0, 1, 1)).
valueMap()
```

results in:

```
{line=[LINESTRING (0 0, 1 1)], name=[l1]}
```

Find the stored data that matches a polygon mapped to the specified points:

```
g.V().
has('polyLocation', 'polygon',Geo.polygon(0,0,1,1,0,1,0,0)).
valueMap()
```

results in:

```
{polygon=[POLYGON ((0 0, 1 1, 0 1, 0 0))], name=[g1]}
```

## Find stored Cartesian spatial points or linestrings within a specified radius from a specified point

These queries, as well as the queries that use a specified Cartesian spatial polygon use a method `Geo.inside()` that specifies a point and a radius.

Find all the points within a radius from a particular point (centerpoint):

```
g.V().
has('location', 'point', Geo.inside(Geo.point(0, 0), 1)).
values('name')
```

lists:

```
==>p0
```

Centering the query on (0, 0) and searching within 1 unit returns one point, *p0*, from the dataset.

Find all the linestrings within a radius from a particular location (centerpoint):

```
g.V().has('lineLocation', 'line', Geo.inside(Geo.point(0.0, 0.0),
1.415)).valueMap()
```

lists:

```
==>{line=[LINESTRING (0 0, 1 1)], name=[l1]}
==>{line=[LINESTRING (0 0, -1 1)], name=[l2]}
```

Centering the query on (0,0) and searching within 1.415 units returns two stored linestrings: *l1* and *l2*.

## Find stored Cartesian spatial points or linestrings within a specified Cartesian spatial polygon

Polygons may be used in these queries if a JAR file from the [JTS library is added to DSE Search](#). Be sure to add the JAR file before attempting to insert the graph data.

Find all points within a specified Cartesian spatial polygon:

```
g.V().
has('location', 'point', Geo.inside(Geo.polygon(0, 0, 1, 0, 1, 1,
0, 1, 0, 0))).
values('name')
```

lists:

```
==>p0
```

find linestrings within a polygon

```
g.V().
has('lineLocation', 'line', Geo.inside(Geo.polygon(0, 0, 1, 0, 1, 1,
0, 1, 0, 0))).
values('name')
```

lists:

```
==>l1
```

## Schema and data

The examples here use the following schema:

```

///SCHEMA
// PROPERTY KEYS
// Check for previous creation of property key with ifNotExists()
schema.propertyKey('name').Text().ifNotExists().create()
schema.propertyKey('address').Text().ifNotExists().create()
schema.propertyKey('location').Point().withBounds(-100,-100,100,100).ifNotExists().create()
// VERTEX LABELS
schema.vertexLabel('person').properties('name').ifNotExists().create()
schema.vertexLabel('home').properties('address','location').ifNotExists().create()
schema.vertexLabel('store').properties('name','location').ifNotExists().create()
schema.vertexLabel('ingredient').properties('name').ifNotExists().create()
// EEDGE LABELS
schema.edgeLabel('livesIn').connection('person','home').ifNotExists().create()
schema.edgeLabel('isStockedWith').connection('store','ingredient').multiple().ifNotExists()

// SEARCH INDEXES
schema.vertexLabel('person').index('search').search().by('name').asString().ifNotExists()
schema.vertexLabel('home').index('search').search().by('name').by('location').add()
schema.vertexLabel('store').index('search').search().by('name').by('location').add();
schema.vertexLabel('ingredient').index('search').search().by('name').add();

```

The examples use the following data:

```

//VERTICES
// PERSON VERTICES
pam = graph.addVertex(label, 'person', 'name', 'Pam')
les = graph.addVertex(label, 'person', 'name', 'Les')
paul = graph.addVertex(label, 'person', 'name', 'Paul')
victoria = graph.addVertex(label, 'person', 'name', 'Victoria')
terri = graph.addVertex(label, 'person', 'name', 'Terri')

// HOME VERTICES
home1 = graph.addVertex(label, 'home', 'address', '555 4th St',
'location', Geo.point(7,2));
home2 = graph.addVertex(label, 'home', 'address', '1700 Coyote Rd',
'location', Geo.point(-2,1));
home3 = graph.addVertex(label, 'home', 'address', '99 Mountain Pass
Hwy', 'location', Geo.point(0,0));

// STORE VERTICES
store1 = graph.addVertex(label, 'store', 'name', 'ZippyMart',
'location', Geo.point(1,5));
store2 = graph.addVertex(label, 'store', 'name', 'Quik Station',
'location', Geo.point(7,-1));
store3 = graph.addVertex(label, 'store', 'name', 'Mamma's Grocery',
'location', Geo.point(-3,-3));

// INGREDIENT VERTICES
celery = graph.addVertex(label, 'ingredient','name', 'celery');
milk = graph.addVertex(label, 'ingredient','name', 'milk');
bokChoy = graph.addVertex(label, 'ingredient','name', 'bok choy');
steak = graph.addVertex(label, 'ingredient','name', 'steak');

```

```

carrots = graph.addVertex(label, 'ingredient','name', 'carrots');
porkChops = graph.addVertex(label, 'ingredient','name', 'pork chops');

// PERSON - HOME EDGES
pam.addEdge('livesIn', home1);
les.addEdge('livesIn', home1);
paul.addEdge('livesIn', home3);
victoria.addEdge('livesIn', home3);
terri.addEdge('livesIn', home2);

// STORE - INGREDIENT EDGES
store1.addEdge('isStockedWith', milk);
store1.addEdge('isStockedWith', bokChoy);
store1.addEdge('isStockedWith', steak);
store2.addEdge('isStockedWith', steak);
store2.addEdge('isStockedWith', carrots);
store2.addEdge('isStockedWith', porkChops);
store3.addEdge('isStockedWith', milk);
store3.addEdge('isStockedWith', carrots);
store3.addEdge('isStockedWith', celery);

```

## Finding celery

You are a mathematics teacher writing simple Cartesian problem for your students. They are great fans of *ants on a log*, a snack made with celery, cream cheese, and raisins. So, you decide to help them find the nearest store to their house which has celery in stock.

Paul is the student whose home we'll use as the starting point. First, list all stores within the given radius (10 units of distance) from Paul's home (the centerpoint):

```

g.V().has('store', 'location', Geo.inside(Geo.point(0,0),10)).
values('name')

```

results in:

```

==>ZippyMart
==>Quik Station
==>Mamma's Grocery

```

Note that this exercise is using Cartesian coordinates and distances calculated between Cartesian points, but a similar exercises can use [geospatial data \(page 597\)](#).

Now list the stores within a 10 unit radius from Paul's home that have celery:

```

g.V().has('store', 'location',
Geo.inside(Geo.point(0,0),10)).as('Store').
out().has('name','celery').as('Ingred').
select('Store', 'Ingred').
by('name').
by('name')

```

finds:

```

==>{Store=ZippyMart, Ingred=celery}
==>{Store=Quik Station, Ingred=celery}

```

```
==>{Store=Mamma's Grocery, Ingred=celery}
```

This query uses methods that are common, such as `as()`, `out()`, and `select()` that are explained in [Simple Traversals \(page 582\)](#) to narrow the query.

Finally, list the stores within a 10 unit radius of Paul's home that have celery, and sort them by the distance from Paul's home:

```
// List store name, location, and ingredient in order by distance from
// the store
g.V().has('store', 'location',
 Geo.inside(Geo.point(0,0),25)).as('Store').
order().by{it.value('location').getOgcGeometry().distance(Geo.point(0,0).getOgcGeometry())
as('Location')).
out().has('name','celery').as('Ingred').
select('Store', 'Location','Ingred').
by('name').
by('location').
by('name')

==>{Store=Mamma's Grocery, Location=POINT (-3 -3), Ingred=celery}
==>{Store=ZippyMart, Location=POINT (1 5), Ingred=celery}
==>{Store=Quik Station, Location=POINT (7 -1), Ingred=celery}
```

This query adds the method `order()` to sort the results; it is also explained in [Simple Traversals \(page 582\)](#). The query must also use a method that must be imported in order for the query to succeed: `getOgcGeometry()` and `distance()`. Importing the library is accomplished in the original script using:

```
import com.esri.core.geometry.ogc.OGCGeometry;
```

The students working on this problem now know that *Mamma's Grocery* is the place to head to get the celery they need to make their favorite snack!

## Branching Traversals

Branching traversals allow decision points to be inserted into the traversal processing.

Branching traversals allow decision points to be inserted into the traversal processing. Prior to trying out branching traversals shown here, you must create the data as described in [Simple Traversals \(page 582\)](#).

This branching traversal example chooses between two labels, either `author` or `reviewer` to fork the traversal. If the vertex label is `author`, the edges labeled `created` are counted. If the vertex label is `reviewer`, the edges labeled `rated` are counted.

```
g.V().choose(label()).
 option('author', out('created').count()).
 option('reviewer', out('rated').count())
```

The output for this traversal lists each result, the count returned. This type of traversal is useful as an intermediary step in a query process, but clearly the output is not useful without reference.

```
==>0
```

```

==>0
==>2
==>0
==>0
==>2
==>1
==>0
==>0
==>0
==>3
==>0
==>2
==>2
==>1
==>2

```

## Recursive Traversals

Recursive traversals allow iterative processing over traversal paths.

Recursive traversals allow iterative processing over traversal paths. Prior to trying out branching traversals shown here, you must create the data as described in [Simple Traversals \(page 582\)](#).

This recursive traversal example returns the names of vertices that are two outgoing steps from the `author` vertex named *Julia Child* using the `times(2)` step. Books, meals, and ingredients are returned by this query.

```
g.V().has('name','Julia Child').repeat(out()).times(2).valueMap()
```

The output for this traversal lists each result:

```

==>{name=[onion]}
==>{name=[beef]}
==>{name=[mashed garlic]}
==>{name=[butter]}
==>{name=[tomato paste]}
==>{name=[JuliaDinner], calories=[900],
 timestamp=[2016-01-14T00:00:00Z]}
==>{year=[1961], name=[The Art of French Cooking, Vol. 1]}
==>{name=[Saturday Feast], calories=[1000],
 timestamp=[2015-11-30T00:00:00Z]}
==>{name=[olive oil]}
==>{name=[green beans]}
==>{name=[tuna]}
==>{name=[hard-boiled egg]}
==>{name=[tomato]}
==>{name=[JuliaDinner], calories=[900],
 timestamp=[2016-01-14T00:00:00Z]}
==>{year=[1961], name=[The Art of French Cooking, Vol. 1]}
==>{name=[olive oil]}
==>{name=[yellow onion]}
==>{name=[zucchini]}

```

```
=>{name=[mashed garlic]}\n=>{name=[eggplant]}
```

## Path Traversals

Path traversals map traversal steps to a location to use in the event that a previous location must be revisited.

Path traversals map traversal steps to a location to use in the event that a previous location must be revisited.

This path traversal starts at an ingredient, traverses to a recipe, and eventually finds a book that contains the recipe with the ingredients specified.

```
g.V().has('ingredient',\n 'name',within('beef','carrots')).in('includes').as('Recipe') .\n out().hasLabel('book').as('Book') .\n select('Book','Recipe') .\n by('name').by('name').path()
```

The output for this traversal lists each result:

```
=>[v[{\~label=ingredient, member_id=2, community_id=1442590464}],\n v[{\~label=recipe, member_id=2, community_id=473764096}],\n v[{\~label=book, member_id=0, community_id=568859392}],\n {Book=The Art of French Cooking, Vol. 1, Recipe=Beef Bourguignon}]\n=>[v[{\~label=ingredient, member_id=1, community_id=684566272}],\n v[{\~label=recipe, member_id=0, community_id=1462084224}],\n v[{\~label=book, member_id=1, community_id=1620680576}],\n {Book=The Art of Simple Food: Notes, Lessons, and Recipes from a\n Delicious Revolution, Recipe=Carrot Soup}]
```

Another path traversal creates a tree that emanates from a vertex label, in this case a book.

```
g.V().hasLabel('book').in().tree().by('name').next()
```

The output for this traversal lists each result:

```
=>Simca's Cuisine: 100 Classic French Recipes for Every Occasion=\n {Patricia Simon={}, Simone Beck={}}\n=>The Art of French Cooking, Vol. 1=\n {Simone Beck={}, Julia Child={}, Beef Bourguignon={}, Louisette\n Bertholie={}, Salade Nicoise={}}\n=>The Art of Simple Food: Notes, Lessons, and Recipes from a Delicious\n Revolution=\n {Alice Waters={}, Kelsie Kerr={}, Roast Pork Loin={}, Carrot Soup={},\n Fritz Streiff={}, Patricia Curtan={}}\n=>The French Chef Cookbook=\n {Julia Child={}}
```

Each book lists the authors and recipes that are included in the book.

Another tree traversal discovers all the vertices that are on outgoing tree branch from a recipe.

```
g.V().hasLabel('recipe').out().tree().by('name').next()
```

The output for this traversal lists each result:

```
==>Roast Pork Loin=
{red wine={}, pork loin={}, chicken broth={}, The Art of Simple Food:
Notes, Lessons, and Recipes from a Delicious Revolution={}}
==>Spicy Meatloaf=
{bacon={}, celery={}, pork sausage={}, onion={}, ground beef={}, green
bell pepper={}}
==>Beef Bourguignon=
{mashed garlic={}, butter={}, The Art of French Cooking, Vol. 1={},
onion={}, tomato paste={}, beef={}}
==>Carrot Soup=
{butter={}, onion={}, chicken broth={}, carrots={}, The Art of Simple
Food: Notes, Lessons, and Recipes from a Delicious Revolution={},
thyme={}}
==>Rataouille=
{mashed garlic={}, yellow onion={}, olive oil={}, zucchini={},
eggplant={}}
==>Salade Nicoise=
{tuna={}, The Art of French Cooking, Vol. 1={}, hard-boiled egg={},
olive oil={}, tomato={}, green beans={}}
==>Wild Mushroom Stroganoff=
{mushrooms={}, yellow onion={}, egg noodles={}}
==>Oysters Rockefeller=
{oyster={}, chervil={}, parsley={}, celery={}, fennel={}, shallots={},
Pernod={}}
```

Each recipe lists the ingredients for the recipe and the books that include the recipe.

## DSE Graph Analysis with DSE Analytics

Perform analytics jobs on graph data using DSE.

### DSE Graph and Graph Analytics

DSE Graph allows you to perform OLAP queries using Spark.

Many local graph traversals can be executed in real time at high transactional loads.

When the density of the graph is too high or the branching factor too large (the number of connected nodes at each level of the graph), the memory and computation requirements to answer OLTP queries go beyond what is acceptable under typical application workloads. These type of queries are called *deep queries*.

*Scan queries* are queries that touch either an entire graph or large parts of the graph. They typically traverse a large number of vertices and edges. For example, a query on a social network graph that searches for friends of friends is a scan query.

For applications that use deep and scan queries, using a OLAP query will result in better performance.

## Performing OLAP queries using DSE Graph

Every graph created in DSE Graph has an OLAP traversal source `a` that is available to `gremlin-console` and DataStax Studio. This traversal source uses the `SparkGraphComputer` to analyze queries and execute them against the underlying DSE Analytics nodes. The nodes must be started with Graph and Spark enabled to access the OLAP traversal source. You must connect to the [Spark Master \(page 217\)](#) node for the datacenter. For one-off or single-session OLAP queries, alias `database.a` to `g` and create the query. For example in the Gremlin console:

```
:remote config alias g database.a

g.V().count()
```

If you are performing multiple queries against different parts of the graph, use `graph.snapshot()` to return an OLAP traversal source for each part of the graph. For example, in the Gremlin console:

```
categories = graph.snapshot().vertices('category1',
'category2').create()
```

To create a snapshot, supply all the vertices the snapshot will traverse. For example, the following query touches both the `Person` and `Address` vertices.

```
def person = graph.snapshot().vertices('Person', 'Address').create()
person.V().hasLabel('Person').out('HAS_ADDRESS').count()
```

Use the `conf()` method on the snapshot before you call `create()` to set [TinkerPop's SparkGraphComputer configuration options](#). For example, to explicitly set the storage level for the snapshot to `MEMORY_ONLY`:

```
graph.snapshot().vertices("vertexlabel_alice",
"vertexlabel_bob").edges("edgelabel_carol").conf("gremlin.spark.persistStorageLevel",
"MEMORY_ONLY").create()
```

## Setting Spark properties from Gremlin

[Spark properties \(page 226\)](#) can be set from Gremlin using the `graph.configuration.setProperty` method on the graph.

```
:remote config alias g database.a

g.graph.configuration.setProperty("property name", value)
```

By default, Spark applications will use all available resources on the node, so no other Spark application can run. Limit the application's resources before running OLAP traversals by setting the maximum number of cores and the amount of memory used by the traversal. This is particularly important on servers with very large amounts of cores and memory.

For example this request sets 10 executors with 1 core and 4 GB of memory each:

```
:remote config alias g example_graph.a
```

```
==>g=example_graph.a

g.graph.configuration.setProperty("spark.cores.max", 10)
g.graph.configuration.setProperty("spark.executor.memory", "4g")
g.graph.configuration.setProperty("spark.executor.cores", "1")
```

The `spark.cores.max` property sets the maximum number of cores used by Spark. Setting this property lower than the total number of cores limits the number of nodes on which the queries will be run. The `spark.executor.memory` property sets the amount of memory used for each executor. The `spark.executor.cores` property sets the number of cores used for each executor.

Before you configure Spark properties from Gremlin kill the currently-running Spark context from the [Spark web UI \(page 202\)](#). This will kill all currently running Gremlin OLAP queries. From the Spark web UI, find the application named Apache TinkerPop's Spark-Gremlin and click `kill` next to the Application ID.

OLAP traversals create many intermediate objects during execution. These objects are garbage-collected by the JVM, so we recommend configuring a larger pool of executors each with smaller memory and CPU resources, compared to non-graph Spark jobs which typically perform better with fewer executors with higher memory and CPU resources.

We recommend allocating executors with no more than 8 cores (1 should work in most cases) to reduce garbage collection pauses and improve OLAP traversal performance. The memory available to Spark should be equally spread among the cores. For example, if you have 3 nodes and each has 24 cores and 96 GB dedicated to Spark you have  $24 * 3 = 72$  cores and  $96 \text{ GB} * 3 = 188 \text{ GB}$  memory. To allocate all resources you should request 72 single core executors with 4 GB of memory each:

```
:remote config alias g example_graph.a
==>g=example_graph.a

g.graph.configuration.setProperty("spark.cores.max", 72)

g.graph.configuration.setProperty("spark.executor.memory", "4g")
g.graph.configuration.setProperty("spark.executor.cores", "1")
```

Some OLAP queries and most `DseGraphFrame` queries use Spark SQL joins for traversals. Spark has a predefined number of partitions to perform merge joins, by default set to 200. This can create huge Spark partitions for very large graphs, which slows query execution.

To reduce the size of single partitions and improve performance increase the number of shuffle partitions by setting the `spark.sql.shuffle.partitions` property to a larger number. We recommend the `spark.sql.shuffle.partitions` is set to 2-4 times the number of Spark cluster cores. So if you have a 200 core cluster, set `spark.sql.shuffle.partitions` to 400 or 800.

```
:remote config alias g example_graph.a
==>g=example_graph.a

g.graph.configuration.setProperty("spark.sql.shuffle.partitions", 500)
```

## When to use analytic OLAP queries

On large graphs, OLAP queries typically perform better for deep queries. However, executing deep queries as part of an OLTP load may make sense if they are rarely performed. For example, on online payment provider will favor OLTP queries to process payments quickly, but may require a deep query if there are indications of fraud in the transaction. While the deep query may take much longer as an OLTP workload, on the whole the performance of the application will be faster than segmenting the application into OLTP and OLAP queries.

Long running and periodic processes like recommendation engines and search engines that analyze an entire graph are the ideal use cases for OLAP queries. However, one-off data analysis operations that involve deep queries or that scan the entire database also can benefit from being run as OLAP queries. See [DSE Graph, OLTP, and OLAP \(page 404\)](#) for detailed information on performance differences between OLTP and OLAP queries.

## Best practices for deleting large numbers of edges and vertices

When deleting large numbers of edges or vertices from a graph, you may end up getting error messages in subsequent queries due the large number of tombstones left in the database before they are automatically removed.

The log entries for such errors resemble the following:

```
ERROR [ReadStage-32] 2017-05-18 11:18:29,289 StorageProxy.java:2068
- Scanned over 100001 tombstones during query 'SELECT * FROM
t33215.PhoneNumber_p WHERE token(community_id) > -7331398285705078207
AND token(community_id) <= -6858404847917653807 LIMIT 1000' (last
scanned row partition key was ((216134144), 1250272)); query aborted
```

To avoid these errors, reduce the number of tombstones per request by setting the `spark.cassandra.input.split.size_in_mb` property to a smaller size than the default of 64 MB. The `spark.cassandra.input.split.size_in_mb` property sets the approximate size of data the Spark Cassandra Connector will request with each individual CQL query.

The following example shows how to set the `spark.cassandra.input.split.size_in_mb` property to 1 MB and then to drop all phone number vertices from a graph.

```
:remote config alias g example_graph.a

g.graph.configuration.setProperty("spark.cassandra.input.split.size_in_mb",
"1")
g.V().hasLabel("PhoneNumber").drop().iterate()
```

## DSE authentication and OLAP queries

If DSE authentication is enabled, the internal user `dse_inproc_user` runs the application, not the user who submitted the Graph OLAP query.

## Using the DseGraphFrame framework for graph analytics queries

The DseGraphFrame framework provides the Spark base API for analytics operations on DSE Graph.

The `DseGraphFrame` framework allows you create applications that use the Spark API for analytics operations on DSE Graph. It is inspired by the Databricks `GraphFrame` library and supports a subset of the Gremlin graph traversal language. You can read DSE Graph data into a `GraphFrame` and write `GraphFrame` objects from any format supported by Spark into DSE Graph. You can also query `DseGraphFrame` vertices and edges in [Spark SQL \(page 237\)](#).

## Choosing when to use `DseGraphFrame` or DSE Graph OLAP queries

[DSE Graph OLAP \(page 607\)](#) has broader support for Gremlin than the `DseGraphFrame` API. While Graph OLAP is the best choice for deep queries, simple filtering and counts are much faster using the `DseGraphFrame` API.

## Overview of `DseGraphFrame`

`DseGraphFrame` represents a graph as two virtual tables: a vertex and an edge `DataFrame`. The `V()` method returns the vertex `DataFrame` of a graph. The `E()` method returns the edge `DataFrame` of a graph.

```
val g = spark.dseGraph("test")
g.V.show
g.E.show
```

`DseGraphFrame` uses a `GraphFrame`-compatible format. This format requires the vertex `DataFrame` to have only one `id` column and the edge `DataFrame` to have hard coded `src` and `dst` columns. Since DSE Graph allows users to define any arbitrary set of columns as the vertex `id` and since there is no concept of labels in `GraphFrame`, `DseGraphFrame` will serialize the entire DSE Graph `id` into one `id` column. The label is represented as part of the `id` and also as the `~label` property column.

## Using `DseGraphFrame`

The starting point for all operations is the `DseGraphFrame` object. In Scala, there's an implicit conversion between `DseGraphFrame` objects and `GraphFrame` objects.

```
// load a graph
val graph = spark.dseGraph("my_graph")
//use the TinkerPop API
graph.V().has("edge", gt(100)).count().next()
// use the GraphFrame API
graph.find("(a)-[e]->(b); (b)-[e2]->(c)").filter("e2.~label` =
'includedIn'").select("a.name", "e.~label`", "b.name", "e2.~label`",
"c.name").distinct.show
// Use both the TinkerPop and GraphFrame APIs:
graph.V().out().hasLabel("label").df.show
```

In Java, use the `gf()` method, or use the `DseGraphFrameBuilder.dseGraph(String graphName, GraphFrame gf)` method to return a `GraphFrame` instance.

```
//load a graph
GraphFrame graph = DseGraphFrameBuilder.dseGraph("my_graph", spark);
//use the TinkerPop API
graph.V().has("edge", gt(100)).count().next()
```

```
// use the GraphFrame API
graph.find("(a)-[e]-(b); (b)-[e2]-(c)").filter("e2.label =
'includedIn'").select("a.name", "e.`~label`", "b.name", "e2.`~label`",
"c.name").distinct().show()
// Use both the TinkerPop and GraphFrame APIs:
graph.V().out().hasLabel("label").df().show()
```

Before doing complex queries, it is strongly recommended you cache the graph. You can do so using the `cache()` or `persist(level)` methods.

```
g.cache()
```

The `persist()` method requires one of the Spark persist levels as a parameter.

```
g.persist(MEMORY_AND_DISK_SER)
```

**Table 11: DseGraphFrame method list**

Method	Description
<code>gf()</code>	Returns a <code>GraphFrame</code> object.
<code>V()</code>	Returns a <code>DseGraphTraversal[Vertex]</code> object to start a vertex traversal.
<code>E()</code>	Returns a <code>DseGraphTraversal[Edge]</code> object to start an edge traversal.
<code>cache()</code>	Cache the graph data with Spark.
<code>persist(level)</code>	Cache the graph data with one of the <a href="#">Spark persist levels</a> .
<code>deleteVertices()</code>	Delete vertices.
<code>deleteVertices(label: String)</code>	Delete all vertices with the specified label.
<code>deleteEdges()</code>	Delete edges.
<code>deleteVertexProperties()</code>	Delete vertex properties.
<code>deleteEdgeProperties()</code>	Delete edge properties.
<code>updateVertices(df: DataFrame, labels: Seq[String] = Seq.empty)</code>	Change the properties of an existing vertex or insert a new vertex. The optional parameter <code>labels</code> improves the performance of the update by iterating over the provided labels rather than deriving the vertices to update from the <code>DataFrame</code> .

Method	Description
<code>updateEdges(df: DataFrame, labels: Seq[String] = Seq.empty)</code>	<p>Change the properties of an existing edge or insert a new edge.</p> <p>The optional parameter <code>labels</code> improves the performance of the update by iterating over the provided labels rather than deriving the edges to update from the <code>DataFrame</code>.</p>

## Configuring DSE Smart Analytics query routing

By default DSE Graph uses the `DseGraphFrameInterceptorStrategy` which will automatically intercept `count`, `groupCount`, and `drop` queries and route them to `DseGraphFrame` to improve performance. These simpler queries skip `StarGraph` RDD creation, allowing for faster execution times.

If a `count` or `groupCount` query is against a snapshot or has a path length longer than 2, DSE will not intercept the query.

To disable `DseGraphFrameInterceptorStrategy`, call the `withoutStrategies` method on the graph.

```
g.withoutStrategies(com.datastax.bdp.graph.impl.tinkerpop.optimizer.DseGraphFrameIntercept
 .v()
 .count()
```

## Using authorization with `DseGraphFrame`

If you have enabled authorization on DSE, [grant execute permissions](#) to the `DseGraphRpc` object.

## TinkerPop API support in `DseGraphFrame`

`DseGraphFrame` supports a subset of the Apache TinkerPop traversal API.

`DseGraphFrame` supports a subset of the Apache TinkerPop traversal API.

`DseGraphFrame` does not support

`org.apache.tinkerpop.gremlin.process.traversal.Traverser` or  
`org.apache.tinkerpop.gremlin.process.traversal.TraversalSideEffects`.

## Supported methods

`DseGraphFrame` mimics the TinkerPop graph traversal source by defining two methods: `E()` and `v()`. These methods return a `GraphTraversal` that has all methods defined below. Only a limit set of TinkerPop's Step classes are supported. Steps other than the ones in the following table will throw an `UnsupportedException`.

**Table 12: TinkerPop read methods**

Steps	Methods
<code>CountGlobalStep</code>	<code>count()</code>
<code>GroupCountStep</code>	<code>groupCount()</code>

Steps	Methods
IdStep	id()
PropertyValuesStep	values()
PropertyMapStep	propertyMap()
HasStep	has(), hasLabel()
IsStep	is()
VertexStep	to(), out(), in(), both(), toE(), outE(), inE(), bothE()
EdgeVertexStep	toV(), inV(), outV(), bothV()
NotStep	not()
TraversalFilterStep	where()
AndStep	and(A, B)
PageRankVertexProgramStep	pageRank()
DedupGlobalStep	dedup()
OrderGlobalStep	order()
LimitGlobalStep	limit()
SelectStep	as() and select()
OrStep	or()

This query finds people who know each other and demonstrates the `as()` and `select()` methods:

```
g.V().as("a").out("knows").as("b").out("knows")
.where(P.eq("a")).select("a", "b").by("name").show
```

```
+----+----+
| a | b |
+----+----+
| Alice| Bob |
| Bob | Alice|
+----+----+
```

**Table 13: TinkerPop update steps and methods**

Steps	Methods
DropStep	V().drop(), E().drop(), properties().drop()
AddPropertyStep	property(name, value, ...)

`DseGraphFrame` can be used to drop millions of vertices or edges at once, and is much faster for bulk property updates than Gremlin OLAP or OLTP.

For example this query drops all `person` vertices and their associated edges:

```
g.V().hasLabel("person").drop().iterate()
```

## Using DseGraphFrame in Scala

`GraphTraversal` is a Java interface, and extends the Java `Iterator` interface. To iterate through the results of a traversal as a `DataFrame` use the `df()` method. `DseGraphFrame` supports implicit conversion to `DataFrame`.

The following example will traverse the vertices of a graph using TinkerPop and then show the result as a `DataFrame`.

```
g.V().out().show
```

In some cases you may need to use the TinkerPop Java API to get the correct TinkerPop objects.

For example, to extract the DSE Graph `Id` object the `Traversal` Java iterator can be converted to a Scala iterator which allows direct access to the TinkerPop representation of the `Id`. This method allows you to use the original `Id` instead of the `DataFrame` methods which return the `DataFrame String` representation of the `Id`, you can also use the `toList()` and `toSet()` methods to set the appropriate ID.

```
import scala.collection.JavaConverters._
for(i <- g.V().id().asScala) println (i)

{~label=vertex, community_id=748226688, member_id=0}
{~label=custom, name=Name, value=1}

g.V.id.toSet

res18: java.util.Set[Object] = [{~label=demigod,
 community_id=224391936, member_id=0}, ...]
```

The TinkerPop `P (predicate)` and `T (constant)` classes are imported by the Spark shell automatically.

```
g.E().groupCount().by(T.label)
g.V().has("age", P.gt(30)).show
```

For standalone applications, import these classes.

```
import org.apache.tinkerpop.gremlin.structure.T
import org.apache.tinkerpop.gremlin.process.traversal.P
import org.apache.tinkerpop.gremlin.process.traversal.dsl.graph._
```

Scala is not always able to infer the return type, especially in the Spark shell. The property values of the type should be provided explicitly.

```
g.V().values[Any]("name").next()
```

Or similarly:

```
val n: String = g.V().values("name").next()
```

Explicitly set the type when dropping properties.

```
g.V().properties[Any]("age", "name").drop().iterate()
```

In this case, using the `DataFrame` API is easier as you do not need to specify the type.

```
g.V().properties("age", "name").drop().show()
```

```
++
||
++
++
```

```
g.V().values("age").show()
```

```
+---+
| age |
+---+
| 10000 |
```

**Table 14: Using Java methods in DseGraphFrame Scala applications**

Method	Use case	Example
<code>hasNext()</code>	You want to know if there's a result, but you don't care about the value.	Did Alice create any other vertices <pre>g.V().has("name", "Alice").outE("created").hasNext()</pre>
<code>next()</code>	You know that there is at least 1 result and you want to get the first one (or the second if you call it twice, and so on).	Get the vertex label distribution. Group steps will always return exactly 1 result. <pre>g.V().groupCount().by(label).next()</pre>
<code>iterate()</code>	You just want to execute the traversal, but don't care about the result and whether it did anything at all.	Set all person's ages to 10. <pre>g.V().property("age", 10).iterate()</pre>

Method	Use case	Example
<code>toList()</code> , <code>toSet()</code>	You expect the result to contain an arbitrary number of items and you want to get all of them.	Get all the people Alice knows.  <code>g.V().has("name", "Alice").out("knows").toList()</code>

## Mapping rules for DseGraphFrame

DseGraphFrame uses mapping rules for column names and types.

DseGraphFrame uses mapping rules for column names and types.

### Column mapping rules

DataFrame column names are the same as graph property names except in the following cases.

- Conflict with column names reserved by GraphFrame will result in an underscore (\_) added to the property name. For example, the `id` column will result in a property named `_id`.

**Table 15: Reserved column names in GraphFrame**

Reserved column name
<code>id</code>
<code>src</code>
<code>dst</code>
<code>new_id</code>
<code>new_src</code>
<code>new_dst</code>
<code>graphx_attr</code>

- DseGraphFrame and Spark SQL are case insensitive by default. Column names that differ only in case will result in conflicts. Set the Spark property `spark.sql.caseSensitive=true` to avoid case conflicts.

```
dse spark --conf spark.sql.caseSensitive=true
```

### Type mapping rules

DseGraphFrame and Spark SQL have a limited set of supported types. A vertex is represented by a `Row` instance.

If the vertex has multiple properties, each property will be represented as a Spark SQL array with property values. If a property has meta-properties it will be represented as `StructType`. The `value` field of the struct contains the property value. All other fields will represent the meta-properties.

**Table 16: DSE Graph to Spark SQL and DseGraphFrame type mapping**

DSE Graph type	Spark SQL type	Conversion rules
boolean	BooleanType	
smallint	ShortType	
int	IntegerType	
bigint	LongType	
float	FloatType	
double	DoubleType	
decimal	DecimalType(38, 18)	
varint	DecimalType(38, 0)	
timestamp	TimestampType	
date	DateType	
time	LongType	The number of nanoseconds from the beginning of the day.
text	StringType	
uuid	StringType	The <code>UUID.toString()</code> and <code>UUID.fromString()</code> methods are used to convert the value.
inet	StringType	The <code>toString</code> and <code>InetAddress.getByName()</code> methods are used to convert the value.
blob	BinaryType	
'PointType'	StringType	The <code>toWellKnownText()</code> and <code>fromWellKnownText()</code> methods are used to convert the value.

DSE Graph type	Spark SQL type	Conversion rules
'LineStringType'	StringType	The <code>toWellKnownText()</code> and <code>fromWellKnownText()</code> methods are used to convert the value.
'PolygonType'	StringType	The <code>toWellKnownText()</code> and <code>fromWellKnownText()</code> methods are used to convert the value.

## Exporting graphs using DseGraphFrame

Use DseGraphFrame to export the graph to any format supported by Spark.

Use DseGraphFrame to export the graph to any format supported by Spark.

1. Export the vertices and edges to a Spark-supported using the `write` method.

Export the graph to a JSON file in the DSEFS file system.

```
g.V.write.json("/tmp/v_json")
g.E.write.json("/tmp/e_json")
```

That will create two directories in the DSEFS file system with vertex and edge JSON files. You can get data locally if they are not too large for the local file system:

```
dse hadoop fs -cat /tmp/v_json/* > local_vertices.json &&
dse hadoop fs -cat /tmp/e_json/* > local_edges.json
```

## Importing graphs using DseGraphFrame

Use DseGraphFrame to import a graph to DataStax Enterprise.

Use DseGraphFrame to import a graph to DataStax Enterprise.

### Prerequisites:

This feature is experimental.

The graph schema should be created manually in the Gremlin console or DataStax Studio before importing the graph. Import only works with custom ID mapping.

1. Start the Spark shell.

```
dse spark
```

2. If you exported the graph to JSON using DseGraphFrame, import it in the Spark shell.

```
val g = spark.dseGraph("gods_import")
g.updateVertices(spark.read.json("/tmp/v.json"))
g.updateEdges(spark.read.json("/tmp/e.json"))
```

```
val g = spark.dseGraph("graph name")
g.updateVertices(spark.read.json("path to exported vertices JSON"))
g.updateEdges(spark.read.json("path to exported edges JSON"))
```

**3.** If you have a custom graph:

- a.** Examine the schema of the graph and note how to map it to the expected schema of a DSE Graph schema.

This example will use the `friends` graph from the [GraphFrame project](#).

```
import org.graphframes._
val g: GraphFrame = examples.Graphs.friends
g.vertices.printSchema
```

```
root
|-- id: string (nullable = true)
|-- name: string (nullable = true)
|-- age: integer (nullable = false)
```

```
g.edges.printSchema
```

```
root
|-- src: string (nullable = true)
|-- dst: string (nullable = true)
|-- relationship: string (nullable = true)
```

- b.** In the Gremlin console or DataStax Studio create the schema.

```
system.graph('friends').create()
:remote config alias g friends.g
 schema.propertyKey("age").Int().create()
 schema.propertyKey("name").Text().create()
 schema.propertyKey("id").Text().single().create()

 schema.vertexLabel('people').partitionKey("id").properties("name",
 "age").create();
 schema.edgeLabel("friend").create()
 schema.edgeLabel("follow").create()
```

- c.** In the Spark shell create an empty `DseGraphFrame` graph and check the target schemas.

```
val d = spark.dseGraph("friends")
```

```
d.V.printSchema
```

```
root
|-- id: string (nullable = false)
|-- ~label: string (nullable = false)
|-- _id: string (nullable = true)
|-- name: string (nullable = true)
|-- age: integer (nullable = true)
```

```
d.E.printSchema
```

```

root
|-- src: string (nullable = false)
|-- dst: string (nullable = false)
|-- ~label: string (nullable = true)
|-- id: string (nullable = true)

```

**d.** Convert the edges and vertices to the target format.

```

val v = g.vertices.select($"id" as "_id", lit("people") as
 "~label", $"name", $"age")
val e = g.edges.select(d.idColumn(lit("people")), $"src"
 as "src", d.idColumn(lit("people")), $"dst" as "dst",
 $"relationship" as "~label")

```

**e.** Append the converted vertices and edges to the target graph.

```

d.updateVertices (v)
d.updateEdges (e)

```

## Using the Northwind demo graph with Spark OLAP jobs

Run OLAP queries against the Northwind demo graph data.

The Northwind demo included with the DSE demos has a script for creating a graph of the data for a fictional trading company.

In this task, you'll use the Gremlin console to create the Northwind graph, snapshot part of the graph, and run a count operation on the subgraph using the `SparkGraphComputer`.

**Prerequisites:**

- [Enable DSE Graph, DSE Search, and DSE Analytics modes \(page 867\)](#) in your datacenter.
- Install the [DSE Graph Loader \(page 472\)](#).
- Clone the [graph-examples](#) Git repository to the machine on which you are running the Gremlin console.

```
git clone https://github.com/datastax/graph-examples.git
```

**1.** Load the Northwind graph and supplemental data using the `graphloader` tool:

```

graphloader -graph northwind -address localhost graph-examples/
northwind/northwind-mapping.groovy -inputpath graph-examples/
northwind/data &&
graphloader -graph northwind -address localhost graph-examples/
northwind/supplemental-data-mapping.groovy -inputpath graph-
examples/northwind/data/

```

**2.** Start the Gremlin console using the `dse gremlin-console` command:

```
dse gremlin-console
```

3. Alias the traversal to Northwind graph using the default OLTP traversal source:

```
:remote config alias g northwind.g
```

4. Set the schema mode to Development.

To allow modifying the schema for the connected graph database, you must set the mode to Development each session. The default schema mode for DSE Graph is Production, which doesn't allow you to modify the graph's schema.

```
schema.config().option('graph.schema_mode').set('Development')
```

5. Enable the use of scans and lambdas.

```
schema.config().option('graph.allow_scan').set('true')
graph.schema().config().option('graph.traversal_sources.g.restrict_lambda').set(false)
```

6. Look at the schema of the northwind graph:

```
schema.describe()
```

7. Alias the traversal to the Northwind analytics OLAP traversal source a. Alias g to the OLAP traversal source for one-off analytic queries:

```
:remote config alias g northwind.a
```

```
==>g=northwind.a
```

8. Count the number of vertices using the OLAP traversal source:

```
g.V().count()
```

```
==>3294
```

When you alias g to the OLAP traversal source `database name.a`, DSE Analytics is the workload back-end.

9. Store subgraphs into snapshots using `graph.snapshot()`.

When you need to run multiple OLAP queries on a graph in one session, use snapshots of the graph as the traversal source.

```
employees = graph.snapshot().vertices('employee').create()
```

```
==>graphtraversalsource[hadoopgraph[persistedinputrdd->persistedoutputrdd], sparkgraphcomputer]
```

```
categories = graph.snapshot().vertices('category').create()
```

```
==>graphtraversalsource[hadoopgraph[persistedinputrdd->persistedoutputrdd], sparkgraphcomputer]
```

The `snapshot()` method returns an OLAP traversal source using the `SparkGraphComputer`.

## 10. Run an operation on the snapshot graphs.

Count the number of employee vertices in the snapshot graph:

```
employees.V().count()
```

```
==> 9
```

Count the number of category vertices in the snapshot graph:

```
categories.V().count()
```

```
==> 8
```

## DSE Graph Tools

Introduce tools available for DSE Graph.

In addition to the Gremlin console, other tools are available for working with DSE Graph:

### DataStax Studio (page 874)

Web-based notebook-style visualization tool. Currently supports Markdown and Gremlin. Includes a variety of list and graph functions.

```
gremlin> g.V().count()
53
gremlin> g.V().hasLabel('recipe').values('recipeTitle').limit(10)
"Spicy Meatloaf",
"Oysters Rockefeller",
"Carrot Soup",
"Beef Bourguignon",
"Ratatouille",
"Salade Niçoise",
"Roast Pork Loin",
"Wild Mushroom Stroganoff"
```

### DSE OpsCenter

Visual management and monitoring tool.



## DSE Lifecycle Manager

Powerful provisioning and configuration management tool.

The figure shows a screenshot of the OpsCenter Lifecycle Manager: Deploy interface. On the left, there is a sidebar with options: Clusters, Jobs, SSH Credentials, Repositories, and Config Profiles. 'Config Profiles' is selected and highlighted in blue. In the main area, the title is 'Lifecycle Manager: Config Profiles'. Below it, there is a sub-section titled 'DSE5GraphCP'. It shows the last modified and created times as '5/26/2016, 11:02PM UTC' and '5/26/2016, 11:00PM UTC' respectively. There are 'Save' and 'Cancel' buttons. A list of files under 'Config Profile' includes: Cassandra (cassandra.yaml, dse.yaml, cassandra-env.sh, logback.xml), Hadoop (hive-site.xml), and Spark. To the right, there are fields for 'Name \*' (set to 'DSE5GraphCP'), 'DataStax Enterprise Version \*' (set to 'dse v5.0.0'), and a 'Comment' field. A note at the bottom states: 'DSE Version cannot be edited after saving the Config Profile. After upgrading DSE, add a Config Profile with the new DSE version. Apply the new Config Profile where appropriate.'

## Starting the Gremlin console

Starting the Gremlin console for an interactive environment.

Gremlin is the query language used to interact with DSE Graph. One method of inputting Gremlin code is to use the Gremlin console. The Gremlin console is a useful interactive environment for directly inputting Gremlin to create graph schema, load data, administer graph, and retrieve traversal results. The Gremlin Console is an interface to the Gremlin Server that can interact with DSE Graph.

- Start the Gremlin console using the `dse` command and passing the additional command `gremlin-console`:

```
bin/dse gremlin-console
```

```
\.,./
(o o)
----o0oo-(3)-o0oo----
plugin activated: tinkerpop.server
plugin activated: tinkerpop.utilities
plugin activated: tinkerpop.tinkergraph
gremlin>
```

Three plugins are activated by default, as shown. The Gremlin Server, `tinkerpop.server`, is started so that commands can be issued to DSE Graph. The utilities plugin, `tinkerpop.utilities`, provides various functions, helper methods and imports of external classes that are useful in Gremlin console. TinkerGraph, an in-memory graph that is used as an intermediary for some graph operations is started with `tinkerpop.tinkergraph`. The Gremlin console automatically connects to the remote Gremlin Server.

**Note:** The Gremlin console packaged with DataStax Enterprise does not allow plugin installation like the Gremlin console packaged with Apache TinkerPop.

- Gremlin console help can be displayed with the `-h` flag:

**Tip:** Use `-v` to display all lines when loading a file, to discover which line of code causes an error.

- Run the Gremlin console with the `host:port` option to specify a specific host and port:

```
bin/dse gremlin-console 127.0.0.1:8182
```

Any hostname or IP address will work to specify the host.

- Run Gremlin console with the `-e` flag to execute one or more scripts:

```
bin/dse gremlin-console -e test1.groovy -e test2.groovy
```

If the scripts run successfully, the command will return with the prompt after execution. If errors occur, the standard output will show the errors.

- If you prefer to have Gremlin console open at the script completion, run Gremlin console with the `-i` flag instead of the `-e` flag:

```
bin/dse gremlin-console -i test1.groovy -i test2.groovy
```

If the scripts run successfully, the command will return with the Gremlin console prompt after execution. If errors occur, the console will show the errors.

- Discover all Gremlin console commands with `help`. Console commands are not Gremlin language commands, but rather commands issued to the Gremlin console for shell functionality. The Gremlin console is based on the [Groovy shell](#).

```
:help
```

For information about Groovy, visit:  
<http://groovy-lang.org>

Available commands:

```
:help (:h) Display this help message
? (:?) Alias to: :help
:exit (:x) Exit the shell
:quit (:q) Alias to: :exit
import (:i) Import a class into the namespace
:display (:d) Display the current buffer
:clear (:c) Clear the buffer and reset the prompt counter.
:show (:S) Show variables, classes or imports
:inspect (:n) Inspect a variable or the last result with the
GUI object browser
:purge (:p) Purge variables, classes, imports or
preferences
:edit (:e) Edit the current buffer
:load (:l) Load a file or URL into the buffer
. (:..) Alias to: :load
:save (:s) Save the current buffer to a file
:record (:r) Record the current session to a file
:history (:H) Display, manage and recall edit-line history
:alias (:a) Create an alias
:register (:rc) Registers a new command with the shell
:doc (:D) Opens a browser window displaying the doc for
the argument
:set (:=) Set (or list) preferences
```

```

:uninstall (:-) Uninstall a Maven library and its dependencies
from the Gremlin Console
:install (:+) Install a Maven library and its dependencies
into the Gremlin Console
:plugin (:pin) Manage plugins for the Console
:remote (:rem) Define a remote connection
:submit (:>) Send a Gremlin script to Gremlin Server

For help on a specific command type:
:help command

```

The Gremlin Console provides code help via auto-complete functionality, using the <TAB> key to trigger a list of possible options.

**Note:** :install and :plugin should not be used with DSE Graph. These commands will result in gremlin console errors.

## DSE Graph Reference

Reference commands and other information for DSE Graph.

### The graph API

Reference guide to graph commands.

graph commands add data to an existing graph.

#### addEdge

How to specify an edge.

#### Synopsis

```
vertex1.addEdge('edgeLabel', vertex2, [T.id, 'edge_id'], ['key',
'value'] [,....])
```

#### Description

Edge data is inserted using addEdge. A previously created [edge label \(page 637\)](#) must be specified. An edge\_id may be specified, to upsert data for a multiple cardinality edge to prevent creation of a new edge. Property key-value pairs may be optionally specified.

#### Examples

Create an edge with an *edge label* rated between the vertices johnDoe and beefBourguignon with the *properties* timestamp, stars, and comment.

```
johnDoe.addEdge('rated', beefBourguignon, 'timestamp',
'2014-01-01T00:00:00.00Z', 'stars', 5, 'comment', 'Pretty tasty!')
```

Update an edge with an *edge label* created between the vertices juliaChild and beefBourguignon, specifying the edge with an edge id of 2c85fabd-7c49-4b28-91a7-ca72ae53fd39, and a *property* createDate of 2017-08-22:

```
juliaChild.addEdge('created', 'beefBourguignon', T.id,
 java.util.UUID.fromString('2c85fabd-7c49-4b28-91a7-ca72ae53fd39'),
 'createDate', '2017-08-22')
```

Note that a conversion function must be used to convert a string to the UUID. `T.id` is a literal that must be included in the statement.

## addVertex

How to specify a vertex.

### Synopsis

```
addVertex(label, 'label_name', 'key', 'value', 'key', 'value')
```

### Description

Vertex data is inserted using `addVertex`. A previously created [vertex label \(page 644\)](#) must be specified.

### Examples

Create a vertex with a *vertex label* reviewer with the *properties* location and status.

