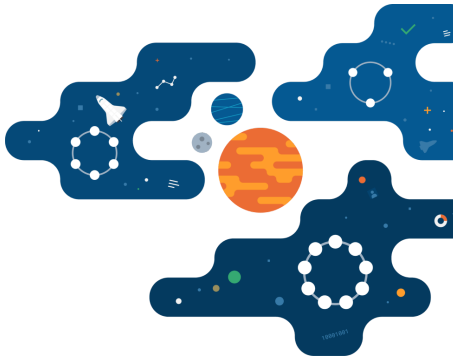
WHITEPAPER



Apache Cassandra™ Tools and Drivers

Everything you need to build and run business-critical applications

For business-critical applications, Apache Cassandra is the practitioners choice and thus Cassandra is among the most successful NoSQL databases today.

That said, the challenges of constructing globally-distributed applications, integrating rapidly expanding data volumes in modern architectures, and running the database at extreme scale are daunting.

Fortunately, the Cassandra community has been working relentlessly on an arsenal of tooling to simplify each stage of operating, integrating and developing applications with Cassandra. In this paper, tailored for a technical audience with a moderate experience with database systems, we discuss the range of utilities at your disposal for building and running successful Cassandra applications.

Engage

- Community: Forums, Slack, Mailing Lists
- DataStax Academy
- DataStax Luna Support

Deployment

- DataStax Desktop
- DataStax Astra
- Kubernetes
- Docker
- Automation Scripting

Development

- CQLSH
- DataStax Bulk Loader
- CQL Client Drivers
- Frameworks: Spring, Quarkus, Django, Express
- NSQLBench
- Fallout

Integration

- Spark Connector
- Kafka Connector

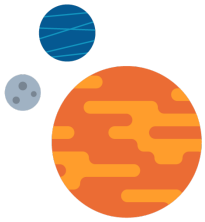
Operation

- JMX metrics
- Prometheus & Grafana
- Nodetool
- Reaper
- Medusa

DataStax

01

Engage



Community

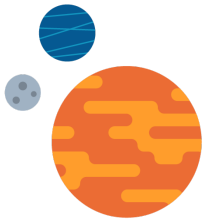
The most valuable resource for open source technology is the people that build and use it. The Cassandra community is one that shares their experiences, knowledge, and best practices to ensure that all users are successful. There are many ways to get in touch via ***forums, slack or mailing lists*** and self-paced learning available on ***DataStax Academy.***

Support

Have you ever got that call in the middle of the night because your production cluster was down? DataStax has you covered with a flexible, subscription-based support model for Cassandra called ***DataStax Luna.*** Luna has coverage for Cassandra versions 2.x and 3.x and Bronze, Silver, and Gold packages to fit your business and technical needs.

02

Deployment



On a Laptop

If you are able to install Docker on your local machine, the quickest way to get going locally with Cassandra is to use the cross-platform (Windows, MacOS, Linux) application called ***DataStax Desktop.*** This application provides a click-button experience to install and start Cassandra and comes with pre-packaged tutorials and samples to get off the ground in minutes.

If you are unable to install Docker on your local machine, head to the ***Cassandra Installation documentation*** to download the tarball and start the database. Note that the latest version of Java 8 is a dependency to run Cassandra and Python 2.7 or greater is required to run the command line shell, CQLSH.

In Any Cloud

Datastax Astra

To avoid the steps of manual installation and operation, consider ***DataStax Astra*** - a fully-managed Cassandra service complete with REST and GraphQL APIs for data access. There is a forever-free, no cost tier for exploring the service.

Deployment Tooling

Cassandra provides builds in ***tarball, Debian,*** and ***RHEL*** package formats. Manual installation is trivial for a single node but becomes difficult as the node count increases. There are many tools built by the Cassandra community that make multi-node deployments less of a burden for users.

Kubernetes

Kubernetes is the de-facto standard for managing containerized workloads and the Cassandra community is ***coming together*** to make Cassandra the default database for Kubernetes. There are many Cassandra Kubernetes operators in the ecosystem today, including an ***open source version*** built by DataStax, and now the community is joining forces to collaborate on a single, project-backed operator. Where Kubernetes goes, Cassandra will follow.

