

Chapter 9: Broker Configuration

Reminiscing on our Kafka journey, so far we have *mostly* gotten away with running a fairly ‘vanilla’ broker setup. The exception, of course, being the tweaks to the `listeners` and `advertised.listeners` configuration properties that were explored in the course of [Chapter 8: Bootstrapping and Advertised Listeners](#).

As one might imagine, Kafka offers a myriad of configuration options that affect various aspects of its behaviour, ranging from the substantial to the minute. The purpose of this chapter is to familiarise the reader with the *core* configuration concepts, sufficient to make one comfortable in making changes to all aspects of Kafka’s behaviour, using a combination of static and dynamic mechanisms, covering a broad set of configurable entities, as well as targeted canary-style updates.

The knowledge gained in this chapter should be complemented with the official Apache Kafka online reference documentation available at kafka.apache.org/documentation¹³. There are literally hundreds of configurable elements described in rigorous detail in the reference documentation. Furthermore, each major release of Kafka often brings about new configuration and deprecates some of the existing. There is little value in regurgitating online reference material in a published book; instead, this chapter will focus on the fundamentals, which are incidentally least catered to by the online documentation.

Entity types

There are four *entity types* that a configuration entry may apply to:

1. `brokers`: One or more Kafka brokers.
2. `topics`: Existing or future topics.
3. `clients`: Producer and consumer clients.
4. `users`: Authenticated user principals.

The `brokers` entity type can be configured either statically — by amending `server.properties`, or dynamically — via the Kafka Admin API. Other entity types may only be administered dynamically.

Dynamic update modes

Orthogonal to the entity types, there are three *dynamic update modes* of configuration entries that affect broker behaviour:

¹³<http://kafka.apache.org/documentation>

1. `read-only`: The configuration is effectively static, requiring a change to `server.properties` and a subsequent broker restart to come into effect.
2. `per-broker`: May be updated dynamically for each broker. The configuration is applied immediately, without requiring a broker restart.
3. `cluster-wide`: May be updated dynamically as a cluster-wide default. May also be updated as a `per-broker` value for canary testing.

These modes apply to the `brokers` entity type. Each subsequent mode in this list is a strict superset of its predecessors. In other words, properties that support the `cluster-wide` mode, will also support `per-broker` and `read-only` modes. However, supporting `per-broker` does not imply supporting `cluster-wide`. As an example, the `advertised.listeners` property can be defined on a `per-broker` basis, but it cannot be applied `cluster-wide`.

For other entity types, such as `per-topic` or `per-user` configuration, the settings can *only* be changed dynamically via the API. The scoping rules are also different. For example, there is no concept of `per-broker` configuration for topics; one can target all topics or individual topics, but the configuration always applies to the entire cluster. (It would make no sense to roll out a topic configuration to just one broker.)



The term *dynamic update mode* might sound a little confusing, particularly for the `read-only` mode. The latter is effectively a static configuration entry defined in the `server.properties` file — it cannot be assigned or updated via the Admin API, and is anything but dynamic.

Dynamic configuration (which excludes the `read-only` mode) is persisted centrally in the ZooKeeper cluster. You don't need to interact with ZooKeeper directly to assign the configuration entries. (This option is still supported for backward compatibility; however, it is deprecated and its use is strongly discouraged.) Instead, the Kafka Admin API and the built-in CLI tools allow an authorized user to impart changes to the Kafka cluster directly, which in turn, will be written back to ZooKeeper.

The 'Broker configs' section in the official Kafka documentation indicates the highest supported dynamic update mode for each configuration entry.

The documentation has a separate '[Topic configs](http://kafka.apache.org/documentation.html#topicconfigs)'¹⁴ section, listing configuration properties that target individual topics. There is no dynamic update mode indicated in the documentation; in all cases, the update mode is either `per-topic` or `cluster-wide`.

Configuration precedence and defaults

There is a strict precedence order when configuration entries are applied. Stated in the order of priority (the highest being at the top) the precedence chain is made up of:

1. Dynamic per-entity configuration;

¹⁴<http://kafka.apache.org/documentation.html#topicconfigs>

2. Dynamic cluster-wide default configuration; followed by
3. Static read-only configuration from `server.properties`.

A seasoned Kafka practitioner may have picked up on two additional, easily overlooked, configuration levels which we have not mentioned — *default configuration* and *deprecated configuration*. The default configuration is applied automatically if no other configuration can be resolved. There is a caveat: A property may be a relatively recent addition, replacing an older, deprecated property. If the newer property is not set, Kafka will apply the value of a deprecated property *if one is set*, otherwise, and only then, will the default value be applied. (In fact, a configuration property may shadow multiple deprecated properties, in which case a more complex chain of precedence is formed.)