```
graph.addVertex(label, 'reviewer', 'location', 'Santa Cruz, CA',
 'status', 'Rock Star')
```

## io

How to read or write a graph using low-level input/output.

### Synopsis

```
io([gryo() | graphson() | graphml()]).[readGraph | writeGraph]
(file_name)
```

### Description

Graph data is written to a file or read from a file using `io`. The file to read must be located on a DSE cluster node, and the written file will be written

### Examples

Write the graph data to a file using the Gryo format:

```
graph.io(gryo()).writeGraph('/tmp/test.gryo')
```

Read the graph data from a file using the Gryo format:

```
graph.io(gryo()).readGraph('/tmp/test.gryo')
```

**Restriction:** This method of reading a graph is not recommended, and will not work with graphs larger than 10,000 vertices or elements. [DSE Graph Loader \(page 471\)](#) is a better choice in production. Additionally, a schema setting may need modification for this method to work:

```
schema.config().option("tx_autostart").set(true)
```

## property

How to specify a property.

### Synopsis

```
vertex1.property(['key', 'value'] [,...], [T.id, 'property_id'])
```

### Description

Property data is inserted using `property`. Property key-value pairs are specified. A `property_id` may be specified, to upsert data for a multiple cardinality property to prevent creation of a new property.

### Examples

Create a property with values for `gender` and `nickname`.

```
jamieOliver.property('gender', 'M', 'nickname', 'jimmy')
```

Update the property `gender` for the vertex `juliaChild` specifying a property with a property id of `2c85fabd-7c49-4b28-91a7-ca72ae53fd39`:

```
uuid = java.util.UUID.fromString('2c85fabd-7c49-4b28-91a7-
ca72ae53fd39')
juliaChild.property('gender', 'F', T.id, uuid)'
```

Note that a conversion function must be used to convert a string to the UUID. `T.id` is a literal that must be included in the statement.

## tx().config().option()

How to specify a transaction-wide option.

### Synopsis

```
tx().config().option(option).open()
```

### Description

### Examples

Change the value of `allow_scan` for a transaction. The effect of this change is to allow all commands executed in the gremlin-console on a particular node to do full graph scans, even if the consistency level for the cluster is not `QUORUM`, the value required to change this option in the appropriate system table.

```
graph.tx().config().option("allow_scan", true).open()
```

Note that the previous transaction (automatically opened in gremlin-console or Studio) must be committed before the new configuration option value is set.

## The schema API

Reference guide to schema commands.

schema commands are used to create schema such as vertex labels, edge labels, property keys and indexes.

### drop

How to drop schema information for a particular graph.

#### Synopsis

```
schema.drop()
```

#### Description

Drop either all schema information or a particular element of the schema for a particular graph using this command. All data will also be dropped if all schema is dropped.

#### Examples

If using the Gremlin console, an alias must be created to bind the graph to a graph traversal before running this command. If using Studio, no prerequisite is required. To drop all schema and data for a particular graph:

```
schema.drop()
```

The result if the drop command is successful:

```
==>null
```

**Danger:** If this command is used, the graph will no longer have any schema or data!

To drop a single schema element, such as a property key, specify the schema to drop:

```
schema.propertyKey('gender').drop()
```

All schema elements (properties, edge labels, vertex labels, and indexes) can be removed with this method.

### connection

How to create an adjacency between two vertices with an edge label creation

#### Synopsis

```
connection('outV', 'inV')
```

## Description

An adjacency between two vertices is created using an edge label and the vertex labels of the outgoing and incoming vertices. This step is used in conjunction with `edgeLabel()`.

## Examples

Create an edge label `isA` specifying that the outgoing vertex label is `ingredient` and the incoming vertex label is `FridgeItem`.

```
schema.edgeLabel('isA').connection('ingredient', 'FridgeItem').create()
```

An adjacency between the `vertexLabel` `author` and `author` specifying the `edgeLabel` `knows`.

```
schema.edgeLabel('knows').connection('author', 'author').add()
```

## config

How to configure graph schema.

## Synopsis

```
schema.config().option(arg).[set(value) | unset(value) | get() |
exists() | describe()]
```

Schema can be configured per graph using the `config()` command. An option and value can be `set()` or `unset()`. An option's value can be retrieved with the `get()` command. Whether or not the option is configured can be discovered with the `exists()` command. The `describe()` command returns a value if the option has been set manually.

**Table 17: Graph-specific options**

[Graph-specific options are preceded by `graph`. For example, `graph.schema_mode`.]

Option argument	Setting Example	Description	Default
<code>allow_scan</code>	<code>true</code>	Setting to allow costly graph scan queries.	<code>true</code>
<code>schema_mode</code>	<code>Production</code>	Set mode to Production or Development.	<code>Development</code>
<code>default_property_key_cardinality</code>	<code>single</code>	Set the cardinality that will be used by default unless otherwise specified.	<code>single</code>
<code>tx_autostart</code>	<code>true</code>	Set whether transactions are started automatically or must be manually opened.	<code>false</code>

**Table 18: TraversalSource-specific options**

[TraversalSource-specific options are preceded by `graph.traversal_sources.*` where \* must be a specified traversal source such as the graph traversal `g`. For example, `graph.traversal_sources.g.type`. The most common TraversalSource is the graph traversal `g`.]

Option argument	Setting Example	Description	Default
<code>evaluation_timeout</code>	PT10S (10 seconds) or "1500 ms"	Maximum time to wait for a traversal to evaluate - this will override other system level settings for the current TraversalSource.	0 days
<code>restrict_lambda</code>	false	Prevent the use of lambdas with this TraversalSource. A particular traversal source can be identified.	true
<code>type</code>	read-only	Specify type of TraversalSource. A particular traversal source can be identified.	default

**Table 19: Transaction-specific options**

[Transaction-specific options are preceded by `graph.tx_groups.*` where \* must be specified as a transaction group or default. For example, `graph.tx_groups.default.read_only` will make all transactions which aren't explicitly named `read_only`, whereas `graph.tx_groups.myTxGroup.read_only` would apply only to transactions which are given the group name `myTxGroup`.]

Option	Setting Example	Description	Default
<code>authenticated_user</code>	<code>test_user</code>	The username to use as the current user for a transaction.	ANONYMOUS_USER

Option	Setting Example	Description	Default
cache	true	Cache retrievals and data store calls within a transaction in transaction-level caches. This setting provides a restricted type of isolation within a transaction (concurrent modifications in other transactions aren't visible and result sets remain consistent between calls) and can improve performance at the expense of additional memory consumption.	true
deep_profiling	true	Enable CQL tracing for <code>profile()</code> in queries. Very costly profiling.	false
internal_vertex_verify	true	Set whether a transaction should verify that vertices for internally provided vertex ids (autogenerated vertex ids) actually exist.	false
external_vertex_verify	false	Set whether a transaction should verify that vertices for externally provided vertex ids (custom vertex ids) actually exist.	true
logged_batch	true	Use a logged batch when committing changes. This guarantees that all mutations will eventually occur at the expense of performance.	false

Option	Setting Example	Description	Default
max_mutations	5000	The maximum number of vertices, properties and edges (cumulatively) that may be added or removed in a single transaction.	10000
max_profile_events	5	The maximum number of profiling events to report for an individual traversal step. Restricting the number of reported events makes output manageable, but can hide important information.	10
prefetch	true	Sets whether the query executor should asynchronously pre-fetch data based on its expected execution of the traversal prior to the data being requested. This can reduce transaction latency but can cause throughput to worse.	true
read_only	true	Set whether a transaction is read-only.	false
read_consistency	ALL	Specify the consistency level for read operations of a transaction.	ONE
single_thread	true	Set whether a transaction is only accessed by a single thread.	false
thread_bound	true	Set whether a transaction is bound to a particular thread.	false

Option	Setting Example	Description	Default
transaction_timestamp		The timestamp at which all mutations of this transaction are persisted.	Instant.EPOCH
verify_unique	false	Set whether transactions should ensure that uniqueness constraints are enforced.	true
vertex_cache_size	4000	Maximum size of the transaction-level cache of recently-used vertices	20000l
vertex_dirty_size		This is a performance hint for write-heavy, performance-sensitive transactional workloads. If set, it should roughly match the median vertices modified per transaction.	32
write_consistency	ANY	Specify the consistency level for write operations of a transaction	QUORUM

## Description

Configure a graph. Options can be `set`, `unset`, or `get` (retrieve the value).

## Examples

Set the current graph to disallow full graph scans in the currently aliased graph.

```
schema.config().option('graph.allow_scan').set('false')
```

To retrieve all traversal sources that have been set, use the `get()` command with the traversal source type option:

```
schema.config().option('graph.traversal_sources.*.type').get()
```

resulting in a list of values for the option that have been manually set:

```
REAL_TIME
```

Set the default write consistency for a transaction to ALL in the currently aliased graph.

```
schema.config().option('graph.tx_groups.default.write_consistency').set('ALL')
```

Get the current write consistency for a transaction in the currently aliased graph.

```
schema.config().option('graph.tx_groups.default.write_consistency').get()
```

To confirm that an option setting has been set manually, use the exists() command:

```
schema.config().option('graph.tx_groups.default.write_consistency').exists()
```

This command will return:

```
true
```

if the setting has been set to a value, otherwise it returns false.

To enable CQL tracing during traversal query profiling, set the deep\_profiling() option:

```
schema.config().option('graph.tx_groups.default.deep_profiling').set('TRUE')
```

To verify that external vertex ids exist, set the external\_vertex\_verify() option:

```
schema.config().option('graph.tx_groups.default.external_vertex_verify').set('TRUE')
```

If this setting is true, then a vertex will not be returned if it doesn't exist. However, if external\_vertex\_verify() is set to false, then a vertex will be returned even if the vertex does not exist given an id. Applications should ensure that vertices exist using the exists() method for expected behavior.

To retrieve a list of configuration options that have been set, use the describe() command:

```
schema.config().describe()
```

resulting in a list of all options that have been manually set:

```
graph.tx_groups.default.write_consistency: ALL
graph.allow_scan: False
```

There are some configuration options for which the default (for example, values are not explicitly set) is determined by using the value of other configuration options. For instance, if allow\_scan is not explicitly set, the default value is true if schema\_mode is set to Development, but false if the schema\_mode is set to Production. These configuration options are not linked to the default settings, leading to potentially misleading information when using schema.config().get() to discover the setting value because the default value is displayed rather than a set value.

To set restrict\_lambda to FALSE in order to test lambda functions (only appropriate for non-production systems):

```
schema.config().option('graph.traversal_sources.g.restrict_lambda').set('FALSE')
```

Full graph scan settings are as follows:

setting	schema mode	scans allowed
dse.yaml schema_mode:Production	Production	no
dse.yaml schema_mode:Development	Development	yes
graph.schema_mode:Production	Production	no
graph.schema_mode:Development	Development	no
graph.allow_scan:true	Production	yes
graph.allow_scan:true	Development	yes

## describe

How to list schema information about a particular graph.

### Synopsis

```
schema.describe()
```

### Description

List schema information about a particular graph using this command. An alias must be created to bind the graph to a graph traversal before running this command.

### Examples

Discover if a particular graph exists. The return value is a boolean value.

```
gremlin> schema.describe()
```

The resulting list:

```
-->schema.vertexLabel("FridgeItem").create()
```

## edgeLabel

How to create an edge label.

### Synopsis

```
schema.edgeLabel('edgeLabel').
 [single() | multiple()].
 [properties(property[, property]).[add() | drop()]].
 [connection(outVertex, inVertex)].
 [ttl].
 [ifNotExists()].
 [create() | add() | drop() | describe() | exists()]
```

## Description

An edge label specifies a type of edge that can be stored in DSE Graph. An edge label can have cardinality specified (default is multiple), properties that an edge has defined, the connections that are defined between two types of vertices, and a time-to-live (TTL) to determine the lifecycle of an edge. The order that the options are added to the schema statement matter: cardinality, properties associated with the edge label, then connection.

## Examples

Create an edgeLabel `created`:

```
schema.edgeLabel('created').create()
```

**Note:** Naming convention used to allow nearly unrestricted unicode. Now the only allowed characters are [a-zA-Z0-9], underscore, hyphen, and period.

Create an edgeLabel `includedIn` if the edge label doesn't already exist:

```
schema.edgeLabel('includedIn').ifNotExists().create()
```

Create an edgeLabel with multiple cardinality:

```
schema.edgeLabel('reviewed').multiple().create()
```

Add properties to an edgeLabel:

```
schema.edgeLabel('reviewed').properties('rating','last_date').add()
```

Create an edgeLabel with both a property and a connection:

```
schema.edgeLabel('reviewed').properties('rating').connection('recipe','reviewer').create()
```

Create a time-to-live (TTL) for an edgeLabel of 60 seconds. Setting a TTL will expire all edges inserted with the edgeLabel at the set TTL value:

```
schema.edgeLabel('createDate').ttl(60).create()
```

**Note:** DSE Graph sets TTL differently from the DSE database. The DSE database sets TTL per mutation (insertion or update) or can inherit a default value from the table schema. DSE Graph sets TTL per vertex label or edge label, and all vertices or edges will be affected by the TTL setting. DSE Graph cannot set TTL for an individual vertex or edge.

Check if an edgeLabel exists:

```
schema.edgeLabel('reviewed').exists()
```

Get the schema creation command for an edgeLabel using the `describe()` command:

```
schema.edgeLabel('createDate').describe()
```

Remove a edge label with the `drop()` command:

```
schema.edgeLabel('reviewed').drop()
```

Remove a property rating from an edge label:

```
schema.edgeLabel('reviewed').properties('rating').drop()
```

## exists

How to identify if a schema element exists.

### Synopsis

```
schema.<schema_element>('author').exists()
```

### Description

Discover if a particular schema element exists using this command. This command can be used with `vertexLabel`, `edgeLabel`, or `propertyKey`.

### Examples

Discover if a particular vertex label exists. The return value is a boolean value.

```
gremlin> schema.vertexLabel('author').exists()
```

The resulting list:

```
=>true
```

## index - edge index

How to add an edge index.

### Synopsis

```
index('index_name')[outE('edgeLabel') | inE('edgeLabel') | bothE('edgeLabel')].by('propertykey_name').add()
```

### Description

An edge index specifies an index that is built using an edge property key in DSE Graph. A vertex label must be specified, and edge indexes are only defined in relationship to a vertex label. The index name must be unique.

An edge index can be created using either outgoing edges (`outE()`) from a vertex label, incoming edges (`inE()`) from a vertex label, or both outgoing and incoming (`bothE()`). The last type, `bothE()`, is rarely used, but could be used in a situation where the index must track both the incoming and outgoing edges from a particular vertex label. An example would be a graph storing reviewers who can both be liked and like other reviewers. To search for reviewers who are liked and who like a particular reviewer, both incoming and outgoing edges would be searched.

## Examples

Create an `index ratedByStars` with an `outE` edge label using the `property key stars`. The `vertex label` is specified as `reviewer`.

```
schema.vertexLabel('reviewer').index('ratedByStars').outE('rated').by('stars').add()
```

Create an `index ratedByStars2Way` with a `bothE` edge label using the `property key year`. The edge index allows queries that find both recipes with a certain year and reviewers who gave a review in a certain year.

```
schema.vertexLabel('recipe').index('byAuthOrRecipe').bothE('created').by('year').ifNotExists().add()
```

It can replace two indexes:

```
schema.vertexLabel('recipe').index('toRecipesRated').inE('rated').by('year').add()
schema.vertexLabel('reviewer').index('toReviewersWhoRated').outE('rated').by('year').add()
```

## index - property index

How to add a property index.

### Synopsis

```
index('index_name').property('propertykey_name').by('meta-propertykey_name').add()
```

### Description

A property index specifies an index that is built using the [meta-property \(page 448\)](#) of a vertex property key in DSE Graph. A vertex label must be specified. The index name must be unique. The property key specified must have `multiple` cardinality.

## Examples

Create an `index byLocation` index using the `property key country` and `meta-property key of livedIn`. The `vertex label` is specified as `author`.

```
schema().vertexLabel('author').index('byLocation').property('country').by('livedIn').add()
```

## index - vertex index

How to add a vertex index.

### Synopsis

```
index('index_name')[secondary() | materialized() |
search()].by('propertykey_name')[asText() | asString()].add()
```

### Description

A vertex index specifies an index that is built using a vertex property key in DSE Graph. A vertex label must be specified. Vertex indexes can be specified as `secondary`, `materialized`, or `search`. The index name must be unique.

A search vertex index must be named `search`; only one search index can exist. Multiple property keys can be specified in a single search index definition. The options `asText()` and `asString()` must be specified for a search index.

## Examples

Create an `index` by `Recipe` as a secondary index using the property key `name`. The `vertex label` is specified as `recipe`.

```
schema.vertexLabel('recipe').index('byRecipe').secondary().by('name').add()
```

Create an `index` by `Meal` as a materialized index using the property key `name`. The `vertex label` is specified as `meal`.

```
schema.vertexLabel('meal').index('byMeal').materialized().by('name').add()
```

Create an `index` `search` as a search index using the property key `instructions` and specify that the index is a `asText()`. The `vertex label` is specified as `recipe`.

```
schema.vertexLabel('recipe').index('search').search().by('instructions').asText().add()
```

Create an `index` `search` as a search index using multiple property keys `instructions` with `asText()` and `category` with `asString()`. The `vertex label` is specified as `recipe`.

```
schema.vertexLabel('recipe').index('search').search().by('instructions').asText().by('cate
```

## partitionKey - clusteringKey

How to define a partition key or clustering key.

### Synopsis

```
partitionKey('id_name')[clusteringKey('id_name')]
```

### Description

`partitionKey` and `clusteringKey` are used to specify a [user-defined vertex id \(page 451\)](#) in conjunction with `vertexLabel`. The `partitionKey` sets a partition key. A composite partition key can also be set by chaining `partitionKey` items. The `clusteringKey` sets a clustering key. The property keys used must be created prior to use.

## Examples

Create a `propertyKey` `city_id`.

```
schema.propertyKey('city_id').Int().create()
```

Create a `vertexLabel` using `sensor_id` as a partitioning key.

```
schema().vertexLabel('FridgeSensor').partitionKey('sensor_id').create()
```

Create a vertex label with a custom partitioning key `city_id` and clustering key `sensor_id`.

```
schema().vertexLabel('FridgeSensor').partitionKey('city_id').clusteringKey('sensor_id').cr
```

Create a *vertex* using `city_id` as a partitioning key and `sensor_id` as a clustering key. The property key `sensor_id` must already exist and be an UUID.

```
graph.addVertex(label, 'FridgeSensor', 'city_id', 100, 'sensor_id',
'60bcae02-f6e5-11e5-9ce9-5e5517507c66')
```

Create a *vertexLabel* using `city_id` and `sensor_id` as a composite partitioning key.

```
schema().vertexLabel('FridgeSensor').partitionKey('city_id',
'sensor_id').create()
```

## properties

How to add properties to a vertex label or edge label.

### Synopsis

```
properties('name').add()
```

### Description

Properties can be added to vertices and edges. A [property key \(page 642\)](#) must be created prior to adding it to either type of element. Allowed characters for the name are alphabetical or underscore.

### Examples

Add a property key to a vertex label. The property key `nationality` must exist prior to adding it to the vertex label.

```
schema.vertexLabel('author').properties('nationality').add()
```

Add more than one property to a vertex label.

```
schema.vertexLabel('author').properties('nationality', 'age',
'assocRestaurants').add()
```

## propertyKey

How to create a property key.

### Synopsis

```
propertyKey('name').
type().
[single() | multiple()].
[properties(metadata_property).[add() | drop()]].
[ttl].
[ifNotExists()].
[create() | add() | drop() | describe() | exists()]
```

## Description

Property keys are created for vertices and edges. A property key must be created prior to adding it to either type of element. Allowed characters for the name are alphabetical or underscore. The [data type \(page 740\)](#) must be included. A property key can have cardinality specified, single(default) or multiple, properties ([meta-properties](#)), and a time-to-live (TTL) to determine the lifecycle of a property.

### Caution:

Multiple cardinality (multi-properties) will be retrieved in graph traversals more slowly than single cardinality properties, because vertices with multi-properties will default to requesting properties individually.

## Examples

Create a property key with the `name` name of `Text` type:

```
schema.propertyKey('name').Text().create()
```

**Note:** Naming convention used to allow nearly unrestricted unicode. Now the only allowed characters are [a-zA-Z0-9], underscore, hyphen, and period.

Create a property key with the `name num_items` of `Integer` type if the property key doesn't already exist:

```
schema.propertyKey('num_items').Int().ifNotExists().create()
```

Create a property key with the `name createDate` of `Timestamp` type with multiple property cardinality:

```
schema.propertyKey('createDate').Timestamp().multiple().create()
```

Add a meta-property for a property. The meta-property, `first_publication`, must first be created as a property key.

```
schema.propertyKey('createDate').properties('first_publication').add()
```

Create a time-to-live (TTL) for a property key of 60 seconds. Setting a TTL will expire all properties inserted with the `propertyKey` at the set TTL value:

```
schema.propertyKey('createDate').ttl(60).create()
```

**Note:** DSE Graph sets TTL differently from the DSE database. The DSE database sets TTL per mutation (insertion or update) or can inherit a default value from the table schema. DSE Graph sets TTL per vertex label or edge label, and all vertices or edges will be affected by the TTL setting. DSE Graph cannot set TTL for an individual vertex or edge.

Check if a property key exists:

```
schema.propertyKey('name').exists()
```

Get the schema creation command for a property key using the `describe()` command:

```
schema.propertyKey('name').describe()
```

Remove a property key with the `drop()` command:

```
schema.propertyKey('gender').drop()
```

Remove a meta-property `first_publication`:

```
schema.propertyKey('createDate').properties('first_publication').drop()
```

## vertexLabel

How to create a vertex label.

### Synopsis

```
schema.vertexLabel('vertexLabel').
 [partitionKey(propertyKey, [partitionKey(propertyKey)]].
 [clusteringKey(propertyKey)].
 [ttl].
 [ifNotExists()].
 [index].
 [properties(property, property).[add() | drop()]].
 [partition()].
 [cache()].
 [create() | drop() | describe() | exists()]
```

### Description

A vertex label specifies a type of vertex that can be stored in DSE Graph. A vertex label can have properties defined, a partition key, clustering key, indexes, cache, and a time-to-live (TTL) to determine the life cycle of an vertex.

DSE Graph limits the number of vertex labels to 200 per graph.

### Examples

Create a `vertexLabel` `author`:

```
schema.vertexLabel('author').create()
```

Create a `vertexLabel` `ingredient` if the vertex label doesn't already exist:

```
schema.vertexLabel('ingredient').ifNotExists().create()
```

For partition and clustering keys, see [partitionKey-clusteringKey \(page 641\)](#).

Add properties to a `vertexLabel`:

```
schema.vertexLabel('author').properties('location','restaurant').add()
```

For indexes, see each index entry ([edge index \(page 639\)](#), [property index \(page 640\)](#), [vertex index \(page 640\)](#)) in the Schema API.

Cache all properties for `author` vertices up to an hour (3600 seconds):

```
schema.vertexLabel('author').cache().properties().ttl(3600).add()
```

Enabling property cache causes index queries to use `IndexCache` for the specified vertex label.

Cache both incoming and outgoing `created` edges for `author` vertices up to a minute (60 seconds):

```
schema.vertexLabel('author').cache().bothE('created').ttl(60).add()
```

Partition a `vertexLabel` based on a particular `edgeLabel`:

```
schema.vertexLabel('author').partition().inE('created').add()
```

Create a time-to-live (TTL) for an `vertexLabel` of 60 seconds. Setting a TTL will expire all vertices inserted with the `vertexLabel` at the set TTL value:

```
schema.vertexLabel('author').ttl(60).create()
```

**Note:** DSE Graph sets TTL differently from the DSE database. The DSE database sets TTL per mutation (insertion or update) or can inherit a default value from the table schema. DSE Graph sets TTL per vertex label or edge label, and all vertices or edges will be affected by the TTL setting. DSE Graph cannot set TTL for an individual vertex or edge.

Check if a `vertexLabel` exists:

```
schema.vertexLabel('author').exists()
```

Get the schema creation command for a `vertexLabel` using the `describe()` command:

```
schema.vertexLabel('author').describe()
```

Remove a vertex label with the `drop()` command:

```
schema.vertexLabel('author').drop()
```

Remove a property `gender` from a vertex label:

```
schema.vertexLabel('author').properties('gender').drop()
```

## The system API

Reference guide to system commands.

The `system` commands create, drop, and describe graphs, as well as list existing graphs and check for existence. Graph and system configuration can also be set and unset with `system` commands.

### create

How to create a new graph.

## Synopsis

```
system.graph('graph_name').create()
```

## Description

Create a new graph. The `graph_name` specified is used to create two DSE database keyspaces, `graph_name` and `graph_name_system`, and can only contain alphanumeric and underscore characters.

## Examples

Create a new graph.

```
gremlin> system.graph('FridgeItem').create()
```

The resulting list:

```
==>FridgeItem
```

Create a new graph if it doesn't currently exist by modifying with `ifNotExists()`.

```
gremlin> system.graph('FridgeItem').ifNotExists().create()
```

The resulting list:

```
==>FridgeItem
```

Creating a graph should include setting the replication factor for the [graph \(page 649\)](#) and the [graph\\_system \(page 650\)](#). It can also include other [options \(page 647\)](#).

## drop

How to drop a graph.

## Synopsis

```
system.graph('graph_name').[IfExists()].drop()
```

## Description

Drop an existing graph using this command. All data and schema will be lost.

## Examples

Drop a graph.

```
gremlin> system.graph('FridgeItem').drop()
```

The resulting list:

```
==>null
```

Drop an existing graph if it exists.

```
gremlin> system.graph('FridgeSensors').ifExists().drop()
```

The resulting list:

```
==>null
```

## exists

How to identify if a graph exists.

### Synopsis

```
system.graph('graph_name').exists()
```

### Description

Discover if a particular graph exists using this command.

### Examples

Discover if a particular graph exists. The return value is a boolean value.

```
gremlin> system.graph('FridgeItem').exists()
```

The resulting list:

```
==>true
```

## graphs

How to identify known graphs.

### Synopsis

```
system.graphs()
```

### Description

Discover what graphs currently exist using this command.

### Examples

Discover all graphs that exist in a DSE cluster.

```
gremlin> system.graphs()
```

The resulting list:

```
==>quickstart
==>test
```

## option

How to configure a graph using a system option.

## Synopsis

```
option(arg).set(value)
```

Graphs can be configured per graph using the following options. The Gremlin console must be used to set system commands.

**Table 20: Graph-Specific Options**

[Graph-specific options are preceded by *graph*. For example, *graph.replication\_config*.]

Option argument	Setting Example	Description	Default
replication_config (replaced by <a href="#">replication (page 649)</a> in DSE 5.1.3 and later)	{ 'class' : 'NetworkTopologyStrategy', 'dc1' : 3 }	Set replication configuration for a single graph.	{'class' : 'SimpleStrategy', 'replication_factor' : 1 }
system_replication_config (replaced by <a href="#">systemReplication (page 650)</a> in DSE 5.1.3 and later)	{ 'class' : 'NetworkTopologyStrategy', 'dc1' : 3 }	Set replication configuration for a single <a href="#">graph_system (page 742)</a> data.	{'class' : 'SimpleStrategy', 'replication_factor' : 1 }
default_property_key_cardinality	Multiple	The default cardinality for automatically defined properties	Single

## Description

Configure a graph. Options can be set.

**Restriction:** The replication factor and system replication factor cannot be altered once set for the *graph\_name* and *graph\_name\_system* keyspaces.

## Examples

Create a new graph and set the *graph* replication configuration and the *graph\_system* replication configuration to the DSE database settings shown.

```
system.graph('food').
 option("graph.replication_config").set("{ 'class' :
'NetworkTopologyStrategy', 'dc1' : 3 }").
 option("graph.system_replication_config").set("{ 'class' :
'NetworkTopologyStrategy', 'dc1' : 3 }").
 ifNotExists().create()
```

The resulting list:

```
==>null
```

The replication settings can be verified using the `cqlsh` tool, running the CQL command `DESCRIBE keyspace food;`.

**Note:** The options shown (`graph.replication_config` and `graph.system_replication_config`) have been replaced in DSE 5.1.3 and later; see the table above.

Other [schema settings \(page 631\)](#) can be set at graph creation, but must be changed using `schema.config()` if modified later.

```
system.graph('food2').
 option("graph.replication_config").set("{'class' : 'SimpleStrategy',
'replication_factor' : 1 }").
 option("graph.system_replication_config").set("{'class' :
'SimpleStrategy', 'replication_factor' : 1 }").
 option("graph.schema_mode").set("Development").
 option("graph.allow_scan").set("false").
 option("graph.default_property_key_cardinality").set("multiple").
 option("graph.tx_groups.*.write_consistency").set("ALL").
 create()
```

To check the schema settings:

```
:remote config alias g food2.g
schema.config().describe()
```

to get the results:

```
graph.schema_mode: Development
graph.allow_scan: False
graph.tx_groups.*.write_consistency: ALL
graph.default_property_key_cardinality: Multiple
gremlin> schema.config().option("graph.allow_scan").set("true")
```

Note the use of a wildcard \* to set the write consistency for all transaction groups.

## replication

How to set the replication factor for a new graph.

### Synopsis

```
system.graph('graph_name').replication("{'class' :
'NetworkTopologyStrategy', 'dcl' : 3}")
```

### Description

Create a new graph. The `graph_name` specified is used to create two DSE database keyspaces, `graph_name` and `graph_name_system`, and can only contain alphanumeric and underscore characters.

### Examples

Create a new graph and set the `graph_name` replication configuration using `replication()` as well as the `graph_name_system` configuration using `systemReplication()`. [DSE database settings for replication factor](#) are used, either

`SimpleStrategy` for single nodes or `NetworkTopologyStrategy` for multiple nodes or multiple datacenters.

The default replication strategy for a multi-node or multi-datacenter graph is `NetworkTopologyStrategy`, whereas for a single node, the replication strategy will default to `SimpleStrategy`. The number of nodes will determine the default replication factor:

number of nodes per datacenter	graph_name replication factor	graph_name_system replication factor
1-3	number of nodes per datacenter	number of nodes per datacenter
greater than 3	3	5

```
system.graph('food').
 replication("{'class' : 'NetworkTopologyStrategy', 'dc1' : 3 }").
 systemReplication("{'class' : 'NetworkTopologyStrategy', 'dc1' :
 3 }").
 ifNotExists().create()
```

The resulting list:

```
==>null
```

The replication settings can be verified using the `cqlsh` tool, running the CQL command `DESCRIBE keyspace food;`.

In addition to setting the replication factor for the `graph_name` keyspace, the [replication factor for the `graph\_name\_system` \(page 650\)](#) must also be set.

**Restriction:** The replication factor and system replication factor cannot be altered once set for the `graph_name` and `graph_name_system` keyspaces.

## systemReplication

How to set the replication factor for the graph system keyspace of a new graph.

### Synopsis

```
system.graph('graph_name').systemReplication("{'class' :
'NetworkTopologyStrategy', 'dc1' : 3}")
```

### Description

Create a new graph. The `graph_name` specified is used to create two DSE database keyspaces, `graph_name` and `graph_name_system`, and can only contain alphanumeric and underscore characters.

### Examples

Create a new graph and set the `graph_name` replication configuration using `replication()` as well as the `graph_name_system` configuration using `systemReplication()`. [DSE database settings for replication factor](#) are used, either `SimpleStrategy` for single nodes or `NetworkTopologyStrategy` for multiple nodes.

The default replication strategy for a multi-node or multi-datacenter graph is `NetworkTopologyStrategy`, whereas for a single node, the replication strategy will default to `SimpleStrategy`. The number of nodes will determine the default replication factor:

number of nodes per datacenter	<code>graph_name</code> replication factor	<code>graph_name_system</code> replication factor
1-3	number of nodes per datacenter	number of nodes per datacenter
greater than 3	3	5

```
system.graph('food').
replication("{'class' : 'NetworkTopologyStrategy', 'dc1' : 3 }").
systemReplication("{'class' : 'NetworkTopologyStrategy', 'dc1' :
3 }").
ifNotExists().create()
```

The resulting list:

```
==>null
```

The system replication settings can be verified using the `cqlsh` tool, running the CQL command `DESCRIBE keyspace food_system;`.

**Important:** Because the graph's schema is stored in `graph_name_system`, it is extremely important that the replication factor is set consistent with the table values above. If the graph's schema is lost, it renders the entire graph inoperable.

In addition to setting the replication factor for the `graph_name_system` keyspace, the [replication factor for the `graph\_name` \(page 649\)](#) must also be set.

## The traversal API

Reference commands and other information for Apache TinkerPop's Gremlin as used by DSE Graph.

## Apache TinkerPop™ graph computing framework

Describe the Apache TinkerPop framework.

Apache TinkerPop™ is a graph abstraction layer that works with numerous different graph databases and graph processors. TinkerPop is composed of two elements: a structure API and a process API.

The primary components of the TinkerPop structure API are:

### Graph

Maintains a set of vertices and edges.

### Vertex

Extends a general class `Element` and maintains a set of incoming and outgoing edges as well as a collection of properties and a vertex type. DSE Graph schema stores `VertexLabel` - ID, name, Time-To-Live (TTL).

## Edge

Extends Element and maintains an incoming and outgoing vertex as well as a collection of properties and an edge type. DSE Graph schema stores EdgeLabel - ID, name, TTL, multiplicity (multi, simple), unidirected, visible, sort-key.

## Property

A string key associated with a value. DSE Graph schema stores PropertyKey - ID, name, TTL, datatype, cardinality (single, list).

## VertexProperty

A string key associated with a value as well as a collection of metadata properties (vertices only).

The primary components of the TinkerPop process API are:

### TraversalSource

A generator of traversals for a particular graph, domain specific language (DSL), and execution engine.

### Traversal<S,E>

A functional data flow process transforming objects of type S into object of type E.

### GraphTraversal

A traversal domain-specific language (DSL) that is oriented towards the semantics of the raw graph (i.e. vertices, edges, etc.).

### GraphComputer

A system that processes the graph in parallel and potentially, distributed over a multi-machine cluster.

### VertexProgram

Code executed at all vertices in a logically parallel manner with intercommunication via message passing.

### MapReduce

Computations that analyzes all vertices in the graph in parallel and yields a single reduced result.

A key feature of TinkerPop is Gremlin, a graph traversal language and virtual machine. TinkerPop and Gremlin are to graph databases what JDBC and SQL are to relational databases. Gremlin variants are available for many languages: Java, Groovy, Python, and others.

## General steps, step-modulators, and predicates

TinkerPop general steps, step-modulators, and predicates

Apache TinkerPop<sup>TM</sup> has five general step types:

### map(Traversal<S,E>) or map(Function<Traverser<S>, E>)

Maps the traverser to some object of type E for the next step to process.

### flatMap(Traversal<S,E>) or flatMap(Function<Traverser<S>, Iterator<E>)

Maps the traverser to an iterator of E objects that are streamed to the next step.

### filter(Traversal<?, ?>) or filter(Predicate<Traverser<S>>)

Maps the traverser to either true or false, and where false will not pass the traverser to the next step.

**sideEffect(Traversal<S, S>) or sideEffect(Consumer<Traverser<S>>)**

Performs some operation on the traverser and passes it to the next step.

**branch(Traversal<S, M>) or branch(Function<Traverser<S>, M>)**

Splits the traverser to all the traversals indexed by the M token.

All other steps in the reference can be categorized as one of these steps or a variant, such as a terminal step that completes a traversal. In addition, steps have step-modulators, a helper step that assists a step:

**as()**

Provide a label to the step that can later be accessed by steps and data structures that make use of such labels.

**by()**

If a step is able to accept traversals, functions, or comparators, then by() is the means by which they are added.

**emit()**

If emit() is placed after repeat(), it is evaluated on the traversers leaving the repeat-traversal. If emit() is placed before repeat(), it is evaluated on the traversers prior to entering the repeat-traversal.

**from()**

Adds a string or traversal to a traversal to point the traversal FROM the next supplied step.

**option()**

Provide a option to a branch() or choose() step.

**to()**

Adds a string or traversal to a traversal to point the traversal TO the next supplied step.

**until()**

If until() comes after repeat() it is do/while looping. If until() comes before repeat() it is while/do looping.

Within steps, predicates are used to determine relationships between data:

**eq(object)**

Check if an incoming object is equal to the provided object.

**neq(object)**

Check if an incoming object is not equal to the provided object.

**lt(number)**

Check if an incoming number is less than the provided number.

**lte(number)**

Check if an incoming number is less than or equal the provided number.

**gt(number)**

Check if an incoming number is greater than the provided number.

**gte(number)**

Check if an incoming number is greater than or equal the provided number.

**inside(number, number)**

Check if an incoming number is inside the range of the provided numbers.

**outside(number, number)**

Check if an incoming number is outside the range of the provided numbers.

**between(number, number)**

Check if an incoming number is greater than or equal to the first provided number and less than the second provided number.

### **within(objects...)**

Check if an incoming object is within an array of provided objects.

### **without(objects...)**

Check if an incoming object is not within an array of provided objects.

## TinkerPop Predicates

Predicates are used to determine relationships between data within traversals.

### **eq**

Given some object, return true or false, for equality.

#### Synopsis

```
eq(object)
```

**Table 21: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

#### Description

The `eq()` predicate answers the question: Is the incoming object equal to the provided object?

#### Examples

Find which recipes are main entrees:

```
g.V().hasLabel('recipe').has('cuisine', eq('main entree')).values('name')
```

### **neq**

Given some object, return true or false, for non-equality.

## Synopsis

```
neq(object)
```

**Table 22: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `neq()` predicate answers the question: Is the incoming object not equal to the provided object?

## Examples

Find who the coauthors are of each book:

```
g.V().hasLabel('person').as('oneAuthor').out('authored').in('authored').
 where(neq('oneAuthor')).as('anotherAuthor').dedup().
 select('oneAuthor', 'anotherAuthor').
 by('name')
```

## lt

Given some object, return true or false, for values that are less than the specified integer.

## Synopsis

```
lt(integer_value)
```

**Table 23: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.

Syntax conventions	Description
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `lt()` predicate answers the question: Is the incoming number less than the provided number?

## Examples

Find all meal items with less than 800 calories:

```
g.V().has('meal_item', 'calories', lt(800)).valueMap('name',
 'calories')
```

## Ite

Given some object, return true or false, for values that are less than or equal to the specified integer.

## Synopsis

```
lte(integer_value)
```

**Table 24: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.

Syntax conventions	Description
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `lte()` predicate answers the question: Is the incoming number less than or equal to the provided number?

## Examples

Find all meal items with less than or equal to 1230 calories:

```
g.V().has('meal_item', 'calories', lte(1230)).valueMap('name',
 'calories')
```

## gt

Given some object, return true or false, for values that are greater than the specified integer.

## Synopsis

```
gt(integer_value)
```

**Table 25: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `gt()` predicate answers the question: Is the incoming number greater than the provided number?

## Examples

Find all recipes reviewed with a rating of greater than 3 stars:

```
g.E().hasLabel('reviewed').has('stars', gt(3)).valueMap()
```

## gte

Given some object, return true or false, for values that are greater than or equal to the specified integer.

### Synopsis

```
gte(integer_value)
```

**Table 26: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

### Description

The `gte()` predicate answers the question: Is the incoming number greater than or equal to the provided number?

### Examples

Find all recipes reviewed with a rating of greater than or equal to 3 stars:

```
g.E().hasLabel('reviewed').has('stars', gte(3)).valueMap()
```

## inside

Given some object, return true or false, for values that are between the specified integers.

### Synopsis

```
inside(integer_value, integer_value)
```

**Table 27: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.

## Description

The `inside()` predicate answers the question: Is the incoming number greater than the first provided number and less than the second?

## Examples

Find all books published between 1960 and 1969 (from 1961 to 1968):

```
g.V().hasLabel('book').has('publishYear',
 inside(1960,1969)).values('name', 'publishYear')
```

## outside

Given some object, return true or false, for values that are outside the specified integers.

## Synopsis

```
outside(integer_value, integer_value)
```

**Table 28: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.

Syntax conventions	Description
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `outside()` predicate answers the question: Is the incoming number less than the first provided number or greater than the second?

## Examples

Find all books published outside of 1960 and 1969 (before 1960 or after 1969):

```
g.V().hasLabel('book').has('publishYear',
 outside(1960,1969)).values('name', 'publishYear')
```

## between

Given some object, return true or false, for values that are between the specified integers.

## Synopsis

```
between(integer_value, integer_value)
```

**Table 29: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `between()` predicate answers the question: Is the incoming number greater than or equal to the first provided number and less than the second?

## Examples

Find all books published between of 1960 and 1969 (1960-1968):

```
g.V().hasLabel('book').has('publishYear',
 bwtween(1960,1969)).values('name', 'publishYear')
```

## within

Given a list of objects, return the ones that listed.

### Synopsis

```
within(object, ...)
```

**Table 30: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.

## Description

The `within()` predicate answers the question: Is the incoming object in the array of provided objects?

### Examples

Get the two people who have the specified user-defined vertex ids:

```
g.V().has(id, within("{~label=person, personId=1}", "{~label=person,
 personId=2}"))
```

## without

Given a list of objects, return the ones that do are not listed.

### Synopsis

```
without(object, ...)
```

**Table 31: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.

## Description

The `without()` predicate answers the question: Is the incoming object not in the array of provided objects?

## Examples

Get all the vertices (including recipes, books, etc.) that do not have the specified user-defined vertex ids:

```
g.V().has(id, without(~label=person, personId=1), ~label=person, personId=2))
```

## TinkerPop Step-modulators

Helpers for Gremlin steps in traversals.

### as

Label an object in a traversal to use later in the traversal.

### Synopsis

```
as('variable_name')
```

**Table 32: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.

Syntax conventions	Description
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `as()` step is a [step modulator \(page 653\)](#), a helper step for another traversal step.

## Examples

Label all returned `person` vertices as `PERSON`, and all `created` edges as `RECIPE`, and then `select()` both the vertices and edges using the assigned variable names:

```
g.V().hasLabel('person').as('PERSON').
 out('created').as('RECIPE').
 select('PERSON','RECIPE').
 by('name')
```

## by

### Synopsis

```
by([{ property | traversal }])
```

**Table 33: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.

Syntax conventions	Description
...	Repeatable. An ellipsis ( ...) indicates that you can repeat the syntax element as often as required.

## Description

The `by()` step is a [step modulator \(page 653\)](#) that can be used to modify sort order from a previous step.

## Examples

```
g.V().group().by(inE().count()).by(count())
```

## emit

Emit traversers before or after a `repeat()` step.

## Synopsis

```
emit(['predicate' | 'traversal'])
```

**Table 34: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( {} ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( ...) indicates that you can repeat the syntax element as often as required.

## Description

The `emit()` step is a [step modulator \(page 653\)](#), a helper step for another traversal step. Its main use is to emit either incoming traversers before a `repeat()` ([page 723](#)) step, or emit outgoing traversers after a `repeat()` step. The emission sends a copy of the current objects to the next step in the query. A [predicate \(page 653\)](#) or traversal can be used in an `emit()` step to cause the emission only if the predicate or traversal is true.

## Examples

```
g.V(1).emit().repeat(out()).times(2).path()
```

```

==>[v[1]]
==>[v[1],v[3]]
==>[v[1],v[2]]
==>[v[1],v[4]]
==>[v[1],v[4],v[5]]
==>[v[1],v[4],v[3]]
gremlin> g.V(1).repeat(out()).times(2).emit().path()
==>[v[1],v[3]]
==>[v[1],v[2]]
==>[v[1],v[4]]
==>[v[1],v[4],v[5]]
==>[v[1],v[4],v[3]]

```

## from

Designate the outgoing vertex for an addE() step.

### Synopsis

```
from('vertex_designator')
```

**Table 35: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>Italics</i>	Variable value. Replace with a user-defined value.
[]	Optional. Square brackets ( [] ) surround optional command arguments. Do not type the square brackets.
{}	Group. Braces ( {} ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.

### Description

The `from()` step is a [step modulator \(page 653\)](#), a helper step for another traversal step. Its main use is to designate the outgoing vertex for an [addE\(\) \(page 679\)](#) step. Generally, a [to\(\)](#) ([page 667](#)) step is paired with a `from()` step.

### Examples

```

g.V().has('person', 'name', 'Jim Walsh').as('jim').
V().has('person', 'name', 'Sharon Smith').as('sharon').

```

```
addE('knows').from('jim').to('sharon').property('since', 1980)
```

## option

An option for branch() or choose().

### Synopsis

```
option(value, value_returned)
```

**Table 36: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

### Description

The option() step modulator is used in conjunction with branch() or choose(), and specifies the returned values based on values found at that point in a traversal.

### Examples

Find all people and list whether they are female, male, or unknown:

```
g.V().hasLabel('person').
 project('name', 'gender').
 by('name').
 by(choose(values('gender')).
 option('F', constant('female')).
 option('M', constant('male')).
 option(none, constant('unknown')))
```

## times

Repeat a step for the specified number of times.

### Synopsis

```
times(integer)
```

**Table 37: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.

## Description

The `times()` step is a [step modulator \(page 653\)](#), a helper step for another traversal step. Its main use is to repeat a [repeat\(\) \(page 723\)](#) step for the specified number of times.

## Examples

An example that repeats through all the outgoing *knows* adjacent vertices of *John Doe* three times:

```
g.V().hasLabel('person','name','John Doe').
repeat(out('knows')).times(3)
```

## to

Designate the incoming vertex for an `addE()` step.

## Synopsis

```
to('variable_name')
```

**Table 38: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.

Syntax conventions	Description
{}	Group. Braces ( {} ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.

## Description

The `to()` step is a [step modulator \(page 653\)](#), a helper step for another traversal step. Its main use is to designate the incoming vertex for an `addE()` ([page 679](#)) step. Generally, a `to()` step is paired with a `from()` ([page 665](#)) step.

## Examples

```
// create a new user
//g.addV('person').property('personId', 26).property('name', 'Jim
Walsh').property('gender', 'M')
// use a mid-traversal V() step to create a new edge between two
people
g.V().has('person', 'name', 'Jim Walsh').as('jim').
V().has('person', 'name', 'Sharon Smith').as('sharon').
addE('knows').from('jim').to('sharon').property('since', 1980)
```

## until

Turn a repeat() step into a do-while or while-do loop.

## Synopsis

```
until(['predicate' | 'traversal'])
```

**Table 39: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ()..
<i>Italics</i>	Variable value. Replace with a user-defined value.
[]	Optional. Square brackets ( [] ) surround optional command arguments. Do not type the square brackets.
{}	Group. Braces ( {} ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.

Syntax conventions	Description
...	Repeatable. An ellipsis ( ...) indicates that you can repeat the syntax element as often as required.

## Description

The `until()` step is a [step modulator \(page 653\)](#), a helper step for another traversal step. Its main use is to turn a [repeat\(\) \(page 723\)](#) step into a do-while loop (if used after the `repeat()` step) or a while-do loop (if used before the `repeat()` step). A [predicate \(page 653\)](#) or traversal can be used in an `until()` step to cause the loop to complete only if the predicate or traversal is true.

## Examples

An example that repeats through all the outgoing `knows` adjacent vertices of *John Doe* until the simple paths are exhausted:

```
g.V().hasLabel('person', 'name', 'John Doe').
repeat(out('knows').dedup().
 aggregate('x').
 by(project('person', 'level').
 by('name').
 by(loops())))
).
until(__.not(out('knows').simplePath())).
cap('x').next()
```

## TinkerPop Vertex Steps

Fundamental steps of Gremlin language

All the vertex steps are [flatMap \(page 652\)](#) steps.

### out

Move to the outgoing adjacent vertices, given the edge labels.

### Synopsis

```
out([edgeLabel])
```

**Table 40: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.

Syntax conventions	Description
{}	Group. Braces ( {} ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `out()` step moves the traversal to the outgoing adjacent vertices, given the edge labels. Specifying no edge label will traverse all incident edges.

## Examples

Get all outgoing adjacent vertices for all the vertices in the graph:

```
g.V().out().valueMap()
```

Get all outgoing adjacent vertices for all the vertices with incident edges *knows*:

```
g.V().out('knows').valueMap()
```

## in

Move to the incoming adjacent vertices, given the edge labels.

## Synopsis

```
in([edgeLabel])
```

**Table 41: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[]	Optional. Square brackets ( [] ) surround optional command arguments. Do not type the square brackets.
{}	Group. Braces ( {} ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `in()` step moves the traversal to the incoming adjacent vertices, given the edge labels. Specifying no edge label will traverse all incident edges.

## Examples

Get all incoming adjacent vertices for all the vertices in the graph:

```
g.V().in().valueMap()
```

Get all incoming adjacent vertices for all the vertices with incident edges *knows*:

```
g.V().in('knows').valueMap()
```

## both

Move to both the outgoing and the incoming adjacent vertices, given the edge labels.

## Synopsis

```
both([edgeLabel])
```

**Table 42: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes <code>( )</code> .
<i>Italics</i>	Variable value. Replace with a user-defined value.
<code>[]</code>	Optional. Square brackets <code>( [] )</code> surround optional command arguments. Do not type the square brackets.
<code>{}</code>	Group. Braces <code>( {} )</code> identify a group to choose from. Do not type the braces.
<code> </code>	Or. A vertical bar <code>(   )</code> separates alternative elements. Type any one of the elements. Do not type the vertical bar.
<code>...</code>	Repeatable. An ellipsis <code>( . . . )</code> indicates that you can repeat the syntax element as often as required.

## Description

The `both()` step moves the traversal to both the outgoing and the incoming adjacent vertices, given the edge labels. Specifying no edge label will traverse all incident edges.

## Examples

Get all outgoing and incoming adjacent vertices for all the vertices in the graph:

```
g.V().both().valueMap()
```

Get all outgoing and incoming adjacent vertices for all the vertices with incident edges *knows*:

```
g.V().both('knows').valueMap()
```

## outE

Move to the outgoing incident edges, given the edge labels.

### Synopsis

```
outE([edgeLabel])
```

**Table 43: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

### Description

The `outE()` step moves the traversal to the outgoing incident edges, given the edge labels. Specifying no edge label will traverse all incident edges.

### Examples

Get all outgoing incident edges for all the vertices in the graph:

```
g.V().outE().valueMap()
```

Get all outgoing incident edges for all the vertices with incident edges *knows*:

```
g.V().outE('knows').valueMap()
```

## inE

Move to the incoming incident edges, given the edge labels.

## Synopsis

```
inE([edgeLabel])
```

**Table 44: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `inE()` step moves the traversal to the incoming incident edges, given the edge labels. Specifying no edge label will traverse all incident edges.

## Examples

Get all incoming incident edges for all the vertices in the graph:

```
g.V().inE().valueMap()
```

Get all incoming incident edges for all the vertices with incident edges *knows*:

```
g.V().inE('knows').valueMap()
```

## bothE

Move to both the outgoing and the incoming incident edges, given the edge labels.

## Synopsis

```
bothE([edgeLabel])
```

**Table 45: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.

Syntax conventions	Description
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `bothE()` step moves the traversal to both the outgoing and the incoming incident edges, given the edge labels. Specifying no edge label will traverse all incident edges.

## Examples

Get all outgoing and incoming incident edges for all the vertices in the graph:

```
g.V().bothE().valueMap()
```

Get all outgoing and incoming incident edges for all the vertices with incident edges `knows`:

```
g.V().bothE('knows').valueMap()
```

## outV

Move to the outgoing vertex.

## Synopsis

```
outV()
```

**Table 46: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.

Syntax conventions	Description
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.

## Description

The `outV()` step moves the traversal to the outgoing vertices.

## Examples

Get all outgoing adjacent vertices for all the edges in the graph:

```
g.E().outV().valueMap()
```

Get all outgoing incident edges for all vertices, then move to the incoming vertices that have those incident edges:

```
g.V().outE('knows').inV().valueMap()
```

Get all outgoing incident edges for all vertices, then move to the incoming vertices that have those incident edges and a vertex label *person*:

```
g.V().outE('knows').inV().hasLabel('person')
```

## inV

Move to the incoming vertex.

## Synopsis

```
inV()
```

**Table 47: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.

Syntax conventions	Description
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `inV()` step moves the traversal to the incoming vertices.

## Examples

Move to all incoming adjacent vertices for all the edges in the graph:

```
g.E().inV().valueMap()
```

Get all incoming incident edges for all vertices, then move to the incoming vertices that have those incident edges:

```
g.V().inE('knows').inV().valueMap()
```

Get all incoming incident edges for all vertices, then move to the incoming vertices that have those incident edges and a vertex label `person`:

```
g.V().inE('knows').inV().hasLabel('person')
```

## bothV

Move to both the outgoing and the incoming vertex.

## Synopsis

```
bothV()
```

**Table 48: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `bothV()` step moves the traversal to both the outgoing and the incoming vertices.

## Examples

Move to all adjacent vertices for all the edges in the graph:

```
g.E().bothV().valueMap()
```

Get all incoming incident edges for all vertices, then move to the all vertices that have those incident edges:

```
g.V().inE('knows').bothV().valueMap()
```

Get all incoming incident edges for all vertices, then move to the all vertices that have those incident edges and a vertex label `person`:

```
g.V().inE('knows').bothV().hasLabel('person')
```

## otherV

Move to the vertex that was not the vertex that was moved from.

## Synopsis

```
otherV()
```

**Table 49: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes <code>()</code> .
<i>Italics</i>	Variable value. Replace with a user-defined value.
<code>[]</code>	Optional. Square brackets <code>( [] )</code> surround optional command arguments. Do not type the square brackets.
<code>{}</code>	Group. Braces <code>( {} )</code> identify a group to choose from. Do not type the braces.
<code> </code>	Or. A vertical bar <code>(   )</code> separates alternative elements. Type any one of the elements. Do not type the vertical bar.
<code>...</code>	Repeatable. An ellipsis <code>( . . . )</code> indicates that you can repeat the syntax element as often as required.

## Description

The `otherV()` step moves the traversal to the vertex that was not the vertex that was moved from.

## Examples

Get all the recipe vertices, then traverse to the edges labeled with the edge labels specified, and finally move to the vertices other than the recipe vertices where the traversal started:

```
g.V().hasLabel('recipe').bothE('reviewed', 'created', 'includedIn').otherV()
```

## addV

Specify a vertex from a traversal.

### Synopsis

```
g.addV('vertexLabel')
 .[property ('property_key') ...]
```

**Table 50: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ()..
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.

## Description

The `addV()` step is a [map \(page 652\)](#)/[sideEffect \(page 653\)](#). A vertex is added from a traversal `g` using `addV`. A previously created [vertex label \(page 644\)](#) must be specified. Property key-value pairs may be optionally specified.

## Examples

Create a vertex with a [vertex label](#) `person` with the [properties](#) `personId`, `name`, and `gender`.