Docker

The Docker community maintains images for all Cassandra versions and these see hundreds of thousands of pulls per week. These images are used frequently for end-to-end testing in development and also in production deployments. For recommended production docker settings visit the ***DataStax documentation.***

Cassandra Docker Images >

Automation Scripting

There are a handful of utilities that automate the provisioning, configuration, and operations management of software packages including CStar, Ansible, Terraform, Puppet, Packer, and Chef. Over the years these have been customized in various forms to make Cassandra easier to deploy across environments. More often than not, there are changes required to tailor these scripts and tools to fit the requirements of a given set of infrastructure but the existing samples are helpful to bootstrap this process. Below we consolidate a group of user-provided templates that can be extended to work in your architecture.

CStar orchestration tooling, built by Spotify >

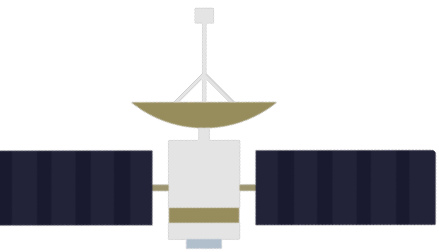
Terraform & Packer, Cassandra in AWS GitHub >

Terraform & Chef, Cassandra in AWS Blog Post: Easy Cassandra Scaling >

Cassandra Chef Cookbook GitHub >

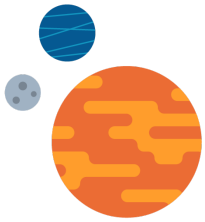
Puppet module for Cassandra GitHub >

Ansible playbook for Cassandra GitHub >



03

Development



CQLSH

CQLSH is a command line shell for interacting with Cassandra through CQL (Cassandra Query Language). This tool provides a useful interface for administrators and developers to access the database and issue ***CQL commands.*** CQLSH requires Python 2.7+ and is bundled with Cassandra in the /bin directory.

By default CQLSH connects to the Cassandra server running on the same machine. Starting the shell is as simple as executing <cassandra-install-dir>/bin/cqlsh or docker exec -it <container-name> cqlsh if using docker.

To connect to a remote machine with authentication enabled the following command line options are used.

```
cqlsh -u <username> -p <password> <ip-address>
```

The following output is expected when CQLSH connects to a Cassandra database. This was using Cassandra version 3.11.6 as shown in the second line.

```
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.6 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh> help

Documented shell commands:
==========================
CAPTURE  CLS          COPY  DESCRIBE  EXPAND  LOGIN   SERIAL  SOURCE   UNICODE
CLEAR    CONSISTENCY  DESC  EXIT      HELP    PAGING  SHOW    TRACING

CQL help topics:
================
```

```
AGGREGATES
ALTER_KEYSPACE
ALTER_MATERIALIZED_VIEW
ALTER_TABLE
ALTER_TYPE
ALTER_USER
APPLY
ASCII
BATCH
BEGIN
BLOB
BOOLEAN
COUNTER
CREATE_AGGREGATE
CREATE_COLUMNFAMILY
CREATE_FUNCTION
CREATE_INDEX
CREATE_KEYSPACE
CREATE_MATERIALIZED_VIEW
CREATE_ROLE
CREATE_TABLE
CREATE_TRIGGER
CREATE_TYPE
CREATE_USER
DATE
DELETE
DROP_AGGREGATE
DROP_COLUMNFAMILY
DROP_FUNCTION
DROP_INDEXDROP_KEYSPACE
DROP_MATERIALIZED_VIEW
DROP_ROLE
DROP_TABLE
DROP_TRIGGER
DROP_TYPE
DROP_USER
FUNCTIONS
GRANT
INSERT
INSERT_JSON
INT
JSON
KEYWORDS
LIST_PERMISSIONS
LIST_ROLES
LIST_USERS
PERMISSIONS
REVOKE
SELECT
SELECT_JSON
TEXT
TIME
TIMESTAMP
TRUNCATE
TYPES
UPDATE
USE
UUID
```

Bulk Loading and Unloading

The DataStax Bulk Loader (DSBulk for short) is a command line utility that enables fast loading and unloading of data from/to CSV or JSON files. Files, directories, stdin/stdout, and web URLs can be used for either source or destination and fields in the sources can be mapped to specific columns in the database. DSBulk is also used to efficiently count the number of rows in Cassandra tables and also the number of rows in the largest partitions which are operations that typically timeout when using CQL's SELECT COUNT(*) command.