Configuration defaults are also specified in the official Kafka documentation page, but there is a snag: the default value often does not convey any practical meaning. For example, what does it mean when the default value of `advertised.listeners` is stated as `null`? In addition to viewing the default, one must carefully read the description of the property, particularly when a property replaces one or more deprecated properties. The documentation delineates the behaviour of an entity (broker, topic, etc.) when the property is not set.

Dynamic update modes were introduced in Kafka 1.1 as part of [KIP-226^a](#) (*KIP* stands for Kafka Improvement Proposal) to reduce the overhead of administering a large Kafka cluster and potential for interruptions and performance degradation caused by rolling broker restarts. Some key motivating use cases included —

- Updating short-lived SSL keystores on the broker;
- Performance-tuning based on metrics (e.g. increasing network or I/O threads);
- Adding, removing, or reconfiguring metrics reporters;
- Updating configuration of all topics consistently across the cluster;
- Updating log cleaner configuration for tuning; and
- Updating listener/security configuration.

The original proposal considered deprecating the static configuration entries in `server.properties` altogether for those settings that could be altered programmatically. In the course of analysis and community consultation, this measure was found to be too drastic and consequently rejected — entries in `server.properties` were allowed to stay, as it was considered useful to maintain a simple quick-start experience of working with Kafka without setting up configuration options in ZooKeeper first.

^a<https://cwiki.apache.org/confluence/display/KAFKA/KIP-226+-+Dynamic+Broker+Configuration>

Applying broker configuration

Static configuration

The static configuration for all Kafka components is housed in `$KAFKA_HOME/config`. A broker sources its read-only static configuration from `server.properties`.

With one exception, all entries in `server.properties` are optional. If omitted, the fallback chain comprising deprecated properties and the default value takes effect. The only mandatory setting is `zookeeper.connect`. It specifies the ZooKeeper connection string as a comma-separated list of `host:port` pairs, in the form `host1:port1,host2:port2,...,hostN:portN` — allowing for multiple redundant ZooKeeper connections.



Other than sharing the comma-separated `host:port` format, the ZooKeeper connection string has nothing in common with the bootstrap servers list used to configure Kafka clients. While a Kafka client employs a two-step process to connect to all brokers in a cluster, a Kafka broker needs only one connection to a ZooKeeper node and will establish it directly. Under the hood, ZooKeeper employs an atomic broadcast protocol to syndicate updates among all peer nodes, eliminating the need for every broker to connect to every node in the ZooKeeper ensemble.

Another crucial configuration entry in `server.properties` is `broker.id`, which specifies the unique identifier of the broker within the cluster. This is being brought up now because awareness of the broker IDs for nodes in a Kafka cluster is required to administer targeted configuration, where changes may be progressively applied to select subsets of the broker nodes.

If left unspecified or set to `-1`, the broker ID will be dispensed automatically by the ZooKeeper ensemble, using an atomically-generated sequence starting from `1001`. (This can be configured by setting `reserved.broker.max.id`; the sequence will begin at one higher than this number.)

It is considered good practice to set `broker.id` explicitly, as it makes it easy to quickly determine the ID by looking at `server.properties`. Once the ID has been assigned, it is written to a `meta.properties` file, residing in the Kafka logs directory. This directory is specified by the `log.dirs` property, defaulting to `/tmp/kafka-logs`.



While the naming might appear to suggest otherwise, the Kafka logs directory is not the same as the directory used for application logging. The logs directory houses the log segments, indexes, and checkpoints — used by the broker to persist topic-partition data. The contents of this directory are essential to the operation of the broker; the loss of the logs directory amounts to the loss of data for the broker in question.

To change the ID of an existing broker, you must first stop the broker. Then remove the `broker.id` entry from `meta.properties` or delete the file altogether. You can then change `broker.id` in

`server.properties`. Finally, restart the broker. Upon startup, the Kafka application log should indicate the new broker ID.

