TopLevelPackages

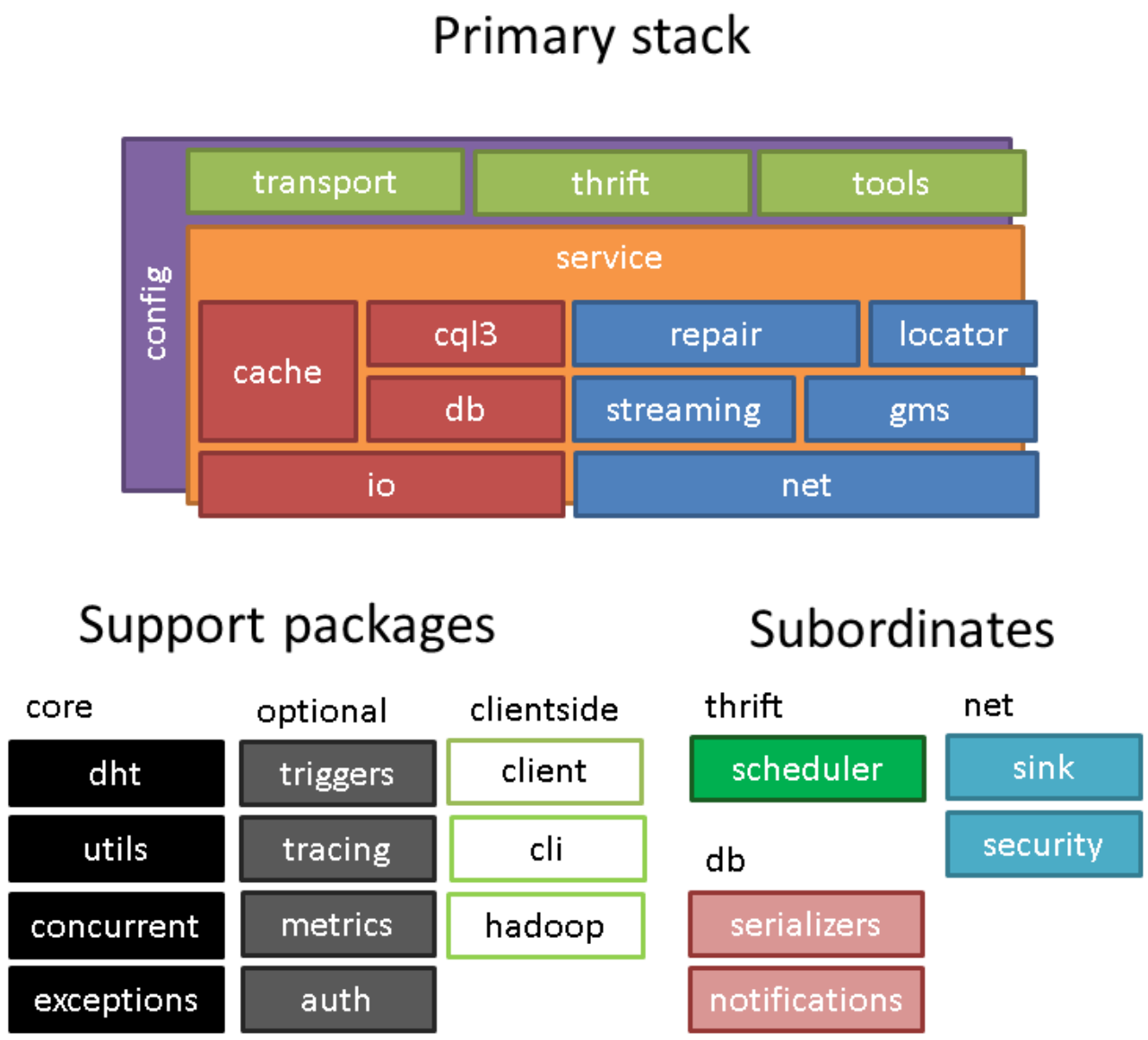
Top Level Packages

Current as of: v2.1 beta 2

Cassandra is written in Java, which uses packages to organize related functionality at the source code level. Understanding scope of individual packages can give insights into overall architecture.

Interpackage structure

Packages can be arranged in a sort of a stack representing their relationship to each other and how they fit into a whole.



(An editable .pptx source of this diagram is packageDiagram.pptx.)

The **primary stack** group includes packages that can be easily arranged in layers. The lower layers are closer to the operating system and hardware (more specifically, to their Java abstractions), the middle layers form the storage and distribution engine. The top layers are various interfaces exposing the service to external clients. A couple of omnipresent classes span all layers, and are deeply involved in all of them.