```
g.addV('person').
 property('personId', 25).
 property('name', 'Stephen Smith').
```

```
property('gender', 'M')
```

## addE

Specify an edge from a traversal.

### Synopsis

```
g.V(vertex)
 .addE('edgeLabel')
 .to(vertex)
```

**Table 51: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

### Description

The `addE()` step is a [map \(page 652\)](#)/[sideEffect \(page 653\)](#). An edge is added from a traversal `g` using `addE` between two existing vertices. A previously created [edge label \(page 637\)](#) must be specified.

### Examples

Create an edge with an `edge label` `knows` between two vertices, `stephenSmith` and `johnDoe`. The first two lines assign the already existing vertices to variable names for use in the `addE()` step.

```
// Get two users to join with an edge
johnDoe = g.V().has('name', 'John Doe').next()
stephenSmith = g.V().has('name', 'Stephen Smith').next()
// Create the edge between Stephen and John
g.V(stephenSmith).addE('knows').to(johnDoe)
```

## property

How to specify a property from a traversal.

## Synopsis

```
g.addV('vertexLabel')
 .[property ('property_key') ...]
```

**Table 52: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ()..
<i>italics</i>	Variable value. Replace with a user-defined value.
[]	Optional. Square brackets ( [] ) surround optional command arguments. Do not type the square brackets.
{}	Group. Braces ( {} ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.

## Description

The `property()` step is a [sideEffect \(page 653\)](#). When a vertex is added from a traversal `g` using `addV`, properties can be added with `property()`. For a previously created vertex, properties can be added. A previously created [vertex label \(page 644\)](#) must be specified. Property key-value pairs are specified.

## Examples

Create a vertex with a `vertex label` `person` with the `properties` `personId` with value `17` and `name` with value `Jamie Oliver`:

```
g.addV('person').
 property('personId', 17).
 property('name', 'Jamie Oliver').next()
```

Add the `properties` `gender` with value `M` and `nickname` with value `jimmy` to a `person` vertex previously created:

```
g.V().has('person', 'name', 'Jamie Oliver').
 property('gender', 'M').
 property('nickname', 'jimmy')
```

Add the `property` `withSuppliedId` with value `341f9950-997c-11e7-9579-7f50358d3f8d` to a `person` vertex:

```
// user-supplied property ID
g.addV('person').
```

```
property('withSuppliedId', 'propValue', T.id,
UUID.fromString('341f9950-997c-11e7-9579-7f50358d3f8d'))
```

## mid-traversal V()

Fetch a vertex in the middle of a graph traversal.

### Synopsis

```
V().has('vertexLabel', 'propertyKey', 'PropertyValue')
```

**Table 53: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

### Description

A mid-traversal V() step can be used to fetch an additional vertex in the middle of a graph traversal, to use for some additional operation.

### Examples

Use a mid-traversal V() to fetch the person named Sharon Smith, so that an edge can be created between Jim Walsh and Sharon Smith:

```
// graph step V() in mid-traversal
// create a new user to use in the addE() step named Jim Walsh
//g.addV('person').property('personId', 26).property('name','Jim
Walsh').property('gender', 'M')

// use a mid-traversal V() step to create a new edge between two people
g.V().has('person', 'name', 'Jim Walsh').as('jim').
V().has('person', 'name', 'Sharon Smith').as('sharon').
addE('knows').from('jim').to('sharon').
property('since', 1980)
```

## aggregate

Aggregate all objects into a collection at a particular point in a graph traversal.

## Synopsis

```
aggregate('variable_name')
```

**Table 54: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `aggregate()` step is a [sideEffect \(page 653\)](#). A vertex is added from a traversal `g` using `addV`. A previously created [vertex label \(page 644\)](#) must be specified.

## Examples

Return a collection of all the people that John Doe knows as an aggregate:

```
g.V().has('person', 'name', 'John Doe').
 out('knows').aggregate('x')
```

Return all the friends of John Doe's friends, including those that are friends of John Doe:

```
g.V().has('person', 'name', 'John Doe').
 out('knows').aggregate('x').
 in('knows').out('knows').valueMap('name', 'gender')
```

Find all the friends of John Doe's friends that are not friends of John Doe:

```
g.V().has('person', 'name', 'John Doe').
 out('knows').aggregate('x').
 in('knows').out('knows').
 where(without('x'))
```

Note the use of the `aggregate('x')` in the later step `where(without('x'))` that is used to exclude John Doe's friends using the aggregate assigned to the variable `x`.

## and

Filter traversals using the AND boolean.

## Synopsis

```
and(traversal, traversal, ...)
```

**Table 55: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.

## Description

The `property()` step is a [filter \(page 652\)](#). The `and()` step can take an arbitrary number of traversals to filter the returned objects using a boolean AND function. The related boolean OR function can be done with the [or\(\) \(page 715\)](#) step.

## Examples

Find all the person vertices that have created recipes AND authored books:

```
g.V().hasLabel('person').
 and(outE('created'), outE('authored')).
 values('name')
```

## barrier

Causes all steps prior to the barrier step to be executed before moving onto additional steps.

## Synopsis

```
barrier()
```

**Table 56: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>italics</i>	Variable value. Replace with a user-defined value.

Syntax conventions	Description
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `barrier()` step is a bulk optimizing step that can create a single traverser for a repeated step.

## Examples

Repeat this traversal with the `barrier()` step and then without, to see the effect the barrier step has on the operation. The basic query starts at a particular book, then traverses to the `includedIn` edges that point to that book to

```
g.V().has('book', 'bookId', 1004).
 in('includedIn').
 in('includedIn').
 groupCount('x').
 by('name').
 barrier().
 project('a', 'b').
 by('name').
 by(select('x'))
```

## branch

Split a traversal based on values.

## Synopsis

```
branch(traversal)
```

**Table 57: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.

Syntax conventions	Description
{}	Group. Braces ( {} ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.

## Description

The `branch` step is a [general step \(page 653\)](#).

## Examples

Find all ingredients in a particular fridge (based on the `fridgeSensor id`), count, and print an message depending on the count returned.

```
g.V().hasLabel('ingredient').as('ingredient').

branch(inE('contains').filter(outV().has('stateId',31).has('cityId',200)).count()).
 option(0L, constant('unavailable')).
 option(1L, constant('time to reorder')).
 option(none, constant('available')).as('status').
select('ingredient','status').
 by('name').
 by()
```

## cap

### Synopsis

```
cap('variable_name' , ...)
```

**Table 58: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>italics</i>	Variable value. Replace with a user-defined value.
[]	Optional. Square brackets ( [] ) surround optional command arguments. Do not type the square brackets.
{}	Group. Braces ( {} ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.

Syntax conventions	Description
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `cap()` step is a barrier step that iterates the traversal up to itself and returns the referenced object by the provided key. If multiple keys are provided, then a map of objects is returned.

## Examples

Get the count of each vertex label and return as a map of vertex label : count value key-value pairs:

```
g.V().groupCount('a').by(label).cap('a')
```

## choose

Route a traverser to a particular traversal branch option.

## Synopsis

```
choose(traversal,choice_1,choice_2,choice_3)
```

**Table 59: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `choose()` step is a [branch step \(page 653\)](#) that can be used to route a traverser to a particular traversal branch, similar to if-then-else logic.

## Examples

```
g.V().hasLabel('store').as('store').
V().hasLabel('ingredient').as('ingredient').
```

```

choose(where(__.in('isStockedWith').as('store')),
 constant('Y'), constant('N')).as('in stock').
select('store','ingredient','in stock').
by('name').
by('name').
by()

```

## coalesce

Return a value based on the first traversal that has an element.

### Synopsis

```
coalesce(traversal,traversal_1, ...)
```

**Table 60: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.

### Description

The `coalesce()` step evaluates the provided traversals in order and returns the first traversal that emits at least one element.

### Examples

Create a list of each person and their mean average of stars given for recipe reviews, using `coalesce()` to either print the mean or print a constant zero value if no reviews are found:

```

"g.V().hasLabel('person').as('person','starCount').
select('person','starCount').
by('name').
by(coalesce(outE('reviewed').values('stars'),constant(0)).mean()).
order().by(select('starCount'), desc)"

```

## coin

Get a random coin toss value with stated odds.

## Synopsis

**Table 61: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `coin()` step is a [filter step \(page 652\)](#) that can be used to get a random coin toss with the inserted odds between 0.0 and 1.0.

## Examples

Find a random set of ingredients that, for instance, must be used in a cooking contest:

```
g.V().hasLabel('ingredient').coin(0.1)
```

## constant

Set a constant value.

## Synopsis

```
constant(value)
```

**Table 62: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.

Syntax conventions	Description
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `constant()` step is a [map step \(page 652\)](#) that is often useful with [choose\(\)](#) ([page 686](#)) or [coalesce\(\)](#) ([page 687](#)) steps.,

## Examples

Use `constant(0)` to designate zero as the value for any returns that are not a number, such as `NaN` (not a number):

```
g.V().hasLabel('person').as('person', 'starCount').
 select('person', 'starCount').
 by('name').
 by(coalesce(outE('reviewed').values('stars'), constant(0)).mean()).
 order().by(select('starCount'), decr)
```

## count

Get a count of the traverser that precedes the count step.

## Synopsis

```
count()
```

**Table 63: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ()..
<i>Italics</i>	Variable value. Replace with a user-defined value.
[]	Optional. Square brackets ( [] ) surround optional command arguments. Do not type the square brackets.
{}	Group. Braces ( {} ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `count()` step is a [map step \(page 652\)](#) that counts whatever precedes it.

## Examples

Count the number of reviews for each recipe, then print the recipe name, the review count, and the review average stars given"

```
g.V().hasLabel('recipe').as('recipe','numberOfReviews','meanRating').
 select('recipe','numberOfReviews','meanRating').
 by('name').
 by(inE('reviewed').count()).
 by(inE('reviewed').values('stars').mean())
```

## cyclicPath

Filter out repeated steps in a traversal

## Synopsis

**Table 64: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `cyclicPath()` step is a [filter step \(page 652\)](#) that allows a traversal to filter out any repeats as the traversal proceeds.

## Examples

Find all people where a person both created and reviewed a recipe:

```
g.V().hasLabel('person').as('a').
 out('created').as('b').
 filter(_.in('reviewed').cyclicPath()).
 select('a','b').
```

```
by('name')
```

## dedup

Deduplicate returned objects from a query.

### Synopsis

```
dedup()
```

**Table 65: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

### Description

The `dedup()` step is a [filter step \(page 652\)](#) that can be used to eliminate duplication of returned objects in a traversal query.

### Examples

Find all the ingredients in Julia Child's recipes, and deduplicate the returned list:

```
g.V().has('person', 'name', 'Julia Child').
 out('created').
 in('includedIn').dedup().
 values('name')
```

## drop

Drop a traversal object.

### Synopsis

```
drop()
```

**Table 66: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `drop()` step is a [map \(page 652\)](#)/[sideEffect \(page 653\)](#). Appending `drop()` to any traversal will drop the preceding objects from the graph.

## Examples

Drop all outgoing edges from all vertices:

```
g.V().outE().drop()
```

Drop the property key *name* from all vertices; the values stored for the property will also be dropped from the graph.

```
g.V().properties('name').drop()
```

Drop all vertices from a graph:

```
g.V().drop()
```

## explain

Explain the traversal strategies used in a traversal.

## Synopsis

```
explain()
```

**Table 67: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.

Syntax conventions	Description
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `explain()` step is a terminal step that will return a traversal explanation detailing how the traversal is compiled, and the [TinkerPop traversal strategies](#) that are used.

## Examples

```
g.V().not(hasLabel('fridgeSensor')).or().hasLabel('meal')).
group().by(label).by('name').explain()
```

The return of `explain()` for the example:

```
Traversal Explanation
=====
Original Traversal [GraphStep(vertex,
[]), NotStep([HasStep([~label.eq(fridgeSensor)]),
OrStep, HasStep([~label.eq(meal)]))), GroupStep(label,
[TraversalMapStep(value(name)), FoldStep])]

ConnectiveStrategy [D] [GraphStep(vertex,
[]), NotStep([OrStep([[HasStep([~label.eq(fridgeSensor)]),
[HasStep([~label.eq(meal)])]]], GroupStep(label,
[TraversalMapStep(value(name)), FoldStep])]
RepeatUnrollStrategy [O] [GraphStep(vertex,
[]), NotStep([OrStep([[HasStep([~label.eq(fridgeSensor)]),
[HasStep([~label.eq(meal)])]]], GroupStep(label,
[TraversalMapStep(value(name)), FoldStep])]
MatchPredicateStrategy [O] [GraphStep(vertex,
[]), NotStep([OrStep([[HasStep([~label.eq(fridgeSensor)]),
[HasStep([~label.eq(meal)])]]], GroupStep(label,
[TraversalMapStep(value(name)), FoldStep])]
PathRetractionStrategy [O] [GraphStep(vertex,
[]), NotStep([OrStep([[HasStep([~label.eq(fridgeSensor)]),
[HasStep([~label.eq(meal)])]]], GroupStep(label,
[TraversalMapStep(value(name)), FoldStep])]
IncidentToAdjacentStrategy [O] [GraphStep(vertex,
[]), NotStep([OrStep([[HasStep([~label.eq(fridgeSensor)]),
[HasStep([~label.eq(meal)])]]], GroupStep(label,
[TraversalMapStep(value(name)), FoldStep])]
```

```

FilterRankingStrategy [O] [GraphStep(vertex,
[]), NotStep([OrStep([[HasStep([~label.eq(fridgeSensor)]),
[HasStep([~label.eq(meal)])]]]), GroupStep(label,
[TraversalMapStep(value(name)), FoldStep]])]
InlineFilterStrategy [O] [GraphStep(vertex,
[]), NotStep([HasStep([~label.or(eq(fridgeSensor), eq(meal))])]),
GroupStep(label,[TraversalMapStep(value(name)), FoldStep]])]
AdjacentToIncidentStrategy [O] [GraphStep(vertex,
[]), NotStep([HasStep([~label.or(eq(fridgeSensor), eq(meal))])]),
GroupStep(label,[TraversalMapStep(value(name)), FoldStep]])]
CountStrategy [O] [GraphStep(vertex,
[]), NotStep([HasStep([~label.or(eq(fridgeSensor), eq(meal))])]),
GroupStep(label,[TraversalMapStep(value(name)), FoldStep]])]
LazyBarrierStrategy [O] [GraphStep(vertex,
[]), NotStep([HasStep([~label.or(eq(fridgeSensor), eq(meal))])]),
GroupStep(label,[TraversalMapStep(value(name)), FoldStep]])]
DseIncidentToAdjacentStrategy [O] [GraphStep(vertex,
[]), NotStep([HasStep([~label.or(eq(fridgeSensor), eq(meal))])]),
GroupStep(label,[TraversalMapStep(value(name)), FoldStep]])]
HasStepStrategy [P] [GraphStep(vertex,[]),
NotStep([DsegHasStep([~label.or(eq(fridgeSensor), eq(meal))])]),
GroupStep(label,[TraversalMapStep(value(name)), FoldStep]])]
QueryStrategy [P] [DsegGraphStep(vertex,
[],true,Unordered), NotStep([DsegHasStep([~label.or(eq(fridgeSensor),
eq(meal))])]), GroupStep(label,[TraversalMapStep(value(name)),
FoldStep])]]
AdjacentVertexFilterOptimizerStrategy [P] [DsegGraphStep(vertex,
[],true,Unordered), NotStep([DsegHasStep([~label.or(eq(fridgeSensor),
eq(meal))])]), GroupStep(label,[TraversalMapStep(value(name)),
FoldStep])]]
DsegPropertyLoadStrategy [F] [DsegGraphStep(vertex,
[],true,Unordered), NotStep([DsegHasStep([~label.or(eq(fridgeSensor),
eq(meal))])]), GroupStep(label,[TraversalMapStep(value(name)),
FoldStep])]]
ProfileStrategy [F] [DsegGraphStep(vertex,
[],true,Unordered), NotStep([DsegHasStep([~label.or(eq(fridgeSensor),
eq(meal))])]), GroupStep(label,[TraversalMapStep(value(name)),
FoldStep])]]
StandardVerificationStrategy [V] [DsegGraphStep(vertex,
[],true,Unordered), NotStep([DsegHasStep([~label.or(eq(fridgeSensor),
eq(meal))])]), GroupStep(label,[TraversalMapStep(value(name)),
FoldStep])]]
LambdaRestrictionStrategy [V] [DsegGraphStep(vertex,
[],true,Unordered), NotStep([DsegHasStep([~label.or(eq(fridgeSensor),
eq(meal))])]), GroupStep(label,[TraversalMapStep(value(name)),
FoldStep])]]

Final Traversal [DsegGraphStep(vertex,
[],true,Unordered), NotStep([DsegHasStep([~label.or(eq(fridgeSensor),
eq(meal))])]), GroupStep(label,[TraversalMapStep(value(name)),
FoldStep])]]

```

The codes for various steps:

**[D]ecoration**

There is an application-level feature that can be embedded into the traversal logic.

**[O]ptimization**

There is a more efficient way to express the traversal at the TinkerPop level.

**[P]rovider optimization**

There is a more efficient way to express the traversal at the graph system, language, or driver level.

**[F]inalization**

There are some final adjustments, cleanups, or analyses required before executing the traversal.

**[V]erification**

There are certain traversals that are not legal for the application or traversal engine.

**fill**

Put all results in the provided collection, then return the collection.

**Synopsis**

```
fill(collection_name)
```

**Table 68: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

**Description**

The `fill()` step is a terminal step that will put all the results of a traversal into the named collection, such as an array. The collection is returned when the filling is complete.

**Examples**

```
results = []
```

```
g.V().out('authored').fill(results)
```

## filter

Filter the traversal to eliminate some of the returned objects based on a criteria.

### Synopsis

```
filter(traversal)
```

**Table 69: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

### Description

The `filter()` step is a [general filter step \(page 652\)](#). This step is used to eliminate some of the returned objects at a particular point in a traversal based on some criteria.

### Examples

Find the books that include an edge from a recipe:

```
g.V().hasLabel('book').filter(__.in('includedIn').hasLabel('recipe'))
```

## flatMap

Create a map of traversal objects and stream to the next traversal step.

### Synopsis

```
flatMap(traversal)
```

**Table 70: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>italics</i>	Variable value. Replace with a user-defined value.

Syntax conventions	Description
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `flatMap()` step is a [general flatMap step \(page 652\)](#). This step creates a map of the traversal objects and streams the map to the next traversal step.

## Examples

Return a map of the recipe properties included in the cookbooks in the graph, but limit the return to 3 recipes:

```
g.V().hasLabel('book').flatMap(__.in('includedIn').hasLabel('recipe')).limit(3))
```

## fold

Aggregate all the returned objects into a list, then emit the list.

## Synopsis

```
fold()
```

**Table 71: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `fold()` step is a [map step \(page 652\)](#). It aggregates all the returned objects into a list, then emits the list into the next traversal step.

## Examples

Get back a list of all the persons that each person knows:

```
g.V().hasLabel('person').out('knows').values('name').fold()
```

## group

Organize returned objects by a function of the objects.

## Synopsis

```
group()
```

**Table 72: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes <code>( )</code> .
<i>italics</i>	Variable value. Replace with a user-defined value.
<code>[ ]</code>	Optional. Square brackets <code>( [ ] )</code> surround optional command arguments. Do not type the square brackets.
<code>{ }</code>	Group. Braces <code>( { } )</code> identify a group to choose from. Do not type the braces.
<code> </code>	Or. A vertical bar <code>(   )</code> separates alternative elements. Type any one of the elements. Do not type the vertical bar.
<code>...</code>	Repeatable. An ellipsis <code>( . . . )</code> indicates that you can repeat the syntax element as often as required.

## Description

The `group()` step is a [map \(page 652\)](#)/[sideEffect \(page 653\)](#). The step organizes the objects according to some function of the object, and reduces the returned objects to a list. .

## Examples

List recipes by name, grouping by cuisine:

```
g.V().hasLabel('recipe').
 group().
 by('cuisine').
```

```
by('name')
```

## groupCount

Determines the number of objects specified.

### Synopsis

```
groupCount()
```

**Table 73: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

### Description

The `groupCount()` step is a [map \(page 652\)](#)/[sideEffect \(page 653\)](#). The step determines the number of objects specified.

### Examples

Print the number of ingredients per recipe:

```
g.V().hasLabel('recipe').
 groupCount().
 by(inE('includedIn').count())
```

## has

Filter based on a particular label or value.

### Synopsis

```
has({ 'vertexLabel' | 'edgeLabel' } , ['propertyKey_name' ,
 'propertyKey_value'] . . .)
```

**Table 74: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `has()` step is a [filter \(page 652\)](#) step. It is the most common step used for graph traversals, since this step narrows the query to find particular vertices or edges with certain property values.

## Examples

Find the meal item that is called a taco:

```
g.V().has('meal_item', 'name', 'taco')
```

## hasNext

Determine if results are available from a traversal.

## Synopsis

```
hasNext()
```

**Table 75: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.

Syntax conventions	Description
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `hasNext()` step is a terminal step. It determines whether or not there are available results from a traversal, returning a Boolean value of true or false.

## Examples

```
// determines whether there are available results
g.V().hasLabel('book').hasNext()
```

## id

Fetch the id of a graph object.

## Synopsis

```
id()
```

**Table 76: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `id()` step is a [map \(page 652\)](#) step that retrieves the id of a graph object.

## Examples

Fetch the id of the person named James Beard:

```
g.V().has('person', 'name', 'James Beard').id()
```

Fetch the id of the edge with an edgeLabel created:

```
g.E().hasLabel('created').id()
```

## inject

Inject an object arbitrarily into a traversal stream.

### Synopsis

```
inject('object_name')
```

**Table 77: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

### Description

The `inject()` step is a [side effect \(page 653\)](#). It insert objects arbitrarily into a traversal stream, but does not change the graph permanently.

### Examples

```
g.V().has('person', 'name', 'James
Beard').out().values('name').inject('Cream Biscuits')
```

## is

Filter for scalar values.

### Synopsis

**Table 78: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>italics</i>	Variable value. Replace with a user-defined value.

Syntax conventions	Description
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `is()` step is a [filter \(page 652\)](#) step that is used to filter for scalar values.

## Examples

```
g.V().
 where(__.hasLabel('meal_item').values('calories').is(lte(1200))).
 values('name', 'calories')
```

Note that this example could be accomplished with a `has()` step.

But this example could not be accomplished with a `has()` step:

```
g.V().hasLabel('meal').filter(out('includes').values('calories').sum().is(gte(1000)))
```

## key

Extracts property keys for specified object.

## Synopsis

```
key()
```

**Table 79: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ()..
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.

Syntax conventions	Description
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `key()` is a [map \(page 652\)](#) step that extracts the property keys for a specified object.

## Examples

Get a deduplicated list of all the `vertexLabel` property keys:

```
g.V().properties().key().dedup()
```

## label

Extracts the specified labels.

## Synopsis

```
label()
```

**Table 80: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ()..
<i>italics</i>	Variable value. Replace with a user-defined value.
[]	Optional. Square brackets ( [] ) surround optional command arguments. Do not type the square brackets.
{}	Group. Braces ( {} ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `label()` step is a [map \(page 652\)](#) step that extracts the specified labels.

## Examples

Get all the vertex and edge labels in a deduplicated list:

```
g.V().label().dedup()
```

## limit

Limit the objects returned.

### Synopsis

```
limit(integer_value)
```

**Table 81: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

### Description

The `limit()` step allows a limit to be set for the number of returned items. It can also be used with the `local()` step to scope the effect within a traversal.

### Examples

Find all vertices with a vertex label *ingredient*, and limit the return to 10 items starting with 0 (zero).

```
g.V().hasLabel('ingredient').limit(10)
```

## local

Limit the operation of the traversal to a single element within the traversal.

### Synopsis

```
local(traversal)
```

**Table 82: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).

Syntax conventions	Description
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `local()` step is a [branch \(page 653\)](#) step that allows the action within the step to be applied to the element within the `local()` step.

## Examples

Get a list of all people and the countries they have lived in, ordered by the year that the person started living in the country:

```
g.V().hasLabel('person').as('person').
 local(properties('country').order().by('startYear',
 incr).limit(2)).value().as('country').
 select('person', 'country').
 by('name').by()
```

Here, `local()` is used to restrict the sorting by the meta-property `startYear` for each person.

```
g.V().has('person', 'personId', '1').local(outE('created').has('createDate',
 gte('1962-03-03')).order().by('createDate'))
```

```
// explain
//g.V().hasLabel('person').outE('reviewed').has('stars',
 gte(1)).order().by('stars').profile()
// Sorts the reviews by stars for each user - uses the edge index
//g.V().hasLabel('person').local(outE('reviewed').has('stars',
 gte(1)).order().by('stars')).profile()
// Sorts the users by name, and then their reviews by stars
//
g.V().hasLabel('person').order().by('name').local(outE('reviewed').has('stars',
 gte(1)).order().by('stars')).profile()
```

## loops

Get the number of times a query has gone through a particular loop.

## Synopsis

```
loops()
```

**Table 83: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `loops()` step is a [map \(page 652\)](#) step that extracts the number of times the traverser has gone through the current loop.

## Examples

```
g.V().hasLabel('person','name','John Doe').
repeat(out('knows').dedup().
 aggregate('x').
 by(project('person','level').
 by('name').
 by(loops())))
).
until(__.not(out('knows').simplePath())).
cap('x').next()
```

## map

Map a portion of the query.

## Synopsis

```
map(traversal)
```

**Table 84: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `map()` step is a general map step ([page 652](#)). This step creates a map of the traverser to some object for the next step to process.

## Examples

List all recipes and a map of their ingredients:

```
g.V().hasLabel('recipe').as('recipe').
 map(__.in('includedIn').values('name').fold()).as('ingredients').
 select('recipe','ingredients').
 by('name').
 by()
```

## match

Provides a more declarative form of graph querying based on pattern matching.

## Synopsis

```
match(traversal_fragment, . . .)
```

**Table 85: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.

Syntax conventions	Description
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `match()` step is a [map \(page 652\)](#) step provides a more declarative form of graph querying based on pattern matching.

## Examples

Find the ingredients in the recipe for *Beef Bourguignon*:

```
g.V().match(
 __.as('INGREDIENT').hasLabel('ingredient'),
 __.as('INGREDIENT').out('includedIn').has('name', 'Beef
 Bourguignon')).
select('INGREDIENT').by('name')
```

## math

Enables scientific calculator functionality.

## Synopsis

```
math('math_function')
```

**Table 86: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>Italics</i>	Variable value. Replace with a user-defined value.
[]	Optional. Square brackets ( [] ) surround optional command arguments. Do not type the square brackets.
{}	Group. Braces ( {} ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `math()` step enables scientific calculator functionality. The basic operations ( +, -, \*, /, ^, and %) are available, as well as:

Function	Description
abs	absolute value
acos	arc cosine
asin	arc sine
atan	arc tangent
cbrt	cubic root
ceil	nearest upper integer
cos	cosine
cosh	hyperbolic cosine
exp	Euler's number raised to the power e <sup>x</sup>
floor	nearest lower integer
log	natural logarithm (base e)
log10	logarithm (base 10)
log2	logarithm (base 2)
sin	sine
sinh	hyperbolic sine
sqrt	square root
tan	tangent
tanh	hyperbolic tangent
signum	signum function

## Examples

Find the calories per meal item, then multiply by the number of servings for a meal:

```
g.V().hasLabel('meal_item').as('a').
 inE('includes').as('b').
 math('a*b').
 by('calories').
 by('numServ')
```

Note that two `by()` statements are used to retrieve a particular property for each math variable used, `calories` for `a` and `numServ` for `b`.

## max

Get the largest number in a traversal.

## Synopsis

```
max()
```

**Table 87: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `max()` step is a [map \(page 652\)](#) step that will discover the largest number generated from the previous step in the traversal.

## Examples

Find the maximum number of reviews written by any person:

```
g.V().hasLabel('person').map(outE('reviewed').count()).max()
```

## mean

Get the mean average in a traversal.

## Synopsis

```
mean()
```

**Table 88: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.

Syntax conventions	Description
{}	Group. Braces ( {} ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.

## Description

The `mean()` step is a [map \(page 652\)](#) step that will discover the mean average of the numbers generated from the previous step in the traversal.

## Examples

Find the mean average of the stars for reviews for each reviewer:

```
g.V().hasLabel('person').as('person', 'starCount').
 select('person', 'starCount').
 by('name').
 by(outE('reviewed').values('stars').mean()).
 order().by(select('starCount'), decr)
```

Find the mean average of the number of ingredients for each recipe:

```
g.V().hasLabel('recipe').
 map(inE('includedIn').count()).
 mean()
```

## min

Get the smallest number in a traversal.

## Synopsis

**Table 89: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ()..
<i>Italics</i>	Variable value. Replace with a user-defined value.
[]	Optional. Square brackets ( [] ) surround optional command arguments. Do not type the square brackets.
{}	Group. Braces ( {} ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.

Syntax conventions	Description
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `min()` step is a [map \(page 652\)](#) step that will discover the smallest number generated from the previous step in the traversal.

## Examples

Find the minimum number of recipes created by any person:

```
g.V().hasLabel('person').map(outE('created').count()).min()
```

## next

Find the next results.

## Synopsis

```
next(integer)
```

**Table 90: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `next()` step is a terminal step. It returns the next number of steps, based on a supplied integer value.

## Examples

Return the next two results found:

```
g.V().hasLabel('store').next(2)
```

## not

Exclude a specified object from the traversal.

### Synopsis

```
not(traversal)
```

**Table 91: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

### Description

The `not()` step is a [filter \(page 652\)](#) step that excludes the objects specified.

### Examples

If a vertex is not a `fridgeSensor` or a `meal`, group each vertex type and list by name:

```
g.V().not(hasLabel('fridgeSensor')).or().hasLabel('meal')).
group().by(label).by('name')
```

## optional

Returns an optional result if the specified traversal yields a result; otherwise, returns the calling element.

### Synopsis

```
optional(traversal_fragment)
```

**Table 92: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .

Syntax conventions	Description
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `optional()` step is a [branch \(page 653\)](#)/[flatMap \(page 652\)](#) step that will either return the result of the specified traversal if it yields a result, or the return of the calling element.

## Examples

Returns the friends of John Doe, since he has friends:

```
g.V().has('person', 'name', 'John Doe').optional(out('knows'))
```

Returns Julia Child, since she has no linked friends:

```
g.V().has('person', 'name', 'Julia Child').optional(out('knows'))
```

or

Yield at least one result from choices.

## Synopsis

```
optional()
```

**Table 93: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.

Syntax conventions	Description
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `or()` step is a [filter \(page 652\)](#) step that ensures that at least one of the provided traversals yield a result.

## Examples

Finds all vertices that are not a `fridgeSensor` or a `meal` and groups them by name:

```
g.V().not(hasLabel('fridgeSensor')).or().hasLabel('meal')).
group().by(label).by('name')
```

## order

Sorts objects given a certain criteria.

## Synopsis

```
order()
```

**Table 94: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `order()` step is a map step that sorts returned objects given a certain criteria.

## Examples

```
g.V().hasLabel('person').values('name').order().by(decr).dedup()
```

## pageRank

Calculates pageRank.

### Synopsis

```
pageRank()
```

**Table 95: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `pageRank()` step is a map/sideEffect step that calculates [PageRank](#). It only runs in Analytics (OLAP) mode.

## Examples

```
g = graph.traversal().withComputer()
g.V().pageRank().by('pageRank').values('pageRank')
```

## path

Examine the history of a traversal's path.

### Synopsis

```
path()
```

**Table 96: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `path()` step is a [map \(page 652\)](#) step that examines the history of the traversal path.

## Examples

Find which recipes are included in which books that include the ingredients of *beef* or *carrots*:

```
g.V().has('ingredient', 'name',
within('beef', 'carrots')).out('includedIn').as('Recipe').
out().hasLabel('book').as('Book').
select('Book', 'Recipe').
by('name').by('name').path()
```

## peerPressure

Executes a Peer Pressure community detection algorithm over the graph.

## Synopsis

```
peerPressure()
```

**Table 97: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.

Syntax conventions	Description
{}	Group. Braces ( {} ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.

## Description

The `peerPressure()` step is a map/sideEffect step that executes a Peer Pressure community detection algorithm over the graph. It only runs in Analytics (OLAP) mode.

## Examples

```
g = graph.traversal().withComputer()
g.V().peerPressure().by('cluster').values('cluster')
```

## profile

Profile a traversal to determine statistical information.

## Synopsis

```
profile()
```

**Table 98: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>Italics</i>	Variable value. Replace with a user-defined value.
[]	Optional. Square brackets ( [] ) surround optional command arguments. Do not type the square brackets.
{}	Group. Braces ( {} ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.

## Description

The `profile()` step is a [side effect \(page 653\)](#) step that profiles a traversal to determine statistical information like step runtime, counts, and percent time spent in each traverser. See [Anatomy of a Query \(page 566\)](#).

## Examples

```
g.V().out('created').repeat(both()).times(3).hasLabel('person').profile()
project
```

## Synopsis

```
project('variable_name', ...)
```

**Table 99: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( ... ) indicates that you can repeat the syntax element as often as required.

## Description

The `project()` step is a [map \(page 652\)](#) step that projects the current object into a map keyed by provided labels. It is similar to `select()` step.

## Examples

Create a map of outgoing and incoming edge counts for a recipe:

```
g.V().has('name', 'Beef Bourguignon').
 project('out','in').
 by(outE().count()).
 by(inE().count())
```

## properties

Retrieve properties of a specified element.

## Synopsis

```
properties(['property_name', ...])
```

**Table 100: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `properties()` step retrieves the properties of the specified element.

## Examples

Find the properties of *Julia Child*:

```
g.V().has('person', 'name', 'Julia Child').properties()
```

Find the countries that *Julia Child* has lived in:

```
g.V().has('person', 'name', 'Julia Child').properties('country')
```

## propertyMap

Yields a map representation of the properties of an element.

## Synopsis

**Table 101: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.

Syntax conventions	Description
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `propertyMap()` step is a [map \(page 652\)](#) step that yields a map representation of the properties of a specified element.

## Examples

Return a map of the properties of all stores:

```
g.V().hasLabel('store').propertyMap()
```

## range

Filter only a specified number of objects into the next step.

## Synopsis

```
range([local], low_integer, high_integer)
```

**Table 102: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `range()` step is a [filter \(page 652\)](#) step that allows only the specified number of objects to pass onto the next step.

## Examples

Find the first two countries listed for each person:

```
g.V().valueMap()
 select('country')
 range(local, 0, 2)
```

Find the first two countries that each person has lived in ordered by `startYear`:

```
g.V().hasLabel('person').
 local(properties('country').order().by('startYear')).range(0,
2).value().fold()
```

## repeat

Loop until a condition is met.

### Synopsis

```
repeat(traversal)
```

**Table 103: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

### Description

The `repeat()` step is a [branch \(page 653\)](#) step and is used for looping over a traversal to some given break predicate. Three step modulators can be used with the `repeat()` step, [emit\(\) \(page 664\)](#), [times\(\) \(page 666\)](#), or [until\(\) \(page 668\)](#).

### Examples

Repeatedly gets the outgoing vertex, three times, and prints the results:

```
g.V().hasLabel('fridgeSensor').repeat(out()).times(3).valueMap()
```

A more complex example that repeats through all the outgoing *knows* adjacent vertices of *John Doe*:

```
g.V().hasLabel('person', 'name', 'John Doe').
 repeat(out('knows')).dedup()
```

```

aggregate('x').
 by(project('person','level').
 by('name').
 by(loops())
)
).
until(__.not(out('knows').simplePath())).
cap('x').next()

```

## sack

Read and write sacks.

### Synopsis

```
sack(traversal)
```

**Table 104: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

### Description

The `sack()` step is a [map \(page 652\)](#)/[sideEffect \(page 653\)](#) step that is used to read or write sacks, local data structures that traversers use.

### Examples

```

g.withSack(0).V().
 has("person", "name", "John Doe").
 outE("reviewed").
 sack(sum).by("stars").
 outV().
 sack()

```

## sample

Sample previous steps in the traversal.

## Synopsis

```
sample(integer_value)
```

**Table 105: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `sample()` step is useful for sampling some number of traversers previous in the traversal.

## Examples

Get three samples of the property `numServ` from the `includes` edges of a `meal_item`:

```
g.V().hasLabel('meal_item').inE().sample(3).by('numServ')
```

## select

Select labeled steps.

## Synopsis

```
select('variable_name', . . .)
```

**Table 106: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.

Syntax conventions	Description
{}	Group. Braces ( {} ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `select()` step is a [map \(page 652\)](#) step that selects labeled steps designated with `as()` steps. This step is typically used to select particular properties from objects earlier in the traversal.

## Examples

List meal items and the meals that are linked:

```
g.V().hasLabel('meal_item').as('item').in().as('meal').
 select('item','meal').
 by('name').by('mealId')
```

## sideEffect

Perform some operation on the traverser and pass the result to the next step.

## Synopsis

```
sideEffect(traversal)
```

**Table 107: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>italics</i>	Variable value. Replace with a user-defined value.
[]	Optional. Square brackets ( [] ) surround optional command arguments. Do not type the square brackets.
{}	Group. Braces ( {} ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `sideEffect()` step is a [general filter step \(page 652\)](#). This step performs some operation on the traverser and passes the result to the next step.

## Examples

For each person, find the number of people that person knows, and the number of people known by the person and return the results for both counts:

```
g.V().hasLabel('person').
 sideEffect(outE('knows').count().store('knows')).
 sideEffect(inE('knows').count().store('known by')).
 cap('knows','known by')
```

## simplePath

Use to prevent a traverser from repeating a path through the graph.

## Synopsis

```
simplePath()
```

**Table 108: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `simplePath()` step is a [filter \(page 652\)](#) step. When it is important that a traverser not repeat its path through the graph, this step should be used.

## Examples

Find the expanded neighborhood of all the people who know each other, without repeating any edges:

```
g.V().has('person', 'name', 'John Doe').
 repeat(both('knows').simplePath()).
 emit().
```

```
path()
 by('name')
*.objects()
*.join(" > ")
```

**Note:** This query uses Groovy notation (\*.objects, \*.join) in a Gremlin query, to return pretty results.

The results:

```
"John Doe > John Smith",
"John Doe > Jane Doe",
"John Doe > Betsy Jones",
"John Doe > John Smith > Jane Doe",
"John Doe > Jane Doe > Sharon Smith",
"John Doe > Jane Doe > John Smith",
"John Doe > Betsy Jones > Sharon Smith",
"John Doe > John Smith > Jane Doe > Sharon Smith",
"John Doe > Jane Doe > Sharon Smith > Betsy Jones",
"John Doe > Jane Doe > Sharon Smith > Jim Walsh",
"John Doe > Betsy Jones > Sharon Smith > Jane Doe",
"John Doe > Betsy Jones > Sharon Smith > Jim Walsh",
"John Doe > John Smith > Jane Doe > Sharon Smith > Betsy Jones",
"John Doe > John Smith > Jane Doe > Sharon Smith > Jim Walsh",
"John Doe > Betsy Jones > Sharon Smith > Jane Doe > John Smith"
```

## skip

Skip a specified number of return objects.

### Synopsis

```
skip(integer_value)
```

**Table 109: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ()..
<i>italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `skip()` step is a [filter \(page 652\)](#) step that is analogous to the [range\(\) \(page 722\)](#) step with an upper limit of -1.

## Examples

Find all the people who lived in more than one country:

```
g.V().hasLabel('person').
 filter(values('country').skip(1))
```

## store

Store information for later use in the traversal.

## Synopsis

```
store('variable_name')
```

**Table 110: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `store()` step is a [sideEffect \(page 653\)](#) step that stores information for later use in the traversal.

## Examples

Find the number of servings that any *includes* edge has as a value:

```
g.E().hasLabel('includes').store('x').by('numServ').cap('x')
```

## subGraph

Get a subgraph of the full graph.

## Synopsis

```
subGraph('subGraph_name')
```

**Table 111: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `subGraph()` step is a [sideEffect \(page 653\)](#) step that provides a way to produce an edge-induced subgraph from virtually any traversal. This step is useful for visualization or further analysis on a smaller subset of the full graph.

## Examples

Get a subgraph of all the stores and their stocked items, excluding stores without stocked items and items that are not available in any store:

```
g.E().hasLabel('isStockedWith').subgraph('subGraph').cap('subGraph').next()
```

## sum

Sum the previously returned objects.

## Synopsis

```
sum()
```

**Table 112: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.

Syntax conventions	Description
{}	Group. Braces ( {} ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `sum()` step is a [map \(page 652\)](#) step that sums the previously returned objects in the traversal.

## Examples

Get the sum of the calories for all meal items in each meal:

```
g.V().hasLabel('meal').local(out('includes').values('calories')).sum()
```

## tail

Get the last specified number of returned objects.

## Synopsis

```
tail(integer_value)
```

**Table 113: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>Italics</i>	Variable value. Replace with a user-defined value.
[]	Optional. Square brackets ( [] ) surround optional command arguments. Do not type the square brackets.
{}	Group. Braces ( {} ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `tail()` step is analogous to [limit\(\) \(page 705\)](#) step, except that it emits the last `n` objects instead of the first `n` objects.

## Examples

Get the last 10 items that are ingredients:

```
g.V().hasLabel('ingredient').tail(10)
```

## timeLimit

Limit the traversal to a specified number of milliseconds of execution time.

### Synopsis

```
timeLimit(integer_value)
```

**Table 114: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `timeLimit()` step is a [filter \(page 652\)](#) step that limits the traversal to a specified number of milliseconds of execution time.

## Examples

Limit the traversal to 2 milliseconds of execution time:

```
g.V().repeat(timeLimit(2).both().groupCount('m')).times(16).cap('m').order(local).by(value
```

## toBulkSet

Return all results in a weighted set.

### Synopsis

```
toBulkSet()
```

**Table 115: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `toBulkSet()` step is a terminal step that returns all results in a weighted set.

## Examples

Will return all results in a weighted set, with duplicates preserved via weighting:

```
g.V().out('reviewed').toBulkSet()
```

## toList

Return all results in a list.

## Synopsis

```
toList()
```

**Table 116: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.

Syntax conventions	Description
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `toList()` step is a terminal step that will return all the results in a list.

## Examples

```
g.V().out('reviewed').toList()
```

## toSet

Return all results in a set with duplicates removed.

## Synopsis

```
toSet()
```

**Table 117: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ()..
<i>Italics</i>	Variable value. Replace with a user-defined value.
[]	Optional. Square brackets ( [] ) surround optional command arguments. Do not type the square brackets.
{}	Group. Braces ( {} ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `toSet()` step is a terminal step that will return all the results in a set with all duplicates removed.

## Examples

```
g.V().out('reviewed').toSet()
```

## tree

Create a tree of the traversal.

## Synopsis

```
tree()
```

**Table 118: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `tree()` step is a [sideEffect \(page 653\)](#) step that returns a tree of the traversal.

## Examples

Find John Doe's network as a tree:

```
g.V().has('person', 'name', 'John Doe').
repeat(out('knows').simplePath()).
until(__.not(out('knows').simplePath())).
tree().
by('name')
```

## unfold

Unfold a returned object into a linear form.

## Synopsis

```
unfold()
```

**Table 119: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>Italics</i>	Variable value. Replace with a user-defined value.

Syntax conventions	Description
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `unfold()` step is a [flatMap \(page 652\)](#) step. Unfold a returned object into a linear form, such as a map into each individual value.

## Examples

Find the top three countries that the most people have lived in:

```
g.V().hasLabel('person').values('country').
 groupCount().
 order(local).
 by(values, desc).
 limit(local, 3).
 select(keys).unfold()
```

## union

Merge the results of several traversals.

## Synopsis

```
union(traversal, . . .)
```

**Table 120: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.

Syntax conventions	Description
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `union()` step is a [branch \(page 653\)](#) step and merges the results of an arbitrary number of traversals that are specified.

## Examples

Recipes reviewed by John Smith and the people he knows:

```
g.V().has('person', 'name', 'John Smith').union(out('reviewed'),
 both('knows').out('reviewed')).dedup()
```

## value

Get the value of a property given a specified property.

## Synopsis

```
value()
```

**Table 121: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ()..
<i>italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `value()` step is a [map \(page 652\)](#) step that gets just the value of a property, not the key/value pair for a property, given a specified property.

## Examples

Get the value of a properties for all `person` vertices:

```
g.V().hasLabel('person').properties().value()
```

## valueMap

Yields a map representation of the properties of an element.

### Synopsis

```
valueMap()
```

**Table 122: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes () .
<i>italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

### Description

The `valueMap()` step yields a map representation of the properties of an element.

### Examples

Get a map of the properties and their values for *Jane Doe*:

```
g.V().has('person', 'name', 'Jane Doe').valueMap()
```

Get a map of the properties and their values for the outgoing edges from *Jane Doe*:

```
g.V().has('person', 'name', 'Jane Doe').outE().valueMap()
```

## values

Extract the value of properties for an element.

### Synopsis

```
values(['property_name', ...])
```

**Table 123: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `values()` step is a map step that extracts the values of either all the properties or specified properties for an element.

## Examples

Get all the values of the properties for a meal item of `taco`:

```
g.V().has('meal_item', 'name', 'taco').values()
```

Get only the value of the property `macro` for a meal item of `taco`:

```
g.V().has('meal_item', 'name', 'taco').values('macro')
```

## where

Filters based on a predicate, a sideEffect or a traversal.

## Synopsis

```
where()
```

**Table 124: Legend**

Syntax conventions	Description
Lowercase and uppercase	Literal keyword. Includes ( ).
<i>Italics</i>	Variable value. Replace with a user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
{ }	Group. Braces ( { } ) identify a group to choose from. Do not type the braces.

Syntax conventions	Description
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

## Description

The `where()` step is a [filter \(page 652\)](#) step that filters the returned objects based on a [predicate \(page 653\)](#), a [sideEffect \(page 653\)](#), or a traversal.

## Examples

First find all the reviewers that rated *Beef Bourguignon* 5-stars. Then find what other recipes those reviewers rated 5-stars, and count how many reviewers did by recipe, without including *Beef Bourguignon* in the count. It illustrates the use of the `where()` step with `where(neq('a'))` to exclude *Beef Bourguignon*.