Dynamic configuration

Dynamic configuration is applied remotely over a conventional client connection. This can be done using a CLI tool — `kafka-configs.sh`, or programmatically — using a client library.



The following examples assume that the Kafka CLI tools in `$KAFKA_HOME/bin` have been added to your path. Run `export PATH=$KAFKA_HOME/bin:$PATH` to do this for your current terminal session.

We are going to start by viewing a fairly innocuous configuration entry — the number of I/O threads. Let's see if the value has been set in `server.properties`:

```
grep num.io.threads $KAFKA_HOME/config/server.properties
```

```
num.io.threads=8
```

Right, `num.io.threads` has been set to 8. This means that our broker maintains a pool of eight threads to manage disk I/O. We can take a peek into the broker process to see these threads:

```
KAFKA_PID=$(jps -l | grep kafka.Kafka | awk '{print $1}')
jstack $KAFKA_PID | grep data-plane-kafka-request-handler
```

```
"data-plane-kafka-request-handler-0" #50 daemon prio=5 os_prio=31 □
  cpu=672.65ms elapsed=18234.06s tid=0x00007faf670dc000 □
  nid=0x13b03 waiting on condition [0x0000700007836000]
"data-plane-kafka-request-handler-1" #51 daemon prio=5 os_prio=31 □
  cpu=671.82ms elapsed=18234.06s tid=0x00007faf670d9000
  nid=0xc803 waiting on condition [0x0000700007939000]
"data-plane-kafka-request-handler-2" #52 daemon prio=5 os_prio=31 □
  cpu=672.39ms elapsed=18234.06s tid=0x00007faf670da000 □
  nid=0xca03 waiting on condition [0x0000700007a3c000]
"data-plane-kafka-request-handler-3" #53 daemon prio=5 os_prio=31 □
  cpu=676.05ms elapsed=18234.06s tid=0x00007faf68802800 □
  nid=0xcc03 waiting on condition [0x0000700007b3f000]
"data-plane-kafka-request-handler-4" #72 daemon prio=5 os_prio=31 □
  cpu=104.53ms elapsed=16207.48s tid=0x00007faf67b5c800 □
  nid=0x1270b waiting on condition [0x0000700007f4b000]
```

```
"data-plane-kafka-request-handler-5" #73 daemon prio=5 os_prio=31 □
  cpu=97.75ms elapsed=16207.48s tid=0x00007faf67b5d800 □
  nid=0xe107 waiting on condition [0x0000700008866000]
"data-plane-kafka-request-handler-6" #74 daemon prio=5 os_prio=31 □
  cpu=96.60ms elapsed=16207.48s tid=0x00007faf650a8800 □
  nid=0x1320b waiting on condition [0x0000700008969000]
"data-plane-kafka-request-handler-7" #75 daemon prio=5 os_prio=31 □
  cpu=105.47ms elapsed=16207.48s tid=0x00007faf68809000 □
  nid=0x12507 waiting on condition [0x0000700008a6c000]
```

Indeed, eight threads have been spawned and are ready to handle I/O requests.

Let's use the `--describe` switch to list the dynamic value. The property name passed to `--describe` is optional. If omitted, all configuration entries will be shown.

```
kafka-configs.sh --bootstrap-server localhost:9092 \
  --entity-type brokers --entity-name 0 \
  --describe num.io.threads
```

Configs for broker 0 are:

It's come up empty. That's because there are no matching per-broker dynamic configuration entries persisted in ZooKeeper. Also, this tool does not output static entries in `server.properties` unless they have been overridden by dynamic entries.



A minor note on terminology: this book is aligned with the official Kafka documentation in referring to the 'name' part of the configuration entry as the 'property name'. The CLI tools that ship with Kafka refer to the property name as the 'key'. Working with Kafka, you will eventually become used to seeing the same thing being referred to by vastly different names. Even within the CLI tools, there is little consistency — for example, some tools refer to the list of bootstrap servers with the `--bootstrap-server` parameter, while others use `--broker-list`. To add to the confusion, some tools will accept multiple bootstrap servers as a comma-separated list, but still insist on the singular form of 'bootstrap server' (without the trailing 's'). For all the undisputed merits of Kafka, do try to keep your expectations grounded when working with the built-in tooling.