In terms of colors, *red packages* form the database substack. They are mainly responsible for handling data stored on the local node. *Blue packages* are the networking substack. These form what is commonly called the “Dynamo layer”. Their main responsibility is to distribute work and data among nodes forming the cluster. Even though these Dynamo packages are shown on the same level as the DB stack, most of them are aware of and depend on database classes. *Green packages* contain external-facing interfaces and tools through which a C* node communicates with the outside world. `service` and `config` packages get their own colors, because they are omnipackages that don’t fit into any single category.

Besides the primary stack, there are some support packages that don’t quite fit into the layer structure. They are best thought of as helper libraries that the primary stack calls into for certain tasks. **Core** support packages form a set of helpers that are essential to C* operation. **Optional** packages are responsible for nonessential features that are not always leveraged or enabled depending on cluster configuration. **Clientside** packages never run in the C* cluster at all; they are specialized client libraries or drivers that ship with Cassandra for convenience.

Finally, **subordinate packages** contain specialized helpers, that are really only used by a single top-level package. So they are best thought of as belonging to some other package, even though they appear on the top level for historical reasons.

Following is the list of all packages with along with a short description.

io

This large package talks to the file system on behalf of C*. The bulk of this work consists of creating and using `SSTables`, which is the format that C* uses to store data. Other responsibilities include on-disk compression as well as some general-purpose I/O functionality for other code to use, including facilities for custom memory management. The main class here is `SSTable`, which represents an abstract persistent container of sorted data. `SSTableWriter` and `SSTableReader` are derived from `SSTable` and expose additional read and write functionality, and are used quite extensively by other packages (primarily `db`).

db

This huge package takes up almost a quarter of the entire codebase and implements the database engine. It operates in familiar database terms including `Cells`, `Rows`, `ColumnFamilies` (tables) and `Keyspaces` (databases). `db` heavily relies on `io.SSTables` for data persistence, but also reaches into many other packages for various tasks. Internally, `db` can be broken down into its own sublayers and also contains multiple subpackages for things such as data marshalling, commit log management, storage compaction and others. Overall, `db` is large and complicated enough to deserve an architectural study of its own.

serializers

This small package is subordinate to `db`, and contains utility methods for converting primitive types to byte buffers.

notifications

This tiny package contains interfaces and classes that allow other code to hook certain internal `db` events with custom code. It can be used for things such as unit tests, but may also be hooked into with other external functionality.

cache

`cache` is a smaller package containing primitives used to implement key and row caching. The class that actually orchestrates all that caching activity (`CacheService`) lives in service, not `cache`. `cache` is primarily used for the benefit of `db`, but `db` almost never uses `cache` directly, instead proxying through `service.CacheService`. Without this extra indirection, `cache` could easily be structured as a subpackage under `db`.

cql3

Read and write APIs provided by `db` are difficult to use directly, so C* provides a query language for easier access to the underlying data–Cassandra Query Language or CQL. The language is implemented in its own package `cql3` (the third major release of the language). `cql3` defines the language grammar and implements the `QueryProcessor` as well as all the `Statements` and related functionality available in CQL. It is interesting that `cql3` is not a purely externally facing API; some internal code actually leverages it to store and retrieve system state information. In that respect, CQL is becoming a core component.

net

This package implements `MessagingService`, which abstracts away most networking machinery from the rest of the codebase. Other packages can then set up communication protocols represented by different `Verbs` and send custom `Messages` carrying those verbs to remote nodes. `Verbs` are handled on the receiving side with `VerbHandlers`. `net` provides only base types and functionality common to all messages; specialized implementation live in various other packages.

sink

Used by `net` and `service`, this tiny package is used to hook into messaging events. This is primarily useful for unit tests.

security

This tiny package, currently containing only one class `SSLFactory`, is used to encrypt communication over the network.

gms

`gms` (possibly standing for Gossip Message Service) implements the `Gossiper`. `Gossiper` is a peer-to-peer service that deals with disseminating cluster state information among member nodes. Gossiping consists of detecting unresponsive nodes using heart beat messages, and sharing liveness data among peers.

locator

`locator` is responsible for two separate tasks. One is discovering cluster topology through a pluggable component called `Snitch`. A few snitches are available out of the box, and some dynamic implementations heavily rely on `Gossiper` to detect up-to-date cluster topology. The second responsibility is deciding how to optimally distribute replicas based on discovered topology (handled by a class called `ReplicationStrategy` and its subclasses).