Example Load:

```
> dsbulk load -h localhost -url /tmp/dsbulk.csv -k example_keyspace -t example_table
Operation directory: /tmp/logs/LOAD_20200218-120147-640593
total      | failed | rows/s  | mb/s | kb/row | p50ms | p99ms | p999ms | batches
12,610,407 | 0      | 150,328 | 3.02 | 0.25   |  9.84 | 39.58 | 45.88  | 200.25
Operation LOAD_20200218-120147-640593 completed successfully in 1 minutes and 27 seconds.
```

Example Unload:

```
> dsbulk unload -h localhost -url /tmp/dsbulk.csv -k example_keyspace -t example_table
Operation directory: /tmp/logs/UNLOAD_20200218-120147-465240
total      | failed |  rows/s | p50ms | p99ms | p999ms
12,610,407 |      0 | 102,478 |  6.77 | 97.52 | 480.25
Operation UNLOAD_20200218-120147-465240 completed successfully in 2 minutes and 2 seconds.
```

Example Count:

```
dsbulk count -h localhost -k example_keyspace -t example_table
Operation directory: /tmp/logs/COUNT_20200218-120147-307117.
total      | failed | rows/s   | mb/s | kb/row | p50 ms | p99ms | p999ms
12,610,407 |      0 |  99,238 | 2.09 |   0.53 |  41.55 | 41.68 |  41.68
Operation COUNT_20200218-120147-307117 completed successfully in 1 minutes and 57 seconds.
```

Download DataStax Bulk Loader >

Documentation: Bulk Loader Reference >

Blog Series: Migration techniques, advanced use cases, helpers for large data sets, uncommon data sources, and bulk operation automation >

Writing Applications

CQL Client Drivers

The primary way to access Cassandra is to use a client driver to connect to and query the database within the application. Cassandra drivers are available in all of the major programming languages and they communicate with the database over Cassandra's Native Protocol which is it's custom TCP interface built for concurrent, asynchronous requests. For information on the seven open source drivers that DataStax maintains, reference the ***DataStax Application Developer Guide*** and visit ***DataStax Examples*** for application code samples.

There are a set of best practices that apply to all Cassandra applications. See the Designing Fault Tolerant Applications with Cassandra ***whitepaper*** and ***demo*** for an in-depth dive on building reliable services.

- **Use a single driver session instance throughout the application**
  A single driver session instance can handle thousands of requests concurrently and creates long-lasting connections to each of the Cassandra machines in the data center with which it is connected.

- **Run queries asynchronously for higher throughput**
  Most Cassandra drivers have both synchronous and asynchronous APIs. The asynchronous APIs provide execution methods that return immediately without blocking the application's progress, allowing a single application thread to run many queries concurrently. Asynchronous execution methods return future objects that can be used by the application to obtain query results and errors if they occur. Running many queries concurrently allows applications to optimize their query processing, improves the driver's ability to coalesce query requests, and maximizes use of server-side resources.

- **Use prepared statements for frequently run queries**
  Cassandra and the drivers have a concept called ***"Prepared Statements"***. Preparing queries allows the server and driver to reduce the amount of processing and network data required to run a query. For prepared statements, the server parses the query once and it is then cached for the lifetime of an application. The server also avoids sending response metadata after the initial prepare step, which reduces the data sent over the network and the corresponding client side processing.