Let's change the `num.io.threads` value by invoking the following command:

```
kafka-configs.sh --bootstrap-server localhost:9092 \
  --entity-type brokers --entity-name 0 \
  --alter --add-config num.io.threads=4
```

Completed updating config for broker: 0.

Run the `kafka-configs.sh ... --describe ...` command:

Configs for broker 0 are:

```
num.io.threads=4 sensitive=false synonyms= []
{DYNAMIC_BROKER_CONFIG:num.io.threads=4, []
STATIC_BROKER_CONFIG:num.io.threads=8, []
DEFAULT_CONFIG:num.io.threads=8}
```

Not only is it now telling us that a per-broker entry for `num.io.threads` has been set, but it is also echoing the static value and the default.

View the threads again using the `jstack` command:

```
"data-plane-kafka-request-handler-0" #50 daemon prio=5 os_prio=31 []
  cpu=730.89ms elapsed=18445.03s tid=0x00007faf670dc000 []
  nid=0x13b03 waiting on condition [0x0000700007836000]
"data-plane-kafka-request-handler-1" #51 daemon prio=5 os_prio=31 []
  cpu=728.40ms elapsed=18445.03s tid=0x00007faf670d9000 []
  nid=0xc803 waiting on condition [0x0000700007939000]
"data-plane-kafka-request-handler-2" #52 daemon prio=5 os_prio=31 []
  cpu=729.16ms elapsed=18445.03s tid=0x00007faf670da000 []
  nid=0xca03 waiting on condition [0x0000700007a3c000]
"data-plane-kafka-request-handler-3" #53 daemon prio=5 os_prio=31 []
  cpu=737.62ms elapsed=18445.03s tid=0x00007faf68802800 []
  nid=0xcc03 waiting on condition [0x0000700007b3f000]
```

Bingo! The number of threads has been reduced, with the action taking effect almost immediately following the update of the dynamic configuration property.

Dynamic configuration is an impressively powerful but dangerous tool. A bad value can take an entire broker offline or cause it to become unresponsive. As a precautionary measure, Kafka caps changes of certain numeric values to either half or double their previous value, forcibly smoothening out changes in the configuration. Increasing a setting to over double its initial value or decreasing it to under half of its initial value requires multiple incremental changes.

Where a setting supports cluster-wide scoping, a good practice is to apply the setting to an individual broker before propagating it to the whole cluster. This is effectively what we've done with `num.io.threads`.

The `--entity-type` argument is compulsory and can take the value of `brokers`, `users`, `clients`, or `topics`. The `--entity-name` argument is compulsory for per-broker dynamic updates and can be replaced with `--entity-default` for cluster-wide updates. Let's apply the `num.io.threads` setting to the entire cluster.

```
kafka-configs.sh --bootstrap-server localhost:9092 \
  --entity-type brokers --entity-default \
  --alter --add-config num.io.threads=4
```

Then list the configuration:

```
kafka-configs.sh --bootstrap-server localhost:9092 \
  --entity-type brokers --entity-name 0 \
  --describe num.io.threads
```

Configs for broker 0 are:

```
num.io.threads=4 sensitive=false synonyms= []
{DYNAMIC_BROKER_CONFIG:num.io.threads=4, []
DYNAMIC_DEFAULT_BROKER_CONFIG:num.io.threads=4, []
STATIC_BROKER_CONFIG:num.io.threads=8, []
DEFAULT_CONFIG:num.io.threads=8}
```

The resulting list has an extra value: `DYNAMIC_DEFAULT_BROKER_CONFIG:num.io.threads=4`. As previously mentioned, the `cluster-wide` entries are second in the order of precedence, which is also reflected in the list above. When using the `--describe` switch, we can specify `--entity-default` to isolate cluster-wide configuration.

```
kafka-configs.sh --bootstrap-server localhost:9092 \
  --entity-type brokers --entity-default \
  --describe num.io.threads
```

Default config for brokers in the cluster are:

```
num.io.threads=4 sensitive=false synonyms= []
{DYNAMIC_DEFAULT_BROKER_CONFIG:num.io.threads=4}
```

Having applied the `cluster-wide` setting, and assuming the update has proved to be stable, we can now remove the `per-broker` entry. Run the following:

```
kafka-configs.sh --bootstrap-server localhost:9092 \
  --entity-type brokers --entity-name 0 \
  --alter --delete-config num.io.threads
```



Always remove per-broker settings once you apply the cluster-wide defaults. Unless there is a compelling reason for a per-broker setting to vary from a cluster-wide default, maintaining both settings creates clutter and may lead to confusion in the longer-term — when the cluster-wide setting is altered later down the track.