streaming

Another core networking package, `streaming` is responsible for moving bulk data between the nodes in the cluster.

repair

This smaller package deals with running `RepairSessions`, which redistribute data after a change in the cluster, or when corruption is detected in one of the existing nodes. Repair events are just one example where `streaming` is used.

service

`service`, although not the largest package, can be thought of as the skeleton upon which all other functionality builds. `service` consists of an executable class `CassandraDaemon` (this class contains the `main()` function of the C* daemon), along with a set of core services, including `StorageProxy` and `StorageService`. A lot of, if not most, of inter-package communication within the codebase is brokered through one of those two. `StorageService` is more involved in orchestrating Dynamo-level activities in the cluster, whereas `StorageProxy` is more focused on handling data transfer.

`service` is critical to most other modules, many of which are free to call into it from arbitrary places. A traditional weakness of such omnipresent uberpackages is that they attract all sort of miscellaneous functionality that doesn't seem to belong anywhere else, and `service` is no exception: expect to see a lot of random bits and pieces residing here.

config

Another omnipackage seemingly accessible from anywhere, `config` is a repository for configurable settings as well as a static entry point into the data store (through a class `Schema` which contains a reference to all keyspaces residing in the local cluster).

transport

This is one of external API providers. `transport` implements a Server that listens for connecting clients that want to use C* Native protocol, which as of Cassandra 2.0 is the primary communication protocol both for external clients and within the cluster.

thrift

A logical peer to transport, this package contains a server implementation for Thrift-based communication. This functionality may be deprecated in future versions, but for now it remains fairly popular in legacy deployments.

scheduler

A small package that is used by the Thrift server to schedule incoming requests to attempt a level of QoS.

tools

This package contains the implementation of several administrative utilities shipped with C*, including the `Node` tool, tools for import and export, as well as several utilities for `SSTable` maintenance. All these tools are available under `/bin` in a typical distribution.

dht

`dht` (Distributed Hash Table) is a core support class that is responsible for partitioning data among the nodes in the cluster. It contains several pluggable implementations of `AbstractPartitioner` class that handles the mechanics of data partitioning. In addition it defines `Range` and `Token`, which are primitives used by other packages to work with partition key ranges.

utils

This hefty package is a grab bag of miscellaneous classes typical to any software project, the proverbial "other" section. It is not the best place to look for architectural pillars, but it contains some clever code that is partially responsible for Cassandra's impressive perf and reliability, including implementations of `BloomFilter` and `MerkleTree` among others.

concurrent

`concurrent` deals with threading and thread pools. Interestingly, the few custom concurrency primitives that C* uses belong to a level 2 package under `utils`, and not to this package.

exceptions

This tiny package consists of a set of custom exception classes used in C*. It is not comprehensive: many custom exceptions belong to other packages whence they are thrown.

triggers

This package implements support for triggers, which are optional user-defined actions invoked during writes. Trigger support is not as comprehensive as some of the relational database engines, so the package is quite small.

tracing

`tracing` implements support for request tracing, whereupon some or all requests to Cassandra will cause verbose logging to be output for the purposes of debugging or performance tuning.

metrics

This package allows collecting quantitative data about various aspects of C* operation. Metric data can be accessed through `Node` tool that ships with C*. Like tracing, this capability can be important for operational maintenance and troubleshooting.

auth

`auth` enables support for authentication and authorization, providing a measure of access control for C* service.

cli

`cli` implements a Command Line Interface client for interacting with a C* cluster from a remote node.

hadoop

`hadoop` exposes C* in terms of MapReduce/Pig primitives, allowing integration with Hadoop clients. This package is expected to be used as an adapter in external Hadoop applications; there is no code here that actually runs server side.

client

A tiny package that provides helper functionality to code that runs against C* on the client side. Only `hadoop` uses `client` out of the box.

Packages listed by size

(Mar 2014)

db	1.5 M
cql3	650 K
service	570 K
io	530 K
utils	500 K
hadoop	300 K
tools	280 K
config	230 K
streaming	190 K
cli	190 K

thrift	180 K
transport	180 K
locator	140 K
gms	130 K
dht	100 K
net	100 K
repair	85K
auth	82K
metrics	74K
cache	55K
serializers	51K
concurrent	37K
exceptions	25K
tracing	15K
scheduler	13K
triggers	12K
notifications	8.0 K
security	5.8 K
sink	5.6 K
client	4.4 K

This page was originally adapted from here.