- **Have the application instance connect only to a single data center**
  It is recommended to pin an application to a single data center using the driver's configuration setting for "local data center". Failover is more robust when controlled at a layer above the application in the stack and cross-datacenter traffic often has higher latency and is more monetarily expensive than inter-datacenter traffic. This is covered in depth in the Designing Fault Tolerant Applications ***whitepaper*** and ***demo.***

  Some drivers try to infer the data center to connect to from the contact points configured in the application. Users have frequently been surprised by cross-data center traffic because it is easy to include contact points in remote data centers or invalid data centers. For example, an application might include contact points for an internal datacenter used during testing. Explicitly setting the local datacenter avoids these types of errors.

- **Avoid delete heavy workloads and writing nulls**
  In Cassandra, a tombstone is a marker that indicates that table data is logically deleted. DSE and Cassandra store updates to tables in immutable SSTable files to maintain throughput and avoid reading stale data. Deleted data, time-to-live (TTL) data, and null values will create tombstones, which allows the database to reconcile the logically deleted data with new queries across the cluster. While tombstones are a necessary byproduct of a distributed database, limiting the number of tombstones and avoiding tombstone creation increases database and application performance. Heavy deletes and nulls use extra disk space and decrease performance on reads. Tombstones can cause warnings and log errors. Deletes can often be avoided through data modeling techniques and can be avoided with proper query construction. See the Cleaning up Tombstones in Cassandra ***blog post*** for more.

- **Instrument the application with metrics**
  One useful best practice is to instrument the application with metrics related to executing queries to the database. This can greatly help when investigating query performance of your application. Information on metrics for the drivers that DataStax maintains can be found in the ***documentation.***

- **Supply multiple contact points**
  When configuring the connection with the driver, supply multiple contact points. With this, your application will still be able to make a connection even if one or more of the nodes is offline. There is no benefit to choosing or avoiding seed nodes. You should only supply contact points for a single data center, the data center that you will set as the local data center described in point 5 above. It is unnecessary to list all the nodes in a data center as contact points. Once the driver makes an initial connection, it will discover all of the rest of the nodes in the cluster and make direct connections to the nodes per the load balancing policy.

- **Beware of query idempotency**
  A CQL query is ***idempotent*** if it can be applied multiple times without changing the result of the initial application. Examples of operations that are not idempotent are adding to collections, incrementing counters, and queries that insert the result of non-deterministic functions such as now() and uuid(). The drivers assume that queries are not idempotent by default though this can be explicitly configured in the application.

- **Carefully craft batches**
  Batch statements are frequently used in Cassandra to handle writing to multiple denormalized tables. At the database level, when the Cassandra coordinator receives a batch it is responsible for dispatching all of the queries within the batch to the other nodes in the cluster. For the case when the initial batch execution was not successful, the batch will be saved and retried by the coordinator. Large batches can put harmful pressure on the coordinator nodes so it's advised to keep batches small, 20 statements is a good ballpark to have in mind as the upper bar. Batches can also be logged or unlogged and logged batches should only be used when the use case requires it.

- **Take the time to enable security**
  Cassandra ships with username / password authentication and SSL. Take the time to enable and configure these settings in your application to prevent headaches down the road. Prepared Statements also prevent against CQL injection attacks, which is another reason to use them for frequently executed queries. See the Securing Cassandra Clusters ***blog post*** for a step-by-step walkthrough.

LANGUAGE	GITHUB SOURCE CODE	DOCUMENTATION
Java	*Java GitHub >*	*Java Documentation >*
Python	*Python GitHub >*	*Python Documentation >*
Node.js	*Node.js GitHub >*	*Node.js Documentation >*
Go	*Go GitHub >*	*Go Documentation >*
C#	*C# GitHub >*	*C# Documentation >*
C++	*C++ GitHub >*	*C++ Documentation>*
PHP	*PHP GitHub >*	*PHP Documentation >*
Ruby	*Ruby GitHub >*	*Ruby Documentation >*
Scala	*Scala GitHub >*	*Scala Documentation >*

Frameworks

The Cassandra ecosystem is full of integrations with various frameworks and application toolsets. The most popular of those are detailed in this section.