Having just deleted the `per-broker` configuration, presumably the only remaining entry is the `cluster-wide` one. Let's describe the configuration for broker 0:


```
kafka-configs.sh --bootstrap-server localhost:9092 \  
  --entity-type brokers --entity-name 0 \  
  --describe num.io.threads
```

Configs for broker 0 are:

This may leave the reader in a slightly puzzled state. Where did the `DYNAMIC_DEFAULT_BROKER_CONFIG` entry go? When invoked with the `--entity-name` parameter, `kafka-configs.sh` will only display an entry if a per-broker configuration has been assigned, irrespective of whether or not a cluster-wide entry has also been set. Running the command with the `--entity-default` switch still works as expected, showing the cluster-wide defaults:

```
Default config for brokers in the cluster are:  
  num.io.threads=4 sensitive=false synonyms= []  
  {DYNAMIC_DEFAULT_BROKER_CONFIG:num.io.threads=4}
```

Now that we're done with our examples, we can revert the configuration to its original state by deleting the cluster-wide entry:

```
kafka-configs.sh --bootstrap-server localhost:9092 \  
  --entity-type brokers --entity-default \  
  --describe num.io.threads
```

Applying topic configuration

Settings that apply broadly to all topics, as well as topic-wide defaults, can be edited using the static configuration in `server.properties` or via dynamic updates. In addition, some settings can be altered on a per-topic basis using just the dynamic approach.

The next example will tinker with another fairly benign setting — `flush.messages` — which controls the number of messages that can be written to a log before it is forcibly flushed to disk with the `fsync` command. Log flushing is disabled by default. (Actually, the value is set to a very high number: $2^{63} - 1$.)

Kafka requires that the topic exists before making targeted changes. We are going to create a test topic named `test.topic.config` for this demonstration. Run the command below.

```
kafka-topics.sh --bootstrap-server localhost:9092 --create \  
  --topic test.topic.config --replication-factor 1 --partitions 1
```

The next command will apply an override for `flush.messages` for the `test.topic.config` topic.

```
kafka-configs.sh --zookeeper localhost:2181 \  
  --entity-type topics --entity-name test.topic.config \  
  --alter --add-config flush.messages=100
```

Note: When referring to the topics entity type, you must substitute the `--bootstrap-server` argument with `--zookeeper`, specifying the host:port combination of any node in the ZooKeeper ensemble.



This is not a limitation of Kafka — the standard Admin API supports setting and querying topic-level configuration; rather, the limitation is with the CLI tools that seem to have taken a back seat to the rapid improvements in the Kafka wire protocol and client APIs. This limitation applies to Kafka 2.4.0 and earlier versions, verified at the time of writing. For those interested, a solution proposal is being floated in [KIP-248¹⁵](#). There is a sliver of hope that by the time this book falls into the reader's hands, the maintainers of Kafka will have addressed it.

We can now use the `--describe` switch to read back the configuration:

```
kafka-configs.sh --zookeeper localhost:2181 \  
  --entity-type topics --entity-name test.topic.config \  
  --describe flush.messages
```

```
Configs for topic 'test.topic.config' are flush.messages=100
```

It was previously stated that the entity name is an optional argument to the `--describe` switch. To further broaden the query to *all* topics, use the `--entity-default` switch, as shown below.

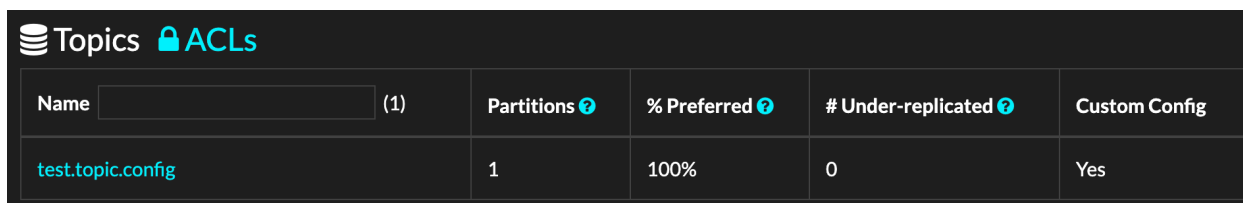
```
kafka-configs.sh --zookeeper localhost:2181 \  
  --entity-type topics --entity-default --describe
```