```
g.V().has('recipe', 'name', 'Beef Bourguignon').as('a').
 inE('reviewed').has('stars', 5).outV().outE('reviewed').has('stars',
 5).
 inV().where(neq('a')).groupCount().by('name')
```

## DSE Graph data types

Describes the DSE Graph data types.

DSE Graph has many data types that are aligned with CQL data types. For search indexes, see the relationship between DSE Graph and Solr data types.

**Table 125: DSE Graph Data Types**

DSE Graph Data Type	Description	Schema example
bigint	64-bit signed long	<code>schema.propertyKey('big_number').BigInt().create()</code>
blob	Arbitrary bytes (no validation), expressed as base64 strings  <code>graph.addVertex(T.label,   'answer', 'blob', '42');</code>	<code>schema.propertyKey('serial_string').Blob().create()</code>
boolean	True or false	<code>schema.propertyKey('alive').Boolean().create()</code>
date	Date, in the format of '1940' or '1940-01-01'.	<code>schema.propertyKey('review_date').Date().create()</code>

DSE Graph Data Type	Description	Schema example
decimal	Variable-precision decimal <b>Note:</b> When dealing with currency, DataStax recommends a currency class that serializes to and from an int or use of the decimal data type.	schema.propertyKey('book_price').Decimal().create()
double	64-bit IEEE-754 floating point	schema.propertyKey('stars').Double().create()
duration	Time duration in milliseconds	schema.propertyKey('until').Duration().create()
float	32-bit IEEE-754 floating point	schema.propertyKey('precise').Float().create()
inet	IP address string in IPv4 or IPv6 format, used by the python-cql driver and CQL native protocols	schema.propertyKey('website_ip').Inet().create()
int	32-bit signed integer	schema.propertyKey('age').Int().create()
linestring	Used for geospatial and Cartesian linestrings (double .... points)	schema.propertyKey('road').Linestring().withGeoBounds().create() schema.propertyKey('road').Linestring().withBounds(-1, -1, 1, 1).create()
point	Used for geospatial and Cartesian points (double x, double y); note that this corresponds to longitude/latitude, in that order, for mapping geospatial points.	schema.propertyKey('coordinates').Point().withGeoBounds().create() schema.propertyKey('coordinates').Point().withBounds(-1, -1, 1, 1).create()
polygon	Used for geospatial and Cartesian polygons (double .... points)	schema.propertyKey('block').Polygon().withGeoBounds().create() schema.propertyKey('block').Polygon().withBounds(-1, -1, 1, 1).create()
smallint	2 byte integer	schema.propertyKey('age').Smallint().create()
text	String or UTF-8 encoded string	schema.propertyKey('name').Text().create()
time	Time in the format of '10:00:00' or '10:00'.	schema.propertyKey('time').Time().create()

DSE Graph Data Type	Description	Schema example
timestamp	Date, or date plus time, encoded as 8 bytes since epoch. The <code>timestamp</code> data type must be specified as a valid DSE database <code>timestamp</code> :  <pre>johnDoe.addEdge('rated', beefBourguignon, 'timestamp', '2014-01-01T00:00:00.00Z', 'stars', 5, 'comment', 'Pretty tasty!')</pre>	schema.propertyKey('mealCreationDate').Timestamp().create()
uuid	A UUID in <a href="#">standard UUID</a> format or timeuuid format	schema.propertyKey('authorID').Uuid().create()
varint	Arbitrary-precision integer	schema.propertyKey('number').Varint().create()

## Graph storage in the DSE database keyspace and tables

Describes graph storage in the DSE database at a high level.

DSE Graph uses the DSE database to store schema and data. Three DSE database keyspaces are created for each graph, `<graphname>`, `<graphname_system>`, and `<graphname_pvt>`. For example, for a graph called `food`, the three keyspaces created will be `food`, `food_system`, and `food_pvt`. The first keyspace `food` will hold the data for the graph. The second keyspace `food_system` holds schema and other system data about the graph. The third keyspace `food_pvt` holds information about partitioning for vertices should the graph contain a vertex with a large number of edges that requires the vertex table to be partitioned across the cluster.

In the `<graphname>` keyspace, two tables are created for each vertex label to store vertex and edge information, `vertexLabel_p` and `vertexLabel_e`, respectively. For example, for a vertex label `author`, two tables are created, `author_p` and `author_e`.

## DSE Advanced Replication

Documentation for configuring and using configurable distributed data replication.

DSE Advanced Replication supports configurable distributed data replication from source clusters to destination clusters. It is designed to tolerate sporadic connectivity that can occur in constrained environments, such as retail, oil-and-gas remote sites, and cruise ships.

**Note:** To learn about replication, see [About data distribution and replication](#).

## About DSE Advanced Replication

Configurable distributed data replication from source clusters to destination clusters.

DSE Advanced Replication supports configurable distributed data replication from source clusters to destination clusters. It is designed to tolerate sporadic connectivity that can occur in constrained environments, such as retail, oil-and-gas remote sites, and cruise ships.

**Note:** To learn about replication, see [About data distribution and replication](#).

## Features

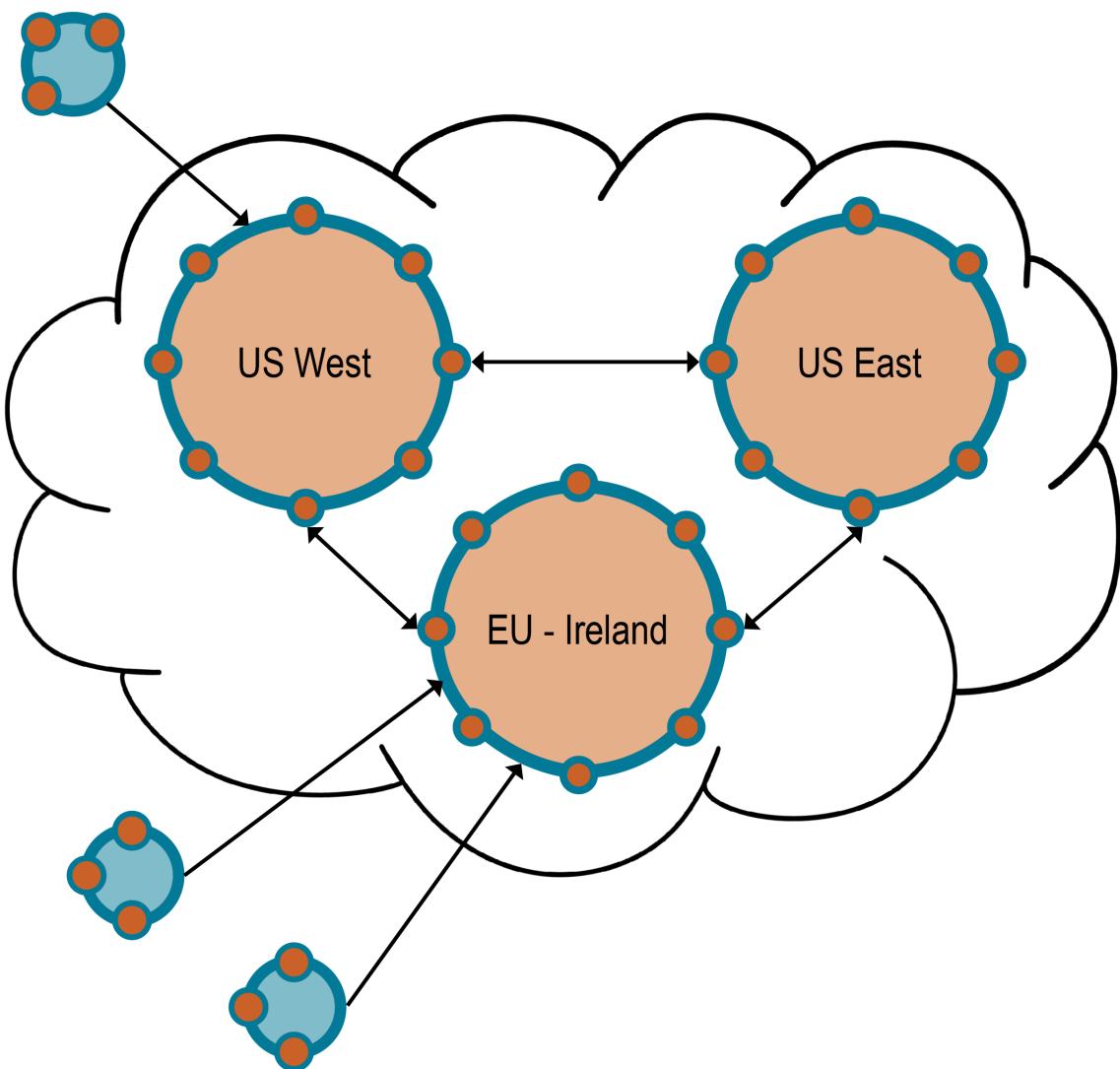
<b>Smartly replicates data from source clusters to destination clusters</b>	Supports replicating data in a spoke and hub configuration from remote locations to central data hubs and repositories. Enterprise customers with remote clusters are able to establish a cluster presence in each location. In addition, mesh configuration can replicate data from any source cluster to another destination cluster within reasonable limits.
<b>Prioritizes data streams</b>	Allows higher priority data streams to be sent from the source cluster to a destination cluster ahead of lower priority data streams.
<b>Supports ingestion and querying of data at every source</b>	DSE Advanced Replication enables ingesting and querying data at any source and sent to any destination that collects and analyzes data from all of the sites.
<b>Solves problem of periodic downtime</b>	Useful for energy (oil and gas), transportation, telecommunications, retail (point-of-sale systems), and other vertical markets that might experience periods of network or internet downtime at the remote locations.
<b>Satisfies data sovereignty regulations</b>	Provides configurable streams of selected outbound data, while preventing data changes to inbound data.
<b>Satisfies data locality regulations</b>	Prevents data from leaving the current geography.

## DSE Advanced Replication architecture

DSE Advanced Replication enables configurable replication between clusters, identifying source and destination clusters with replication channels. Topologies such as hub-and-spoke or mesh networks can differentially push or pull data depending on operational needs.

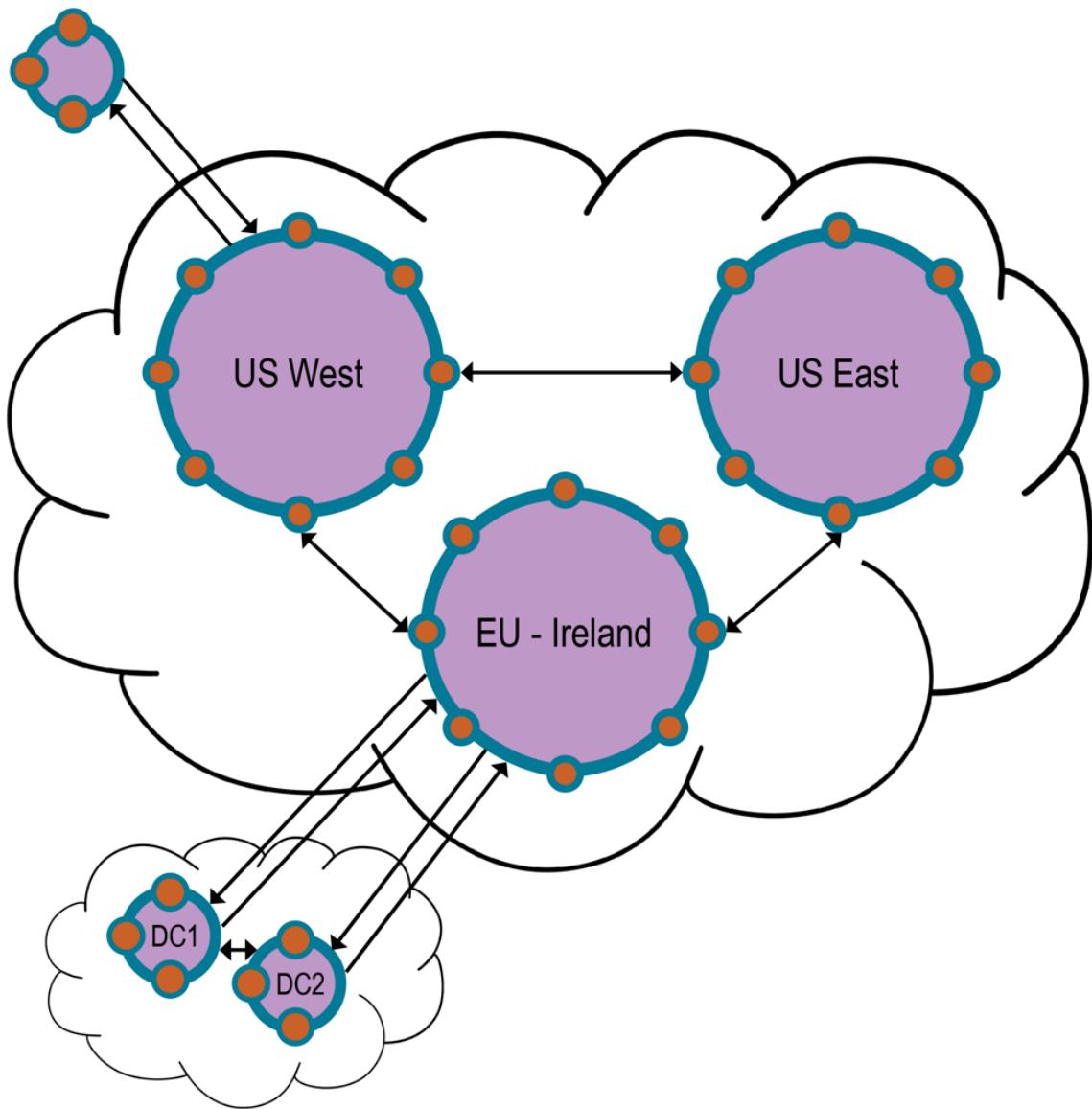
DSE Advanced Replication enables configurable replication between clusters, identifying source and destination clusters with replication channels. Topologies such as hub-and-spoke or mesh networks can differentially push or pull data depending on operational needs.

A common operational scenario for DSE Advanced Replication is a network of remote sensors with poor network connection to a centrally located storage and analytics network. The remote edge clusters collect data, but can experience disconnections from the network and periodically send one-way updates to the central hub clusters when a connection is available. Some sensors may be deemed more important than others, requiring prioritization of transmission. All sensors can continue to collect data, and to transmit in a specified manner, or have collection turned off as needed. Each remote sensor cluster would be designated as a source, while the central database cluster would be a destination.

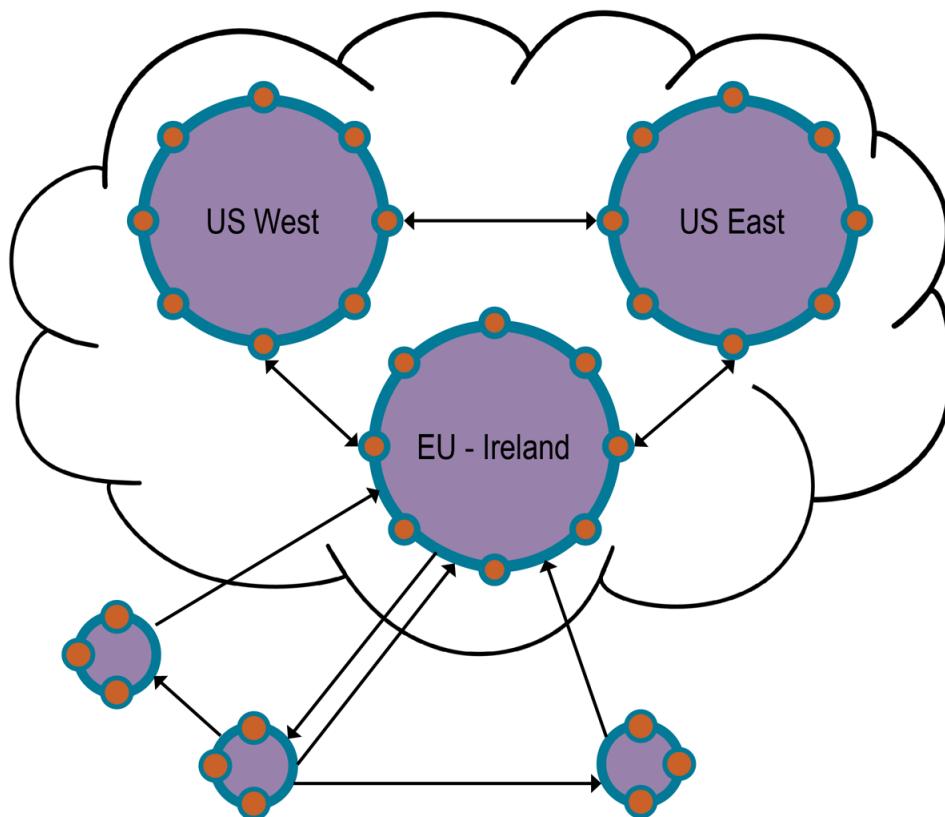


This configuration would also be suitable to a network of microservices clusters that report data to a central analytics cluster.

Another scenario may include similar remote sites that mainly send data to a centralized location, but must periodically be updated with information from the centralized location. In this scenario, each remote cluster would be both a source and a destination, with two channels designated, one upstream and one downstream. A small Point of Sale (POS) system serves as a possible model for this scenario, with periodic updates to the remote systems.



A mesh network can also use advanced replication, with remote clusters receiving updates from either a central location or another remote cluster.



Although any cluster, remote or centralized, may serve as a source for an advanced replication channel, a limited number of destinations can be configured for any one source. In general, consider the flow of replication as many sources to few destinations, rather than few sources to many destinations.

## Traffic between the clusters

Traffic between the source cluster and the destination cluster is managed with permits, priority, and configurable behavior for multi-datacenter operation.

Traffic between the source cluster and the destination cluster is managed with permits, priority, and configurable failover behavior for multi-datacenter operation.

### Permits

Traffic between the source cluster and the destination cluster is managed with permits. When a permit cannot be acquired, the message is postponed and waits in the replication log until it is processed when a permit becomes available. Permits are global and not per destination.

To manage permits and set the maximum number of messages that can be replicated to all destinations simultaneously, use `dse advrep conf`:

```
dse advrep conf update --permits 1000
```

The default is 30,000.

Channel with a higher priority will take precedence in acquiring permits. Permits are required to transmit data from a source to a destination.

## Priority and FIFO/LIFO enablement

The commit log is flushed from memory to disk, writing the data to the appropriate table. A Capture-Data-Change (CDC) collection agent additionally filters the data written and creates replication log files on disk. Each channel source table will have a separate data directory created on disk into which data is appended each time the commit log is flushed, storing all the messages that are to be replicated to a destination. Several replication log files may exist per source table at any given time. Each file stores a contiguous time-slice, configurable with `dse advrep conf update` command and the `--collection-time-slice-width` option (default: 60 seconds). A CDC transmission agent then sends the messages stored in the replication log files to the destination, where the data is processed and written to the appropriate database table. The order in which source table data is transmitted can be altered with the `priority` option when creating a channel, and the order in which a source table's replication log files are read can be tuned with the `--fifo-enabled` and `--lifo-enabled` options.

The replication log files are processed according to the time and priority of the replication channel. Replication channel priorities are set per table, and determines how the transmission agent orders the transmission of replication log files from the source to the destination. The replication log files can be passed to the destination in either last in, first out (LIFO) or first in, first out (FIFO); FIFO is the default. If the newest messages should be read first, use LIFO; if the oldest messages should be read first, use FIFO. Once an individual replication log file is transmitted, the messages it contains are read FIFO. Both options, priority and read order, can be set during channel creation:

```
dse advrep --host 192.168.3.10 channel create --source-keyspace foo --
source-table bar --source-id source1 --source-id-column source_id --
destination mydest --destination-keyspace foo --destination-table bar --
collection-enabled true --priority 1 --lifo-enabled
```

This example sets the channel for table `foo.bar` to the top priority of one, so that the table's replication log files will be transmitted before other table's replication log files. It also sets the replication log files to be read from newest to oldest.

## Configure automatic failover for hub clusters with multiple datacenters

DSE Advanced Replication uses the DSE Java driver load balancing policy to communicate with the hub cluster. You can explicitly define the local datacenter for the datacenter-aware round robin policy ([DCAwareRoundRobinPolicy](#)) that is used by the DSE Java driver.

You can enable or disable failover from a local datacenter to a remote datacenter. When multiple datacenter failover is configured and a local datacenter fails, data replication from the edge to the hub continues using the remote datacenter. Tune the configuration with these parameters:

### **driver-local-dc**

For destination clusters with multiple datacenters, you can explicitly define the name of the datacenter that you consider local. Typically, this is the datacenter that

is closest to the source cluster. This value is used only for clusters with multiple data centers.

#### **driver-used-hosts-per-remote-dc**

To use automatic failover for destination clusters with multiple datacenters, you must define the number of hosts per remote datacenter that the datacenter aware round robin policy ([DCAwareRoundRobinPolicy](#)) considers available.

#### **driver-allow-remote-dcs-for-local-cl**

Set to true to enable automatic failover for destination clusters with multiple datacenters. The value of the **driver-consistency-level** parameter must be LOCAL\_ONE or LOCAL\_QUORUM.

To enable automatic failover with a consistency level of LOCAL\_QUORUM, use `dse advrep destination update`:

```
dse advrep destination update --name mydest --driver-allow-remote-dcs-
for-local-cl true --driver-consistency-level LOCAL_QUORUM
Destination mydest updated
Updated driver_allow_remote_dcs_for_local_cl from null to true
Updated driver_consistency_level from ONE to LOCAL_QUORUM
```

## DSE Advanced Replication terminology

Terminology that is specific to DSE Advanced Replication that supports distributed data replication from a DataStax Enterprise source cluster to a destination cluster.

This terminology is specific to DSE Advanced Replication that supports distributed data replication from a DataStax Enterprise source cluster to a destination cluster.

#### **collection agent**

The process thread that runs on the source cluster that captures the incoming changes and populates the replication log.

#### **destination cluster**

The cluster to which the data flow is going from the source cluster.

#### **source cluster**

A cluster that primarily sends data to one or more destination clusters. DSE Advanced Replication must be enabled on the source cluster.

#### **source datacenter**

A datacenter of a source cluster.

#### **destination cluster**

A cluster that generally supports one or more source clusters that replicate data to the destination cluster. DSE Advanced Replication is not required on the destination cluster.

#### **destination datacenter**

A datacenter of a destination cluster.

#### **isolated**

The state of a cluster when there is not a live connection between the source cluster and the destination cluster.

#### **replication agent**

The process thread that runs on the source cluster that reads data from the replication log and transmits that data to the destination cluster.

#### **replication channel**

A defined channel of change data between source clusters and destination clusters. A replication channel is defined by the source cluster, source keyspace, source table name, destination cluster, destination keyspace, and destination table name.

#### **replication channel priority**

The priority order of which replication channel has precedence when limited bandwidth occurs between the source cluster and the destination cluster.

#### **replication log**

The replication log on the source cluster stores data in preparation for transmission to the destination cluster.

#### **tethered**

The state when there is a live connection between the source cluster and the destination cluster.

## Getting started with DSE Advanced Replication

Getting started steps to set up source and destination DataStax Enterprise clusters to test DSE Advanced Replication.

To test Advanced Replication, you must set up an source cluster and a destination cluster. These steps set up one node in each cluster.

Getting started overview:

1. [Setting up the destination cluster node \(page 749\)](#)
2. [Setting up the source cluster \(page 750\)](#)
3. [Creating sample keyspace and table \(page 750\)](#)
4. [Configuring replication on the source node \(page 751\)](#)
5. [Creating the replication channel \(page 756\)](#)
6. [Starting replication from source to destination \(page 757\)](#)
7. [Inserting data on the source \(page 758\)](#)
8. [Testing loss of connectivity \(page 758\)](#)
9. [Testing replication start and stop \(page 759\)](#)

**Note:** Due to [Cassandra-11368](#), list inserts might not be idempotent (unchanged).

Because DSE Advanced Replication might deliver the same message to the destination more than once, this Cassandra bug might lead to data inconsistency if lists are used in a column family schema. DataStax recommends using other collection types, like sets or frozen lists, when ordering is not important.

### Setting up the destination cluster node

**Attention: Prerequisite:** If you are using Advanced Replication V1 from DSE 5.0, you must upgrade to DSE 5.1 and [migrate to Advanced Replication V2](#).

On the destination node:

1. [Install DataStax Enterprise](#).
2. Start DataStax Enterprise as a transactional node with the command that is appropriate for the [installation method \(page 867\)](#).

3. Note the public IP address for the destination node.

## Setting up the source cluster

Advanced replication can operate in a mixed-version environment. The source cluster requires DataStax Enterprise 5.1 or later. On the source node:

1. [Install DataStax Enterprise](#) 5.1 or later.
2. To enable replication, edit the `dse.yaml` file.

At the end of the file, uncomment the `advanced_replication_options` setting and options, and set `enabled: true`.

```
Advanced Replication configuration settings
advanced_replication_options:
 enabled: true
```

3. [Enable Capture-Data-Change \(CDC\)](#) ([page 68](#)) in the `cassandra.yaml` file on a per-node basis for each source:

```
cdc_enabled: true
```

**Note:** Advanced Replication will not start if CDC is not enabled, since CDC logs are used to implement the feature.

4. Consider increasing the default CDC disk space, depending on the load (default: 4096 or 1/8 of the total space where `cdc_raw_directory` resides):

```
cdc_total_space_in_mb: 16384
```

5. The commitlog compression should be checked for the following value:

```
commitlog_compression:
 - class_name: LZ4Compressor
```

6. Start DataStax Enterprise as a transactional node with the command that is appropriate for the [installation method](#) ([page 867](#)).
7. Once advanced replication is started on a cluster, the source node will create keyspaces and tables that need alteration. See [Keyspaces](#) ([page 760](#)) for information.

## Creating the sample keyspace and table

These steps show you how to create the demonstration keyspace and table.

1. On the source node and the destination node, create the sample keyspace and table:

```
CREATE KEYSPACE foo
WITH REPLICATION = {
 'class': 'SimpleStrategy',
 'replication_factor':1};
```

**Remember:** Remember to use escaped quotes around keyspace and table names as command line arguments to preserve casing: `dse advrep create --keyspace \"keyspaceName\" --table \"tableName\"`

**2. On the source node:**

```
CREATE TABLE foo.bar (
 name TEXT,
 val TEXT,
 scalar INT,
 PRIMARY KEY (name));
```

**3. On the destination node:**

```
CREATE TABLE foo.bar (
 name TEXT,
 val TEXT,
 scalar INT,
 source_id TEXT,
 PRIMARY KEY (name, source_id));
```

**Note:** The `source_id` column is required on the destination node.

## Configuring a replication destination on the source node

DSE Advanced Replication stores all of its settings in CQL tables. To configure replication, use the [dse advrep command line tool](#).

When you configure replication on the source node:

- The source node points to its destination using the public IP address that you saved earlier.
- The `source_id` value is a unique identifier for all data that comes from this particular source node.
- The `source_id` unique identifier is written to the `source-id-column` that was included when the `foo.bar` table was created on the destination node.

To configure a replication destination, run this command:

```
dse advrep --verbose destination create --name mydest --addresses
10.200.182.148 --transmission-enabled true
```

```
Destination mydest created
```

To verify the configuration, run this command:

```
dse advrep destination list-conf
```

destination	name	value
mydest	driver_ssl_enabled	false
mydest	addresses	10.200.182.148



```
| | | |
| TLS_DHE_DSS_WITH_AES_256_CBC_SHA256, | |
| | | ,|
| | |
| | |
| TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA, | |
| | | ,|
| | |
| | |
| TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA, | |
| | | ,|
| | |
| | |
| TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA, | |
| | | ,|
| | |
| | |
| TLS_ECDH_RSA_WITH_AES_256_CBC_SHA, | |
| | | ,|
| | |
| | |
| TLS_DHE_RSA_WITH_AES_256_CBC_SHA, | |
| | | ,|
| | |
| | |
| TLS_DHE_DSS_WITH_AES_256_CBC_SHA, | |
| | | ,|
| | |
| | |
| TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256, | |
| | | ,|
| | |
| | |
| TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256, | |
| | | ,|
| | |
| | |
| TLS_RSA_WITH_AES_128_CBC_SHA256, | |
| | | ,|
| | |
| | |
| TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256, | |
| | | ,|
| | |
| | |
| TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256, | |
| | | ,|
| | |
| | |
| TLS_DHE_RSA_WITH_AES_128_CBC_SHA256, | |
```

```
TLS_DHE_DSS_WITH_AES_128_CBC_SHA256,
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,
TLS_RSA_WITH_AES_128_CBC_SHA,
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA,
TLS_ECDH_RSA_WITH_AES_128_CBC_SHA,
TLS_DHE_RSA_WITH_AES_128_CBC_SHA,
TLS_DHE_DSS_WITH_AES_128_CBC_SHA,
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,
TLS_RSA_WITH_AES_256_GCM_SHA384,
TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384,
```

TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384,		
		,
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384,		
		,
TLS_DHE_DSS_WITH_AES_256_GCM_SHA384,		
		,
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,		
		,
TLS_RSA_WITH_AES_128_GCM_SHA256,		
		,
TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256,		
		,
TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256,		
		,
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256,		
		,
TLS_DHE_DSS_WITH_AES_128_GCM_SHA256,		
		,
TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA,		
		,
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA,		
		,
SSL_RSA_WITH_3DES_EDE_CBC_SHA,		
		,
TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA,		
		,
TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA,		

SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA,		
SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA,		
TLS_ECDHE_ECDSA_WITH_RC4_128_SHA,		
TLS_ECDHE_RSA_WITH_RC4_128_SHA,		
SSL_RSA_WITH_RC4_128_SHA,		
TLS_ECDH_ECDSA_WITH_RC4_128_SHA,		
TLS_ECDH_RSA_WITH_RC4_128_SHA,		
SSL_RSA_WITH_RC4_128_MD5,		
TLS_EMPTY_RENEGOTIATION_INFO_SCSV]		
-----		
mydest	source_id	source1
-----		
mydest	transmission_enabled	true

## **Creating the replication channel**

A replication channel is a defined channel of change data between source clusters and destination clusters. A replication channel is defined by the source cluster, source keyspace, source table name, destination cluster, destination keyspace, and destination table name. Source clusters can exist in multi-datacenter clusters, but a replication channel is configured with only one datacenter as the responsible party.

The keyspace and table names on the destination can be different than on the source, but in this example they are the same. You can also set the `source-id` and `source-id-column` differently from the global setting.

To create the replication channel for our keyspace and table:

```
dse advrep channel create --source-keyspace foo --source-table bar --
source-id source1 --source-id-column source_id --destination mydest --
destination-keyspace foo --destination-table bar --collection-enabled
true --transmission-enabled true --priority 1
```

```
Created channel dc=Cassandra keyspace=foo table=bar to mydest
```

```
dse advrep channel status
```

dc	keyspace	table	collecting	transmitting	replication		
order	priority	dest ks	dest table	src id	src id col	dest	dest enabled
Cassandra	foo	bar	true	true	FIFO		
1	foo	bar	source1	source_id	mydest	true	

**Warning:** The designated keyspace for a replication channel must have durable writes enabled. If `durable_writes = false`, then an error message will occur and the channel will not be created. If the durable writes setting is changed after the replication channel is created, the tables will not write to the commit log and CDC will not work. The data will not be ingested through the replication channel and a warning is logged, but the failure will be silent.

## Starting replication from source to destination

At this point, the replication is configured and the replication channel is enabled and replication has been started.

1. On the destination, use cqlsh to verify that no data is present:

```
SELECT * FROM foo.bar;
```

name	source_id	scalar	val
( 0 rows )			

2. On the source, replication to the destination can be paused or resumed, the latter shown here:

```
dse advrep channel resume --source-keyspace foo --source-table bar --
transmission
```

```
Channel dc=Cassandra keyspace=foo table=bar collection to mydest was resumed
```

Notice that either `--transmission` or `--collection` can be specified, to resume transmission from the source to the destination or to resume collection of data on the source.

3. Review the number of records that are in the replication log. Because no data is inserted yet, the record count in the replication log is 0:

```
dse advrep replog count --destination mydest --source-keyspace foo --source-table bar
```

```
0
```

## Inserting data on the source

Insert data on the source for replication to the destination.

1. On the source, insert data using cqlsh:

```
INSERT INTO foo.bar (name, val, scalar) VALUES ('a', '1', 1);
INSERT INTO foo.bar (name, val, scalar) VALUES ('b', '2', 2);
```

2. On the destination, verify that the data was replicated:

```
SELECT * FROM foo.bar;
```

name	source_id	scalar	val
a	source1	1	1
b	source1	2	2
(2 rows)			

## Checking data on the destination

Check data on the destination.

1. On the destination, verify that the data was replicated:

```
SELECT * FROM foo.bar;
```

name	source_id	scalar	val
a	source1	1	1
b	source1	2	2
(2 rows)			

## Testing loss of connectivity

To test loss of connectivity to the destination, stop the DataStax Enterprise process on the destination, and insert more data on the source. The expected result is for data to be replicated quickly after the destination cluster resumes.

1. On the destination cluster, stop DataStax Enterprise:

```
dse cassandra-stop
```

2. On the source, insert more data:

```
INSERT INTO foo.bar (name, val, scalar) VALUES ('c', '3', 3);
INSERT INTO foo.bar (name, val, scalar) VALUES ('d', '4', 4);
```

3. Review the number of records that are in the replication log. The replication log should have 2 entries:

```
dse advrep replog count --destination mydest --source-keyspace foo --
source-table bar
```

```
2
```

4. On the destination, restart DataStax Enterprise.

```
dse cassandra
```

Wait a moment for communication and data replication to resume to replicate the new records from the source to destination.

```
SELECT * FROM foo.bar;
```

name	source_id	scalar	val
a	source1	1	1
c	source1	3	3
d	source1	4	4
b	source1	2	2

5. On the source, the replication log count should be back to 0:

```
dse advrep replog count --destination mydest --source-keyspace foo --
source-table bar
```

```
0
```

## Testing replication start and stop

Similar to testing loss of connectivity, you can pause and resume individual replication channels by using the [advrep command line tool](#). The expected result is that newly inserted data is not saved to the replication log and will never be sent to the destination.

1. On the source, pause the replication channel:

```
dse advrep --verbose channel pause --keyspace foo --table bar --
collection
```

2. Insert more data.
3. On the source, resume the replication channel:

```
dse advrep --verbose channel resume --keyspace foo --table bar --
collection
```

## DSE Advanced Replication keyspace overview

Information about DSE Advanced Replication keyspaces and tables.

Keyspaces and tables are automatically created on the source cluster when DSE Advanced Replication runs for the first time. Two keyspaces are used, *dse\_system* and *dse\_advrep*. Each keyspace is configured differently.

**Note:** System keyspaces on the source and destination are not supported for advanced replication.

The *dse\_system* keyspace uses the EverywhereStrategy replication strategy by default; this setting must not be altered. The *dse\_advrep* keyspace is configured to use the SimpleStrategy replication strategy by default and this setting must be updated in production environments to avoid data loss. After starting the cluster, alter the keyspace to use the NetworkTopologyStrategy replication strategy with an appropriate settings for the replication factor and datacenters. For example, use a CQL statement to configure a replication factor of 3 on the DC1 datacenter using NetworkTopologyStrategy:

```
ALTER KEYSPACE dse_advrep
WITH REPLICATION = {
 'class': 'NetworkTopologyStrategy',
 'DC1': '3' };
```

For most environments using DSE Advanced Replication, a replication factor of 3 is suitable. The strategy must be configured for any datacenters which are serving as an advanced replication source.

`nodetool repair` must be run on each node of the affected datacenters. to repair the altered keyspace:

```
nodetool repair -full dse_advrep
```

For more information, see [Changing keyspace replication strategy](#).

## DSE Advanced Replication data types

Information about DSE Advanced Replication data types.

DSE data types are supported for most operations in DSE Advanced Replication. The following table shows the supported data types and operations:

Data Type	Advanced Replication Operations
Primitive data types: int, ascii, bigint, blob, boolean, decimal, double, float, inet, text, timestamp, timeuuid, uuid, varchar, varint	All types are implemented for insert/update/delete.

Data Type	Advanced Replication Operations
Frozen collections: frozen<list< <i>data_type</i> >>, frozen<set< <i>ddata_type</i> >>, frozen<map< <i>data_type</i> , <i>data_type</i> >>	All frozen collections are implemented for insert/update/delete, as values are immutable blocks - entire column value is replicated.
Tuples: tuple< <i>data_type</i> , <i>data_type</i> , <i>data_type</i> >, frozen<tuple< <i>data_type</i> , <i>data_type</i> , <i>data_type</i> >>	All tuples are implemented for insert/update/delete, as values are immutable blocks - entire column value is replicated.
Frozen user-defined type (UDT): UDT type and frozen UDT type	All UDTs are implemented for insert/update/delete, as values are immutable blocks - entire column value is replicated.
Geometric types: Point, LineString, Polygon	All geometric types are implemented for insert/update/delete.

The following table shows the data type and operations that are not supported in DSE Advanced Replication:

Data Type	Advanced Replication Operations
Unfrozen updatable collections: <list< <i>data_type</i> >>, <set< <i>ddata_type</i> >>, <map< <i>data_type</i> , <i>data_type</i> >>	All unfrozen updatable collections are implemented for insert/delete if the entire column value is replicated. Unfrozen collections cannot update values.
Unfrozen updatable user-defined type (UDT)	All unfrozen updatable UDTs are implemented for insert/delete if the entire column value is replicated. Unfrozen UDTs cannot update values.

## Using DSE Advanced Replication

Operations including starting, stopping, and configuring DSE Advanced Replication.

Operations including starting, stopping, and configuring DSE Advanced Replication.

1. Starting DSE Advanced Replication ([page 761](#))
2. Stopping DSE Advanced Replication ([page 762](#))
3. Configuring global configuration settings ([page 763](#))
4. Configuring destination settings ([page 764](#))
5. Configuring channel settings ([page 777](#))
6. Security ([page 778](#))
7. Data insert methods ([page 780](#))
8. Monitoring operations ([page 780](#))

### Starting DSE Advanced Replication

**Attention: Prerequisite:** If you are using Advanced Replication V1 from DSE 5.0, you must upgrade to DSE 5.1 and [migrate to Advanced Replication V2](#).

Before you can start and use DSE Advanced Replication, you must create the user [keyspaces \(page 760\)](#) and tables on the source cluster and the destination cluster.

On all nodes in the source cluster:

1. Enable replication in the `dse.yaml` file.

Uncomment all `advanced_replication_options` entries, set `enabled: true`, and specify a directory to hold advanced replication log files with `advanced_replication_directory`:

```
Advanced Replication configuration settings
advanced_replication_options:
 enabled: true
 advanced_replication_directory: /var/lib/cassandra/advrep
```

2. Enable [Capture-Data-Change \(CDC\) \(page 68\)](#) in the `cassandra.yaml` file on a per-node basis for each source:

```
cdc_enabled: true
cdc_raw_directory: /var/lib/cassandra/cdc_raw
```

**Note:** Advanced Replication will not start if CDC is not enabled. Either use the default directory or change it to a preferred location.

3. Consider increasing the default CDC disk space, depending on the load (default: 4096 MB or 1/8 of the total space where `cdc_raw_directory` resides):

```
cdc_total_space_in_mb: 16384
```

4. Commitlog compression is turned off by default. To avoid problems with advanced replication, this option should NOT be used:

```
commitlog_compression:
- class_name: LZ4Compressor
```

5. Do a rolling restart: restart the nodes in the source cluster one at a time while the other nodes continue to operate online.

## Disabling DSE Advanced Replication

When replication is not enabled, data is not written to the replication log. On all nodes in the source cluster:

1. To disable replication, edit the `dse.yaml` file.

In the `advanced_replication_options` section, set `enabled: false`.