- **Spring Framework**
  Spring is an ecosystem of Java projects that streamline development by providing boilerplate libraries and integrations for common application tasks such as security, cloud deployment, web, metrics, and data access. For Cassandra applications, the two widely used pieces are Spring Boot and Spring Data Cassandra. ***Spring Boot*** enables rapid bootstrapping of applications by taking care of the dependency management and functionality such as Actuator for metrics, Health Checks, and auto-configuration. ***Spring Data Cassandra*** is an opinionated take on data access that provides standard API constructs across data sources. DataStax actively collaborates with the Spring community and contributes to these utilities.

- **Quarkus**
  Quarkus is a full-stack, Kubernetes-native Java framework made for Java virtual machines (JVMs) and native compilation, optimizing Java specifically for containers and enabling it to become an effective platform for serverless, cloud, and Kubernetes environments. DataStax actively collaborates with the Quarkus community and authors a *Cassandra Quarkus extension* to make it easy for users to build applications with this stack.

- **Django Cassandra Engine**
  For Python developers, *Django Cassandra Engine* fast-tracks the construction of Cassandra-backed Python applications in Django projects. It leverages the *cqlengine Object Mapper* that ships with the Datastax maintained Python driver and is under active development at the time of this writing.

- **Express Cassandra**
  Any Node.js or JavaScript enthusiasts? *Express-Cassandra* automatically loads your models and provides an object-oriented mapping to Cassandra tables like a standard ORM/ODM. Built-in support for *Elassandra* and *JanusGraph* allows automatic management of synced Elasticsearch and JanusGraph indexes stored in Cassandra. The DataStax supported Node.js driver also *ships with an Object Mapper* for those that still want to interact more closely with their application models.

Relational Developers

Cassandra's query language CQL may seem similar to relational SQL but there are several key differences to understand. For more on those, check out the *Cassandra Data Modeling white paper* that goes into depth about the things to watch out for if transitioning your relational experience to Cassandra.

There are *JDBC and ODBC drivers* available for those comfortable with these standard programming interfaces. These drivers are backed by DataStax and Simba and provide Cassandra *access to tooling such as Tableau.*

Testing Applications

Continuous Integration

Despite that we want to believe that the software we author is bug-free, it's quite literally never the case. It's important to test the things that we build to guarantee that our users have the experience they deserve. Testing Cassandra-backed applications is tricky because the database is an external system and only so much of the application behavior can be isolated in unit tests. The Cassandra community is chipping away at making the data access code within applications faster to test while maintaining the realistic database interaction that the application codebase endures in production.

- **Cassandra Cluster Manager (CCM)**
  CCM is used thoroughly in the Cassandra project's testing framework and the goal of CCM is to make it easy to create, manage and destroy a small Cassandra cluster on a local machine. DataStax actively maintains and contributes to this python-based project.

- **TestContainers (Java)**
  TestContainers serve the main purpose of facilitating application integration tests that rely on external systems such as databases. It is a Java library that supports JUnit tests, providing lightweight, throwaway instances of common databases. Learn how to build more reliable and faster integration tests with Cassandra in this ***blog post.***

- **CassandraUnit (Java)**
  CassandraUnit is another Java test utility for writing isolated Java tests with JUnit, similar to that of DBUnit. This allows you to start an embedded Cassandra to test interactions with the mock database with defined schema's and datasets.

Performance

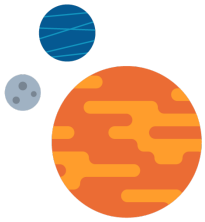
Whether you are determining how a new data model will behave, figuring out how many machines you will need to support your workload, or performance testing a new configuration, ***NoSQLBench*** provides a powerful toolkit to accurately test real-life traffic at scale. Define your schema and queries via a yaml file and NoSQLBench will take care of the rest with an efficient harness to adequately stress your Cassandra database deployment.

Reliability

Cassandra's masterless architecture is uniquely suited to handle all types of failures that happen frequently in modern computing environments. Though it is not enough to simply take this at face value and the most resilient systems require thorough testing, including chaos and resiliency scenarios. DataStax open sourced Fallout to bring operational, correctness, and scenario based testing to the hands of all practitioners. Based on Kubernetes and Jepsen, ***Fallout*** streamlines this type of validation testing at scale. The Last Pickle's ***tlp-cluster*** tool can also be used to deploy and test Cassandra clusters on Amazon Web Services.