Similarly, using `--entity-default` with the `--alter` switch will apply the dynamic configuration defaults to *all* topics. For example, if we wanted to apply `flush.messages=100` to all topics, we could have run the following:

```
kafka-configs.sh --zookeeper localhost:2181 \  
  --entity-type topics --entity-default \  
  --alter --add-config flush.messages=100
```

¹⁵<https://cwiki.apache.org/confluence/display/KAFKA/KIP-248+-+Create+New+ConfigCommand+That+Uses+The+New+AdminClient>

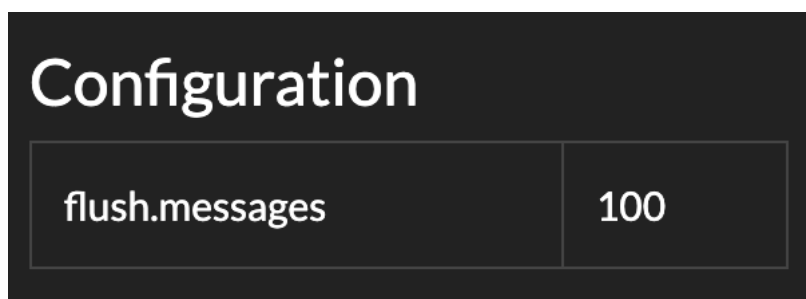
Using `kafka-configs.sh` is one way of viewing the topic configuration; however, the limitation of requiring ZooKeeper for topic-related configuration prevents its use in most production environments. There are other alternatives — for example, Kafdrop — that display topic configuration using the standard Kafka Admin API. Switch to Kafdrop and refresh the cluster overview page. You should see the recently-created `test.topic.config` topic appear in the topics list. Furthermore, there is an indication that a custom configuration has been assigned.



Name	Partitions	% Preferred	# Under-replicated	Custom Config
test.topic.config	1	100%	0	Yes

Kafdrop cluster overview, showing topic names

Click through to the topic. The custom configuration is displayed in the top-right corner of the topic overview screen.



Kafdrop topic overview, showing custom configuration

Having done with this example, we can revert the configuration to its original state by running the command below.

```
kafka-configs.sh --zookeeper localhost:2181 \
  --entity-type topics --entity-name test.topic.config \
  --alter --delete-config flush.messages
```

Users and Clients

The lion's share of configuration use cases relates to brokers, with the remainder mostly falling on topics. Users and clients are configured within the broader context of security and quota management — topic areas that will be covered separately in [Chapter 16: Security](#) and [Chapter 17: Quotas](#).

Although users and clients are not going to be covered in this chapter, their configuration closely resembles that of topics. In other words, they support two dynamic update modes: per-entity and

cluster-wide, with the per-entity taking precedence, followed by cluster-wide defaults, and finally by the hard defaults.

Kafka is a highly tunable and adaptable event streaming platform. Understanding the ins and outs of Kafka configuration is essential to operating a production cluster at scale.

Kafka provides a fair degree of scope granularity of an individual configuration entry. Configuration entries may apply to a range of entity types, including brokers, topics, users, and clients. Depending on the nature of the configuration, a change may be administered either statically — by amending `server.properties`, or dynamically — via the Kafka Admin API. A combination of whether remote changes are supported and the targeted scope of a configuration entry is referred to as its *dynamic update mode*. The broadest of supported dynamic update modes applies a cluster-wide setting to all entities of a given type. The per-entity dynamic mode allows the operator to target an individual entity — a specific broker, user, topic, or client. The read-only dynamic update mode bears the narrowest of scopes, and is reserved for static, per-broker configuration.

In addition to navigating the theory, we have also racked up some hands-on time with the `kafka-configs.sh` built-in CLI tool. This utility can be used to view and assign dynamic configuration entries. And while it is effective, `kafka-configs.sh` has its idiosyncrasies and limitations — for example, it requires a direct ZooKeeper connection for administering certain entity types.