```
Advanced Replication configuration settings
advanced_replication_options:
 enabled: false
```

2. Do a rolling restart: restart the nodes in the source cluster one at a time while the other nodes continue to operate online.

3. To clean out the data that was used for DSE Advanced Replication, use `cqlsh` to remove these [keyspaces \(page 760\)](#):

```
DROP TABLE dse_system.advrep_source_config;
DROP TABLE dse_system.advrep_destination_config;
DROP TABLE dse_system.advrep_repl_channel_config;
DROP KEYSPACE dse_advrep;
```

## Configuring global configuration settings

Global settings apply to the entire source cluster. These global settings are stored in the CQL table `dse_system.advrep_source_config` that is automatically created.

Change global settings by using the [dse advrep command line tool](#) with this syntax:

```
dse advrep conf ...
```

To view the source node configuration settings:

```
dse advrep conf list
```

The result is:

```

| name | value |

| audit_log_file | /tmp/myaudit.gz |

| audit_log_enabled | true |

```

The following table describes the configuration keys, their default values, and identifies when a restart of the source node is required for the change to be recognized.

The `dse advrep` command line tool uses these configuration keys as command arguments to the [dse advrep](#) command line tool.

Configuration key	Default value	Description	Restart required
permits	30,000	Maximum number of messages that can be replicated in parallel over all destinations.	No
source-id	N/A	Identifies this source cluster and all inserts from this cluster. The source-id must also exist in the primary key on the destination for population of the source-id to occur.	No
collection-expire-after-write	N/A		
collection-time-slice-count	5	The number of files which are open in the ingestor simultaneously.	Yes

Configuration key	Default value	Description	Restart required
collection-time-slice-width	60 seconds	The time period in seconds for each data block ingested. Smaller time widths => more files. Larger timer widths => larger files but more data to resend on CRC mismatches.	Yes
invalid-message-log	SYSTEM_LOG	Select one of these logging strategies to adopt when an invalid message is discarded:  SYSTEM_LOG: Log the CQL query and the error message in the system log on the destination.  CHANNEL_LOG: Store the CQL query and the error message in files in <code>/var/lib/cassandra/advrep/invalid_queries</code> on the destination.  NONE: Perform no logging.  See <a href="#">Managing invalid messages (page 790)</a> . Requires node restart.	No
audit-log-enable	false	Specifies whether to store the audit log.	Yes
audit-log-file	/tmp/advrep_rl_audit.log	Specifies the file name prefix template for the audit log file. The file name is appended with <code>.gz</code> if compressed using gzip.	Yes
audit-log-max-life-span-mins	0	Specifies the maximum lifetime of audit log files. Periodically, when log files are rotated, audit log files are purged when they: <ul style="list-style-type: none"><li>• Match the audit log file template</li><li>• And they have not been written to for more than the specified maximum lifespan minutes</li></ul> To disable purging, set to 0.	Yes
audit-log-rotate-time-mins	60	Specifies the time interval to rotate the audit log file. On rotation, the rotated file is appended with the log counter <code>[logcounter]</code> , incrementing from [0]. To disable rotation, set to 0.	Yes

## Configuring destination settings

A destination is a location to which source data will be written. Destinations are stored in the CQL table `dse_system.advrep_destination_config` that is automatically created.

Change destination settings by using the [dse advrep command line tool](#) with this syntax:

```
dse advrep destination ...
```

You can verify the channel configuration before you change it. For example:

```
dse advrep destination list-conf
```

The result is:

destination	name	value
mydest	driver_ssl_enabled	false
mydest	addresses	10.200.182.251
mydest	driver_read_timeout	15000
mydest	driver_connections_max	8
mydest	source_id_column	source_id
mydest	driver_connect_timeout	15000
mydest	driver_ssl_protocol	TLS
mydest	driver_consistency_level	QUORUM
mydest	driver_used_hosts_per_remote_dc	0
mydest	driver_allow_remote_dcs_for_local_cl	false
mydest	driver_compression	lz4
mydest	driver_connections	1
mydest	driver_ssl_cipher_suites	
	[TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384,	,
		,
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384,	,
		,
	TLS_RSA_WITH_AES_256_CBC_SHA256,	,
		,

```
| |
| TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384, | |
| | |
| | |
| | |
| | |
| TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384, | |
| | |
| | |
| | |
| | |
| TLS_DHE_RSA_WITH_AES_256_CBC_SHA256, | |
| | |
| | |
| | |
| | |
| TLS_DHE_DSS_WITH_AES_256_CBC_SHA256, | |
| | |
| | |
| | |
| | |
| TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA, | |
| | |
| | |
| | |
| | |
| TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA, | |
| | |
| | |
| | |
| | |
| TLS_RSA_WITH_AES_256_CBC_SHA, | |
| | |
| | |
| | |
| | |
| TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA, | |
| | |
| | |
| | |
| | |
| TLS_ECDH_RSA_WITH_AES_256_CBC_SHA, | |
| | |
| | |
| | |
| | |
| TLS_DHE_RSA_WITH_AES_256_CBC_SHA, | |
| | |
| | |
| | |
| | |
| TLS_DHE_DSS_WITH_AES_256_CBC_SHA, | |
| | |
| | |
| | |
| | |
| TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256, | |
| | |
| | |
| | |
| | |
| TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256, | |
| | |
| | |
| | |
| | |
| TLS_RSA_WITH_AES_128_CBC_SHA256, | |
```

TLS\_ECDH\_ECDSA\_WITH\_AES\_128\_CBC\_SHA256 , |  
| | |  
| | |  
TLS\_ECDH\_RSA\_WITH\_AES\_128\_CBC\_SHA256 , |  
| | |  
| | |  
TLS\_DHE\_RSA\_WITH\_AES\_128\_CBC\_SHA256 , |  
| | |  
| | |  
TLS\_DHE\_DSS\_WITH\_AES\_128\_CBC\_SHA256 , |  
| | |  
| | |  
TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_SHA , |  
| | |  
| | |  
TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA , |  
| | |  
| | |  
TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA , |  
| | |  
| | |  
TLS\_ECDH\_ECDSA\_WITH\_AES\_128\_CBC\_SHA , |  
| | |  
| | |  
TLS\_ECDH\_RSA\_WITH\_AES\_128\_CBC\_SHA , |  
| | |  
| | |  
TLS\_DHE\_RSA\_WITH\_AES\_128\_CBC\_SHA , |  
| | |  
| | |  
TLS\_DHE\_DSS\_WITH\_AES\_128\_CBC\_SHA , |  
| | |  
| | |  
TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_GCM\_SHA384 , |  
| | |  
| | |  
TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_GCM\_SHA256 , |  
| | |

TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,		
		,
TLS_RSA_WITH_AES_256_GCM_SHA384,		,
TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384,		,
TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384,		,
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384,		,
TLS_DHE_DSS_WITH_AES_256_GCM_SHA384,		,
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,		,
TLS_RSA_WITH_AES_128_GCM_SHA256,		,
TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256,		,
TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256,		,
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256,		,
TLS_DHE_DSS_WITH_AES_128_GCM_SHA256,		,
TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA,		,
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA,		

			,
			,
SSL_RSA_WITH_3DES_EDE_CBC_SHA,			,
			,
TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA,			,
			,
TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA,			,
			,
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA,			,
			,
SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA,			,
			,
TLS_ECDHE_ECDSA_WITH_RC4_128_SHA,			,
			,
TLS_ECDHE_RSA_WITH_RC4_128_SHA,			,
			,
SSL_RSA_WITH_RC4_128_SHA,			,
			,
TLS_ECDH_ECDSA_WITH_RC4_128_SHA,			,
			,
TLS_ECDH_RSA_WITH_RC4_128_SHA,			,
			,
SSL_RSA_WITH_RC4_128_MD5,			,
			,
TLS_EMPTY_RENEGOTIATION_INFO_SCSV]			
<hr/>			
mydest	source_id		source1
<hr/>			
mydest	transmission_enabled		true

```
| llpdest | driver_ssl_enabled | false
| |
|-----|
| llpdest | addresses | 10.200.177.184
| |
|-----|
| llpdest | driver_read_timeout | 15000
| |
|-----|
| llpdest | driver_connections_max | 8
| |
|-----|
| llpdest | source_id_column | source_id
| |
|-----|
| llpdest | driver_connect_timeout | 15000
| |
|-----|
| llpdest | driver_ssl_protocol | TLS
| |
|-----|
| llpdest | driver_consistency_level | ONE
| |
|-----|
| llpdest | driver_used_hosts_per_remote_dc | 0
| |
|-----|
| llpdest | driver_allow_remote_dcs_for_local_cl | false
| |
|-----|
| llpdest | driver_compression | lz4
| |
|-----|
| llpdest | driver_connections | 1
| |
|-----|
| llpdest | driver_ssl_cipher_suites |
[TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384, |
| | |
| | |
| | |
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384, |
| | |
| | |
| | |
TLS_RSA_WITH_AES_256_CBC_SHA256, |
| | |
| | |
| | |
TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384, |
| | |
| | |
| | |
```

TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384,		
		,
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256,		
		,
TLS_DHE_DSS_WITH_AES_256_CBC_SHA256,		
		,
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA,		
		,
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,		
		,
TLS_RSA_WITH_AES_256_CBC_SHA,		
		,
TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA,		
		,
TLS_ECDH_RSA_WITH_AES_256_CBC_SHA,		
		,
TLS_DHE_RSA_WITH_AES_256_CBC_SHA,		
		,
TLS_DHE_DSS_WITH_AES_256_CBC_SHA,		
		,
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,		
		,
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,		
		,
TLS_RSA_WITH_AES_128_CBC_SHA256,		
		,
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256,		



```
| |
| TLS_RSA_WITH_AES_256_GCM_SHA384, | |
| | ,|
| |
| |
| TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384, | |
| | ,|
| |
| |
| TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384, | |
| | ,|
| |
| |
| TLS_DHE_RSA_WITH_AES_256_GCM_SHA384, | |
| | ,|
| |
| |
| TLS_DHE_DSS_WITH_AES_256_GCM_SHA384, | |
| | ,|
| |
| |
| TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256, | |
| | ,|
| |
| |
| TLS_RSA_WITH_AES_128_GCM_SHA256, | |
| | ,|
| |
| |
| TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256, | |
| | ,|
| |
| |
| TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256, | |
| | ,|
| |
| |
| TLS_DHE_RSA_WITH_AES_128_GCM_SHA256, | |
| | ,|
| |
| |
| TLS_DHE_DSS_WITH_AES_128_GCM_SHA256, | |
| | ,|
| |
| |
| TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA, | |
| | ,|
| |
| |
| TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA, | |
| | ,|
| |
| |
| SSL_RSA_WITH_3DES_EDE_CBC_SHA, | |
```

			,
TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA,			,
TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA,			,
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA,			,
SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA,			,
TLS_ECDHE_ECDSA_WITH_RC4_128_SHA,			,
TLS_ECDHE_RSA_WITH_RC4_128_SHA,			,
SSL_RSA_WITH_RC4_128_SHA,			,
TLS_ECDH_ECDSA_WITH_RC4_128_SHA,			,
TLS_ECDH_RSA_WITH_RC4_128_SHA,			,
SSL_RSA_WITH_RC4_128_MD5,			,
TLS_EMPTY_RENEGOTIATION_INFO_SCSV]			
<hr/>			
llpdest	source_id		source1
<hr/>			
llpdest	transmission_enabled		false
<hr/>			

The following table describes the configuration keys, their default values, and identifies when a restart of the source node is required for the change to be recognized.

Configuration key	Default value	Description	Restart required
separator	N/A	Field separator.	No
name	N/A	Name for destination (required).	No
addresses	none	REQUIRED. A comma separated list of IP addresses that are used to connect to the destination cluster using the DataStax Java driver.	No
driver-allow-remote-dcs-for-local-cl	false	Set to true to enable automatic failover for destination clusters with multiple datacenters. The value of the <b>driver-consistency-level</b> parameter must be LOCAL_ONE or LOCAL_QUORUM.	Yes
driver-compression	lz4	The compression algorithm the DataStax Java driver uses to send data from the source to the destination. Supported values are lz4 and snappy.	Yes
driver-connect-timeout	15000	Time in milliseconds the DataStax Java driver waits to connect to a server.	No
driver-connections	32	The number of connections the DataStax Java driver will create.	Yes
driver-connections-max	256	The maximum number of connections the DataStax Java driver will create.	Yes
driver-max-requests-per-connection	1024	The maximum number of requests per connection the DataStax Java driver will create.	
driver-consistency-level	ONE	The consistency level used by the DataStax Java driver when executing statements for replicating data to the destination. Specify a valid DSE <b>CONSISTENCY</b> level: ANY, ONE, TWO, THREE, QUORUM, ALL, LOCAL_QUORUM, EACH_QUORUM, SERIAL, LOCAL_SERIAL, or LOCAL_ONE.	No
driver-local-dc	N/A	For destination clusters with multiple datacenters, you can explicitly define the name of the datacenter that you consider local. Typically, this is the datacenter that is closest to the source cluster. This value is used only for clusters with multiple data centers.	Yes

Configuration key	Default value	Description	Restart required
driver-pwd	none	Driver password if the destination requires a user and password to connect. Changing the driver-pwd value for connection to a destination will automatically connect, but with a slight delay.  <b>Note:</b> By default, driver user names and passwords are plain text. DataStax recommends encrypting the driver passwords before you add them to the CQL table.	Yes
driver-read-timeout	15000	Time in milliseconds the DataStax Java driver waits to read responses from a server.	No
driver-ssl-enabled	false	Whether SSL is enabled for connection to the destination.	Yes
driver-ssl-disabled		Disable SSL for connection to the destination.	
driver_ssl_keystore_path	none	The path to the keystore for connection to DSE when SSL client authentication is enabled.	Yes
driver_ssl_keystore_password	none	The keystore password for connection to DSE when SSL client authentication is enabled.	Yes
driver_ssl_keystore_type	none	The keystore type for connection to DSE when SSL client authentication is enabled.	Yes
driver_ssl_truststore_path	none	The path to the truststore for connection to DSE when SSL is enabled.	Yes
driver-ssl-truststore-password	none	The truststore password for connection to DSE when SSL is enabled.	Yes
driver-ssl-truststore-type	none	The keystore type for connection to DSE when SSL client authentication is enabled.	Yes
driver-ssl-protocol	TLS	The SSL protocol for connection to DSE when SSL is enabled.	Yes
driver-ssl-cipher-suites	none	A comma-separated list of SSL cipher suites for connection to DSE when SSL is enabled. Cipher suites must be supported by the source machine.	Yes
driver-used-hosts-per-remote-dc	0	To use automatic failover for destination clusters with multiple datacenters, you must define the number of hosts per remote datacenter that the datacenter aware round robin policy ( <a href="#">DCAwareRoundRobinPolicy</a> ) considers available.	Yes

Configuration key	Default value	Description	Restart required
driver-user	none	Driver username if the destination requires a user and password to connect. Changing the driver-user value for connection to a destination will automatically connect, but with a slight delay.	Yes
source-id	N/A	Identifies this source cluster and all inserts from this cluster. The source-id must also exist in the primary key on the destination for population of the source-id to occur.	No
source-id-column	source-id	The column to use on remote tables to insert the source id as part of the update. If this column is not present on the table that is being updated, the source id value is ignored.	No
transmission-enabled	false	Specify if data collector for the table should be replicated to the destination using boolean value.	No

## Configuring channel settings

A replication channel is a defined channel of change data between source clusters and destination clusters.

A replication channel is defined by the source cluster, source keyspace, source table name, destination cluster, destination keyspace, and destination table name. Replications for each channel (unique keyspace and table) are stored in the CQL table `dse_system.advrep_repl_channel_config` that is automatically created.

Change the settings using the `dse advrep` command line tool with this syntax:

```
dse advrep channel ...
```

You can verify the channel configuration before you change it. For example:

```
dse advrep channel status
```

The result is:

```
-----|dc |keyspace|table |collecting|transmitting|replication
 |order|priority|dest ks|dest table |src id |src id col|dest |dest
 |enabled| |true |true |FIFO
-----|Cassandra|foo |bar |true |source1|source_id |mydest|true
 | 2 |foo |bar
 |

```

Properties are continuously read from the metadata, so a restart is not required after configuration changes are made. The following table describes the configuration settings.

Column name	Description
separator	Field separator.
keyspace	The keyspace on the source for the table to replicate.
table	The table name on the source to replicate.
source-id	Placeholder to override the source-id that is defined in the advrep_conf metadata
source-id-column	Placeholder to override the source-id-column that is defined in advrep_conf metadata.
enabled	If true, replication will start for this table. If false, no more messages from this table will be saved to the replication log.
data-center-id	Datacenter this replication channel is meant for, if none specified the replication will happen in all specified dc1.
destination	Destination to which data is written.
destination-keyspace	The keyspace on the destination for the replicated table.
destination-table	The table name on the destination for the replicated table.
priority	Messages are marked by priority in descending order (DESC).
transmission-enabled	Specify if the data collector for the table should be replicated to the destination.
fifo-order	Specify if the channel should be replicated in FIFO order (default).
lifo-order	Specify if the channel should be replicated in LIFO order.

## Security

Authentication credentials can be provided in several ways, see [Providing credentials from DSE tools](#).

The user who is doing the replicating with DSE Advanced Replication requires table and keyspace level authorization. If the same user access is required, then ensure that the authorization is the same on the source and destination clusters.

Advanced Replication also supports setting [row-level permissions](#) on the destination cluster. The user which connects to the destination cluster must have permission to write to the specified destination table at the row level replicated from the source, according to the RLAC restrictions. The user is specified with the `--driver-user destination (page 764)` setting. Row-level access control (RLAC) on the source cluster does not impact Advanced Replication. Because Advanced Replication reads the source data at the raw CDC file layer, it essentially reads as a superuser and has access to all configured data tables.

Advanced Replication supports encrypting the driver passwords. Driver passwords are stored in a CQL table. By default, driver passwords are plain text. DataStax recommends encrypting the driver passwords before you add them to the CQL table. Create a global

encryption key, called a system\_key for [SSTable encryption](#). Each node in the source cluster must have the same system key. The destination does not require this key.

1. In the dse.yaml file:

- Verify that the **config\_encryption\_active** property is false:

```
config_encryption_active: false
```

- Enable driver password encryption with the **conf\_driver\_password\_encryption\_enabled** property:

```
conf_driver_password_encryption_enabled: true
```

- Define where system keys are stored on disk. The location of the key is specified on the command line with the [-d \(page 815\)](#) option or with [system\\_key\\_directory \(page 115\)](#) in dse.yaml. The default filepath is /etc/dse/conf.
- To configure the filename of the generated encryption key, set the [config\\_encryption\\_key\\_name \(page 116\)](#) option in dse.yaml. The default name is system\_key.

2. Generate a system key:

On-server:

```
dsetool createsystemkey cipher_algorithm strength system_key_file
```

Off-server

```
dsetool createsystemkey cipher_algorithm strength system_key_file -kmp=kmip_groupname
```

For example:

```
dsetool createsystemkey 'AES/ECB/PKCS5Padding' 128 system_key_file
```

where *system\_key\_file* is a unique file name for the generated system key file. See [createsystemkey \(page 813\)](#).

**Result:** Configure transparent data encryption (TDE) on a per table basis. You can configure encryption with or without compression. You can create a global encryption key in the location that is specified by [system\\_key\\_directory \(page 115\)](#) in the `dse.yaml` file. This default global encryption key is used when the `system_key_file` subproperty is not specified.

3. Copy the returned value.

4. On any node in the source cluster, use the `dse` command to set the encrypted password in the DSE Advanced Replication environment:

```
dse advrep destination --driver-pwd "Sa9x0Vaym7bddjXUT/eeOQ==" --driver-user "username"
```

5. Start `dse` ([page 867](#)).

## Data insert methods

There are several ways to get data into a DataStax Enterprise cluster. Any normal paths used will result in data replication using DSE Advanced Replication.

Supported data insert methods:

- CQL insert, including cqlsh and applications that use the standard DSE drivers
- [COPY](#) from a CSV file
- Solr HTTP or CQL
- Spark saveToCassandra

Unsupported data insert methods:

- sstableloader ([Cassandra bulk loader](#))
- OpsCenter restore from backup
- Spark bulkSaveToCassandra

## Monitoring operations

Advanced replication can be monitored with JMX metrics. The outgoing replication queue size is a key factor to watch. See [Metrics \(page 782\)](#) for more details.

# CQL queries in DSE Advanced Replication

Supported CQL queries and best practices guidelines for running queries on source and destination clusters in DSE Advanced Replication.

This overview of supported CQL queries and replication concepts for DSE Advanced Replication provide details on supported CQL queries and best practices guidelines.

DSE Advanced Replication replicates data from source clusters to destination clusters. Replication takes the CQL query on the source and then recreates a modified version of the query and runs it on the destination. DataStax Enterprise supports a restricted list of valid CQL queries to manipulate data. In DSE Advanced Replication, the same restrictions apply to the generated CQL queries that are used to replicate data into the destination.

Restrictions apply to the primary key. The primary key consists of two parts: the partition key and the clustering key. The primary key parts plus the optional field values comprise the database row.

If differences exist between the primary key on the source table and the primary key on the destination table, restrictions apply for which CQL queries are supported.

## Best practices

DataStax recommends the following best practices to ensure seamless replication.

### Schema structure on the source table and the destination table

- Maintain an identical primary key (partition keys and clustering keys) format in the same order, with the same columns.
- Add the optional `source_id` as the first clustering column.
- Maintain all, or a subset of, the field values.

**Note:** Although the `source_id` column can be present in the source table schema, values that are inserted into that column are ignored. When records are replicated, the configured `source-id` value is used.

## Partition key columns

The following list details support and restrictions for partition keys:

- In the destination table, only an additional optional `source_id` column is supported in the partition key. Additional destination table partition key columns are not supported. The `source_id` can be either a clustering column or a partition key, but not both.
- Using a subset of source table partition key columns in the destination table might result in overwriting. There is a many-to-one mapping for row entries.
- Order is irrelevant for replication. All permutations are supported.
- CQL `UPDATE` queries require that all of the partition key columns are fully restricted. Restrict partition key columns using `=` or `IN` (single column) restrictions.
- CQL `DELETE` queries require that all of the partition key columns are fully restricted. Restrict partition key columns using `=` or `IN` (single column) restrictions.

## Clustering columns

The following list details support and restrictions for clustering columns:

- In the destination table, only an additional optional `source_id` column is supported in the clustering column. Additional destination table partition key columns are not supported. The `source_id` can be either a clustering column or a partition key, but not both.
- Using a subset of source table clustering columns in the destination table might result in overwriting. There is a many-to-one mapping for row entries.
- Order is irrelevant for replication when using CQL `INSERT` and `UPDATE` queries. All permutations are supported.
- Order **is** relevant for replication when using CQL `DELETE` queries. There are limits to permutation support, all permutations are **not** supported.
- CQL `UPDATE` queries require that all of the clustering columns are fully restricted. Restrict partition key columns using `=` or `IN` (single column) restrictions.
- CQL `DELETE` queries require that the last-specified clustering column be restricted using `=/>/>=/</<=` (single or multiple column) or `IN` (single or multiple column). **All** of the clustering columns that precede the last-specified clustering column must also be restricted using `=` or `IN`.
- Restricting clustering columns is optional. However, if you do restrict clustering columns, then all of the clustering columns that you restrict between the first and last (in order) clustering columns must be restricted.

## Field values

The following list details support and requirements for field values:

- A subset, or all, of the field values on the source are supported for replication to the destination.
- Fields that are present on the source, but absent on the destination, are not replicated.
- Fields that are present on the destination, but absent on the source, are not populated.

## Source ID (`source_id`)

The `source_id` identifies the source cluster and all inserts from the source cluster. The following list details support and requirements for the `source_id`:

- The [source\\_id configuration key \(page 763\)](#) must be present and correct in the metadata.
- The `source_id` must be the first position in the clustering column, or any of the partition keys.  
If not, then the CQL `INSERT` and `UPDATE` queries should work, but the CQL `DELETE` queries with partially restricted clustering columns might fail.
- The `source_id` is always restricted in CQL `DELETE` and `UPDATE` queries. Certain delete statements are not supported where the clustering key is **not** fully restricted, and the `source_id` is **not** the first clustering column

## DSE Advanced Replication metrics

DSE Advanced Replication metrics on the source nodes refer to the current status of that node in the source cluster.

Collect metrics on each source node to review the current status of that node in the source cluster. A working source and destination configuration is required to use the metrics feature. See [Getting started \(page 749\)](#).

### Ensure JMX access

Metrics are stored in the DataStax Enterprise JMX system. JMX access is required.

- For production, DataStax recommends authenticating JMX users, see [Configuring JMX authentication](#).
- Use these steps to enable local JMX access. Localhost access is useful for test and development.

1. On the source node, edit `cassandra-env.sh` and enable local JMX:

```
JVM_OPTS="$JVM_OPTS -Djava.rmi.server.hostname=localhost"
LOCAL_JMX=yes
```

2. On the source node, [stop and restart \(page 867\)](#) DataStax Enterprise to recognize the local JMX change.

### Display metrics on the command line

Use the `dse advrep` command line tool to display metrics on the command line. Ensure that the source node meets the command line prerequisites.

1. On the source node:

```
dse advrep --jmx-port 7199 metrics list
```

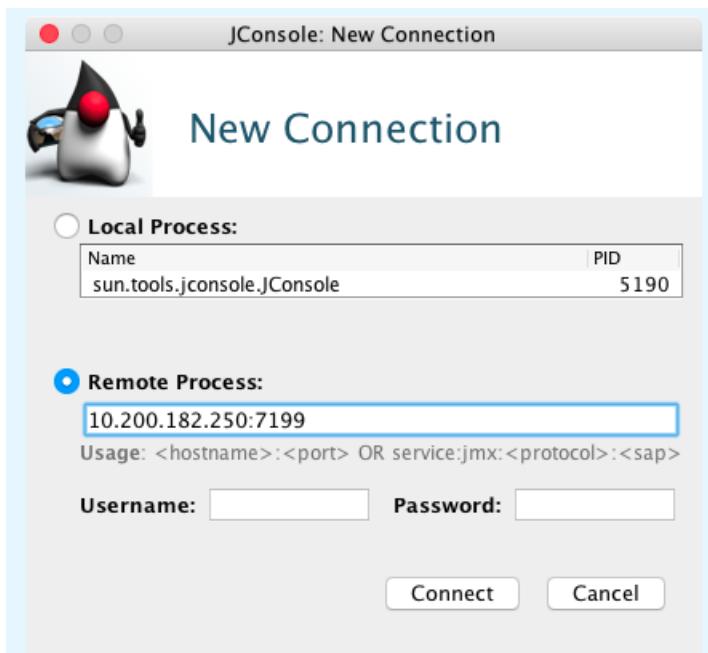
Group	Type	Count
Tables	MessagesDelivered	1002
ReplicationLog	CommitLogsToConsume	1
Tables	MessagesReceived	1002
ReplicationLog	MessageAddErrors	0
ReplicationLog	CommitLogsDeleted	0

Group	Type	Count	RateUnit	MeanRate
		FifteenMinuteRate	OneMinuteRate	FiveMinuteRate
ReplicationLog	MessagesAdded	1002	events/second	
0.012688461014514603	9.862886141388435E-39   2.964393875E-314			
2.322135514219019E-114				
ReplicationLog	MessagesDeleted	0	events/second	0.0
0.0	0.0			0.0
ReplicationLog	MessagesAcknowledged	1002	events/second	
0.012688456391385135	9.86403600116801E-39   2.964393875E-314			
2.3230339468969963E-114				
ReplicationLog	CommitLogMessagesRead	16873	events/second	
0.21366497971804438	0.20580430240786005   0.39126032533612265			
0.2277227124698431				

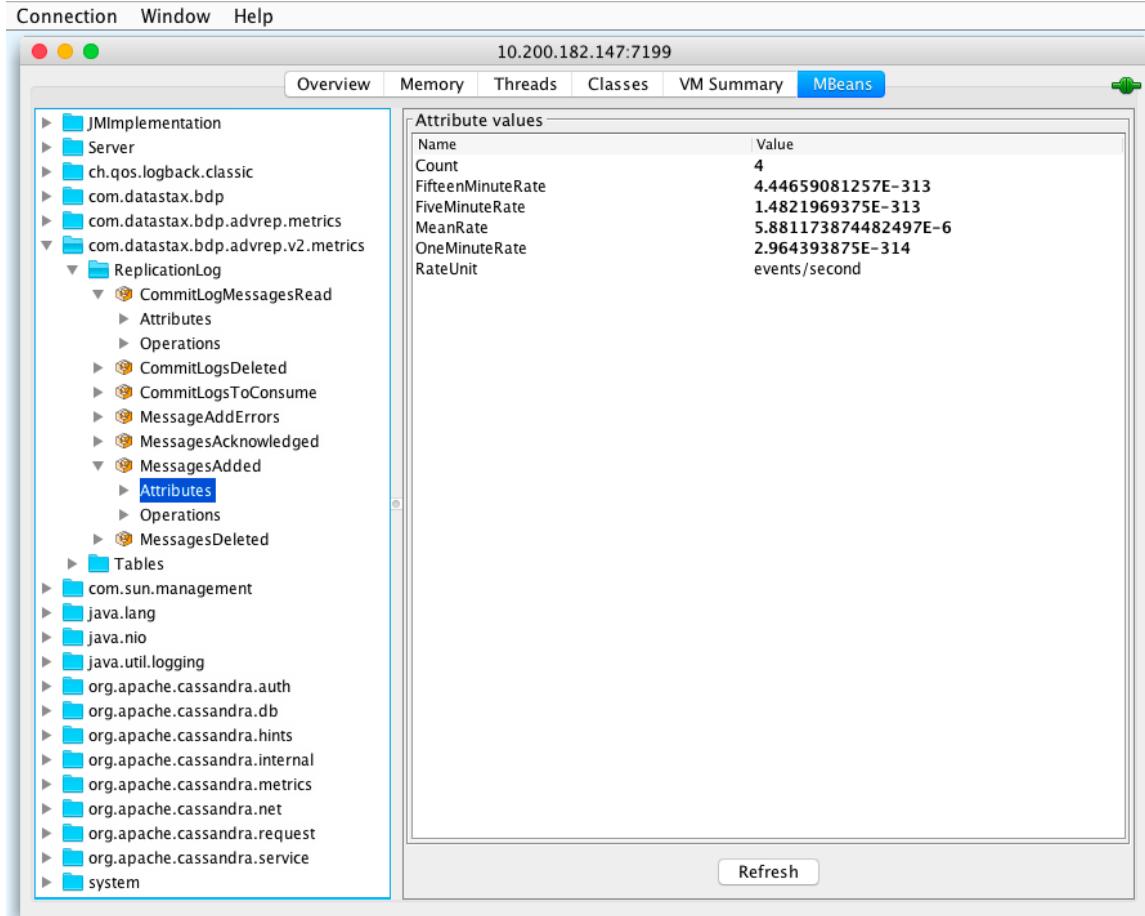
Group	Type	Value
Transmission	AvailablePermits	30000

## Accessing the metrics

Use JMX to access the metrics. Any JMX tool, such as `jconsole`, can access the MBeans for advanced replication. The port listed above, 7199, is used with the hostname or IP address:



Choose the MBeans tab and find `com.datastax.bdp.advrep.v2.metrics` in the left-hand navigation frame:



The example shown here displays the attributes for

`com.datastax.bdp.advrep.v2.metrics:type=ReplicationLog, name=MessagesAdded.`

## Performance metrics

Metrics are exposed as JMX MBeans under the `com.datastax.bdp.advrep.v2.metrics` path and are logically divided into main groups. Each group refers to an architecture component. Metrics types are:

### Counter

A simple incrementing and decrementing 64-bit integer.

### Meter

Measures the rate at which a set of events occur.

### Histogram

Measures the distribution of values in a stream of data.

### Timer

A histogram of the duration of a type of event and a meter of the rate of its occurrence.

### Gauge

A gauge is an instantaneous measurement of a value.

Metrics are available for the following groups:

- [ReplicationLog \(page 786\)](#)
- [Transmission \(page 787\)](#)
- [AdvancedReplicationHub-\[destinationId\]-metrics \(page 787\)](#)

Metrics are also available per table:

- [Performance metrics per table \(page 789\)](#)

Descriptions of each metric is provided.

**Note:** Metrics for DSE 5.0 (V1) are still present; see the [DSE 5.0 documentation](#) for those metrics.

## ReplicationLog

Metrics for the ReplicationLog group:

Metric name	Description	Metric type
MessagesAdded	The number of messages that were added to the replication log, and the rate that the messages were added, per replica.	Meter
MessagesAcknowledged	The number of messages that were acknowledged (and removed) from the replication log. Acknowledgement can be 1 or 1+n if errors occur.	Meter
MessagesDeleted	The number of messages that were deleted from the replication log, including invalid messages and messages that were removed after a channel truncate operation.	Meter
MessageAddErrors	The number of errors that occurred when adding a message to the replication log.	Counter
CommitLogsToConsume	The number of commit logs that need to be consumed that have advanced replication messages.	Counter
CommitLogMessagesRead	The number of commit log messages added to the replication log. The commit log messages are read if a message pertains to a source table that has collection enabled.	Meter

Metric name	Description	Metric type
CommitLogMessagesDeleted	The number of commit log messages deleted from the commit log after adding to the replication log. Like CommitLogMessagesRead, this metric only pertains to messages in tables that are enabled for advanced replication.	Meter

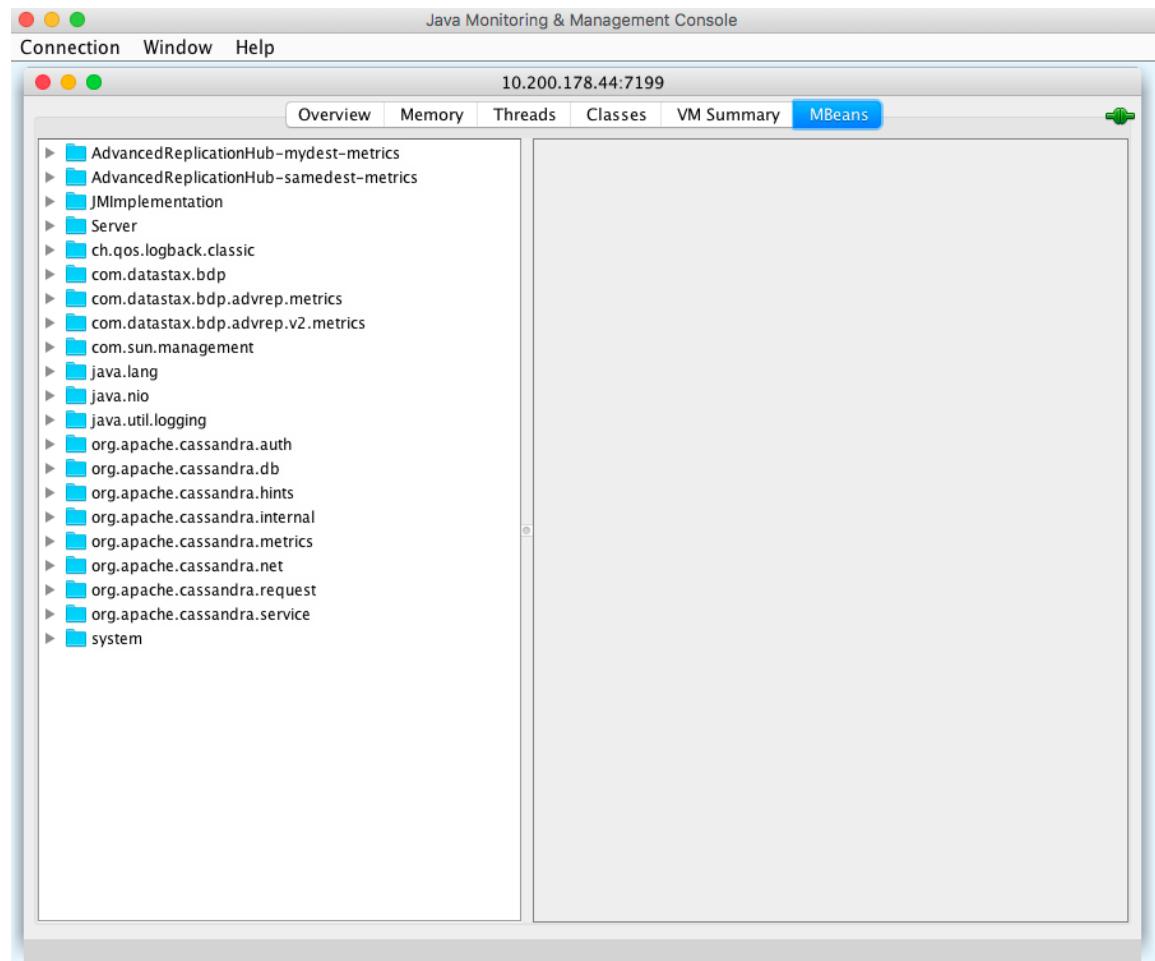
## Transmission

Metrics for the Transmission group:

Metric name	Description	Metric type
AvailablePermits	The current number of available global permits for transmission.	Gauge

## AdvancedReplicationHub-[destinationName]-metrics

Metrics for the AdvancedReplicationHub-[destinationName]-metrics group are provided automatically by the DSE Java driver.



Incomplete examples of per-destination-metrics are:

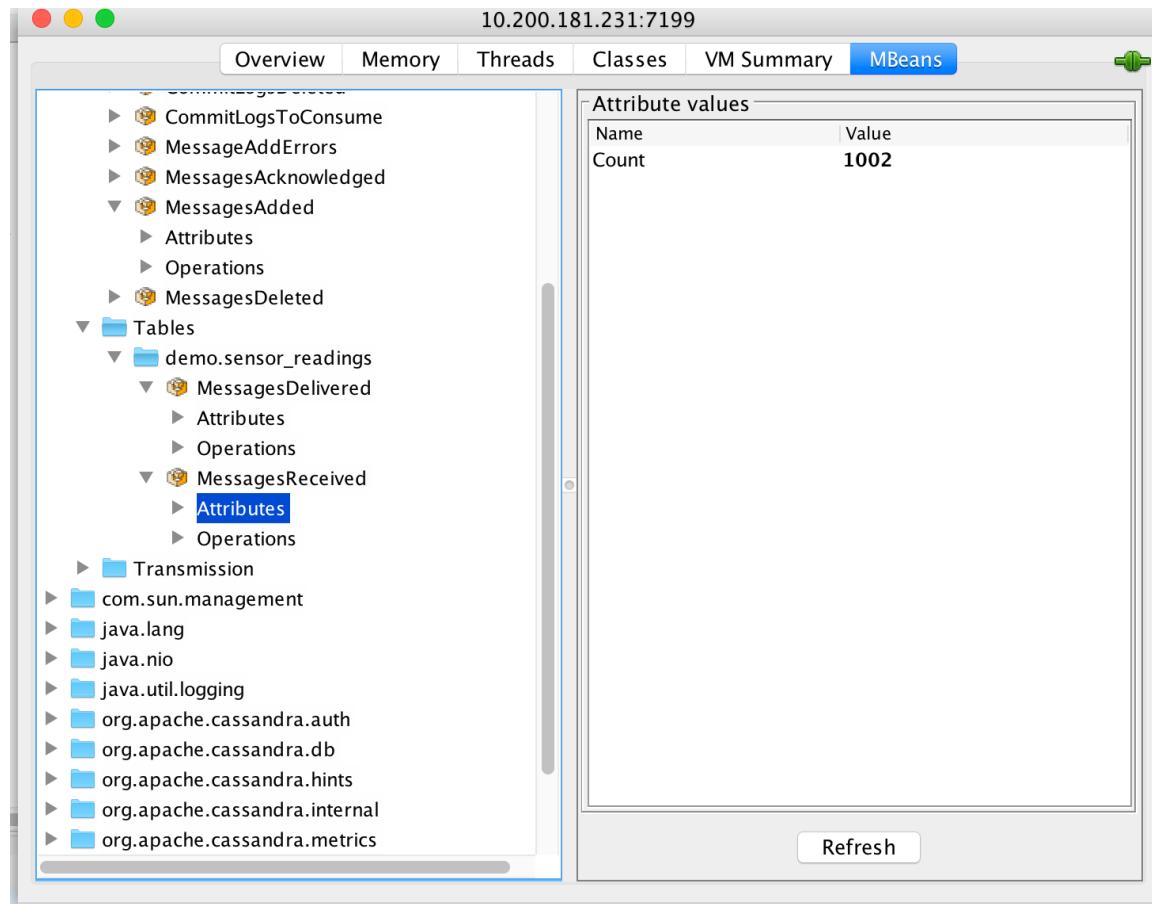
Metric name	Metric type
known-hosts	Counter
connected-to	Counter
open-connections	Counter
requests-timer	Timer
connection-errors	Counter
write-timeouts	Counter
read-timeouts	Counter
unavailables	Counter
other-errors	Counter
retries	Counter

Metric name	Metric type
ignores	Counter

For details, see the [DSE Java driver documentation](#).

## Performance metrics per table

Use JMX to find performance metrics per table, look under the `com.datastax.bdp.advrep.v2.metrics` tab in the left-hand navigation frame for Tables, select a table and inspect the metrics:



For example, to access the `MessagesReceived` metric for the table `sensor_readings` in the keyspace `demo` look at the following path:

```
com.datastax.bdp.advrep.v2.metrics:type=Tables,scope=demo.sensor_readings,name=MessagesRec
```

The following metrics are provided per table:

Metric name	Description	Metric type
MessagesReceived	The number of messages received from the source cluster for this table.	Counter

Metric name	Description	Metric type
MessagesDelivered	The number of messages for the source table that were replicated to the destination.	Counter
MessagesDeleted	The number of messages that were deleted from the replication log, including invalid messages and messages that were removed after a channel truncate operation.	Counter

## Managing invalid messages

DSE Advanced Replication strategies for managing invalid messages when replication fails.

During message replication, DSE Advanced Replicates attempts to manipulate the message to ensure successful replication. In some cases, replication might occur with only a subset of the data.

In other cases, replication fails when there are too many differences between the schema on the source cluster and the schema on the destination cluster. For example, schema incompatibilities occur when a column in the destination has a different type than the same column in the source, or a table in the source doesn't contain all the columns that form the primary key of the same table in the destination.

If a message cannot be replicated, a second transmission will be tried. If replication still fails after that the second try, the message is discarded and removed from the replication log.

The replication log on the source cluster stores data in preparation for transmission to the destination cluster.

When a message is discarded, the CQL query string and the related error message are logged on the destination cluster. To define where to store the CQL strings and the error messages that are relevant to the failed message replication, use one of the following logging strategies:

- SYSTEM\_LOG: Log the CQL query and the error message in the system log on the destination.
- CHANNEL\_LOG: Store the CQL query and the error message in files in `/var/lib/cassandra/advrep/invalid_queries` on the destination. This is the default value.
- NONE: Perform no logging.

For the channel logging strategy, a file is created in the channel log directory on the source node, following the pattern `/var/lib/cassandra/advrep/invalid_queries/<keyspace>/<table>/<destination>/invalid_queries.log` where `keyspace`, `table` and `destination` are:

- `keyspace`: `keyspace` name of the invalid query
- `table`: `table` name of the invalid query
- `destination`: destination cluster of the channel

The log file stores the following data that is relevant to the failed message replication:

- time\_bucket: an hourly time bucket to prevent the database partition from getting too wide
- id: a time based id (timeuuid)
- cql\_string: the CQL query string, explicitly specifies the original timestamp by including the USING TIMESTAMP option.
- error\_msg: the error message

Invalid messages are inserted by time in the log table.

#### Manage invalid messages using channel logging:

1. To store the CQL query string and error message using a channel log, instead of the default system log location, specify the invalid\_message\_log configuration key as CHANNEL\_LOG:

```
dse advrep conf update --invalid_message_log CHANNEL_LOG
```

#### Manage invalid messages using system logging:

2. To store the CQL query string and error message using a system log, instead of the default channel log location, specify the invalid\_message\_log configuration key as SYSTEM\_LOG:

```
dse advrep conf update --invalid_message_log SYSTEM_LOG
```

3. To identify the problem, examine the error messages, the CQL query strings, and the schemas of the data on the source and the destination.
4. Take appropriate actions to resolve the incompatibility issues.

## Managing audit logs

Use metadata configuration to manage the DSE Advanced Replication replication load audit logs.

DSE Advanced Replication provides replication audit logging and commands to manage the audit logs with metadata configuration. Audit logs are stored on the source cluster and are handled by the audit log analyzer (AuditLogAnalyzer). The audit log analyzer reads the log files, including audit log files in GZIP (.gz) format, that might be incomplete because they are still being written or they were improperly closed. The audit log analyzer identifies the list of files which match the template that is defined with the audit\_log\_file configuration key and that have exceeded the maximum time interval since they were written to. Purging is based on these criteria.

Global settings apply to the entire source cluster. These global settings are stored in the CQL table `dse_system.advrep_source_config` that is automatically created. To define configuration keys to [change global settings \(page 763\)](#), use the `dse advrep conf update` command. The audit log files are read/write (RW) only for the file owner, with no permissions for other users.

**Note:** The time stamp for all writes is UTC (Universal Time Coordinated ).

1. Enable replication audit logging:

```
dse advrep conf update --audit-log-enabled true
```

2. The default base audit log directory is `/var/lib/cassandra/advrep/auditlog`. To define a different directory for storing audit log files:

```
dse advrep conf update --audit-log-file /tmp/auditAdvRep
```

If the configured audit log file is a relative path, then the log files be placed in the default base directory. If the configured audit log file is an absolute path, then that path is used.

3. To compress the audit log output using the gzip file format:

```
dse advrep conf update --audit-log-compression GZIP --audit-log-file /tmp/auditAdvRep/myaudit.gz
```

The default value is NONE for compression. If `.gz` is not appended to the audit log filename in the command, it will be appended to the created files. Compressed audit log files will remain locked until rotated out; the active file cannot be opened.

4. Specify the time interval to rotate the audit log file. On rotation, the rotated file is appended with the log counter `.[logcounter]`, incrementing from [0]. To disable rotation, set to 0.

```
dse advrep conf update --audit-log-rotate-mins 120
```

For example, the compressed file from the last step can be uncompressed after rotating out to `/tmp/auditAdvRep/myaudit.[0].gz`.

5. Specify the maximum lifetime of audit log files.

After audit log files are rotated, they are periodically purged when the log files:

- Match the audit log file
- And have not been written to for more than the specified maximum lifespan minutes

To disable purging, set to 0.

```
dse advrep conf update --audit-log-max-life-span-mins 120
```

6. Restart the node to enable the changes.

When logging is enabled, log files that would be overwritten are moved to a subdirectory in the log directory. The subdirectory is named `archive_x`, where x increments from 0 until an unused directory is identified and created.

# DataStax Enterprise tools

Tools include dse commands, dsetool, fs-stress tool, pre-flight check, and yaml\_diff tools, and the sstableloader.

## DataStax Bulk Loader (dsbulk)

The DataStax Bulk Loader tool dsbulk can be used to load and unload DSE data in either CSV or JSON format.

DataStax recommends using the latest DataStax Bulk Loader 1.1.0. For details, see the [DataStax Bulk Loader Guide](#).

## dse command

The dse cassandra commands for starting the database.

The dse commands for starting the database and connecting an external client to a DataStax Enterprise node and performing common utility tasks. For a complete list of administrator commands, see [dse command \(page 793\)](#) in the Administrator Guide.

## dse commands

The dse cassandra commands for starting the database.

The dse commands for starting the database and connecting an external client to a DataStax Enterprise node and performing common utility tasks. For a complete list of administrator commands, see [dse command \(page 793\)](#) in the Administrator Guide.

The dse commands provide additional controls for starting and using DataStax Enterprise (DSE).

- [Synopsis \(page 793\)](#)
- [Connection options \(page 794\)](#)
- [dse subcommands \(page 795\)](#)
- [DSE Multi-Instance commands \(page 798\)](#)

### Synopsis

Package installations:

```
dse connection_options subcommand [command_arguments]
```

Tarball installations: /bin/ dse spark

```
installation_location/bin/dse connection_options subcommand [command_arguments]
```

**Table 126: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>Italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ <i>key:value</i> }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
< <i>datatype1,datatype2&gt;</i>	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
<i>cql_statement</i> ;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' < <i>schema</i> > ... </ <i>schema</i> > '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.
@ <i>xml_entity='xml_entity_type'</i>	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

## Connection options

Through authentication, the database establishes the identity of the person or system that is attempting an operation. Authentication credentials can be provided in several ways, see [Providing credentials from DSE tools](#).

Connection options are authentication command arguments that can be used with all dse subcommands:

```
dse connection_options subcommand [command_arguments]
```

**Table 127: Connection options to authenticate dse command**

Command arguments	Description
-f	Path to <a href="#">configuration file</a> that stores credentials. The credentials in this configuration file override the <code>~/.dserc</code> credentials. If not specified, then use <code>~/.dserc</code> if it exists.
-u	Role to authenticate against the configured authentication schema.
-p	Password to authenticate against the configured authentication schema.
-a	User name to authenticate with secure JMX.
-b	Password to authenticate with secure JMX.
-v	Send the DataStax Enterprise version number to standard output. Does not require authentication.

## dse subcommands

Specify one dse subcommand and none or more optional command arguments.

**Note:** When multiple flags are used, list them separately on the command line. For example, ensure there is a space between `-k` and `-s` in `dse cassandra -k -s`.

Subcommand and command arguments	Description
<code>advrep dse advrep commands</code>	Options for configuring and using <a href="#">DSE Advanced Replication (page 742)</a> .
<code>beeline</code>	Start the <a href="#">Beeline (page 254)</a> shell.
<code>cassandra</code>	Start a real-time transactional node in the background. See <a href="#">Starting DataStax Enterprise (page 870)</a> .
<code>cassandra -f</code>	Start a real-time transactional node in the foreground.
<code>cassandra -g</code>	Start a node in graph mode. See <a href="#">Starting DataStax Enterprise (page 870)</a> . Can be used with <code>-s</code> and <code>-k</code> options.
<code>cassandra -k</code>	Start up an analytics node in Spark mode in the background. See <a href="#">Starting Spark (page 179)</a> .
<code>cassandra -fs</code>	Invoke the DataStax Enterprise file system <a href="#">DSEFS (page 273)</a> shell.
<code>cassandra-stop -p pid</code>	Stop the DataStax Enterprise process number pid. See <a href="#">Stopping a node (page 872)</a> .
<code>cassandra -s</code>	Start a DSE Search node in the background. See <a href="#">Starting DataStax Enterprise (page 870)</a> .

Subcommand and command arguments	Description
<code>cassandra -s -Ddse.solr.data.dir=path</code>	Use <i>path</i> to store DSE Search data. See <a href="#">Managing the location of DSE Search data</a> .
<code>cassandra -Doption</code>	All -D options in <a href="#">DataStax Enterprise start-up parameters (page 155)</a> commands are supported.
<code>client-tool subcommand (page 799) subcommand</code>	See <a href="#">dse client-tool (page 799)</a> .
<code>dse-nodeID DSE Multi-Instance commands</code>	Run standard dse commands for nodes on a <a href="#">DSE Multi-Instance</a> host machine.
<code>exec command</code>	Sets the environment variables required to configure DSE Spark before executing the given command. This command is typically used for third-party tools that integrate with Spark. When you run <code>dse exec</code> it sets <code>SPARK_HOME</code> to point to the DSE Spark directory, sets <code>HADOOP_CONF_DIR</code> to point to the Hadoop configuration directory within DSE, sets other environment variables required by DSE Spark to enable custom DSE, and executes the given shell command.
<code>hadoop fs</code>	Invoke DSEFS operations using the HDFS interface to <a href="#">DSEFS (page 273)</a> .
<code>fs</code>	Run DSEFS shell. See <a href="#">dsefs command line (page 279)</a> tool.
<code>pyspark</code>	<a href="#">PySpark (page 242)</a> .
<code>spark</code>	<a href="#">Accessing database data from Spark (page 184)</a> .

Subcommand and command arguments	Description
<pre>-framework dse spark-2.0</pre>	<p>Use with dse spark commands to specify a different classpath to accommodate applications originally written for open source Apache Spark™. Omit the --framework option for the default (dse) behavior present in earlier versions of DSE.</p> <ul style="list-style-type: none"> <li>• dse           <p>Default. Sets all of the spark classpath to the same classpath that is used by the DSE server.</p> </li> <li>• spark-2.0           <p>Sets a classpath that is used by the open source Spark (OSS) 2.0 release. Uses a <a href="#">BYOS (Bring Your Own Spark) (page 255)</a> JAR with shaded references to internal dependencies to eliminate complexity when porting an app from OSS Spark.</p> <p><b>Note:</b> If the code works on DSE, applications do not require this framework. Full support in the spark-2.0 framework might require specifying additional dependencies. For example: hadoop-aws is included on the dse server path but is not present on the OSS Spark-2.0 classpath. In this example, applications that use S3 or other AWS APIs must include their own aws-sdk on the runtime classpath. This additional runtime classpath is required only for applications that cannot run on the DSE classpath.</p> </li> </ul>
spark-history-server start	Start <a href="#">Spark history server (page 225)</a> .
spark-history-server stop	Stop <a href="#">Spark history server (page 225)</a> .
spark-sql-thriftserver start	Start <a href="#">Spark SQL Thrift server (page 243)</a> .
spark-sql-thriftserver stop	Stop <a href="#">Spark SQL Thrift server (page 243)</a> .
spark-jobserver start submission_arguments	Launch applications on a cluster and use <a href="#">Spark Jobserver (page 258)</a> . See <a href="#">Spark submit options (page 227)</a> .
spark-jobserver stop	Stop the <a href="#">Spark Jobserver (page 258)</a> .
spark-sql	<a href="#">Spark SQL command line (page 235)</a>
spark-submit submission_arguments	Launch applications on a cluster and use Spark cluster managers. See <a href="#">dse spark-submit (page 181)</a> .
sparkR	Starts the R shell configured with DSE Spark. See <a href="#">Using SparkR with DataStax Enterprise (page 245)</a> .

Subcommand and command arguments	Description
spark-sql-metastore-migrate	<p>Map custom external tables to the new release format of the Hive metastore used by Spark SQL after upgrading. For example:</p> <pre data-bbox="660 418 1330 475">dse spark-sql-metastore-migrate --to 5.1.0 -- from 5.0.4</pre>

## DSE Multi-Instance commands

To run standard DataStax Enterprise commands for nodes on a DSE Multi-Instance host machine, specify the node name using this syntax:

```
sudo dse dse-nodeId subcommand [command_arguments]
```

For details, see [DSE Multi-Instance commands](#).

## dse cassandra start parameters

List of DataStax Enterprise start parameters.

Start parameters can be run from the command line or specified in the `cassandra-env.sh` file.

### Usage

Specify the `start` option on the command line:

```
dse cassandra option
```

For example:

```
bin/dse cassandra -
Dcassandra.prepared_statements_cache_size_in_bytes=345678912
```

You can set start options and set options to the JVM instead of setting them in the environment. Add an entry for each option to the `cassandra-env.sh` file:

```
JVM_OPTS="$JVM_OPTS -Doption"
```

For example:

```
JVM_OPTS="$JVM_OPTS -
Dcassandra.prepared_statements_cache_size_in_bytes=345678912"
```

## Start parameters

All of the [start parameters \(page 155\)](#) are supported for starting a DSE node and for use in the `cassandra-env.sh` for DSE. The following start parameters are supported for DSE only:

Option	Description
<code>dse.solr.data.dir=path</code>	The <i>path</i> to store DSE Search data. See <a href="#">Set the location of search indexes</a> .

## dse client-tool

Connects an external client to a DataStax Enterprise node and performs common utility tasks.

### About dse client-tool

A command line interface that connects an external client to a DataStax Enterprise node and performs common utility tasks.

The dse client-tool command line interface connects an external client to a DataStax Enterprise node and performs common utility tasks.

For full dse client-tool command documentation, see the [DSE Administrator Guide](#).

### Connection options

Connection options specify how to connect and authenticate for all dse client-tool commands:

Short	Long	Description
	<code>--port</code>	Port number.
<code>-p</code>	<code>--password</code>	Password.
<code>-u</code>	<code>--username</code>	Username.
<code>-a</code>		DSE authorization username if <a href="#">proxy authentication</a> is used.
<code>-t</code>		Delegation token which can be used to login. Alternatively, you can use the <code>DSE_TOKEN</code> environment variable.
<code>--</code>		Separates command parameters from a list of options.

#### Note:

- If a username and password for RMI authentication are set explicitly in the `cassandra-env.sh` file for the host, then you must specify credentials.
- The repair and rebuild commands can affect multiple nodes in the cluster.
- Most nodetool commands operate on a single node in the cluster if `-h` is not used to identify one or more other nodes. If the node from which you issue the command is the intended target, you do not need the `-h` option to identify the

target; otherwise, for remote invocation, identify the target node, or nodes, using **-h**.

Example:

```
nodetool -u username -pw password describering demo_keyspace
```

## Using dse client-tool command line help

To show a listing of the dse client-tool subcommands:

```
dse client-tool help
```

To show the command line help for a specific dse client-tool subcommand:

```
dse client-tool help subcommand
```

For example:

```
dse client-tool help configuration
```

## dse cassandra

Starts the database and advanced features on a node.

Starts the database and advanced features on a node.

### Synopsis

```
dse cassandra [-f] [-h] [-k] [-s] [-g] \
[-p pidfile] [-H dumpfile] [-E errorfile] [start_parameters]
```

**Table 128: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>Italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

Syntax conventions	Description
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ <i>key:value</i> }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
< <i>datatype1,datatype2&gt;</i>	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
<i>cql_statement</i> ;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' < <i>schema</i> > ... </ <i>schema</i> > '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.
@ <i>xml_entity</i> =' <i>xml_entity_type</i> '	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

## Options

**-f**  
Start the database in the foreground.

**-h**  
Display help.

Enable advanced features

Use these options to start advanced features on the node. The first time the node starts up the workload type is configured.

**-k**  
Start Analytics service.

**-s**  
Start the Search service.

**-g**  
Start the Graph service.

Configuration parameters

Use the following settings to change the DSE system properties on start up.

**-Dcassandra.auto\_bootstrap**

Set [auto\\_bootstrap](#) ([page 83](#)) to `false` during the initial set up of a node to override the default setting in the `cassandra.yaml` file.

Default: `true`.

**-Dcassandra.available\_processors**

Number of processors available to DSE. In a multi-instance deployment, each instance independently assumes that all CPU processors are available to it. Use this setting to specify a smaller set of processors.

Default: `all_processors`.

**-Dcassandra.config**

Set to the directory location of the `cassandra.yaml` file.

Default: depends on the type of installation.

**-Dcassandra.consistent.rangemovement**

Set to `true`, makes bootstrapping behavior effective.

Default: `false`.

**-Dcassandra.disable\_auth\_caches\_remote\_configuration**

Set to `true` to disable authentication caches, for example the caches used for credentials, permissions, and roles. This will mean those config options can only be set (persistently) in `cassandra.yaml` and will require a restart for new values to take effect.

Default: `false`.

**-Dcassandra.expiration\_date\_overflow\_policy**

Set the policy (`REJECT` or `CAP`) for any TTL (time to live) timestamps that exceeds the maximum value supported by the storage engine,

`2038-01-19T03:14:06+00:00`. The database storage engine can only encode TTL timestamps through January 19 2038 03:14:07 UTC due to the [Year 2038 problem](#).

- `REJECT`: Reject requests that contain an expiration timestamp later than `2038-01-19T03:14:06+00:00`.
- `CAP`: Allow requests and insert expiration timestamps later than `2038-01-19T03:14:06+00:00` as `2038-01-19T03:14:06+00:00`.

Default: `REJECT`.

**-Dcassandra.force\_default\_indexing\_page\_size**

Set to `true` to disable dynamic calculation of the page size used when indexing an entire partition during initial index build or a rebuild. Fixes the page size to the default of 10000 rows per page.

Default: `false`.

**-Dcassandra.ignore\_dc**

Set to `true` to ignore the datacenter name change on startup. Applies only when using `DseSimpleSnitch`.

Default: `false`.

**-Dcassandra.initial\_token**

Use when DSE is not using virtual nodes (vnodes). Set to the initial partitioner token for the node on the first start up.

Default: `blank` (not set).

**Note:** Vnodes automatically select tokens.

**-Dcassandra.join\_ring**

Set to `false` to prevent the node from joining a ring on startup.

**Tip:** Add the node to the ring afterwards using `nodetool join` and a JMX call.

Default: `true`.

**-Dcassandra.load\_ring\_state**

Set to `false` to clear all gossip state for the node on restart.

Default: `true`.

#### **-Dcassandra.metricsReporterConfigFile**

Enables pluggable metrics reporter and configures it from the specified file.

Default: `blank` (not set).

#### **-Dcassandra.native\_transport\_port**

Set to the port number that CQL native transport listens for clients.

Default: 9042.

#### **-Dcassandra.native\_transport\_startup\_delay\_seconds**

Set to the number of seconds to delay the native transport server start up.

Default: 0 (no delay).

#### **-Dcassandra.partitioner**

Set to the partitioner name.

Default: `org.apache.cassandra.dht.Murmur3Partitioner`.

#### **-Dcassandra.partition\_sstables\_by\_token\_range**

Set to `false` to disable JBOD SSTable partitioning by token range to multiple `data_file_directories`.

**Caution:** Advanced setting that should only be used with guidance from [DataStax Support](#).

Default: `true`.

#### **-Dcassandra.replace\_address**

Set to the [listen\\_address \(page 69\)](#) or the [broadcast\\_address \(page 84\)](#) when replacing a dead node with a new node. The new node must be in the same state as before bootstrapping, without any data in its data directory.

**Note:** The `broadcast_address` defaults to the `listen_address` except when the ring is using the [Configuring Amazon EC2 multi-region snitch \(page 153\)](#).

#### **-Dcassandra.replayList**

Allows restoring specific tables from an archived commit log.

#### **-Dcassandra.ring\_delay\_ms**

Set to the number of milliseconds the node waits to hear from other nodes before formally joining the ring.

Default: 30000.

#### **-Dcassandra.ssl\_storage\_port**

Sets the SSL port for encrypted communication.

Default: 7001.

#### **-Dcassandra.start\_native\_transport**

Enables or disables the native transport server. See [start\\_native\\_transport \(page 91\)](#) in `cassandra.yaml`.

Default: `true`.

#### **-Dcassandra.storage\_port**

Sets the port for inter-node communication.

Default: 7000.

# dsetool

A list of the available commands for DSE operations.

## About dsetool

A command line interface for DSE operations.

dsetool is a command line interface for DSE operations.

### Synopsis

```
dsetool [connection_options] command command_args
```

**Table 129: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ <i>key:value</i> }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
< <i>datatype1,datatype2</i> >	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
<i>cql_statement</i> ;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.

Syntax conventions	Description
' <schema> ... </schema> '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.
@xml_entity='xml_entity_type'	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

## Using dsetool command line help

To view a listing of dsetool commands:

```
dsetool help
```

To view help for a specific command:

```
dsetool command help
```

## dsetool commands for DSE Search

Search [CQL commands](#) are distributed to the entire data center. The dsetool commands for DSE Search distribute search index changes to the data center by default, and are node-specific only when the distributed flag is set to false.

## Connection options

Options for connecting to your cluster with the dsetool utility.

Options for connecting to your cluster with the dsetool utility.

## Synopsis

```
dsetool [connection_options]
```

**Table 130: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.

Syntax conventions	Description
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ <i>key:value</i> }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
< <i>datatype1,datatype2&gt;</i>	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
<i>cql_statement</i> ;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' < <i>schema</i> > ... </ <i>schema</i> > '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.
@ <i>xml_entity</i> =' <i>xml_entity_type</i> '	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

JMX authentication is supported by some dsetool commands. Other dsetool commands authenticate with the user name and password of the configured user. The connection option short form and long form are comma separated.

**Note:**

You can provide authentication credentials in several ways, see [Credentials for authentication](#).

To enable dsetool to use Kerberos authentication, see [Using dsetool with Kerberos enabled cluster](#).

Specify how to connect and authenticate the dsetool command.

This list shows short form (-f *filename*) and long form (--config-file=*filename*):

**-a jmx\_username, --jmxusername jmx\_username**

User name for authenticating with secure local JMX.

**-b jmx\_password, --jmxpassword jmx\_password**

Password for authenticating with secure local JMX. If you do not provide a password, you are prompted to enter one.

**-c dse\_port, --cassandra\_port dse\_port**

DSE port number.

**--cipher-suites ssl\_cipher\_suites**

Specify comma-separated list of SSL cipher suites for connection to DSE when SSL is enabled. For example, --cipher-suites c1,c2,c3.

**-f filename, --config-file filename**

File path to configuration file that stores credentials. The credentials in this configuration file override the [~/.dserv credentials](#).

**-h IP\_address, --host IP\_address**  
 Connect to the specified hostname or IP address. Do not specify to connect to the local node.

**-j jmx\_port, --jmlexport jmx\_port**  
 Remote JMX agent port number.

**--keystore-path ssl\_keystore\_path**  
 Path to the keystore for connection to DSE when SSL client authentication is enabled.

**--keystore-password keystore\_password**  
 Keystore password for connection to DSE when SSL client authentication is enabled.

**--keystore-type ssl\_keystore\_type**  
 Keystore type for connection to DSE when SSL client authentication is enabled. JKS is the type for keys generated by the Java keytool binary, but other types are possible, depending on user environment.

**-l username, --username username**  
 Role to authenticate for database access.

**-p password, --password password**  
 Password to authenticate for database access.

**-s solr\_port, --port solr\_port**  
 Solr port.

**--ssl ( true | false )**  
 Whether to use SSL for native connections.

**--ssl-protocol ssl\_protocol**  
 SSL protocol for connection to DSE when SSL is enabled. For example, --ssl-protocol ssl4.

**--sslauth ( true | false )**  
 Whether to use SSL client authentication.

**--truststore\_password ssl\_truststore\_password**  
 Truststore password to use for connection to DSE when SSL is enabled.

**--truststore-path ssl\_truststore\_path**  
 Path to the truststore to use for connection to DSE when SSL is enabled. For example, --truststore-path /path/to/ts.

**--truststore-type ssl\_truststore\_type**  
 Truststore type for connection to DSE when SSL is enabled. JKS is the type for keys generated by the Java keytool binary, but other types are possible, depending on user environment. For example, --truststore-type jks2.

## dsetool core\_indexing\_status

Retrieves the dynamic indexing status of a search index on a DSE Search node.

Retrieves the dynamic indexing status of a search index on a DSE Search node and displays the percent complete and an estimated completion time in milliseconds.

**Restriction:** Command is supported only on nodes with DSE Search workloads.

## Synopsis

```
dsetool core_indexing_status [keyspace_name.]table_name| --all [--progress]
```

**Table 131: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ <i>key:value</i> }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
< <i>datatype1,datatype2&gt;</i>	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
<i>cql_statement</i> ;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' < <i>schema</i> > ... </ <i>schema</i> > '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.
@ <i>xml_entity</i> =' <i>xml_entity_type</i> '	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

Retrieves the dynamic indexing status (INDEXING, FINISHED, or FAILED) of the specified index or indexes, where:

**[keyspace\_name.]table\_name**

The search index table name is required. The keyspace name is optional. The case of keyspace and table names is preserved. You must use the correct case for the keyspace and table names.

**--all**

Retrieve the dynamic indexing status of all search indexes.

**--progress**

Display the percent complete and an estimated completion time in milliseconds.

This option is ignored and is assumed true. The command always displays the percent complete and an estimated completion time in milliseconds.

See [Verifying indexing status \(page 336\)](#).

**Examples**

To view the indexing status for the local node:

```
dsetool core_indexing_status demo.health_data
```

The results are displayed:

```
[demo.health_data]: INDEXING, 38% complete, ETA 452303 milliseconds (7
minutes 32 seconds)
```

To view the indexing status for a search index on a specified node:

```
dsetool -h 200.192.10.11 core_indexing_status demo.health_data
```

To view indexing status of all search indexes in the data center:

```
dsetool -h 200.192.10.11 core_indexing_status --all
```

## [dsetool create\\_core](#)

Creates the search index table on the local node.

Creates the search index table on the local node. Supports DSE authentication with `[-u username -p password]`.

The CQL command to create a search index is [CREATE SEARCH INDEX](#).

**Restriction:** Command is supported only on nodes with DSE Search workloads.

Auto-generated schemas have default DocValues enabled. See [Creating a search index with default values](#) for details on docValues.

**Note:** If one or more nodes fail to create the core in distributed operations, an error message indicates the failing node or nodes. If it failed to create the core immediately, issue the create again. If it failed to create on some nodes, issue a reload for those nodes to load the newly created core.

## Synopsis

```
dsetool create_core keyspace_name.table_name
[coreOptions=yamlFile | coreOptionsInline=key1:value1#key2:value2#...]
[distributed=true|false]
```

```
[generateResources=true|false]
[schema=path]
[solrconfig=path]
```

**Table 132: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>Italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ <i>key:value</i> }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
< <i>datatype1,datatype2</i> >	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
<i>cql_statement</i> ;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' < <i>schema</i> > ... </ <i>schema</i> > '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.
@ <i>xml_entity</i> =' <i>xml_entity_type</i> '	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

**keyspace\_name.table\_name**

Required. The keyspace and table names of the search index. Keyspace and table names are case-sensitive. Enclose names that contain uppercase in double quotation marks.

**coreOptions=yamlFile**

When `generateResources=true`, specify a customized YAML-formatted file of options. The contents of the file are the same options that can be specified with `coreOptionsInline`. See [Changing auto-generated search index settings \(page 331\)](#).

#### **coreOptionsInline=key1:value1#key2:value2#...**

Use this key-value pair syntax `key1:value1#key2:value2#` to specify values for these settings: See [Changing auto-generated search index settings \(page 331\)](#).

#### **auto\_soft\_commit\_max\_time:ms**

The maximum auto soft commit time in milliseconds.

#### **default\_query\_field:field**

The query field to use when no field is specified in queries.

#### **enable\_string\_copy\_fields:( true | false )**

`true | false` - Generate non-stored string copy fields for non-key text fields. Text data can be tokenized or non tokenized. True creates a non-stored, non-tokenized copy field, so that you can have text both ways. Default: `false`.

#### **exclude\_columns: col1, col2, col3, ...**

A comma-separated (CSV) list of columns to exclude.

#### **generate\_DocValues\_for\_fields: (\* | field1, field2, ...)**

Specify the fields to automatically configure DocValues in the generated search index schema. Specify `'*'` to add all possible fields:

```
generate_DocValues_for_fields: '*'
```

or specify a comma-separated list of fields, for example:

```
generate_DocValues_for_fields: uuidfield, bigintfield
```

**Note:** Solr does not support DocValue on boolean fields.

#### **distributed=( true | false )**

Whether to distribute and apply the operation to all nodes in the local datacenter.

- True applies the operation to all nodes in the local datacenter.
- False applies the operation only to the node it was sent to. False works only when `recovery=true`.

Default: `true`

**Warning:** Distributing a re-index to an entire datacenter degrades performance severely in that datacenter.

#### **generateResources=( true | false )**

Whether to automatically generate search index resources based on the existing CQL table metadata. Cannot be used with `schema=` and `solrconfig=`.

Valid values:

- `true` - Automatically generate search index schema and configuration resources if resources do not already exist. If resources exist,
- `false` - Default. Do not automatically generate search index resources.

#### **include\_columns**

A comma-separated (CSV) list of columns to include. Empty = includes all columns.

#### **index\_merge\_factor**

How many segments of equal size to build before merging them into a single segment.

#### **index\_ram\_buffer\_size**

The index ram buffer size in megabytes (MB).

#### **lenient**

Ignore non-supported type columns and continue to generate resources, instead of erroring out when non-supported type columns are encountered. Default: false

#### **resource\_generation\_profiles**

To minimize index size, specify a CSV list of profiles to apply while generating resources.

**Table 133: Resource generation profiles**

Profile name	Description
spaceSavingAll	Applies all options: spaceSavingNoTextfield, spaceSavingNoJoin, and spaceSavingSlowTriePrecision.
spaceSavingNoTextfield	No TextFields. Use StrField instead.
spaceSavingNoJoin	Do not index a hidden primary key field. Prevents joins across cores.
spaceSavingSlowTriePrecision	Set trie fields precisionStep to '0', allowing for greater space saving but slower querying.

**Note:** Using spaceSavings profiles disables auto generation of DocValues.

For example:

```
resource_generation_profiles: spaceSavingNoTextfield,
spaceSavingSlowTriePrecision
```

#### **rt**

Enable live indexing to increase indexing throughput. Enable live indexing on only one search index per cluster.

```
rt=true
```

#### **recovery=( true | false )**

Whether to delete and recreate the search index if it is not able to load due to corruption.

Valid values:

- true - If search index is unable to load, recover the index by deleting and recreating it.
- false - Default. No recovery.

#### **reindex=( true | false )**

Whether to reindex the data when search indexes are auto-generated with generateResources=true. Reindex works on a datacenter (DC) level. Reindex only once per search-enabled DC.

Valid values:

- true - Reindexes the data. Accepts reads and keeps the current search index while the new index is building.
- false - Default. Does not reindex the data. You can check and customize search index resources before indexing.

**schema=path**

Path of the UTF-8 encoded search index schema file. Cannot be specified when generateResources=true.

**solrconfig=path**

Path of the UTF-8 encoded search index configuration file. Cannot be specified when generateResources=true.

**Examples**

Automatically generate search index for the health\_data table in the demo keyspace

```
dsetool create_core demo.health_data generateResources=true
```

Override the default and reindex existing data, specify the reindex=true option

```
dsetool create_core demo.health_data generateResources=true reindex=true
```

The generateResources=true option generates resources only if resources do not exist in the solr\_resources table.

Use options in a YAML-formatted file

To turn on live indexing, also known as real-time (RT) indexing, the contents of the rt.yaml are rt: true:

```
dsetool create_core udt_ks.users generateResources=true reindex=true
coreOptions=rt.yaml
```

Enable encryption with inline options

Specify the class for directoryFactory to solr.EncryptedFSDirectoryFactory:

```
dsetool create_core keyspace_name.table_name generateResources=true
coreOptionsInline="directory_factory_class:solr.EncryptedFSDirectoryFactory"
```

```
dsetool create_core demo.health_data generateResources=true
coreOptionsInline="directory_factory_class:solr.EncryptedFSDirectoryFactory"
```

Use options in a YAML-formatted file

```
dsetool create_core demo.health_data
coreOptions="directory_factory_class:solr.EncryptedFSDirectoryFactory"
```

## **dsetool createsystemkey**

Creates an encryption/decryption key for transparent data encryption (TDE).

Creates an encryption/decryption key for transparent data encryption (TDE).

See [Transparent data encryption](#).

## Synopsis

```
dsetool createsystemkey
[cipher_algorithm[/mode/padding]
[length] [-d filepath]
[key_name] [-k=kmp_groupname]
[-t kmp_template] [-n namespace]]
```

**Table 134: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>Italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ <i>key:value</i> }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
< <i>datatype1,datatype2&gt;</i>	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
<i>cql_statement</i> ;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' < <i>schema</i> > ... </ <i>schema</i> > '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.
@ <i>xml_entity</i> =' <i>xml_entity_type</i> '	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

### **cipher\_algorithm[/mode/padding]**

DSE supports the following JCE cipher algorithms:

- AES/CBC/PKCS5Padding (valid with length 128, 192, or 256).
- AES/ECB/PKCS5Padding (valid with length 128, 192, or 256)
- DES/CBC/PKCS5Padding (valid with length 56)
- DESede/CBC/PKCS5Padding (valid with length 112 or 168)
- Blowfish/CBC/PKCS5Padding (valid with length 32-448)
- RC2/CBC/PKCS5Padding (valid with length 40-128)

Default value: AES/CBC/PKCS5Padding (with length 128)

#### **-d filepath, --directory filepath**

Key file output directory. Enables creating key files [before DSE is installed](#). This option is typically used by IT automation tools like Ansible. When no directory is specified, keys are saved to the value of [system\\_key\\_directory \(page 115\)](#) in dse.yaml.

#### **length**

Required if cipher\_algorithm is specified. Key length is not required for HMAC algorithms. Default value: 128 (with the default cipher algorithm AES/CBC/ PKCS5Padding)

#### **key\_name**

Unique file name for the generated system key file. Encryption key files can have any valid Unix name. When no name is specified, the default file name is system\_key. The default key file name is not configurable.

#### **-k=kmip\_groupname**

The name of the KMIP group that is defined in the [kmip\\_hosts \(page 116\)](#) section of dse.yaml.

#### **-t kmip\_template**

The key template on the specified KMIP provider.

#### **-n namespace**

Namespace on the specified KMIP provider.

## **Examples**

To create a local key file:

```
dsetool createsystemkey 'AES/ECB/PKCS5Padding' 128 system_key2
```

where system\_key2 is the unique file name for the generated key file.

To create an off-server key file:

```
dsetool createsystemkey 'AES/ECB/PKCS5Padding' 128 system_key2 -
kmip=group2
```

where group2 is the key server group defined in the [kmip\\_hosts \(page 116\)](#) section of dse.yaml.

To create a local key file in a specific directory:

```
dsetool createsystemkey 'AES/ECB/PKCS5Padding' 128 -d /mydir
```

See [Setting up local encryption keys](#).

## dsetool encryptconfigvalue

Encrypts sensitive configuration information.

Encrypts sensitive configuration information. This command takes no arguments and prompts for the value to encrypt.

### Example

```
dsetool encryptconfigvalue
```

## dsetool get\_core\_config

Displays the XML for the specified search index config.

Displays the XML for the specified search index config. Supports DSE authentication with [ -l *username* -p *password* ].

**Restriction:** Command is supported only on nodes with DSE Search workloads.

### Synopsis

```
dsetool get_core_config keyspace_name.table_name [current=true|false]
```

**Table 135: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>Italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ <i>key:value</i> }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.

Syntax conventions	Description
<code>&lt;datatype1,datatype2&gt;</code>	Set, list, map, or tuple. Angle brackets ( <code>&lt; &gt;</code> ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
<code>cql_statement;</code>	End CQL statement. A semicolon ( <code>:</code> ) terminates all CQL statements.
<code>[ -- ]</code>	Separate the command line options from the command arguments with two hyphens ( <code>--</code> ). This syntax is useful when arguments might be mistaken for command line options.
<code>' &lt;schema&gt; ... &lt;/schema&gt; '</code>	Search CQL only: Single quotation marks ( <code>'</code> ) surround an entire XML schema declaration.
<code>@xml_entity='xml_entity_type'</code>	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

### **keyspace\_name.table\_name**

Required. The keyspace and table names of the search index. Keyspace and table names are case-sensitive. Enclose names that contain uppercase in double quotation marks.

### **current=true|false**

Optionally specify to view the current (active) configuration.

- true - Returns the active live search index config.
- false - Default. Returns the pending (latest uploaded) search index configuration.

## Examples

The following examples view the search index config for the demo keyspace and health\_data table.

To view the pending (latest uploaded) configuration:

```
dsetool get_core_config demo.health_data
```

The XML for the auto-generated configuration is displayed:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<config>
 <abortOnConfigurationError>${solr.abortOnConfigurationError:true}</
 abortOnConfigurationError>
 <luceneMatchVersion>LUCENE_6_0_0</luceneMatchVersion>
 <dseTypeMappingVersion>2</dseTypeMappingVersion>
 <directoryFactory class="solr.StandardDirectoryFactory"
 name="DirectoryFactory" />
 <indexConfig>
 <rt>false</rt>
 <rtOffheapPostings>true</rtOffheapPostings>
 <useCompoundFile>false</useCompoundFile>
 <ramBufferSizeMB>512</ramBufferSizeMB>
 ...
 </indexConfig>
</config>
```

```
</requestHandler>
<admin>
 <defaultQuery>*:*</defaultQuery>
</admin>
</config>
```

To view the pending (latest uploaded) search index configuration:

```
dsetool get_core_config demo.health_data current=true
```

To save the XML output to a file:

```
dsetool get_core_config demo.health_data > /Users/maryjoe/Documents/
search/health_data_config.xml
```

The health\_data\_config.xml file is created.

## **dsetool get\_core\_schema**

Displays the XML for the pending or active search index schema.

Displays the XML for the pending or active search index schema. Supports DSE authentication with [-l *username* -p *password*].

**Restriction:** Command is supported only on nodes with DSE Search workloads.

### Synopsis

```
dsetool get_core_schema keyspace_name.table_name [current=true|false]
```

**Table 136: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.

Syntax conventions	Description
{ key:value }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
<datatype1,datatype2>	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
cql_statement;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' <schema> ... </schema> '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.
@xml_entity='xml_entity_type'	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

### keyspace\_name.table\_name

Required. The keyspace and table names of the search index. Keyspace and table names are case-sensitive. Enclose names that contain uppercase in double quotation marks.

### current=true|false

Optionally specify to view the current (active) schema.

- true - Returns the current live search index schema.
- false - Default. Returns the latest uploaded search index schema.

## Examples

The following examples view the search index schema for the demo keyspace and health\_data table.

To save the XML output to a file:

```
dsetool get_core_schema demo.health_data > /Users/maryjoe/Documents/search/health_data_schema.xml
```

The health\_data\_schema.xml file is created.

To view the pending (latest uploaded) search index schema:

```
dsetool get_core_schema demo.health_data
```

To view the active (currently loaded) search index schema:

```
dsetool get_core_schema demo.health_data current=true
```

The XML for the schema is displayed:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<schema name="autoSolrSchema" version="1.5">
```

```

<types>
 <fieldType class="org.apache.solr.schema.TextField" name="TextField">
 <analyzer>
 <tokenizer class="solr.StandardTokenizerFactory"/>
 <filter class="solr.LowerCaseFilterFactory"/>
 </analyzer>
 </fieldType>
 <fieldType class="org.apache.solr.schema.TrieIntField"
name="TrieIntField"/>
</types>
<fields>
 <field indexed="true" multiValued="false" name="grade_completed"
stored="true" type="TextField"/>
 ...
 <field indexed="true" multiValued="false" name="fips" stored="true"
type="TextField"/>
</fields>
<uniqueKey>(id,age)</uniqueKey>
</schema>

```

## dsetool help

Provides a listing of dsetool commands and parameters.

Provides a listing of dsetool commands and parameters.

### Synopsis

```
dsetool help
```

**Table 137: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>Italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.

Syntax conventions	Description
{ key:value }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
<datatype1,datatype2>	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
cql_statement;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' <schema> ... </schema> '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.
@xml_entity='xml_entity_type'	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

Typing `dsetool` or `dsetool help` provides a listing of dsetool commands and parameters.

**Note:** Help is not available on a single command.

## dsetool index\_checks (experimental)

Optional and experimental. Reads the full index and optionally performs sanity checks. No repairs or fixes occur. Run only when index is inactive. No writes are allowed while index check is running.

**Note:** Running this index check is time consuming and implies a hard commit.

**Restriction:** Command is supported only on nodes with DSE Search workloads.

### Synopsis

```
dsetool index_checks keyspace_name.table_name
[coreOptions=yamlfilepath] | [coreOptionsInline=options]
--index_checks=true|false
--index_checks_stop=true|false
```

**Table 138: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>Italics</i>	Variable value. Replace with a valid option or user-defined value.

Syntax conventions	Description
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ <i>key:value</i> }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
< <i>datatype1,datatype2&gt;</i>	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
<i>cql_statement</i> ;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' < <i>schema</i> > ... </ <i>schema</i> > '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.
@ <i>xml_entity</i> =' <i>xml_entity_type</i> '	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

**keyspace\_name.table\_name**

Required. The keyspace and table names of the search index. Keyspace and table names are case-sensitive. Enclose names that contain uppercase in double quotation marks.

**coreOptions=yamlFilepath**

When auto-generation is on with generateResources=true, the file path to a customized YAML-formatted file of options. See [Changing auto-generated search index settings \(page 331\)](#).

**coreOptionsInline=key1:value1#key2:value2#...**

Use this key-value pair syntax key1:value1#key2:value2# to specify values for these settings:

- auto\_soft\_commit\_max\_time:ms
- default\_query\_field:field
- distributed:( true | false )
- enable\_string\_copy\_fields:( true | false )

- exclude\_columns: col1, col2, col3, ...
- generate\_DocValues\_for\_fields:( \* | field1, field2, ... )
- generateResources:( true | false )

See [Changing auto-generated search index settings \(page 331\)](#).

#### --index\_checks=true|false

Specify to run the index check.

- true - Runs the index check to verify index integrity. Reads the full index and has performance impact.
- false - Default. Does not run the index check.

#### --index\_checks\_stop=true|false

Specify to stop the index check.

- true - Requests the index check to stop.
- false - Does not stop the index check.

## Examples

**Important:** Ensure that indexing is inactive before doing an index check.

To do an index check:

```
dsetool index_checks demo.health_data
```

The LUKE handler information is displayed:

```
LUKE handler info:

numDocs:0
maxDoc:0
deletedDocs:0
indexHeapUsageBytes:0
version:2
segmentCount:0
current:true
hasDeletions:false
directory:org.apache.lucene.store.MMapDirectory:MMapDirectory@/
Users/maryjoe/dse/data/solr.data/demo.health_data/index
lockFactory=org.apache.lucene.store.NativeFSLockFactory@5c94e0dd
segmentsFile:segments_1
segmentsFileSizeInBytes:71
userData:{}
```

## dsetool infer\_solr\_schema

Automatically infers and proposes a schema that is based on the specified keyspace and table. Search indexes are not modified. Supports DSE authentication with [-l *username* -p *password*].

**Restriction:** Command is supported only on nodes with DSE Search workloads.

## Synopsis

```
dsetool infer_solr_schema keyspace_name.table_name
[coreOptions=yamlfilepath] | [coreOptionsInline=options]
```

**Table 139: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ <i>key:value</i> }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
< <i>datatype1,datatype2&gt;</i>	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
<i>cql_statement</i> ;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' < <i>schema</i> > ... </schema> '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.
@ <i>xml_entity='xml_entity_type'</i>	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

### **keyspace\_name.table\_name**

Required. The keyspace and table names of the search index. Keyspace and table names are case-sensitive. Enclose names that contain uppercase in double quotation marks.

### **coreOptions=yamlfilepath**

When auto-generation is on with generateResources=true, the file path to a customized YAML-formatted file of options. See [Changing auto-generated search index settings \(page 331\)](#).

#### **coreOptionsInline=key1:value1#key2:value2#...**

Use this key-value pair syntax `key1:value1#key2:value2#` to specify values for these settings:

- `auto_soft_commit_max_time:ms`
- `default_query_field:field`
- `distributed:( true | false )`
- `enable_string_copy_fields:( true | false )`
- `exclude_columns: col1, col2, col3, ...`
- `generate_DocValues_for_fields:( * | field1, field2, ... )`
- `generateResources:( true | false )`

See [Changing auto-generated search index settings \(page 331\)](#).

### **Examples**

To automatically infer and propose a schema that is based on the specified keyspace and table with the tuples and UDTs, specify the keyspace and table that contains tuples and UDTs:

```
dsetool infer_solr_schema demo.health_data_udt
```

## **dsetool inmemorystatus**

Provides the memory size, capacity, and percentage for this node and the amount of memory each table is using.

Provides the memory size, capacity, and percentage for this node and the amount of memory each table is using. The unit of measurement is MB. Bytes are truncated.

### **Synopsis**

```
dsetool inmemorystatus [keyspace_name.table_name]
```

**Table 140: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.

Syntax conventions	Description
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ <i>key:value</i> }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
< <i>datatype1,datatype2&gt;</i>	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
<i>cql_statement</i> ;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' < <i>schema</i> > ... </ <i>schema</i> > '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.
@ <i>xml_entity='xml_entity_type'</i>	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

**[keyspace\_name.table\_name]**

The keyspace name and table name.

**Examples**

To view the status for all tables:

```
dsetool inmemorystatus
```

The results for all tables are displayed:

Max Memory to Lock:	3276MB
Current Total Memory Locked:	0MB
Current Total Memory Not Able To Lock:	0MB
No MemoryOnlyStrategy tables found.	

To view the status for a specific table:

```
dsetool inmemorystatus demo.health_data
```

**dsetool list\_index\_files**

Lists all index files for a search index on the local node.

Lists all index files for a search index on the local node. The results show file name, encryption, disk usage, decrypted size, and encryption overhead. The index file is encrypted only when the backing CQL table is encrypted and the search index uses EncryptedFSDirectoryFactory; otherwise, the index file is decrypted.

**Restriction:** Command is supported only on nodes with DSE Search workloads.

## Synopsis

```
dsetool list_index_files keyspace_name table_name [--index directory]
```

**Table 141: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>Italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ <i>key:value</i> }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
< <i>datatype1,datatype2</i> >	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
<i>cql_statement</i> ;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' < <i>schema</i> > ... </ <i>schema</i> > '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.

Syntax conventions	Description
<code>@xml_entity='xml_entity_type'</code>	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

**keyspace\_name.table\_name**

Required. The keyspace and table names of the search index. Keyspace and table names are case-sensitive. Enclose names that contain uppercase in double quotation marks.

**--index**

The data directory that contains the index files.

- If not specified, the default directory is inferred from the search index name.
- *directory* - A specified file path to the solr.data directory that contains the search index files.

**Examples**

To list the index files:

```
dsetool list_index_files
```

The results show file name, encryption, disk usage, decrypted size, and encryption overhead:

Filename	Decrypted size	Encryption overhead	Disk usage
segments_1	N/A	N/A	7124 bytes
write.lock	N/A	N/A	3240 bytes

To list the index files in a specified directory:

```
dsetool list_index_files /My_data_dir
```

## dsetool list\_subranges

Lists the subranges of data in a keyspace to describe a token range by a number of smaller subranges.

Lists the subranges of data in a keyspace by dividing a token range into a number of smaller subranges. Useful when the specified range is contained in the target node's primary range.

**Synopsis**

```
dsetool list_subranges keyspace_name table_name
keys_per_range start_token end_token
```

**Table 142: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>Italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ <i>key:value</i> }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
< <i>datatype1,datatype2&gt;</i>	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
<i>cql_statement</i> ;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' < <i>schema</i> > ... </ <i>schema</i> > '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.
@ <i>xml_entity</i> =' <i>xml_entity_type</i> '	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

**keyspace\_name table\_name**

Keyspace table pair.

**keys\_per\_range**

The approximate number of rows per subrange.

**start\_token**

The start token of a specified range of tokens.

**end\_token**

The end token of a specified range of tokens.

## Example

To run the command:

```
dsetool list_subranges demo health_data 10000
113427455640312821154458202477256070485 0
```

The subranges are output and can be used as input to the [nodetool repair](#) command.

Start Token Estimated Size	End Token
113427455640312821154458202477256070485	
132425442795624521227151664615147681247	11264
132425442795624521227151664615147681247	
151409576048389227347257997936583470460	11136
151409576048389227347257997936583470460	0
	11264

## dsetool listjt

Lists all Job Tracker nodes grouped by the datacenter that is local to them.

Lists all Job Tracker nodes grouped by the datacenter that is local to them.

**Restriction:** Command is supported only on nodes with analytics workloads.

## Synopsis

```
dsetool listjt
```

**Table 143: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.

Syntax conventions	Description
{ <i>key:value</i> }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
< <i>datatype1,datatype2&gt;</i>	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
<i>cql_statement</i> ;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' < <i>schema</i> > ... </ <i>schema</i> > '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.
@ <i>xml_entity='xml_entity_type'</i>	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

This command takes no arguments.

## Examples

```
dsetool listjt
```

## dsetool managekmip list

Lists encryption/decryption keys on a KMIP server.

Verifies communication with the specified Key Management Interoperability Protocol (KMIP) server and lists the encryption/decryption keys on that server.

## Synopsis

```
dsetool managekmip list kmip_group_name [namespace=key_namespace]
```

**Table 144: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.

Syntax conventions	Description
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
'Literal string'	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ key:value }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
<datatype1,datatype2>	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
cql_statement;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' <schema> ... </schema> '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.
@xml_entity='xml_entity_type'	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

**kmip\_groupname**

The user-defined name of the KMIP group that is configured in the [kmip\\_hosts \(page 116\)](#) section of `dse.yaml`.

**namespace=key\_namespace**

Namespace on the specified KMIP provider.

**Examples**

Get a list of the available keys and states from the KMIP server:

```
dsetool managekmip list kmipgrouptwo
```

The results show that the KMIP server named vormetricgroup has two keys:

Keys on vormetricgroup:		Name	Cipher
ID	State	Activation Date	Creation Date
Protect Stop Date	Namespace		
02-449	82413ef3-4fa6-4d4d-9dc8-71370d731fe4_0		AES/CBC/PKCS5
Deactivated	Mon Apr 25 20:25:47 UTC 2016	n/a	n/a
	n/a	n/a	

02-540 0eb2277e-0acc-4adb-9241-1dd84dde691c\_0  
Active Tue May 31 12:57:59 UTC 2016

AES

n/a

## dsetool managekmip expirekey

Expires encryption/decryption keys on a KMIP server.

Expires encryption/decryption keys on a Key Management Interoperability Protocol (KMIP) server. Database stops using the key for encryption at the specified time and continues to use the expired key to decrypt existing data. Data re-keying is not required. Use this command to satisfy security policies that require periodically switching the encryption key.

DataStax recommends following best practices for key management permission policies. See [Expiring an encryption key](#).

### Synopsis

```
dsetool managekmip expirekey kmip_group_name kmip_key_id [date_time]
```

**Table 145: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>Italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ <i>key:value</i> }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
< <i>datatype1,datatype2&gt;</i>	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
<i>cql_statement</i> ;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.

Syntax conventions	Description
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' <schema> ... </schema> '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.
@xml_entity='xml_entity_type'	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

**kmip\_groupname**

The user-defined name of the KMIP group that is configured in the [kmip\\_hosts \(page 116\)](#) section of `dse.yaml`.

**kmip\_key\_id**

The key id on the KMIP provider.

**date\_time**

After the specified date\_time, new data will not be encrypted with the key. Data can be decrypted with the key after this expire date/time. Format of datetime is YYYY-MM-DD HH:MM:SS:T. For example, use 2016-04-13 20:05:00:0 to expire the encryption key at 8:05 p.m. on 13 April 2016.

**Examples**

To immediately expire an encryption key:

```
dsetool managekmip expirekey kmipgrouptwo 02-540
```

Encryption for new data is prevented, but decryption with the key is still allowed. Because the expire date/time is not specified, the key is expired immediately.

To expire an encryption key at a specific date and time:

```
dsetool managekmip expirekey kmipgrouptwo 02-540 2017-04-13 20:05:00:0
```

**[dsetool managekmip revoke](#)**

Permanently disables the key on the KMIP server.

Permanently disables the key on the KMIP server. Database can no longer use the key for encryption, but continues to use the key for decryption of existing data. Re-encrypt existing data before completely removing the key from the KMIP server. Use this command as the first step when replacing a compromised key.

**Synopsis**

```
dsetool managekmip revoke kmip_group_name kmip_key_id
```

**Table 146: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>Italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ <i>key:value</i> }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
< <i>datatype1,datatype2&gt;</i>	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
<i>cql_statement</i> ;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' < <i>schema</i> > ... </ <i>schema</i> > '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.
@ <i>xml_entity</i> =' <i>xml_entity_type</i> '	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

**kmip\_groupname**

The user-defined name of the KMIP group that is configured in the [kmip\\_hosts \(page 116\)](#) section of `dse.yaml`.

**kmip\_key\_id**

The key id on the KMIP provider.

**Examples**

To revoke a key to prevent decryption:

```
dsetool managekmip revoke kmipgrouptwo 02-540
```

## dsetool managekmip destroy

Completely removes the key from the KMIP server.

Completely removes the key from the KMIP server. Database can no longer use the key for encryption or decryption. Existing data that has not been re-encrypted becomes inaccessible.

**Important:** Use this command only after revoking a key and re-encrypting existing data.

### Synopsis

```
dsetool managekmip destroy kmip_group_name kmip_key_id
```

**Table 147: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>Italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ <i>key:value</i> }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
< <i>datatype1,datatype2</i> >	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
<i>cql_statement</i> ;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.

Syntax conventions	Description
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' <schema> ... </schema> '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.
@xml_entity='xml_entity_type'	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

### kmip\_groupname

The user-defined name of the KMIP group that is configured in the [kmip\\_hosts \(page 116\)](#) section of `dse.yaml`.

### kmip\_key\_id

The key id on the KMIP provider.

## Examples

To revoke a key to prevent decryption:

```
dsetool managekmip revoke kmipgrouptwo 02-540
```

After you revoke a key, you can destroy it:

```
dsetool managekmip destroy kmipgrouptwo 02-540
```

## dsetool node\_health

Retrieves a dynamic score between 0 and 1 that describes the health of a DataStax Enterprise node.

Retrieves a dynamic score between 0 and 1 that describes the health of a DataStax Enterprise node. Node health is a score-based representation of how fit a node is to handle search queries. The node health composite score is based on dropped mutations and uptime. A higher score indicates better node health. Nodes that have a large number of dropped mutations and nodes that are just started have a lower health score.

See [Collecting node health and indexing status scores](#).

## Synopsis

```
dsetool node_health [--all]
```

**Table 148: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>Italics</i>	Variable value. Replace with a valid option or user-defined value.

Syntax conventions	Description
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ <i>key:value</i> }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
< <i>datatype1,datatype2&gt;</i>	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
<i>cql_statement</i> ;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' < <i>schema</i> > ... </ <i>schema</i> > '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.
@ <i>xml_entity</i> =' <i>xml_entity_type</i> '	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

**--all**

Run the operation on all nodes.

**Examples**

To retrieve the health score of the local node:

```
dsetool node_health
```

The result displays a number between 0 and 1:

```
Node Health [0,1]: 0.7
```

To retrieve the health score of a specified node:

```
dsetool -h 200.192.10.11 node_health
```

To retrieve the health score of all nodes:

```
dsetool node_health --all
```

## dsetool partitioner

Returns the fully qualified classname of the IPartitioner that is in use by the cluster.

Returns the fully qualified classname of the IPartitioner that is used by the cluster.

### Synopsis

```
dsetool partitioner
```

**Table 149: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ <i>key:value</i> }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
< <i>datatype1,datatype2&gt;</i>	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
<i>cql_statement</i> ;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' < <i>schema</i> > ... </ <i>schema</i> > '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.

Syntax conventions	Description
<code>@xml_entity='xml_entity_type'</code>	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

This command takes no arguments.

## Examples

```
dsetool partitioner
```

The partitioner in use is displayed:

```
org.apache.cassandra.dht.Murmur3Partitioner
```

## dsetool perf

Temporarily changes the running parameters for the CQL Performance Service.

Temporarily changes the running parameters for the CQL Performance Service. Histogram tables provide DSE statistics that can be queried with CQL.

Changes made with performance object subcommands do not persist between restarts and are useful only for short-term diagnostics.

**Note:** To make these changes permanent, change the [CQL Performance Service options \(page 123\)](#) in `dse.yaml`.

See [DSE Performance Service diagnostic table reference](#) and [Collecting histogram diagnostics](#).

## Synopsis

```
dsetool perf subcommand values
```

**Table 150: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>Italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.

Syntax conventions	Description
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ <i>key:value</i> }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
< <i>datatype1,datatype2</i> >	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
<i>cql_statement</i> ;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' < <i>schema</i> > ... </ <i>schema</i> > '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.
@ <i>xml_entity</i> =' <i>xml_entity_type</i> '	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

**clustersummary enable|disable**

Whether to enable the collection of database-level statistics for the cluster.

**cqlslowlog enable|disable**

Whether to enable the collection of CQL queries that exceed the specified time threshold.

**cqlslowlog threshold**

The CQL slow log threshold as a percentile of the actual request times:

- [0,1] is a percentile threshold
- >1 is an absolute threshold in milliseconds
- 1.0 logs no queries
- 99.9 logs 0.1% of the slowest queries
- 95.0 logs 5% of the slowest queries
- 50.0 logs 50% of the slowest queries
- 0.0 logs all queries

**cqlslowlog skip\_writing\_to\_db**

Keeps slow queries in-memory only.

**cqlslowlog write\_to\_db**

Writes data to the database. When data writes to the database, the threshold must be >= 2000 ms to prevent a high load on database.

Temporary equivalent of `cql_slow_log_options.skip_writing_to_db: false` setting in [dse.yaml \(page 124\)](#).

**cqlslowlog set\_num\_slowest\_queries**

The number of slow queries to keep in-memory.

**cqlslowlog recent\_slowest\_queries**

The specified number of the most recent slow queries to retrieve.

**cqlsysteminfo enable|disable**

Whether to collect CQL system performance information statistics.

**dbsummary enable|disable**

Whether to collect database summary statistics.

**histograms enable|disable**

Whether to collect table histograms that measure the distribution of values in a stream of data. Histogram tables provide DSE statistics that can be queried with CQL. The data in the diagnostic histogram tables is cumulative since the DSE server was started.

**resourcelatencytracking enable|disable**

Whether to collect resource latency tracking statistics.

**solrcachestats enable|disable**

Whether to collect Solr cache statistics.

**solrindexingerrorlog enable|disable**

Whether to log Solr indexing errors.

**solrindexstats enable|disable**

Whether to collect Solr indexing statistics.

**solrlatencysnapshots enable|disable**

Whether to collect Solr latency snapshots.

**solrrequesthandlerstats enable|disable**

Whether to collect Solr request handler statistics.

**solrslowlog threshold enable|disable**

Whether to log the Solr slow sub-query log and set the Solr slow log threshold in milliseconds.

**solrupdatehandlerstats enable|disable**

Whether to collect Solr update handler statistics.

**userlatencytracking enable|disable**

Whether to enable user latency tracking.

## Examples

These example commands make temporarily changes only. Changes made with performance object subcommands do not persist between restarts and are useful only for short-term diagnostics.

See [Collecting database summary diagnostics](#).

To enable the collection of database-level statistics data:

```
dsetool perf clustersummary enable
```

To disable the collection of database-level statistics data:

```
dsetool perf clustersummary disable
```

See [Collecting slow queries](#).

To keep slow queries in-memory only:

```
dsetool perf cqslowlog skip_writing_to_db
```

To set the number of slow queries to keep in-memory:

```
dsetool perf cqslowlog set_num_slowest_queries 5
```

To write slow queries to the database:

```
dsetool perf cqslowlog write_to_db
```

To disable collecting information on slow queries:

```
dsetool perf cqslowlog disable
```

To change the threshold to collect information on 5% of the slowest queries:

```
dsetool perf cqslowlog 95.0
```

To enable collecting information to identify slow search queries:

```
dsetool perf solrslowlog enable
```

To change the threshold value (in milliseconds) at which a sub-query is slow enough to be reported:

```
dsetool perf solrslowlog 200
```

## [dsetool read\\_resource](#)

Reads the specified search index config or schema.

Reads the specified search index config or schema. Supports DSE authentication with [ -l *username* -p *password* ].

**Restriction:** Command is supported only on nodes with DSE Search workloads.

### Synopsis

```
dsetool read_resource keyspace_name.table_name name=res_filename
```

**Table 151: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>Italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.

Syntax conventions	Description
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ <i>key:value</i> }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
< <i>datatype1,datatype2&gt;</i>	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
<i>cql_statement</i> ;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' < <i>schema</i> > ... </ <i>schema</i> > '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.
@ <i>xml_entity='xml_entity_type'</i>	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

### **keyspace\_name.table\_name**

Required. The keyspace and table names of the search index. Keyspace and table names are case-sensitive. Enclose names that contain uppercase in double quotation marks.

### **res\_filename**

The name of the search index resource file to read.

### **Examples**

To read the resource:

```
dsetool read_resource demo.health_data stopwords.xml
```

After reading the resource, then upload the search index.

## **dsetool rebuild\_indexes**

Rebuilds specified secondary indexes for specified keyspace and table.

Rebuilds secondary indexes.

**Restriction:** Command is supported only on nodes with DSE Search workloads.

## Synopsis

```
dsetool rebuild_indexes keyspace_name.table_name [index1,index2,...]
```

**Table 152: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>Italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ <i>key:value</i> }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
< <i>datatype1,datatype2</i> >	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
<i>cql_statement</i> ;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' < <i>schema</i> > ... </schema> '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.
@ <i>xml_entity</i> =' <i>xml_entity_type</i> '	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

### **keyspace\_name.table\_name**

Required. The keyspace and table names of the search index. Keyspace and table names are case-sensitive. Enclose names that contain uppercase in double quotation marks.

**index1,index2,...**

Include one or a comma-separated list of secondary indexes to rebuild. If indexes are not specified, rebuilds all indexes.

**Examples**

To rebuild all secondary indexes:

```
dsetool rebuild_indexes demo.health_data
```

To rebuild only the specified secondary indexes:

```
dsetool rebuild_indexes demo.health_data index1, index2
```

**dsetool reload\_core**

Reloads the search index to recognize changes to schema or configuration.

Reloads the search index to recognize changes to schema or configuration. Supports DSE authentication with [-l *username* -p *password*].

**Note:** To reload the core and prevent reindexing, accept the default values `reindex=false` and `deleteAll=false`.

See [Reloading the search index](#) for details.

**Synopsis**

```
dsetool reload_core keyspace_name.table_name
[coreOptions=yamlFile | coreOptionsInline=key1:value1#key2:value2#...]
[deleteAll=true|false]
[distributed=true|false]
[reindex=(true|false]
[schema=path]
[solrconfig=path]
```

**Table 153: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>Italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.

Syntax conventions	Description
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ <i>key:value</i> }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
< <i>datatype1,datatype2</i> >	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
cql_statement ;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' < <i>schema</i> > ... </ <i>schema</i> > '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.
@ <i>xml_entity</i> =' <i>xml_entity_type</i> '	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

**keyspace\_name.table\_name**

Required. The keyspace and table names of the search index. Keyspace and table names are case-sensitive. Enclose names that contain uppercase in double quotation marks.

**schema=path**

Path of the UTF-8 encoded search index schema file. Cannot be specified when generateResources=true.

**solrconfig=path**

Path of the UTF-8 encoded search index configuration file. Cannot be specified when generateResources=true.

**distributed=( true | false )**

Whether to distribute and apply the operation to all nodes in the local datacenter.

- True applies the operation to all nodes in the local datacenter.
- False applies the operation only to the node it was sent to. False works only when recovery=true.

**Warning:** Distributing a re-index to an entire datacenter degrades performance severely in that datacenter.

**reindex=( true | false )**

Whether to reindex the data when search indexes are auto-generated with generateResources=true. Reindex works on a datacenter (DC) level. Reindex only once per search-enabled DC.

Valid values:

- true - Reindexes the data. Accepts reads and keeps the current search index while the new index is building.
- false - Default. Does not reindex the data. You can check and customize search index resources before indexing.

**deleteAll**

- true - deletes the already existing index before reindexing; search results will return either no or partial data while the index is rebuilding.
- false - does not delete the existing index, causing the reindex to happen in-place; search results will return partially incorrect results while the index is updating. Default.

During reindexing, a series of criteria routes sub-queries to the nodes most capable of handling them. See [Shard routing for distributed queries](#).

**Examples**

To make the pending search index active:

```
dsetool reload_core demo.health_data
coreOptionsInline="directory_factory_class:solr.EncryptedFSDirectoryFactory"
```

To upload the changed resource file:

```
dsetool reload_core demo.health_data
coreOptionsInline="directory_factory_class:solr.EncryptedFSDirectoryFactory"
```

**dsetool ring**

Lists the nodes in the ring.

Lists the nodes in the ring. For more readable output, use dsetool status.

**Synopsis**

```
dsetool ring
```

**Table 154: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>Italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.

Syntax conventions	Description
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ <i>key:value</i> }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
< <i>datatype1,datatype2&gt;</i>	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
<i>cql_statement</i> ;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' < <i>schema</i> > ... </ <i>schema</i> > '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.
@ <i>xml_entity</i> =' <i>xml_entity_type</i> '	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

This command requires no input.

## Examples

```
dsetool ring
```

## dsetool sparkmaster cleanup

Drops and recreates the Spark Master recovery table.

Drops and recreates the Spark Master recovery table.

## Synopsis

```
dsetool sparkmaster cleanup [datacenter]
```

**Table 155: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.

Syntax conventions	Description
<i>Italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ([ ]) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses (( )) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ <i>key:value</i> }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
< <i>datatype1,datatype2&gt;</i>	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
<i>cql_statement</i> ;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' < <i>schema</i> > ... </ <i>schema</i> > '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.
@ <i>xml_entity='xml_entity_type'</i>	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

This command has an optional argument *datacenter*. If a datacenter is specified, it will remove the recovery data for that datacenter.

## Examples

```
dsetool sparkmaster cleanup
```

```
dsetool sparkmaster cleanup dc1
```

## dsetool sparkworker restart

Manually restarts the Spark Worker on the selected node, without restarting the node.

Manually restarts the Spark Worker on the selected node, without restarting the node.

## Synopsis

```
dsetool sparkworker restart
```

**Table 156: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ <i>key:value</i> }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
< <i>datatype1,datatype2</i> >	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
<i>cql_statement</i> ;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' < <i>schema</i> > ... </schema> '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.
@ <i>xml_entity</i> =' <i>xml_entity_type</i> '	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

This command accepts no parameters.

## Examples

```
dsetool sparkworker restart
```

## dsetool status

Lists the nodes in their ring, including the node type and node health. When the datacenter workloads are the same type, the workload type is listed. When the datacenter workloads are heterogeneous, the workload type is shown as mixed.

## Synopsis

```
dsetool status
```

**Table 157: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>Italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ <i>key:value</i> }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
< <i>datatype1,datatype2</i> >	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
<i>cql_statement</i> ;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.

Syntax conventions	Description
' <schema> ... </schema> '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.
@xml_entity='xml_entity_type'	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

This command accepts no parameters.

## Examples

```
dsetool status
```

## dsetool stop\_core\_reindex

Stops reindexing for the specified search index on the node where the command is run. Optionally, specify a timeout in minutes so that the core waits to stop reindexing until the specified timeout is reached, then gracefully stops the indexing. The default timeout is 1 minute.

Stops reindexing for the specified search index on the node where the command is run. Optionally, specify a timeout in minutes so that the core waits to stop reindexing until the specified timeout is reached, then gracefully stops the indexing. The default timeout is 1 minute.

## Synopsis

```
dsetool stop_core_reindex keyspace_name.table_name [timeout_min]
```

**Table 158: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>Italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.

Syntax conventions	Description
{ <i>key:value</i> }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
< <i>datatype1,datatype2&gt;</i>	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
<i>cql_statement</i> ;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' < <i>schema</i> > ... </ <i>schema</i> > '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.
@ <i>xml_entity='xml_entity_type'</i>	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

**keyspace\_name.table\_name**

Required. The keyspace and table names of the search index. Keyspace and table names are case-sensitive. Enclose names that contain uppercase in double quotation marks.

**timeout\_min**

The number of minutes to wait to gracefully stop the indexing.

**Examples**

To stop reindexing after the default 1 minute timeout:

```
dsetool stop_core_reindex demo.health_data
```

To reindexing after 6 minutes:

```
dsetool stop_core_reindex demo.health_data 6
```

## dsetool tieredtablestats

Outputs tiered storage information, including SSTables, tiers, timestamps, and sizes. Provides information on every table that uses tiered storage.

Outputs tiered storage information, including SSTables, tiers, timestamps, and sizes. Provides information on every table that uses tiered storage.

**Synopsis**

```
dsetool tieredtablestats keyspace_name.table_name [-v]
```

**Table 159: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>Italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ <i>key:value</i> }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
< <i>datatype1,datatype2&gt;</i>	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
<i>cql_statement;</i>	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' < <i>schema</i> > ... </ <i>schema</i> > '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.
@ <i>xml_entity='xml_entity_type'</i>	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

**keyspace\_name.table\_name**

Required. The keyspace and table names of the search index. Keyspace and table names are case-sensitive. Enclose names that contain uppercase in double quotation marks.

**-v**

Output statistics for each SSTable, in addition to the tier summaries.

**Examples**

To monitor all tables using tiered storage:

```
dsetool tieredtablestats
```

Output of command:

```
ks.tbl
Tier 0:
Summary:
 max_data_age: 1449178580284
 max_timestamp: 1449168678515945
 min_timestamp: 1449168678515945
 reads_120_min: 5.2188117172945374E-5
 reads_15_min: 4.415612774014863E-7
 size: 4839
SSTables:
/mnt2/ks/tbl-257cecf1988311e58be1ff4e6f1f6740/ma-3-big-Data.db:
 estimated_keys: 256
 level: 0
 max_data_age: 1449178580284
 max_timestamp: 1449168678515945
 min_timestamp: 1449168678515945
 reads_120_min: 5.2188117172945374E-5
 reads_15_min: 4.415612774014863E-7
 rows: 1
 size: 4839
Tier 1:
Summary:
 max_data_age: 1449178580284
 max_timestamp: 1449168749912092
 min_timestamp: 1449168749912092
 reads_120_min: 0.0
 reads_15_min: 0.0
 size: 4839
SSTables:
/mnt3/ks/tbl-257cecf1988311e58be1ff4e6f1f6740/ma-4-big-Data.db:
 estimated_keys: 256
 level: 0
 max_data_age: 1449178580284
 max_timestamp: 1449168749912092
 min_timestamp: 1449168749912092
 reads_120_min: 0.0
 reads_15_min: 0.0
 rows: 1
 size: 4839
```

To monitor the `health_data` table using tiered storage:

```
dsetool tieredtablestats demo.health_data
```

To monitor the `health_data` table with output for each SSTable:

```
dsetool tieredtablestats demo.health_data -v
```

## dsetool tsreload

Reloads the truststores without a restart.

Reloads the truststores without a restart. Specify client or server.

### Synopsis

```
dsetool tsreload client|server
```

**Table 160: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>Italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ <i>key:value</i> }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
< <i>datatype1,datatype2&gt;</i>	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
<i>cql_statement</i> ;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' < <i>schema</i> > ... </ <i>schema</i> > '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.

Syntax conventions	Description
<code>@xml_entity='xml_entity_type'</code>	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

**client**

Reloads the truststore that is used for encrypted client-to-node communications.

**server**

Reloads the server truststore that is used for encrypted node-to-node (internode) SSL communications.

## dsetool unload\_core

Removes a search index.

Removes a search index. Supports DSE authentication with `[-l username -p password]`.

To drop a search index from a table and delete all related data for the entire cluster, see [DROP SEARCH INDEX](#).

The removal of the secondary index from the table schema is always distributed.

**Restriction:** Command is supported only on nodes with DSE Search workloads.

### Synopsis

```
dsetool unload_core keyspace_name.table_name
[deleteDataDir=(true|false)]
[deleteResources=(true|false)]
[distributed=(true|false)]
```

**Table 161: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.

Syntax conventions	Description
{ key:value }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
<datatype1,datatype2>	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
cql_statement;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' <schema> ... </schema> '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.
@xml_entity='xml_entity_type'	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

**deleteDataDir=( true | false )**

Whether to delete index data and any other artifacts in the `solr.data` directory.

Valid values:

- true - Deletes index data and any other artifacts in the `solr.data` directory. It does **not** delete DataStax Enterprise data.
- false - Default. Does not delete index data or other artifacts.

**deleteResources=( true | false )**

Whether to delete the config and schema resources associated with the search index.

Valid values:

- true - Deletes index resources.
- false - Default. Does not delete index resources.

**distributed=( true | false )**

Whether to distribute and apply the operation to all nodes in the local datacenter.

- True applies the operation to all nodes in the local datacenter.
- False applies the operation only to the node it was sent to. False works only when `recovery=true`.

Default: true

**Warning:** Distributing a re-index to an entire datacenter degrades performance severely in that datacenter.

## dsetool upgrade\_index\_files

Upgrades all DSE Search index files.

Upgrades all DSE Search index files.

Requirements:

- The remote node that contains the encryption configuration must be running.
- The local node is offline.
- The user that runs this command must have read and write permissions to the directory that contains the index files.

## Synopsis

```
dsetool upgrade_index_files keyspace_name.table_name
-h IP_address [-c port]
[--backup] [--workspace directory] [--index directory]
```

**Table 162: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>Italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ <i>key:value</i> }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
< <i>datatype1,datatype2&gt;</i>	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
<i>cql_statement</i> ;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' < <i>schema</i> > ... </ <i>schema</i> > '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.

Syntax conventions	Description
<code>@xml_entity='xml_entity_type'</code>	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

**keyspace\_name.table\_name**

Required. The keyspace and table names of the search index. Keyspace and table names are case-sensitive. Enclose names that contain uppercase in double quotation marks.

**-h IP\_address**

Required. Node hostname or IP address of the remote node that contains the encryption configuration that is used for index encryption. The remote node must be running.

**-c port**

The DSE port on the remote node that contains the encryption configuration.

**--backup**

Preserves the index files from the current index as a backup after successful upgrade. The preserved index file backup is moved to the --workspace directory. When not specified, index files from the current index are deleted.

**--workspace directory**

The workspace directory for the upgrade process. The upgraded index is created in this directory. When not specified, the default directory is the same directory that contains the search index files.

**--index directory**

The data directory that contains the search index files. When not specified, the default directory is inferred from the search index name.

**Examples**

To perform offline index encryption:

```
dsetool upgrade_index_files demo.health_data
```

See [Migrating encrypted tables from earlier versions](#) and [Encrypting new Search indexes](#).

## **dsetool write\_resource**

Uploads the specified search index config or schema.

Uploads the specified search index config or schema.

**Restriction:** Command is supported only on nodes with DSE Search workloads.

Resource files are stored internally in the database. You can configure the maximum resource file size or disable resource upload with the [resource\\_upload\\_limit \(page 121\)](#) option in `dse.yaml`.

Supports DSE authentication with `[-l username -p password]`.

## Synopsis

```
dsetool write_resource keyspace_name.table_name name=res_filename
 file=path_to_file_to_upload
```

**Table 163: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ <i>key:value</i> }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
< <i>datatype1,datatype2&gt;</i>	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
<i>cql_statement</i> ;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' < <i>schema</i> > ... </schema> '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.
@ <i>xml_entity='xml_entity_type'</i>	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

### **keyspace\_name.table\_name**

Required. The keyspace and table names of the search index. Keyspace and table names are case-sensitive. Enclose names that contain uppercase in double quotation marks.

### **res\_filename**

The name of the search index resource file to upload.  
**file**

The file path of the file to upload.

## Examples

To write the resource:

```
dsetool write_resource demo.health_data stopwords.xml
```

To specify the uploaded resource file and the path to the resource file:

```
dsetool write_resource demo.health_data name=ResourceFile.xml file=/myPath1/myPath2/schemaFile.xml
```

## DataStax Enterprise stress tools

Tools for stress testing DataStax Enterprise.

## Interpreting the output of cassandra-stress

About the output from the cassandra-stress running tests.

Each line reports data for the interval between the last elapsed time and current elapsed time.

```
Created keyspaces. Sleeping 1s for propagation.
Sleeping 2s...
Warming up WRITE with 50000 iterations...
Running WRITE with 200 threads for 1000000 iteration
 type total ops, op/s, pk/s, row/s, mean, med,
.95, .99, .999, max, time, stderr, errors, gc: #, max
 ms, sum ms, sdv ms, mb
 total, 43148, 42991, 42991, 42991, 4.6, 1.5,
10.9, 106.1, 239.3, 255.4, 1.0, 0.00000, 0, 1,
49, 49, 0, 612
 total, 98715, 43857, 43857, 43857, 4.6, 1.7,
 8.5, 98.6, 204.6, 264.5, 2.3, 0.00705, 0, 1,
45, 45, 0, 619
 total, 157777, 47283, 47283, 47283, 4.1, 1.4,
 8.3, 70.6, 251.7, 286.3, 3.5, 0.02393, 0, 1,
59, 59, 0, 611

Results:
op rate : 46751 [WRITE:46751]
partition rate : 46751 [WRITE:46751]
row rate : 46751 [WRITE:46751]
latency mean : 4.3 [WRITE:4.3]
latency median : 1.3 [WRITE:1.3]
latency 95th percentile : 7.2 [WRITE:7.2]
latency 99th percentile : 60.5 [WRITE:60.5]
latency 99.9th percentile : 223.2 [WRITE:223.2]
latency max : 503.1 [WRITE:503.1]
```

```

Total partitions : 1000000 [WRITE:1000000]
Total errors : 0 [WRITE:0]
total gc count : 18
total gc mb : 10742
total gc time (s) : 1
avg gc time(ms) : 73
stdev gc time(ms) : 16
Total operation time : 00:00:21

END

```

**Table 164: Output of cassandra-stress**

Data	Description
total ops	Running total number of operations during the run.
op/s	Number of operations per second performed during the run.
pk/s	Number of partition operations per second performed during the run.
row/s	Number of row operations per second performed during the run.
mean	Average latency in milliseconds for each operation during that run.
med	Median latency in milliseconds for each operation during that run.
.95	95% of the time the latency was less than the number displayed in the column.
.99	99% of the time the latency was less than the number displayed in the column.
.999	99.9% of the time the latency was less than the number displayed in the column.
max	Maximum latency in milliseconds.
time	Total operation time.
stderr	Standard error of the mean. It is a measure of confidence in the average throughput number; the smaller the number, the more accurate the measure of the cluster's performance.
gc: #	Number of garbage collections.
max ms	Longest garbage collection in milliseconds.
sum ms	Total of garbage collection in milliseconds.
sdv ms	Standard deviation in milliseconds.
mb	Size of the garbage collection in megabytes.

## fs-stress tool

The fs-stress tool performs stress testing of the DSE File System (DSEFS) layer.

## Synopsis

```
fs-stress [options] dsefs_directory listen_address
```

The default IP address is the [listen\\_address \(page 69\)](#) property in the `cassandra.yaml` file. If not using localhost, specify the correct IP address.

`fs-stress` is located in the `tools` directory of your installation.

The default location of the `tools` directory depends on the type of installation:

- Package installations: `/usr/share/dse/tools`
- Tarball installations: `installation_location/dse/tools`

**Table 165: Legend**

Syntax conventions	Description
UPPERCASE	Literal keyword.
Lowercase	Not literal.
<i>Italics</i>	Variable value. Replace with a valid option or user-defined value.
[ ]	Optional. Square brackets ( [ ] ) surround optional command arguments. Do not type the square brackets.
( )	Group. Parentheses ( ( ) ) identify a group to choose from. Do not type the parentheses.
	Or. A vertical bar (   ) separates alternative elements. Type any one of the elements. Do not type the vertical bar.
...	Repeatable. An ellipsis ( . . . ) indicates that you can repeat the syntax element as often as required.
' <i>Literal string</i> '	Single quotation ( ' ) marks must surround literal strings in CQL statements. Use single quotation marks to preserve upper case.
{ <i>key:value</i> }	Map collection. Braces ( { } ) enclose map collections or key value pairs. A colon separates the key and the value.
< <i>datatype1,datatype2</i> >	Set, list, map, or tuple. Angle brackets ( < > ) enclose data types in a set, list, map, or tuple. Separate the data types with a comma.
<i>cql_statement</i> ;	End CQL statement. A semicolon ( ; ) terminates all CQL statements.
[ -- ]	Separate the command line options from the command arguments with two hyphens ( -- ). This syntax is useful when arguments might be mistaken for command line options.
' < <i>schema</i> > ... </ <i>schema</i> > '	Search CQL only: Single quotation marks ( ' ) surround an entire XML schema declaration.

Syntax conventions	Description
<code>@xml_entity='xml_entity_type'</code>	Search CQL only: Identify the entity and literal value to overwrite the XML element in the schema and solrconfig files.

## Description

The `fs-stress` tool performs stress testing of the DSE File System (DSEFS) layer.

Data	Description
progress	Total progress of the stress operation.
bytes	Total bytes written/read.
curr rate	Current rate of bytes being written/read per second.
avg rate	Average rate of bytes being written/read per second.
max latency	Maximum latency in milliseconds during the current reporting window.

# Operations

DataStax Enterprise operation topics, such as node and datacenter operations, changing replication strategies, configuring compaction and compression, caching, and tuning Bloom filters.

## Starting and stopping DataStax Enterprise

You can start and stop DataStax Enterprise as a service or stand-alone process.

After you install and configure DataStax Enterprise on one or more nodes, start your cluster beginning with the seed nodes. In a mixed-workload DataStax Enterprise cluster, you must start the analytics seed node first.

Packaged installations include start-up and stop scripts for running DataStax Enterprise as a service. Binary tarballs do not.

OpsCenter provides options in the Nodes UI for starting, stopping, and restarting DSE on a node. See [Starting DSE on a node](#), [Stopping DSE on a node](#), and [Restarting DSE on a node](#).

### Starting DataStax Enterprise as a service

Steps for starting the DataStax Enterprise service when DataStax Enterprise was installed from a RHEL or Debian package.

Steps for starting the DataStax Enterprise (DSE) service when DataStax Enterprise was installed from RHEL or Debian packages.

All nodes types are DataStax Enterprise nodes and run the database.

### Considerations for starting a cluster

Be aware of the following when starting a DataStax Enterprise cluster:

#### Nodes must be segregated by datacenters

Transactional, DSE Search, DSE Analytics, and [SearchAnalytics \(page 172\)](#) nodes must be in separate datacenters. For example, in a cluster with both DSE Search and transactional nodes, all DSE Search nodes must be in a one or more search datacenters and all transactional nodes must be in one or more datacenters.

DSE Graph can be enabled on any node in any datacenter.

#### Deploying a mixed-workload cluster

When deploying one or more datacenters for each type of node, first determine which nodes to start as transactional, analytic, DSE Graph only, DSE Graph plus other types, DSE Search, and SearchAnalytics nodes. Deploy in this order:

1. Analytic seed nodes.
2. Transactional or DSE Graph-only seed nodes.
3. DSE Search seed nodes.
4. SearchAnalytics nodes.

5. Remaining nodes one at a time. See [Initializing multiple datacenters per workload type](#).

### DSE Analytics nodes

Before starting DSE Analytics nodes, ensure that the [replication factor \(page 171\)](#) is configured correctly for the analytics keyspaces. Every time you add a new datacenter, you **must** manually increase the replication factor of the `dse_leases` keyspace for the new DSE Analytics datacenter.

### Start up commands

Set the type of node in the `/etc/default/dse` file. (Start-up scripts are also available in `/etc/init.d.`)

Command	Description
<code>GRAPH_ENABLED=1</code>	Starts the node as a DSE Graph node.
<code>SPARK_ENABLED=1</code>	Starts the node as a Spark node and starts the Spark Master service.
<code>SOLR_ENABLED=1</code>	Starts the node as a DSE Search node.
Transactional-only or BYOS nodes	<code>NODE_TYPES=0</code> or not present.

**Table 166: Examples**

Node type	Settings
Spark Analytics node	<pre>SPARK_ENABLED=1 SOLR_ENABLED=0 GRAPH_ENABLED=0</pre> <p>or</p> <pre>SPARK_ENABLED=1</pre> <p><b>Note:</b> No entry is the same as disabling it.</p>
Spark Analytics, DSE Graph, and DSE Search node	<pre>SPARK_ENABLED=1 GRAPH_ENABLED=1 SOLR_ENABLED=1</pre>
BYOS (Bring Your Own Spark) Spark nodes run in separate Spark cluster from a vendor other than DataStax.	Set BYOS nodes as transactional nodes: <code>ALL_NODE_TYPES=0</code> or not present.
DSE Graph and BYOS	<pre>GRAPH_ENABLED=1</pre>

Node type	Settings
<p><a href="#">SearchAnalytics (page 172)</a> nodes An integrated DSE SearchAnalytics cluster allows analytics jobs to be performed using CQL queries.</p>	<pre>SPARK_ENABLED=1 SOLR_ENABLED=1</pre>

**Prerequisites:** Be sure to read the [Considerations for starting a cluster \(page 867\)](#).

You can also use [OpsCenter](#) to start and stop nodes.

1. If DataStax Enterprise is running, [stop the node \(page 872\)](#).
2. Set the node type in the `/etc/default/dse` file. For example, to a Spark node:

```
SPARK_ENABLED=1
SOLR_ENABLED=0
GRAPH_ENABLED=0
```

**Note:** Alternately, you can omit the other start up entries and just use `SPARK_ENABLED=1`.

3. Start DataStax Enterprise:

```
sudo service dse start
```

If the following error appears, see [DataStax Enterprise times out when starting](#).

```
WARNING: Timed out while waiting for DSE to start.
```

4. To verify that the cluster is running:

```
nodetool status
```

**Note:** If DSE has problems starting, see [Troubleshooting starting and installing DSE](#).

The nodetool command shows the node type and the status. For a transactional node running in a normal state (UN) with virtual nodes (vnodes) enabled shows:

```
Datacenter: Cassandra
=====
Status=Up/Down
| / State=Normal/Leaving/Joining/Moving
-- Address Load Tokens Owns Host ID
 Rack
UN 127.0.0.1 82.43 KB 128 ?
40725dc8-7843-43ae-9c98-7c532b1f517e rack1
```

For example, a running node in a normal state (UN) with DSE Analytics without vnodes enabled shows:

```
Datacenter: Analytics
```

```
=====
Status=Up/Down
| / State=Normal/Leaving/Joining/Moving
-- Address Load Owns Host ID
 Token
UN 172.16.222.136 103.24 KB ? 3c1d0657-0990-4f78-
a3c0-3e0c37fc3a06 1647352612226902707 rack1
```

## Starting DataStax Enterprise as a stand-alone process

Steps for starting the DataStax Enterprise process when DataStax Enterprise was installed from a tarball.

Steps for starting the DataStax Enterprise (DSE) process when DataStax Enterprise was installed from a tarball.

All nodes types are DataStax Enterprise nodes and run the database.

### Considerations for starting a cluster

Be aware of the following when starting a DataStax Enterprise cluster:

#### Nodes must be segregated by datacenters

Transactional, DSE Search, DSE Analytics, and [SearchAnalytics \(page 172\)](#) nodes must be in separate datacenters. For example, in a cluster with both DSE Search and transactional nodes, all DSE Search nodes must be in a one or more search datacenters and all transactional nodes must be in one or more datacenters.

DSE Graph can be enabled on any node in any datacenter.

#### Deploying a mixed-workload cluster

When deploying one or more datacenters for each type of node, first determine which nodes to start as transactional, analytic, DSE Graph only, DSE Graph plus other types, DSE Search, and SearchAnalytics nodes. Deploy in this order:

1. Analytic seed nodes.
2. Transactional or DSE Graph-only seed nodes.
3. DSE Search seed nodes.
4. SearchAnalytics nodes.
5. Remaining nodes one at a time. See [Initializing multiple datacenters per workload type](#).

#### DSE Analytics nodes

Before starting DSE Analytics nodes, ensure that the [replication factor \(page 171\)](#) is configured correctly for the analytics keyspaces. Every time you add a new datacenter, you **must** manually increase the replication factor of the `dse_leases` keyspace for the new DSE Analytics datacenter.

#### Start up commands

1. Start the node from the `installation_location`.
2. Set the type.

Node/datacenter	Command
Transactional only	bin/dse cassandra
DSE Graph	bin/dse cassandra -g
DSE Analytics with Spark	bin/dse cassandra -k
DSE Search	bin/dse cassandra -s

**Note:** When multiple flags are used, list them separately on the command line. For example, ensure there is a space between `-k` and `-s` in `dse cassandra -k -s`.

**Table 167: Starting examples**

Node type	Settings
From the <code>installation_location</code> :	
Spark Analytics, DSE Graph, and DSE Search node	bin/dse cassandra -k -g -s
BYOS (Bring Your Own Spark) Spark nodes run in separate Spark cluster from a vendor other than DataStax.	bin/dse cassandra
DSE Graph and BYOS	bin/dse cassandra -g
SearchAnalytics (page 172) nodes An integrated DSE SearchAnalytics datacenter allows analytics jobs to be performed using CQL queries.	bin/dse cassandra -k -s

**Prerequisites:** Be sure to read the [Considerations for starting a cluster \(page 870\)](#).

You can also use [OpsCenter](#) to start and stop nodes.

1. If DataStax Enterprise is running, [stop the node \(page 872\)](#).
2. From the install directory, start the node. For example, to set a Spark node:

```
bin/dse cassandra -k
```

3. To check that your ring is up and running:

```
cd installation_location &&
bin/nodetool status
```

where the installation directory is either:

- `/usr/share/dse`
- DataStax Enterprise installation directory

**Note:** If DSE has problems starting, see [Troubleshooting starting and installing DSE](#).

The nodetool command shows the node type and the status. For a transactional node running in a normal state (UN) with virtual nodes (vnodes) enabled shows:

```
Datacenter: Cassandra
=====
Status=Up/Down
| / State=Normal/Leaving/Joining/Moving
-- Address Load Tokens Owns Host ID
 Rack
UN 127.0.0.1 82.43 KB 128 ??
40725dc8-7843-43ae-9c98-7c532b1f517e rack1
```

For example, a running node in a normal state (UN) with DSE Analytics without vnodes enabled shows:

```
Datacenter: Analytics
=====
Status=Up/Down
| / State=Normal/Leaving/Joining/Moving
-- Address Load Owns Host ID
 Token Rack
UN 172.16.222.136 103.24 KB ?? 3c1d0657-0990-4f78-
a3c0-3e0c37fc3a06 1647352612226902707 rack1
```

## Stopping a DataStax Enterprise node

Stopping DataStax Enterprise on a node.

To speed up the restart process, before stopping the dse service, run `nodetool drain`. This step writes the current memtables to disk. When you restart the node, the commit log is not read which speeds the restart process. If you have durable writes set to false, which is unlikely, there is no commit log and you must drain the node to prevent losing data.

To stop DataStax Enterprise running as a service:

```
nodetool drain &&
sudo service dse stop
```

To stop DataStax Enterprise running as a stand-alone process:

Running `nodetool drain` before using the `cassandra-stop` command to stop a stand-alone process is not necessary because the `cassandra-stop` command drains the node before stopping it.

From the installation location:

```
bin/dse cassandra-stop
```

**Note:** Use sudo if required.

In the unlikely event that the `cassandra-stop` command fails because it cannot find the process DataStax Enterprise Java process ID (PID), the output instructs you to find the DataStax Enterprise Java process ID (PID) manually, and stop the process using its PID number.

```
ps auwx | grep dse
```

Use the PID, in the second column of the output, to stop the database.

```
bin/dse cassandra-stop -p PID
```

**Note:** If you have trouble, see [Troubleshooting starting DataStax Enterprise](#).

## Clearing the data from DataStax Enterprise

Remove all data from any type of installation.

Remove all data from any type of installation.

### Package installation

To clear the data from the **default** directories:

1. After [Stop \(page 872\)](#) the service.
2. Run one of the following commands:

```
sudo rm -rf /var/lib/cassandra/* ## Remove all data
```

```
sudo rm -rf /var/lib/cassandra/data/* ## Remove only the data
directories
```

### Tarball installation

To clear all data from the **default** directories:

1. [Stop \(page 867\)](#) the DataStax Enterprise process.
2. Remove the data from the installation location:

```
cd installation_location
```

Run one of the following commands:

```
sudo rm -rf data/* commitlog/* saved_caches/* hints/* ## Remove all
data
```

```
sudo rm -rf data/* ## Remove only the data directories
```

# DataStax Studio

An IDE for CQL (Cassandra Query Language) and DSE Graph.

## About DataStax Studio

DataStax Studio is an interactive developer tool for CQL (Cassandra Query Language), Spark SQL, and DSE Graph.

DataStax Studio is an interactive developer tool for CQL (Cassandra Query Language), Spark SQL, and DSE Graph. Developers and analysts collaborate by mixing code, documentation, query results, and visualizations in self-documenting notebooks.

- For [CQL \(page 886\)](#), use DataStax Studio to visually create and navigate database objects, create complex queries, and tune CQL queries. The schema helps you visualize the keyspaces in a tree-like view.
- For DSE Graph, use DataStax Studio to explore and view large datasets. The code for DSE Graph is written in the [Gremlin graph traversal language](#) and is executed by the Gremlin Server that is a part of the DSE Graph component. The [graph schema view \(page 880\)](#) helps you visualize the graph organization and connections.
- For DSE Analytics, write, test, and run [Spark SQL queries \(page 887\)](#) against DSE clusters.
- The intelligent code editor provides syntax highlighting, validation, intelligent code completion, configuration options, and query profiling.
- Self-documenting [notebooks \(page 875\)](#) mix runnable code, documentation (markdown text), and visualizations for query results. This rich interactive environment provides an intuitive interface for developers and analysts to collaborate and test theories. [Markdown](#) is a simple language for creating human-readable plain text.

See:

- [Installing DataStax Studio](#)
- [Upgrading DSE Studio](#)

## DataStax Studio security

Security of DataStax Studio.

DataStax Studio does not provide a multi-user experience. If deployed in a central fashion, grant only to users who need to see and edit the notebooks and the database connections that are associated with that DataStax Studio instance.

Access to DataStax Studio is not protected by built-in authentication. Exposing direct access to DataStax Studio outside of a host-local network interface is strongly discouraged.

**Important:** Changing the httpBindAddress setting from the default (localhost) can pose a security risk as users on external machines can gain access to notebooks and

the DSE clusters those notebooks are connected to. Studio is designed to be used as a desktop application. Distributed deployment introduces potential security risks.

## About DataStax Studio notebooks

Information about DataStax Studio self-documenting notebooks.

Notebooks provide an intuitive interface for developers and analysts to collaborate by mixing code, documentation, and visualizations for query results. A notebook consists of one or more cells.

Notebooks combine a web browser user interface with the DataStax Enterprise scalable database. Notebooks provide a sharable interface to access data and enable collaboration, sharing, and explanation.

Notebooks contain the inputs and outputs of an interactive session and descriptive text that accompanies the code.

DataStax Studio 6.0 provides these notebooks:

### What's New in Studio v6.0.0

Overview of new features: notebook export and import, interactive graph with neighborhood expansion, notebook revision history, and Spark SQL to write and submit Spark SQL queries against DSE Analytics.

### Working with CQL

Tutorial to create and interact with data in a DSE cluster by writing and executing CQL code in a notebook. Learn about viewing the CQL schema with DESCRIBE commands. Learn to use content assist, syntax validation, and domain validation. View results in table view or different styles of charts. See also [CQL](#) documentation.

### Working with Graph v6.0.0

Introduces the DSE Graph-centric features of Studio and provides a basic introduction to vertices sizes, edge colors, and using [interactive graph \(page 880\)](#) features. Learn how to:

- Use display name templates to improve the display names shown in graph view
- Dynamically view results in multiple views such as JSON, table, charts, and graph
- Examine a subset of a graph based on an edge label, a set of edge labels, or a set of edges
- Select vertices and edges, learn about vertices sizes and colors
- Use full screen
- Hide the code editor
- Use content assist and schema-aware content assist for Gremlin code
- Use schema view to see what is connected and how

Learn about charting to visualize data results. Learn how to view help for efficient [keyboard shortcuts \(page 904\)](#).

### Working with Spark SQL v6.0.0

Quick tips and brief tutorial for working with Spark SQL in Studio. Learn to use content assist while writing a Spark SQL statement. Use schema view for a tree

view of schema elements in a database. If you are new to DSE Analytics, see [DSE Analytics \(page 170\)](#).

### DSE Graph QuickStart v6.0.0

Introduction tutorial for using Studio to learn about DSE Graph. The tutorial provides steps to insert data and run traversals. Also, see the [DSE Graph QuickStart \(page 368\)](#) documentation.

## Using DataStax Studio

Information about starting and using Studio.

### Starting and stopping DataStax Studio

Starting and stopping DataStax Studio.

Steps to start and stop Studio.

1. To start Studio, run the Studio Server shell script:

- Linux:

```
cd installation_location/datastax-studio-6.0.0
./bin/server.sh
```

**Tip:** To start Studio in the background, add & at the end of the command:

```
./bin/server.sh &
```

- Windows:

```
C:/> installation_location\datastax-studio-6.0.0\bin\
C:/> server.bat
```

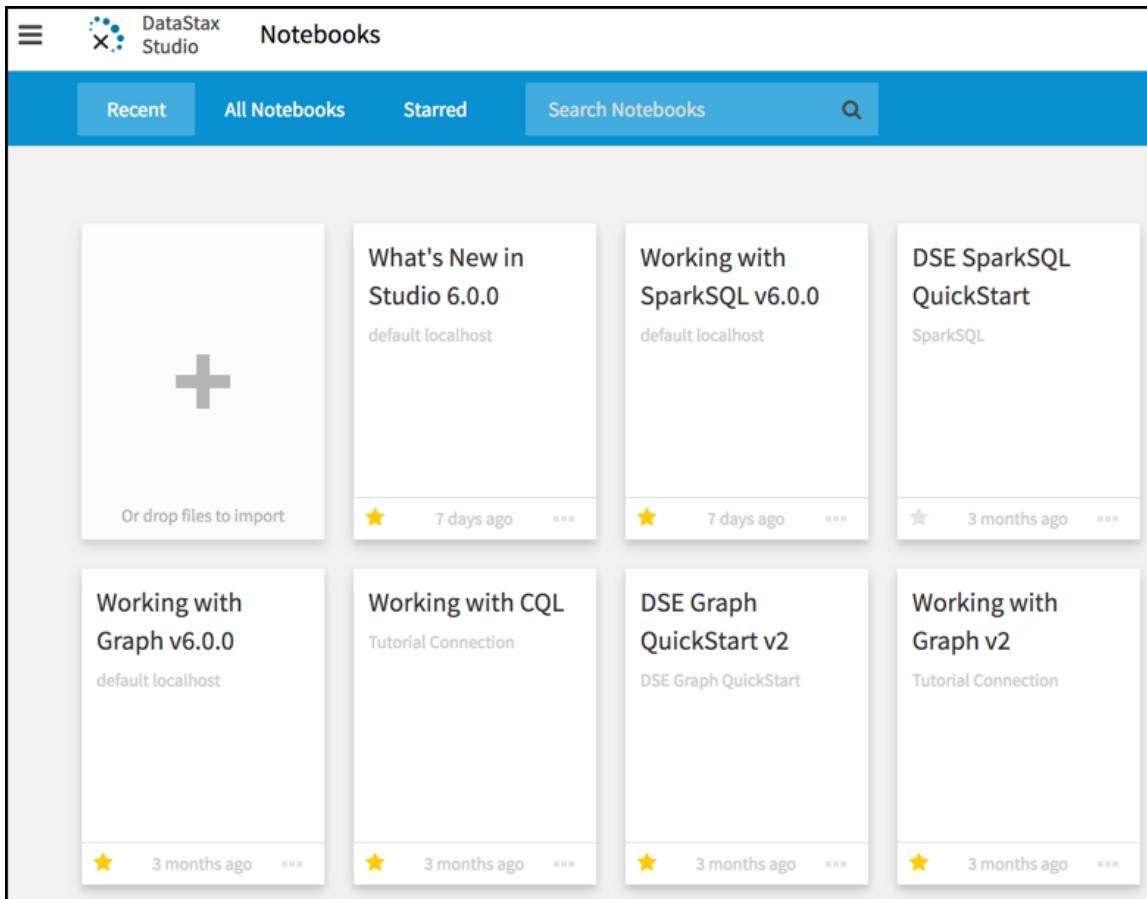
Your result will look similar to:

```
Studio is now running at: http://127.0.0.1:9091
```

2. To run DataStax Studio, enter this URL in your web browser:

`http://URI_running_DSE:9091/`

- For DSE running on localhost, `URI_running_DSE` is localhost.
- For DSE on another machine, `URI_running_DSE` is the hostname or IP address for the remote machine.



### 3. To stop Studio:

```
pkill -f studio
```

## Creating a connection in DataStax Studio

Creating a connection from a Studio notebook to a DSE cluster.

A connection from each notebook to a DSE cluster is required. A notebook persists its data to a DSE cluster.

Each notebook has only one connection. A connection can serve multiple notebooks.

**Tip:** You can configure more than one host in a Studio connection. Hosts initialize the Cassandra driver connection, so more than one host provides redundancy and failover protection.

**Prerequisites:** The DSE cluster must be installed and running.

To create a connection from a notebook to a DSE cluster:

1. Browse to the URL for the Studio installation:

```
http://URI_running_DSE:9091/
```

2. In the menu (#), select **Connections**.

The existing connections are displayed in a list. The list is sorted in alphabetical order and shows the port and host for each connection.

3. Click **+** in the top center of the page.

4. In the **Create Connection** dialog, enter the connection information:

**Name**

Name of the connection from the notebook to a DSE cluster.

**Host/IP (comma delimited)**

The host names or IP addresses of the DSE cluster to connect to. Default: localhost.

**Username**

Optional. DSE username for logging in.

**Password**

Optional. DSE password for logging in.

**Port**

IP connection port. Default: 9042.

For example, to connect to a single-node DSE cluster on the local host using the default port.

**Name**

My First Connection

**Host/IP**

127.0.0.1

**Port**

9042

5. Click **Test** to verify the connection works.

6. To configure a secure encrypted connection, select the **Use SSL** check box.

The Truststore and Keystore fields display. See [Using SSL connections in DataStax Studio \(page 878\)](#).

7. Click **Save**.

A connection is created from the notebook to the DSE cluster.

## Using SSL connections in DataStax Studio

Steps to connect Studio to an SSL-enabled DataStax Enterprise cluster.

**Prerequisites:**

SSL must be configured and working on your DSE cluster. See [Configuring SSL](#) for details on:

- Client-to-node encryption
- Node-to-node encryption
- Preparing server certificates

DataStax recommends using certificates signed by a CA. The truststore or CA certificate can be shared between all DSE servers and clients. See [Setting up SSL certificates](#).

For DataStax Studio, the public key certificate from the CA must be stored in a local truststore file.

To perform server verification, the client needs to have the public key certificate of each node in the cluster stored in a local truststore file. This truststore file is password protected.

1. In the menu (#), select **Connections**.
2. Click **+** to add a connection or click any of the listed connections to edit an existing connection.
3. In the **Create Connection** dialog, enter the connection information:

**Name**

Name of the connection from the notebook to a DSE cluster.

**Host/IP (comma delimited)**

The host names or IP addresses of the DSE cluster to connect to. Default: localhost.

**Username**

Optional. DSE username for logging in.

**Password**

Optional. DSE password for logging in.

**Port**

IP connection port. Default: 9042.

For example, to connect to a single-node DSE cluster on the local host using the default port.

**Name**

My First Connection

**Host/IP**

127.0.0.1

**Port**

9042

4. Select the **Use SSL** check box to show the Truststore and Keystore fields.
5. As appropriate for your environment, enter the local file path and password for the truststore.
6. Click **Test** to verify the connection to the DSE cluster.

## Using graph in DataStax Studio

Interactively create and explore contextual views of data.

Graph databases are useful for discovering simple and complex relationships between objects. Relationships are fundamental to how objects interact with one another and their environment. Graph databases perfectly represent the relationships between objects.

Using DSE Graph in DataStax Studio provides an interactive way to:

- Compose Gremlin traversals and view results in multiple dynamic views such as JSON, table, charts, and graph
- Leverage Studio's schema view and schema-aware content-assist for rapid Gremlin Traversal development

- Manipulate the graph view with interactive graph
  - # Expand vertex neighborhoods
  - # Remove vertices and edges from view

For an interactive introduction to working with DSE Graph in Studio, see the DSE Graph notebook tutorials that are installed with Studio:

### Working with Graph v6.0.0

Introduces the DSE Graph-centric features of Studio and provides a basic introduction to vertices sizes, edge colors, and using [interactive graph \(page 880\)](#) features. Learn how to:

- Use display name templates to improve the display names shown in graph view
- Dynamically view results in multiple views such as JSON, table, charts, and graph
- Examine a subset of a graph based on an edge label, a set of edge labels, or a set of edges
- Select vertices and edges, learn about vertices sizes and colors
- Use full screen
- Hide the code editor
- Use content assist and schema-aware content assist for Gremlin code
- Use schema view to see what is connected and how

Learn about charting to visualize data results. Learn how to view help for efficient [keyboard shortcuts \(page 904\)](#).

### DSE Graph QuickStart v6.0.0

Introduction tutorial for using Studio to learn about DSE Graph. The tutorial provides steps to insert data and run traversals. Also, see the [DSE Graph QuickStart \(page 368\)](#) documentation.

The Studio graph view provides a contextual view of a result by automatically populating the entire sub-graph shared between the result's vertices and edges. The Gremlin result is examined:

- If a result contains vertices, then all edges linking those vertices are automatically populated.
- If a result contains edges, then the adjacent vertices are automatically populated.
- If the result is a subgraph, then it is displayed as-is.

Path results can contain arbitrary Java objects (not just vertices and edges), so path results are not rendered by the graph view.

## Using interactive graph in DataStax Studio graph view

Using interactive graph to expand vertex neighborhoods and remove vertices and edges from the graph view.

Studio provides these interactive graph features:

- Expand vertex neighborhoods, with the ability to filter by edge label

- Remove vertices and edges from the graph view

**Tip:** Right-click vertices and edges to activate these features.

In a gremlin cell, you can interactively expand the neighborhood in graph view.

**Tip:** At any time, double-click the graph view to recenter the display.

Edge population in neighborhood expansion happens in two stages:

1. Adjacent edges and their corresponding other vertices are retrieved.
2. Edges between new vertices and existing vertices may be automatically populated based on the selected Edge Population Strategy.

Neighborhood expansion has configurable view options:

- Populate all common edges, the default complete view.
- Populate only the edges of the selected labels to expand.

All edges that connect a newly retrieved vertex and an existing vertex are added to the view only if their EdgeLabel is selected.

- Do not auto-populate common edges.

Open and use the Working with Graph notebook to learn more about neighborhood expansion.

## Notebook code editor in Gremlin code cells

The Notebook code editor supports the Groovy programming language in Gremlin code cells.

The Notebook code editor supports the [Apache Groovy™](#) programming language in Gremlin code cells. Syntax validation supported for a subset of Groovy.

A notebook connects to a DSE cluster based on the connection information. Each notebook has only one connection. One piece of information that a connection specifies is its **Graph Name**.

The code editor provides pre-defined alias variables:

**graph**

The variable that refers to the graph.

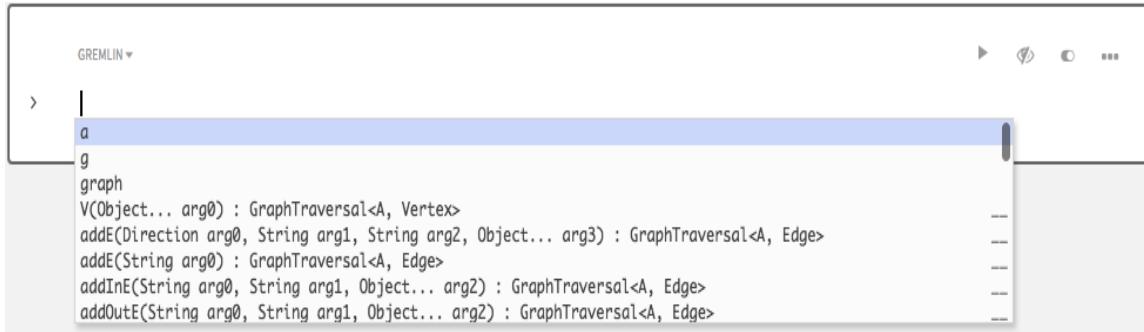
**g**

The traversal source associated with the graph for OLTP graph traversals.

**a**

The traversal source associated with the graph for OLAP graph traversals.

A drop-down list shows for [content-assist \(page 882\)](#) on a blank line in the editor.