03

Integration



Modern architectures are increasingly complex and often consist of many distinct technologies that are purpose-built for a given role within the system as a whole.

Today's flood of data gave rise to a group of new technologies that take advantage of the cutting edge techniques for data storage, processing, analytics and streaming. Apache Cassandra, Apache Spark, Apache Kafka, ElasticSearch, and Snowflake are the stack for today's digital age and there are tools to make these pieces fit seamlessly.

Spark Connector

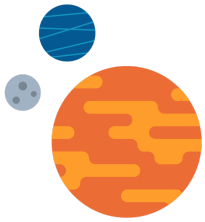
The ***Spark Cassandra Connector*** brings Spark and Cassandra together to serve the large scale analytics and distributed data needs of the data-driven enterprise. This library lets you expose Cassandra tables as Spark RDDs and Datasets/DataFrames, write Spark RDDs and Datasets/DataFrames to Cassandra tables, and execute arbitrary CQL queries in your Spark applications. With support for all major versions of both Spark and Cassandra, this connector is your go-to utility when using these technologies in a single architecture.

Kafka Connector

Cassandra is the perfect fit for event streaming data because it was built for the same high ingest rates that are common for streaming platforms such as Kafka. The DataStax sponsored ***Kafka Connector*** makes it easier to bring these two technologies together so that you can do all of your real-time streaming operations in Kafka and then serve your application APIs with Cassandra. This defines a future proof architecture that handles any needs that microservices and applications throw at it. The Kafka Connector has mapping functionality that lets you selectively choose the Kafka JSON or Avro fields that you want to write to columns in Cassandra. For the common denormalization design pattern, you can also choose to automatically write a single Kafka record to many tables in Cassandra. For ***JSON*** and ***Avro*** code samples see DataStax Examples.

04

Operations



Monitoring

Cassandra is built to handle data volumes that would topple most database systems but like any technology, it's important to be aware of the signals of struggle. Cassandra emits metrics via the ***Dropwizard Metrics*** library, similar to the Java driver. These can be queried via ***JMX*** or pushed to external monitoring systems using a number of ***built in*** and ***third party*** reporter plugins. It's up to the operator to observe and aggregate the critical metrics that each Cassandra node makes available. There are a wide range of ***commercial*** and open-source solutions available to monitor Cassandra clusters.

Prometheus & Grafana

A common solution for monitoring Cassandra clusters is to collect and consolidate database metrics with ***Prometheus*** and ***Grafana*** (Cloud-Native Computing Foundation project and member respectively). The Cassandra community built a ***Prometheus Exporter*** to streamline this integration and there are other actively maintained projects such as ***Seastat*** to make this more feasible for larger deployments. Cassandra consulting experts created ***pre-built dashboards*** for primary health metrics that surface the most important signals and alerts. These dashboards take advantage of ***DataStax's open source Metrics Collector*** which uses collectd to emit both system and Cassandra metrics.

Management

The same toolset used for deployment automation is frequently used for management operations. There are other operational actions with Cassandra that are good to scope off the bat.

Repair

Cassandra is a distributed database that relaxes the Consistency pillar of the ***CAP theorem.*** Because of this, data is continuously being synced in the background, transparent to the application developer. The process of syncing this distributed data is called Repair in Cassandra nomenclature. The leading Cassandra consultants open sourced ***Cassandra Reaper*** to reduce the knowledge-specific action required for Cassandra operators.

Backup & Restore

Every database requires the backup and restore of data to make sure that harmony can be restored in the system if things go wrong. Cassandra supports on-disk snapshots via the nodetool utility but this requires ssh access and delicate management of the files on disk. To make this a click button experience, the leading Cassandra consultants worked with Spotify to deliver ***Cassandra Medusa.***

Cassandra ships with a command line utility for all server operations called ***nodetool.*** This is found in the /bin directory and handles management activities such as getting the status of nodes in the cluster, scaling down clusters via decommission, and scaling up via rebuild. There is also a handy sjk command for diagnostics. For more, reference the documentation.