```
GREMLIN > a
g
graph
V(Object... arg0) : GraphTraversal<A, Vertex>
addE(Direction arg0, String arg1, String arg2, Object... arg3) : GraphTraversal<A, Edge>
addE(String arg0) : GraphTraversal<A, Edge>
addInE(String arg0, String arg1, Object... arg2) : GraphTraversal<A, Edge>
addOutE(String arg0, String arg1, Object... arg2) : GraphTraversal<A, Edge>
```

The code editor supports inline code validation. If some code is invalid or uses Groovy syntax that the editor does not support, the invalid code is displayed with a red underlining.

Use the slider (⌚) to Disable editor validations to turn off.

### Cancel execution

When code is running in a Gremlin cell, you can cancel the execution to kill the gremlin session. This cancel execution is helpful when you want to make a change to the code and do not want to wait until a long running query finishes.

- Click Kill Session.

The confirmation dialog appears:

```
Gremlin cell executions will be stopped by killing the notebook session. Proceed?
```

- Click Yes to proceed, no to cancel.
- After you cancel execution, this message is shown:

```
The Gremlin Session associated with the notebook was killed.
```

- You can make code changes to the cell and run the execution again.

### Content assistance

The code editor provides content assistance (**Ctrl+Space**).

The screenshot shows the DataStax Studio interface with a Gremlin editor tab. In the code editor, the prefix 'g.' is typed, and a dropdown menu appears, listing various Gremlin methods such as E(), V(), addV(), and GraphTraversalSource methods like asBuilder() and getGraph(). The 'GraphTraversalSource' item is highlighted. Below the editor, a message says 'Displaying 1-1 of 1' and '1 ELEMENT RETURNED'.

Content-assistance proposes content based on context:

- Methods
- Variables

Press **Return** to select the highlighted choice from the assistance list.

You can filter proposals by typing enough characters to select a line in the pull-down menu.

Use Shift-Tab to return to the beginning of a line.

## Code validation

The notebook editor supports these validations:

- Groovy syntax—The code in a Gremlin cell is executed within DSE Graph as Groovy. The notebook editor adds enough Groovy syntax support to help you craft Gremlin statements.

The screenshot shows a Gremlin cell in the notebook editor with the following code: 'g.V().has('author', 'name', 'Julia'. A tooltip 'String literal is not properly closed' appears over the closing quote 'Julia'. The cell has a blue vertical bar on its left side.

- Type-checking—The code in a cell is also checked for type. If you try to call a method on an object of the wrong type or pass a parameter of an invalid type, a validation error is displayed.

The screenshot shows two separate Gremlin code editor cells. The top cell contains the code `g.V().hasPropertyKey()`, which is highlighted in red. A tooltip box appears over the code, stating: "The method hasPropertyKey() is undefined for the type GraphTraversal<Vertex, Vertex>". The bottom cell contains the code `g.V().has(123, 'x', 'y')`, where the number 123 is also highlighted in red. A tooltip box appears over the code, stating: "Type mismatch: cannot convert from int to String". Both cells have a blue vertical bar on the left and a plus sign icon at the top right.

- Domain-specific—Common errors in code are pointed out when possible.

The screenshot shows a single Gremlin code editor cell containing the code `g.V()`. The word `g` is highlighted in yellow. A tooltip box appears over the code, stating: "Traversal has not been iterated; call next(), iterate() or forEachRemaining()". The cell has a blue vertical bar on the left and a plus sign icon at the top right.

The notebook editor performs validation as code is entered into the cell. You can turn off validation on a per-cell basis.

The code editor cells in a notebook have no Gremlin session scope context and rely on an implicit scope based on their order. Validation occurs from the top cell down in the order in the notebook. If you execute code out of order, validation errors can occur even if the code executes successfully.

## Groovy language support

Supported:

- variable declarations in Groovy style:

```
def foo
def SomeType foo
SomeType foo
Shell-style variables
```

- method invocations
- optional semicolons to complete statements
- generics
- strings in both forms:

```
'123'
"123"
```

- for loops: basic and for in syntax
- while loops
- try-catch-finally: a type is required for catching the exception
- if-else statements
- switch statements
- casting in both styles: (SomeType) foo and foo as SomeType
- list literals

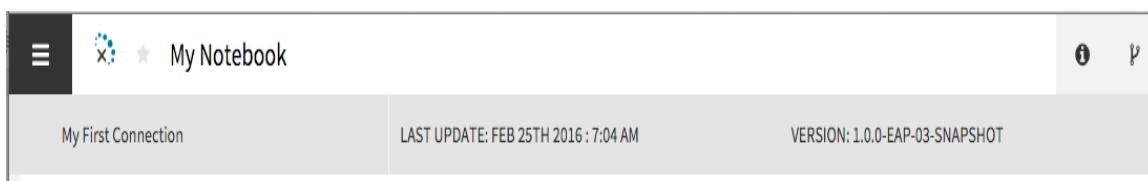
## Unsupported Groovy language

- closures
- multi-variable assignment and other advanced variable assignments like object deconstruction
- import statements explicitly disabled
- map literals or other complex type literals that are not supported by Java
- try-catch-finally: multi-catch is not supported

## Notebook information

Selecting **Information** displays:

- Connection being used for the notebook.
- Last update to the notebook.
- Version of Studio JAR file being run.



## Notebook schema

Selecting **Information** displays the current graph schema for the notebook.

The screenshot shows the DataStax Studio interface. On the left, there's a dark panel titled "product" containing the schema definition:

- ID Properties**

Name	Type
community_id	Int
member_id	Bigint

- Properties**

Name	Type	Cardinality	TTL
id	Int	Single	
name	Text	Single	

To the right, a graph diagram illustrates a relationship between "user" and "product". A blue arrow labeled "bought" points from "user" to "product".

In addition to visual schema representation, the code-assist feature also provides schema-assist

The screenshot shows the GREMLIN editor in DataStax Studio. The user has typed "g.V().has(" and is receiving code-assist proposals:

```
g.V().has(
 product
 product
 user
 user
 name
 name
 name
 name
```

A tooltip displays the details for the first proposal, "product":

Proposed Method	Description	Type
has('product')	has('product', String propertyKey, Object value)	Vertex Label
has('product')	has('product', String propertyKey, P predicate)	Vertex Label
has('user')	has('user', String propertyKey, Object value)	Vertex Label
has('user')	has('user', String propertyKey, P predicate)	Vertex Label
has('name')	has('name')	Vertex Property Key
has('name')	has('name', Object value)	Vertex Property Key
has('name')	has('name', P predicate)	Vertex Property Key
has('name')	has('name', Traversal propertyTraversal)	Vertex Property Key

proposals.

That's a lot of proposals. Let's break them down:

- The editor is intelligent enough to know that this is a Vertex-based traversal because of type inference. The editor presents only the schema proposals that are relevant for vertex properties and keys.
- Because there are multiple `has` methods, Studio proposes possibilities for all of these variations.
- The editor concretely makes a proposal for each possible property key.

## Using CQL in DataStax Studio

Interacting with data in a DSE cluster by writing and executing CQL code in a notebook.

The notebook tutorial **Working with CQL** is installed with Studio. The tutorial provides hands-on steps to create and interact with data in a DSE cluster by writing and executing CQL code in a notebook. This interactive notebook provides interactive steps to use CQL in DataStax Studio to:

- Select CQL as the language to enable intelligent editor features
- View the CQL schema

- Use CQL templates and content assist to help you form valid CQL statements:
  - # Propose valid CQL keywords
  - # Propose valid table names anywhere a table can be referenced in a CQL statement
  - # Propose column names anywhere a table can be referenced
- Identify errors with DSE domain-specific validations when CQL statements violate a constraint
- View results in table view or different styles of charts

You can use the keyspace menu to select the keyspace or specify the CQL [USE](#) statement. The following rules apply to USE statements in Studio:

- USE statements apply only to the current cell.
- A USE statement overrides the default keyspace setting for all statements that follow the USE statement within that cell.
- The keyspace selected in the keyspace drop down menu applies only to the current cell.
- The selected keyspace is copied to new cells created below an existing CQL cell.
- If a default keyspace is not selected, and there are no USE statements in the cell, all statements must be fully qualified to include the desired keyspace.

See the DataStax dev blog post [Studio Multi-Model CQL Support](#).

## Using Spark SQL in DataStax Studio

Writing, testing, and running Spark SQL queries against DSE clusters.

Analyze data stored in DSE clusters with Spark SQL relational queries. Spark SQL is a unified relational query language for traversing over distributed collections of data, and supports a variation of the SQL language used in relational databases.

Spark SQL notebook features include:

- Interactively perform Spark SQL queries against a DSE cluster
- Schema-aware content assist
- Syntax validations to facilitate faster prototyping

To run Spark SQL queries in Studio:

- The DSE cluster must be configured for the [AlwaysOn SQL \(page 245\)](#) service.
- Be familiar with the [Supported syntax of Spark SQL \(page 239\)](#).

If the [AlwaysOnSQL service \(page 245\)](#) is turned on, Studio uses the JDBC interface to pass queries to DSE Analytics. Two tables, `graphName_vertices` and `graphName_edges`, are automatically generated in the Spark database `dse_graph` for each graph, where `graphName` is replaced with the graph used for the Studio connection assigned to a Studio notebook. These tables can be queried with common Spark SQL commands directly in Studio, or can be explored with the [dse spark-sql \(page 235\)](#) shell. To learn more about using Spark SQL to query, see the [Using Spark SQL to query data \(page 235\)](#) documentation.

The notebook tutorial *Working with SparkSQL v6.0.0* is installed with Studio. The tutorial provides hands-on steps to create data and execute Spark SQL code in a notebook. Learn

about exploring the SQL schema in schema view, using content assist for syntax and domain validation. View results in table view and different styles of charts.

## Listing notebooks in DataStax Studio

Listing and filtering notebooks in DataStax Studio.

The Notebook Manager lists all notebooks, starred notebooks, or filtered notebooks.

1. Open the Studio server by entering `http://URI_running_DSE:9091/` in your web browser.

Card 1: Placeholder	Card 2: What's New in Studio 6.0.0	Card 3: Working with SparkSQL v6.0.0	Card 4: DSE SparkSQL QuickStart
+ Or drop files to import	What's New in Studio 6.0.0 default localhost 7 days ago	Working with SparkSQL v6.0.0 default localhost 7 days ago	DSE SparkSQL QuickStart SparkSQL 3 months ago
Card 5: Working with Graph v6.0.0	Card 6: Working with CQL	Card 7: DSE Graph QuickStart v2	Card 8: Working with Graph v2
Working with Graph v6.0.0 default localhost 3 months ago	Working with CQL Tutorial Connection 3 months ago	DSE Graph QuickStart v2 DSE Graph QuickStart 3 months ago	Working with Graph v2 Tutorial Connection 3 months ago

DataStax Studio opens to the Notebooks page.

2. To filter notebooks, click:

- All Notebooks
- Starred
- To filter by notebook name, start typing text in the Search Notebooks field.

The default notebook filter is Recent.

3. To return to the Notebook Manager from a notebook, select **Notebooks** from the menu (#).

## Using notebook history

Saving and restoring notebook revisions in DataStax Studio.

Notebook revisions provide a self-describing persisted state of a notebook. Major changes, like executing a cell and getting a new result, create a new revision immediately.

Changes to a notebook are automatically tracked with notebook revisions that provide a historical dated record with descriptions and change events that support easy retrieval. You can provide revision names and short comments to specific persisted states of a notebook to identify specific milestones in the history of a notebook. You can filter and retrieve notebook revisions. Notebook revisions are the historical versions of a notebook. When viewing notebook history, the notebook preview is read-only.

Events are saved only when the cell is executed. Automatic notebook revisions are saved before these events:

Event	Description
Cell Added	A cell is added to a notebook.
Cell Deleted	A cell is deleted from a notebook.
Cell Language	The cell language is changed: Markdown, Gremlin, CQL, or Spark SQL.
Cell Schema	A schema is created or modified.
Cell View	Changes are made in a cell with returned data.
Cell View Settings	The view settings are changed in a cell with returned data.
Code Changed	Code in a cell is changed.
Gremlin Execution Engine	The run configuration is changed.
Notebook Reverted	A notebook is reverted.

1. In any notebook, click  (View History) in the upper-right corner.

While viewing notebook history, the notebook is read-only.

2. The **Notebook History** panel displays on the right. The default view is to show all revisions in descending date order.

**Notebook History**

All    Named Revisions    All Changes    Search Revisions

**Current**

Feb 26, 2018 9:45 AM	result for count
Feb 26, 2018 9:45 AM	added new cell for count » Code Changed, 1 setting adjusted
Feb 26, 2018 9:45 AM	description » Cell Added
Feb 26, 2018 9:45 AM	description
Feb 26, 2018 9:45 AM	description
Feb 25, 2018 3:27 PM	changed cell view setting » 1 setting adjusted
Feb 23, 2018 1:19 PM	description

- To view revisions for a specific change event, click **All Changes #**, and select a change event.

For example, to view only the change events where code in a cell was changed, select **Code Change #**.

- To view all changes again, click **Code Change #** and select **All Changes** from the list of change events.
- To view only named revisions, click **Named Revisions**.

**Notebook History**

All    **Named Revisions**    All Changes    Search Revisions

**Current**

Feb 26, 2018 9:45 AM	result for count
Feb 26, 2018 9:45 AM	added new cell for count » Code Changed, 1 setting adjusted

All filters apply. For example, to view only named revisions with code changes, apply the **Code Change** filter and then click **Named Revisions**.

- To filter named revisions, enter text in the Search Revisions text box.  
The results are filtered as you type.
- To view a notebook revision, select a revision in the **Notebook History** panel.  
The notebook preview is read-only.
- To restore a notebook revision, click **Restore Revision**.  
The notebook history is updated.
- To delete a notebook revision, hover over the revision and click .

Feb 26, 2018 9:45 AM



10. A notebook revision is view-only. To update a notebook, click Cancel.

## Defining run behavior with execution configurations

Managing execution configurations to define run behavior for a code cell.

With results visualization and profiling capability, Studio is a powerful debugging tool by executing code in a way that reproduces the settings used in their applications. When using Studio to interact with DSE, you are able to execute code written in CQL, Gremlin, or Spark SQL. Each cell language has its own set of configuration options that determine execution behavior.

Execution configurations provide different execution setups by passing a set of options that customize the run execution. Execution configurations are persistent so you can reuse them.

Standard run configurations are provided:

Cell type	Execution configuration	Settings
CQL	CL.ONE	<ul style="list-style-type: none"> <li>Consistency level: ONE</li> <li>Timeout (MS): 10000</li> <li>Max Results: 5000</li> <li>CQL Tracing: Disabled</li> </ul>
CQL	CL.Quorum	<ul style="list-style-type: none"> <li>Consistency level: QUORUM</li> <li>Timeout (MS): 10000</li> <li>Max Results: 5000</li> <li>CQL Tracing: Disabled</li> </ul>
CQL	CL.ALL	<ul style="list-style-type: none"> <li>Consistency level: ALL</li> <li>Timeout (MS): 10000</li> <li>Max Results: 5000</li> <li>CQL Tracing: Disabled</li> </ul>
CQL	CL.ONE TRACE	<ul style="list-style-type: none"> <li>Consistency level: ONE</li> <li>Timeout (MS): 10000</li> <li>Max Results: 5000</li> <li>CQL Tracing: Enabled</li> </ul>
Gremlin	Transactional	Execute statement using real-time (transactional) engine for OLTP (online transaction processing), traversal source <code>g</code> .
Gremlin	Analytic	Execute statement using analytic engine (Spark) for OLAP (online analytical processing), traversal source <code>a</code> .

Cell type	Execution configuration	Settings
Spark SQL	Run	Execute statement with Spark SQL query engine with AlwaysOn SQL service.

You can create a custom CQL execution configuration by adjusting these settings for your environment:

- **Consistency level:** ANY, ONE, TWO, THREE, QUORUM, ALL, LOCAL\_QUORUM, EACH QUORUM, SERIAL, LOCAL\_SERIAL, LOCAL\_ONE
- **Timeout (MS):** *milliseconds*
- **Max Results:** *number of records*
- **CQL Tracing:** Enabled or Disabled

1. In the upper-right corner of a code cell, hover over the play button # and then click # to list the existing configurations.

**Tip:**

- In CQL cells, the default execution configuration is CL.ONE so the play button is **CL.ONE #**.
- In Gremlin cells, the default execution configuration is Execute using real-time (transactional) engine so the play button is **Real-time #**.
- In Spark SQL cells, all statements are executed with Spark SQL query engine with AlwaysOn SQL service.

2. Select **Manage Configurations**.
3. Select an existing configuration to view settings.
4. **For CQL cells only:** Click **+ Add New Configuration** to create an execution configuration.
  - a. Enter a self-describing **Name**.
  - b. Select a **Consistency Level**.
  - c. Enter a **Timeout** value in milliseconds.
  - d. Enter **Max Results**.
  - e. Select the **CQL Tracing** check box to enable CQL tracing.
5. Click **Save**.

New execution configurations are available to be selected from any CQL notebook cell.

## Exporting notebooks in DataStax Studio

Steps to export a notebook.

Export a notebook to a packaged compressed file. Options for notebook export are:

- Cell code and result sets

Export the entire notebook. Useful for collaborating.

- Cell code only

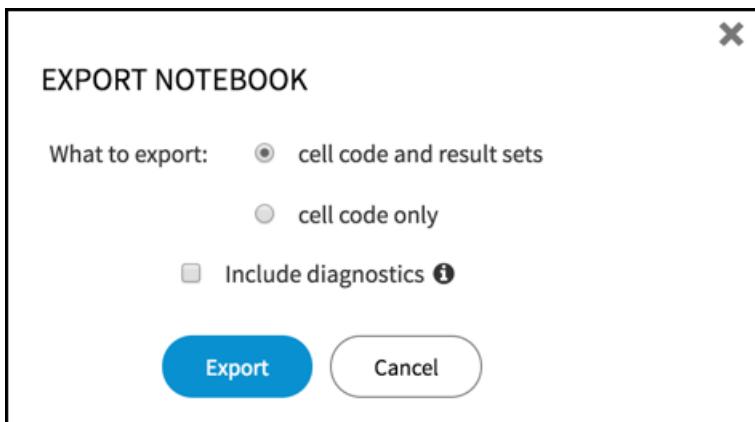
Export only the cell code of the notebook. Useful for creating a clean copy of the notebook. For example, for a tutorial.

- Include diagnostics

Export the notebook with diagnostic data for troubleshooting purposes. May include sensitive information. Credentials are not included.

#### 1. To export a notebook:

- In any open notebook, in the menu (#), select **Export this Notebook**.
- On the Notebook Manager page, click the ellipsis button (...) to show more actions for the notebook you want to export, and click **Export**.



#### 2. Select export options and click **Export**.

As appropriate, select to include cell code and results or cell code only. For either of these actions, you can also select to include diagnostics. Exporting a notebook with diagnostics might include sensitive information. Credentials are not included in the export.

The notebook is exported to the downloads location as a packaged compressed file. For example: DSE\_SparkSQL\_QuickStart\_6.0.0\_2018-02-28\_15\_03\_41.studio-nb.tar.

#### 3. Share the exported notebook using any file manager.

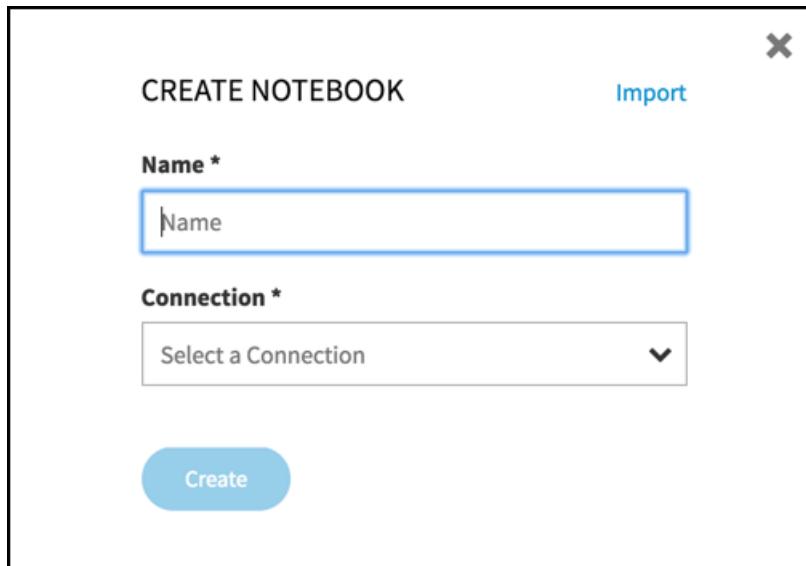
## Importing notebooks in DataStax Studio

Steps to import notebooks.

You can import a notebook using drag and drop or with the file system. The contents of the imported notebooks are defined during the export process. For example, a notebook that was exported with cell code only will be imported without result sets. If a notebook with an existing name is imported, duplicate notebook names are renamed (1), (2), and so on.

#### 1. Open the Notebook Manager page.

2. To import a notebook by dropping a packaged compressed file, open a file manager and drag the file to the Notebook Manager page.
3. To import a notebook using the Import dialog:
  - a. Click + on the empty notebook cell.
  - b. Click **Import** on the Create Notebook dialog.



- c. Navigate the file manager and select the notebook you want to import.
4. Select a connection for the notebook you are importing. Click **Save**.  
If the notebook is a graph notebook, the option to specify a graph is available if the connection is up for a graph-enabled DSE cluster.
5. The notebook is imported with the connection and graph. To change the connection or graph, select **Configure this Notebook** in the menu (#).

## Configuring Studio

Basic and advanced configuration for DataStax Studio.

Studio is a desktop application that automatically saves your work. The default configuration should not require adjusting.

### Configuring DataStax Studio

The default configuration should not require adjusting. You can configure different aspects of how Studio runs.

Studio is a desktop application that automatically saves your work. The default configuration should not require adjusting. However, you can configure different aspects of how Studio runs, including changing the httpBindAddress.

**Important:** Changing the httpBindAddress setting from the default (localhost) can pose a security risk as users on external machines can gain access to notebooks and

the DSE clusters those notebooks are connected to. Studio is designed to be used as a desktop application. Distributed deployment introduces potential security risks.

To change JVM settings, see [Specifying JVM settings for DataStax Studio \(page 899\)](#).

1. Open the `configuration.yaml` file in a text editor.
2. Make changes to any [basic \(page 895\)](#) setting.
3. If an [advanced \(page 897\)](#) setting is not in the file, add it.
4. Save the file.
5. Restart Studio to recognize the changes.

## configuration.yaml Studio configuration file

The DataStax Studio configuration file for basic configuration options.

The configuration file for Studio is `dse-studio-install-dir/configuration.yaml`.

A sample configuration file in XML format:

```
Maximum number of items returned per cell execution. Additional items
will be truncated.
Unit: count / number of items
resultSizeLimit: 1000

Maximum size of a cell result. If a cell result exceeds this size then
the cell execution will fail.
Unit: bytes
maxResultSizeBytes: 524288

Cell execution timeout. A value of 0 indicates no Studio-specific
timeout is applied. Instead,
use the DSE server timeouts that are configured in dse.yaml.
Unit: milliseconds
executionTimeoutMs: 0

Options to configure the Studio web server.
server:
 httpPort: 9091
 # WARNING: Changing the setting from the default (localhost) can pose a
 # security risk as other
 # users on external machines can gain access to notebooks and the DSE
 # clusters those
 # notebooks are connected to. Studio is designed to be used as a desktop
 # application.
 # Distributed deployment introduces potential security risks.
 # See http://docs.datastax.com/en/latest-studio/studio/reference/configuration.html for documentation.
 httpBindAddress: localhost

 # Studio logging options.
logging:
 fileName: studio.log
 maxLogFileSize: 250 MB
 maxFiles: 10
```

```

directory: ./logs
Spark SQL log level
0: Disable all logging
1: Log severe error events that cause the driver to stop
2: Log errors that may allow driver to continue
3: Log events that might result in an error
4: Log general driver progress information
5: Log detailed driver debug information
6: Log all driver activity
sparkSQLLogLevel: 0

The path to the local file system where user data is stored.
userData:
 # Defaults to .datastax_studio folder in your home directory, such as
 # ~/.datastax_studio
 # Set to a non-null value to override.
 baseDirectory: null

 # Time interval between revision saves when only minor changes are
 made.
 # For example, revision cell code and settings changes. Major changes,
 such as executing a cell
 # and getting a new result, always create a revision history unless the
 result is identical
 # to the prior values.
 historySaveFrequencyInSeconds: 300
 # Directory to store notebooks revisions that can be restored.
 historyDirectory: history
 # Enable Gzip compression of history files.
 compressHistoryFiles: true
 # Enable pruning of history revisions. When there are more than
 "minHistoryRevisionsToKeep"
 # history revisions, revisions older than "maxDaysOfHistoryToKeep" days
 are deleted.
 pruneRevisionHistoryEnabled: true
 # Maximum number of days to retain history revisions.
 maxDaysOfHistoryToKeep: 30
 # Minimum number of revisions to retain before pruning by date is
 enforced.
 minHistoryRevisionsToKeep: 25

```

## Cell options

### **resultSizeLimit**

Maximum number of items returned per cell execution. Additional items will be truncated. Default: 1000.

### **maxResultSizeBytes**

Maximum size of a cell result. If a cell result exceeds this size then the cell execution will fail. Default: 10485760.

### **executionTimeoutMs**

Cell execution timeout in milliseconds. A value of 0 indicates no timeout override. Uses the DSE server timeouts configured in the dse.yaml file. Default: 0.

## Studio web server

### **httpPort**

The port on which the Studio server is running. Default: 9091

### **httpBindAddress**

The IP address to which the Studio server is bound. Default: 127.0.0.1.

## Logging options

### **fileName**

Name of the log file. Default: studio.log.

### **maxLogFileSize**

Default: 250 MB.

### **maxFiles**

Maximum number of log files. Default: 10.

### **directory**

Path of the directory in which log files are stored. Default: ./log.

### **sparkSQLLogLevel**

Spark SQL log level. Default: 0

## User data

The path to the local file system where user data is stored.

### **baseDirectory**

Defaults to .datastax\_studio folder in your home directory, such as  
~/.datastax\_studio. Set to a non-null value to override. Default: null

### **historySaveFrequencyInSeconds**

Time interval between revision saves when only minor changes are made. For example, revision cell code and settings changes. Major changes, such as executing a cell and getting a new result, always create a revision history unless the result is identical to the prior values. Default: 300

### **historyDirectory**

Directory to store notebooks revisions that can be restored. Default:  
~/.datastax\_studio/history

### **compressHistoryFiles**

Enable Gzip compression of history files. Default: true

### **pruneRevisionHistoryEnabled**

Enable pruning of history revisions. When there are more than minHistoryRevisionsToKeep history revisions, revisions older than maxDaysOfHistoryToKeep days are deleted. Default: true

### **maxDaysOfHistoryToKeep**

Maximum number of days to retain history revisions. Default: 30

### **minHistoryRevisionsToKeep**

Minimum number of revisions to retain before pruning by date is enforced. Default: 25

## Advanced configuration options for DataStax Studio

Configuration options are not explicitly declared and set in configuration.yaml.

Advanced configuration options are not explicitly declared in the `configuration.yaml` file.

DataStax recommends contacting the DataStax services team before adding advanced configuration options. For the properties in each section, the parent setting has zero spaces. Each child entry requires at least two spaces. Adhere to the YAML syntax and retain the spacing.

## **userData**

### **connectionsDirectory**

The directory where connections are stored. Default: `connections`.

### **snapshotSaveIntervalInSeconds**

Default: 300.

### **entityCacheIdleTimeoutInSeconds**

Default: 3600.

### **maxKeyspaceSessionsPerConnection**

Maximum number of sessions associated with a specific keyspace to keep open at a time. Least recently used sessions are closed first. Default: 5.

### **eventReplayTimeoutInSeconds**

Default: 600.

### **eventReplayBatchSize**

Default: 10.

## **security**

To make encryption of passwords unique for your installation, you can change the password in this file. Use a strong generated password. DataStax recommends following security best practices, including avoiding simple words and phrases.

### **encryptionPasswordFile**

Default: `conf/security/security.properties`.

## **connectionManagement**

### **idleTimeoutInSeconds**

How long before an unused connection expires and is closed when it is not in use.  
Default: 3600 (1 hour).

## **Schema refresh**

### **schemaRefreshIntervalMs**

Schema refresh polling interval in milliseconds. Default: 3000 (3 seconds).

## **User data in DataStax Studio**

All about user data in DataStax Studio.

Studio notebooks, connections, and other user data are stored across multiple file locations. To use multiple instances of Studio, you can have more than one instance of the `.datastax_studio` user data directory.

**Warning:** The `.datastax_studio` is useful only to create backups of user data. Do not change any information or copy files from the user data directories.

## Specifying JVM settings for DataStax Studio

How to specify JVM command-line options for DataStax Studio.

Studio enables you to specify JVM command-line options for the local Studio server. Studio runs with these defaults:

- min heap 256 MB
- max heap 4 GB
- temp dir `/tmp`

The default values for Xmms (min heap), Xmx (max heap), and temp dir are expressed as:

```
export STUDIO_JVM_ARGS="-Xms256m -Xmx4g -Djava.io.tmpdir=/tmp"
```

Adjust as appropriate for your environment.

1. Create a file named `setenv.sh` for Linux operating systems (`setenv.bat` for Windows).
2. Add the content for the JVM arguments to pass to Studio.

The format of the content is:

```
export STUDIO_JVM_ARGS="JVM_options"
```

The default values for Xmms (min heap), Xmx (max heap), and tmpdir are:

```
export STUDIO_JVM_ARGS="-Xms256m -Xmx4g -Djava.io.tmpdir=/tmp"
```

For example, to change the maximum heap size to 8 GB:

```
export STUDIO_JVM_ARGS="$JVM_OPTS -Xmx8g"
```

3. Save the file in the same directory as the `Studio_server.sh` file.

For example: `datastax-studio/bin/setenv.sh`

4. [Restart \(page 876\)](#) Studio to use the JVM options.

## DataStax Studio Reference

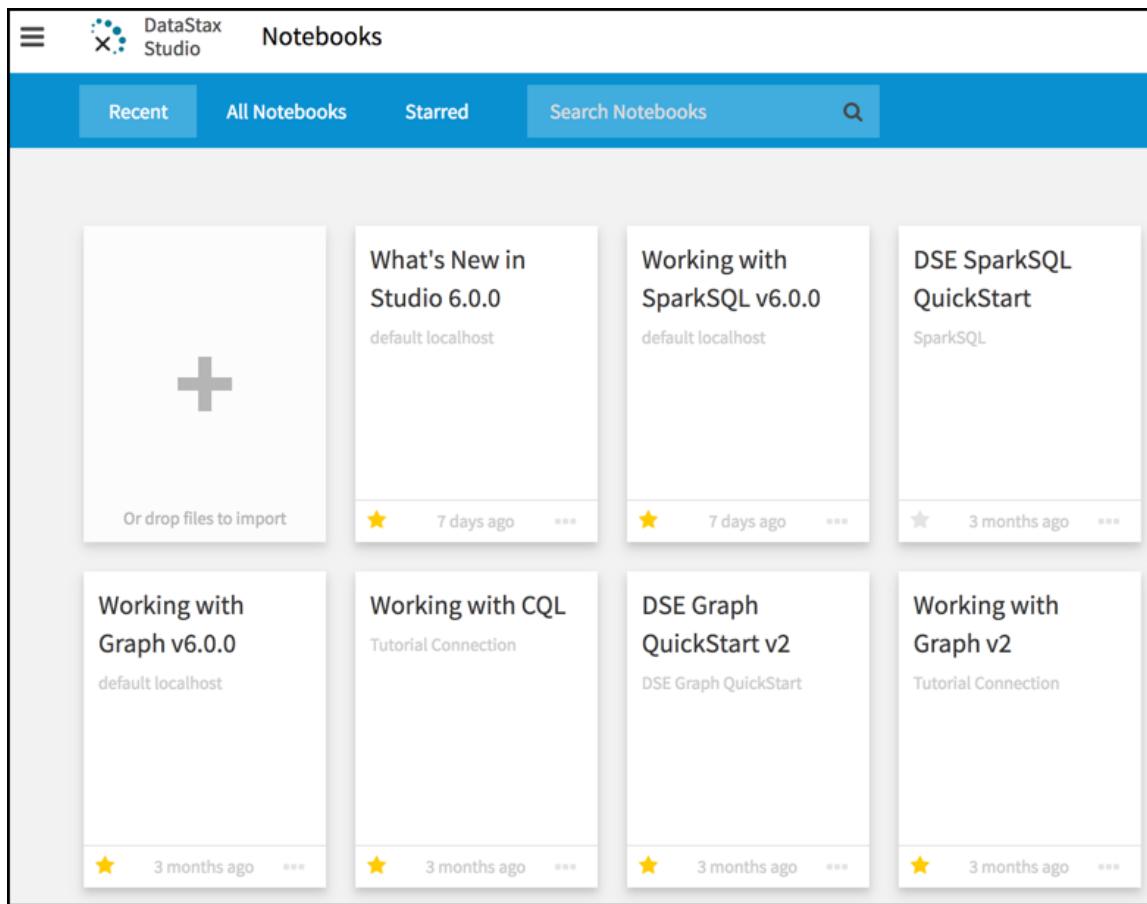
Reference information for DataStax Studio, including default imports and Notebook keyboard shortcuts.

## Creating a simple notebook in DataStax Studio

Steps to create a simple notebook in DataStax Studio.

Steps to create a notebook that contains text and runnable code. A notebook requires a name and a connection to a DSE cluster.

1. Open the Studio server by entering `http://URI_running_DSE:9091/` in your web browser.



DataStax Studio opens to the Notebooks page.

2. Click the plus (+) to add a notebook.
- The **Create Notebook** dialog displays.
3. Enter the notebook name.
  4. Select an existing connection or click **+ Add New Connection**.
  5. For a new connection, enter the connection information:

#### Name

Name of the connection from the notebook to a DSE cluster.

#### Host/IP (comma delimited)

The host names or IP addresses of the DSE cluster to connect to. Default: localhost.

#### Username

Optional. DSE username for logging in.

#### Password

Optional. DSE password for logging in.

#### Port

IP connection port. Default: 9042.

For example, to connect to a single-node DSE cluster on the local host using the default port.

**Name**  
My First Connection  
**Host/IP**  
127.0.0.1  
**Port**  
9091

6. Click **Test** to verify the connection works.
7. To configure a secure encrypted connection, select the **Use SSL** check box.

The Truststore and Keystore fields display. See [Using SSL connections in DataStax Studio \(page 878\)](#).

8. Click **Save**.

A connection is created from the notebook to the DSE cluster.

9. Select **Create**.

The notebook displays with a single empty (default) cell in **CQL** edit mode. This can be changed to Gremlin, SparkSQL, or Markdown.

The following steps show a Markdown example:

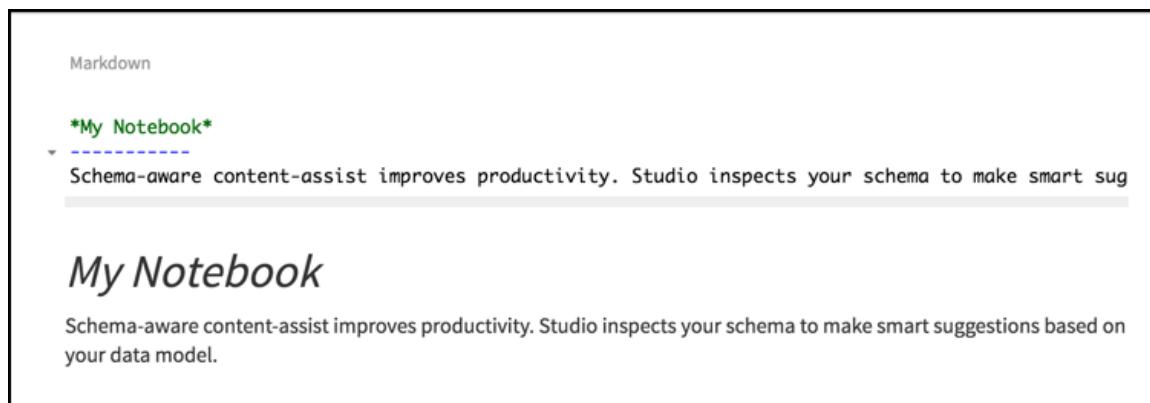
10. In the default cell, change the drop-down to **Markdown**.

11. Add some verbiage to the default cell:

```
My Notebook

Schema-aware content-assist improves productivity. Studio inspects your schema to make smart suggestions based on your data model. When you work with properties of a person, then you'll see suggestions for name and date_of_birth.
```

12. Select **Run Cell** to render the marked down text.



13. Create another Markdown cell.

## Configuring a notebook in DataStax Studio

Steps to change the name, connection, and graph of a Studio notebook.

You can change the name, connection, and graph of a notebook.

Each graph notebook is connected to a particular graph. See [DSE Graph QuickStart \(page 368\)](#).

A connection in Studio defines the graph and assigns a graph traversal *g* for that graph. A graph traversal is the mechanism for visiting each vertex in a graph, based on the filters defined in the graph traversal. To query DSE Graph, the graph traversal *g* must be assigned to a particular graph; Studio manages this assignment with connections.

**1. To configure a notebook:**

- On the Notebook Manager page, click the ellipsis button (...) in the notebook you want to configure.
- Open the notebook you want to configure.

**2. In the menu (#), select **Configure this Notebook**.**

**3. You can change the name of the notebook, the connection, and the graph.**

- The notebook name must be unique. The name field prevents duplicate names.
- Each notebook has only one connection. A connection can serve multiple notebooks.
- Each notebook is connected to a particular graph. Multiple notebooks can be connected to the same graph.

**4. Click **Save**.**

## Default gremlin imports in DataStax Studio

The default imports for DataStax Studio gremlin cells.

The following imports are performed by default for Gremlin cells. These imports are not configurable.

• **Static imports:**

```
org.apache.tinkerpop.gremlin.process.traversal.SackFunctions.Barrier.*
org.apache.tinkerpop.gremlin.util.TimeUtil.*
org.apache.tinkerpop.gremlin.structure.Direction.*
org.apache.tinkerpop.gremlin.process.traversal.Pop.*
org.apache.tinkerpop.gremlin.process.traversal.step.TraversalOptionParent.Pick.*
org.apache.tinkerpop.gremlin.structure.VertexProperty.Cardinality.*
org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.GraphTraversalSource.*
org.apache.tinkerpop.gremlin.process.traversal.P.*
org.apache.tinkerpop.gremlin.process.traversal.Order.*
org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.__.*
org.apache.tinkerpop.gremlin.structure.io.IoCore.*
org.apache.tinkerpop.gremlin.process.traversal.Scope.*
org.apache.tinkerpop.gremlin.structure.Column.*
org.apache.tinkerpop.gremlin.structure.T.*
```

```

org.apache.tinkerpop.gremlin.process.traversal.Operator.*

• Imports:

groovy.grape.Grape
org.apache.commons.configuration.*
org.apache.tinkerpop.gremlin.process.traversal.strategyverification.*
org.apache.tinkerpop.gremlin.process.computer.bulkloading.*
org.apache.tinkerpop.gremlin.process.computer.traversal.*
org.apache.tinkerpop.gremlin.util.function.*
org.apache.tinkerpop.gremlin.structure.io.*
org.apache.tinkerpop.gremlin.process.computer.ranking.pagerank.*
org.apache.tinkerpop.gremlin.groovy.loaders.*
groovy.json.*
org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.*
org.apache.tinkerpop.gremlin.structure.*
org.apache.tinkerpop.gremlin.process.traversal.strategy.decoration.*
org.apache.tinkerpop.gremlin.process.traversal.engine.*
org.apache.tinkerpop.gremlin.structure.io.gryo.*
org.apache.tinkerpop.gremlin.process.traversal.strategy.optimization.*
org.apache.tinkerpop.gremlin.process.traversal.step.util.event.*
org.apache.tinkerpop.gremlin.util.*
org.apache.tinkerpop.gremlin.structure.util.*
org.apache.tinkerpop.gremlin.structure.io.graphml.*
org.apache.tinkerpop.gremlin.process.computer.*
org.apache.tinkerpop.gremlin.process.traversal.strategy.finalization.*
org.apache.tinkerpop.gremlin.process.computer.clustering.peerpressure.*
org.apache.tinkerpop.gremlin.structure.util.detached.*
org.apache.tinkerpop.gremlin.structure.io.graphson.*
org.apache.tinkerpop.gremlin.process.traversal.*
org.apache.tinkerpop.gremlin.process.computer.bulkdumping.*
org.apache.tinkerpop.gremlin.process.traversal.util.*
org.apache.tinkerpop.gremlin.groovy.function.*

• Extra imports:

com.datastax.bdp.graph.api.*
com.datastax.bdp.graph.api.schema.*
com.datastax.bdp.graph.api.id.*
com.datastax.bdp.graph.api.config.*
org.apache.cassandra.db.marshall.geometry.*
com.datastax.bdp.graph.api.system.*
java.time.*
```

**Note:** To get a list from within a Gremlin cell:

```
DseGraphImports.getInstance().getAllImports()
```

## Keyboard shortcuts in DataStax Studio

Studio notebook keyboard shortcuts for edit mode and command mode.

Studio notebooks have keyboard shortcuts to increase your proficiency while writing code and content. Mode depends on cursor location:

- Edit mode when focus is on a cell editor.
- Command mode when focus is on the cell.

### Edit mode keyboard shortcuts

```
##+#
```

Shortcut	Description
Ctrl+Shift+?	Display <b>Shortcut help</b> .
Esc	Unfocus the editor and focus the cell (switch to <b>Command mode</b> ).
Shift+Enter	Save code and execute cell.
Ctrl+Space	Schema aware content assist.
Ctrl+L	Toggle line numbers.
Ctrl+/ (Macintosh OS X: #+/)	Toggle comment.
Ctrl+H	Hide the editor and focus on the cell (switch to <b>Command mode</b> ).
Ctrl+Alt+H	Add a new cell below and switch to its editor.
Ctrl+Alt+Shift+H	Add a new cell above and switch to its editor.
Ctrl+Shift+S	Toggle schema view.
Ctrl+Shift+Down (Macintosh OS X: #+#+Down)	Focus on the cell below, entering its <b>Command mode</b> .
Ctrl+Shift+Up (Macintosh OS X: #+#+Up)	Focus on the cell above, entering its <b>Command mode</b> .
Alt+Backspace (Macintosh OS X: #+Backspace	Delete from the cursor to the beginning of the line.
Alt+Delete (Macintosh OS X: #+Delete	Delete from the cursor to the end of the line.

## Command mode keyboard shortcuts

Shortcut	Description
Ctrl+Shift+?	Display <b>Shortcut help</b> .
Enter	Focus on the cell's code editor (switch to <b>Edit mode</b> ).
Delete (Macintosh OS X: fn+Backspace)	Delete the current cell and focus on next available cell.
H	Toggle editor visibility.
N	Add new cell below and switch to its editor.
Shift+N	Add new cell above and switch to its editor.
S	Toggle Schema view.
Up	Switch to the cell above.
Down	Switch to the cell below.
Shift+Up	Move current cell up.
Shift+Down	Move current cell down.

## Notebook cells in DataStax Studio

Description of notebook cells in DataStax Studio.

Notebook cells contain [markdown](#) text, CQL, Gremlin code, or Spark SQL. A notebook consists of a sequence of cells. A cell is a multi-line text input field.

Execute cell contents by:

- Pressing Shift + Enter with focus in the cell
- Clicking the Play button # on the right
- Selecting **Cell | Run** in the menu bar

The cell language determines the execution behavior of a cell:

- Gremlin

[Execution configurations \(page 891\)](#) define the run behavior.

- CQL

[Execution configurations \(page 891\)](#) define the run behavior. Custom run configurations are supported.

- Spark SQL
- Markdown

Markdown is a superset of HTML. Use Markdown to document the computational process in a literate way, alternating descriptive text with code. You can make text italic

or bold. You can build lists, and provide other structure including headings that link to other sections of the notebook. Click Play to display the formatted markdown text.

Every cell starts off being a code cell, but you can change its type by selecting a different cell language or by using [Keyboard shortcuts in DataStax Studio \(page 904\)](#).

## Notebook

Widget	Name	Description
	View Schema	Switch to Schema to view and interact with contextual views of data.
	View History	Display notebook history to view and manage <a href="#">notebook revisions (page 889)</a> .
	Add Cell	Add a cell in the current location.

## Cells

Widget	Name	Description
	Run Cell	Display the markdown or runs the Gremlin code and displays the results.
	Hide Code Editor	Hide the code editor.
	Disable editor validations	Toggle editor validations to off.
	Enable editor validations	Toggle editor validations to on.
	Maximize cell	Maximize the cell.
	More actions	Display more menu items for the cell.

## Gremlin results display

Display the results returned by the last Gremlin statement executed in a cell in these ways:

Widget	View description
	Raw
	Table

Widget	View description
	Pie chart
	Bar
	Line
	Area
	Scatter
	Graph
	Settings

## Schema view

Widgets for schema view:

Widget	Description
	Reset drop down selections
	Dock schema to the top
	Dock schema to the side

## DataStax Studio FAQ

Frequently asked questions about DataStax Studio.

### Frequently asked questions

#### Why can I configure more than one host in a Studio connection?

Hosts initialize the Cassandra driver connection. Configuring more than one host provides redundancy and failover protection.

#### Which web browsers are supported for Studio?

Studio is tested on [these platforms](#) (all 64-bit) with the latest versions of the specified web browsers.

#### How can I view a list of my notebooks?

Use the [Notebook Manager \(page 888\)](#) to list and filter notebooks.

### How do I specify a default keyspace for a cell?

You can use the keyspace menu to select the keyspace or specify the CQL [USE](#) statement. The following rules apply to USE statements in Studio:

- USE statements apply only to the current cell.
- A USE statement overrides the default keyspace setting for all statements that follow the USE statement within that cell.
- The keyspace selected in the keyspace drop down menu applies only to the current cell.
- The selected keyspace is copied to new cells created below an existing CQL cell.
- If a default keyspace is not selected, and there are no USE statements in the cell, all statements must be fully qualified to include the desired keyspace.

### Does Studio support connection to SSL-enabled clusters?

Yes. See [Using SSL connections in DataStax Studio \(page 878\)](#).

### Where are my notebooks from earlier versions?

When you start Studio, all of your existing notebooks are upgraded to the new version. Notebooks are not backward compatible.

## Troubleshooting DataStax Studio

Fixing problems in DataStax Studio.

Fixing problems in DataStax Studio.

See [Troubleshooting DataStax Studio](#).

# CQL

CQL (Cassandra Query Language) is a query language for the DataStax Enterprise database.

CQL (Cassandra Query Language) is a query language for the DataStax Enterprise database.

See [CQL for DataStax Enterprise 6.0](#).

# Warehouse - sstable tool

## Keyrefs

**PLEASE READ THIS** Create new command reference topics from the Reference.dita template.

Follow the wiki instructions for [command reference syntax](#) to ensure clean text output for the command line help.

## Command options and parameters

### **-b,--backups**

Include backups in the data directories (recursive scans).

### **-c cell\_count\_threshold,--min-cells cell\_count\_threshold**

Partition cell count threshold. When this threshold for cell count is exceeded, identify as a large partition.

### **-k partition\_keys,--key partition\_keys**

Include partition keys.

### **-m,--csv**

Instead of JSON formatted output, produce CSV machine-readable output.

### **-o tombstone\_count\_threshold,--min-tombstones tombstone\_count\_threshold**

Partition tombstone count threshold.

### **-r,--recursive**

Scan for SSTables recursively.

### **-s,--snapshots**

Include snapshots present in data directories (recursive scans).

### **-t partition\_count\_threshold,--min-size partition\_count\_threshold**

Partition size threshold.

### **-u,--current-timestamp**

Include timestamp in output. Timestamp is the number seconds since epoch, unit time for TTL expired calculation.

### **-x partition\_keys,--exclude-key partition\_keys**

Exclude partition key or keys from partition detailed row/cell/tombstone information.

Does not apply if -y option is given.

### **-y,--partitions-only**

Provide brief partition information only. Exclude per-partition detailed row/cell/tombstone information from process and output.